

Вінницький національний технічний університет

Міністерство освіти і науки України

Кваліфікаційна наукова
праця на правах рукопису

МОРОЗОВ ОЛЕКСАНДР СЕРГІЙОВИЧ

УДК 004.8:004.9

ДИСЕРТАЦІЯ
ІНТЕЛЕКТУАЛЬНА ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ
АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-
ОРІЄНТОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Спеціальність 122 «Комп'ютерні науки»

Галузь знань 12 «Інформаційні технології»

Подається на здобуття ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилатись на відповідне джерело

_____ О.С.Морозов

Науковий керівник:

Яровий Андрій Анатолійович
доктор технічних наук, професор

Вінниця – 2026

АНОТАЦІЯ

Морозов О.С. Інтелектуальна інформаційна технологія автоматизованого тестування web-орієнтованого програмного забезпечення. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття ступеня доктора філософії за спеціальністю 122 «Комп'ютерні науки» (12 «Інформаційні технології»). – Вінницький національний технічний університет, Вінниця, 2026.

Дисертаційну роботу присвячено розв'язанню актуальної науково-прикладної задачі створення інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого програмного забезпечення, орієнтованої на підвищення стабільності, адаптивності та якості автоматизованих UI-тестів у сучасних динамічних веб-додатках.

Актуальність теми зумовлена зростаючою складністю web-орієнтованого програмного забезпечення інформаційних систем, активним використанням SPA-архітектур, динамічною модифікацією DOM-структур та широким впровадженням CI/CD-практик. За цих умов традиційні підходи до автоматизованого тестування, що ґрунтуються на ручному або напівавтоматичному створенні DOM-селекторів, виявляються недостатньо ефективними та призводять до появи нестабільних (flaky) тестів, значних витрат часу на супровід і зниження достовірності результатів тестування. Особливої складності набуває задача підтримки селекторів при частих змінах інтерфейсу, що зумовлює необхідність інтелектуальних методів аналізу та адаптації.

Метою дисертаційної роботи є підвищення стабільності та якості DOM-селекторів в автоматизованому тестуванні web-орієнтованого програмного забезпечення інформаційних систем на основі формалізованих метрик з використанням GNN-моделей.

Об'єктом дослідження є процес інтелектуального аналізу інформації щодо структурної організації та автоматизованого тестування web-орієнтованого програмного забезпечення інформаційних систем. Предметом дослідження є моделі, методи та інформаційна технологія інтелектуального аналізу DOM-

структури для побудови та оцінювання селекторів в інформаційних системах автоматизованого тестування.

У роботі запропоновано формалізацію DOM-структури у вигляді орієнтованого графа, де вузли відповідають HTML-елементам, а ребра відображають структурні батьківсько-дочірні та сусідні зв'язки. Для вузлів графа сформовано систему семантичних, атрибутивних і структурних ознак, що дозволяє комплексно описати контекст розташування елементів у DOM-дереві. На основі цього подання розроблено GNN-модель, здатну формувати контекстно-залежні векторні репрезентації DOM-елементів.

У межах запропонованої інформаційної технології реалізовано інтелектуальний механізм формування селекторів, що поєднує результати графової нейронної мережі (GNN) з агентом прийняття рішень. Агент послідовно формує селектор, використовуючи ознаки вузлів і структурні зв'язки, та оцінює його якість за сукупністю формалізованих метрик. Запропоновано систему кількісного оцінювання якості селекторів, що включає показники унікальності, точності (Precision), повноти (Recall), F1-міри та довжини селектора з урахуванням вагових коефіцієнтів для різних компонентів. Такий підхід дозволяє об'єктивно порівнювати різні стратегії побудови селекторів та забезпечує основу для адаптивного вибору найбільш стабільних селекторів.

Розроблено архітектуру інтелектуальної інформаційної технології, яка включає модулі побудови DOM-графа, екстракції ознак, графового навчання, інтелектуального агента та підсистему оцінювання якості селекторів. Проведено експериментальні дослідження на реальних web-ресурсах, що підтвердили здатність запропонованої технології забезпечувати більш стабільні та компактні селектори порівняно з традиційними ручними та автоматизованими підходами, а також зменшувати кількість нестабільних тестів і витрати часу на їх підтримку.

Основний науковий результат та новизна отриманих результатів полягає у формалізації процесу автоматизованого тестування web-орієнтованого програмного забезпечення інформаційних систем як задачі інтелектуального аналізу DOM-графа з використанням графових нейронних мереж; удосконаленні

системи метрик оцінювання якості селекторів з урахуванням їх стабільності та структурної складності; а також у розробці адаптивного підходу до формування селекторів, що враховує контекст елементів у DOM-структурі.

Практичне значення роботи полягає у можливості використання запропонованої інтелектуальної інформаційної технології як основи для створення інструментів підтримки автоматизованого тестування, зокрема у вигляді локальних утиліт або розширень для браузерів, що сприяють підвищенню надійності UI-тестів, зменшенню кількості flaky-тестів та скороченню витрат на супровід тестової інфраструктури інформаційних систем.

Ключові слова: інформаційні технології, штучний інтелект, розробка програмного забезпечення, інтелектуальні системи, нейронні мережі, тестування, веб-застосунок, рядки коду, DOM-структура, IT-проект.

ABSTRACT

Morozov O.S. Intellectual information technology for automated testing of web-oriented software. – Qualification scientific work on the rights of the manuscript.

Thesis for the degree of Doctor of Philosophy in the specialty 122 ‘Computer Science’ (12 ‘Information Technology’). – Vinnytsia National Technical University, Vinnytsia, 2026.

The dissertation is devoted to solving the urgent scientific and applied problem of creating an intellectual information technology for automated testing of web-oriented software, aimed at improving the stability, adaptability and quality of automated UI tests in modern dynamic web applications.

The relevance of the topic is due to the growing complexity of web-oriented software, the active use of SPA architectures, dynamic modification of DOM structures, and the widespread implementation of CI/CD practices. Under these conditions, traditional approaches to automated testing, based on manual or semi-automatic creation of DOM selectors, prove to be insufficiently effective and lead to unstable (flaky) tests, significant time spent on maintenance, and reduced reliability of test results. The task of maintaining selectors becomes particularly difficult with frequent interface changes, which necessitates intelligent methods of analysis and adaptation.

The aim of this thesis is to improve the stability and quality of DOM selectors in the automated testing of web-oriented software based on formalised metrics using GNN models.

The object of the study is the process of intelligent analysis of information regarding structural organisation and automated testing of web-based information system software. The subject of this study is the models, methods and information technology for the intelligent analysis of DOM structures, used to construct and evaluate selectors in automated testing information systems.

The work proposes the formalisation of the DOM structure in the form of an oriented graph, where nodes correspond to HTML elements, and edges represent structural parent-child and sibling relationships. A system of semantic, attributive, and

structural features has been formed for the graph nodes, which allows for a comprehensive description of the context of the location of elements in the DOM tree. Based on this representation, a GNN model has been developed that is capable of generating context-dependent vector representations of DOM elements.

Within the proposed intelligent information technology, an intelligent mechanism for forming selectors has been implemented, combining the results of a graph neural network (GNN) with a decision-making agent. The agent sequentially forms a selector using node features and structural connections and evaluates its quality based on a set of formalised metrics. A system for quantitative assessment of selector quality is proposed, which includes indicators of uniqueness, precision, recall, F1-measure, and selector length, taking into account weight coefficients for different components. This approach allows for an objective comparison of different selector construction strategies and provides a basis for the adaptive selection of the most stable selectors.

The architecture of intelligent information technology has been developed, which includes modules for building a DOM graph, feature extraction, graph learning, an intelligent agent, and a subsystem for evaluating the quality of selectors. Experimental studies have been conducted on real web resources, confirming the ability of the proposed technology to provide more stable and compact selectors compared to traditional manual and automated approaches, as well as to reduce the number of unstable tests and the time spent on their maintenance.

The main scientific contribution and novelty of the results obtained lies in the formalisation of the process of automated testing of web-oriented software as a task of intelligent analysis of the DOM graph using graph neural networks; improving the system of metrics for evaluating the quality of selectors, taking into account their stability and structural complexity; and developing an adaptive approach to the formation of selectors that takes into account the context of elements in the DOM structure.

The practical significance of the work lies in the possibility of using the proposed intellectual information technology as a basis for creating tools to support automated

testing, in particular in the form of local utilities or browser extensions that help to increase the reliability of UI tests, reduce the number of flaky tests, and reduce the cost of maintaining the test infrastructure information systems.

Keywords: information technology, artificial intelligence, software development, intelligent systems, neural networks, testing, web application, lines of code, DOM structure, IT project.

СПИСОК ПУБЛІКАЦІЙ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковані основні наукові результати дисертації:

1. Морозов О. С., Яровий А. А. Експериментальні дослідження підходів до генерації тестових селекторів із застосуванням GNN у процесі автоматизованого тестування вебдодатків. Український журнал інформаційних технологій. 2025, т. 7, № 2. С. 44-49, DOI: <https://doi.org/10.23939/ujit2025.02.044> (ISSN 2707-1898). **Фахове видання категорії Б**
2. Морозов О.С., Яровий А.А., “Аналіз підходів та експериментальні дослідження процесу оптимізації автоматизованого тестування програмного забезпечення із застосуванням паралелізму тестів”. Наука і техніка сьогодні, №9(50), 2025. С. 1361-1374, DOI: [https://doi.org/10.52058/2786-6025-2025-9\(50\)-1361-1374](https://doi.org/10.52058/2786-6025-2025-9(50)-1361-1374) (ISSN 2786-6025). **Фахове видання категорії Б**
3. О. Морозов і А. Яровий, «Інтелектуальна модель генерації адаптивних веб-селекторів на основі GNN», Оптико-електронні інформаційно-енергетичні технології, вип. 50, вип. 2, с. 142–149, Січ 2026. DOI: <https://doi.org/10.31649/1681-7893-2025-50-2-142-149> (ISSN 2311-2662). **Фахове видання категорії Б**
4. Морозов О.С., Яровий А.А., “Аналіз особливостей процесу автоматизованого тестування веб-орієнтованих програмних продуктів”, Наука і техніка сьогодні, № 3(44), 2025. С. 1330-1339, DOI: [https://doi.org/10.52058/2786-6025-2025-3\(44\)-1330-1339](https://doi.org/10.52058/2786-6025-2025-3(44)-1330-1339) (ISSN 2786-6025). **Фахове видання категорії Б**

Наукові праці, що засвідчують апробацію матеріалів дисертації:

5. Morozov Oleksandr, Yarovyi Andrii., METHODS OF EVALUATING SELECTOR LENGTH FOR AUTOMATED TESTING // Cambridge Science and Education Conference : proceedings of the International scientific and practical conference (February 23-25, 2026). – Cambridge, United Kingdom : naukainfo.com, 2026. - Pp. 108-113. - DOI: <https://doi.org/10.64828/conf-100-2026>
6. Морозов О.С., Яровий А.А., Петришин С.І., " Гібридні методи для вибору тестових селекторів у динамічних веб додатках", Молодь науці: дослідження, проблеми, перспективи (МН-2025), 2025. Тип доступу: <https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/23706/19578>
7. Морозов О.С., Яровий А.А., Козловський А.В., "Застосування машинного навчання для аналізу змін у веб-ресурса", LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2025), 2025. Тип доступу: <https://jpvrnd.conf.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23685/19566>
8. Морозов О.С., Яровий А.А., Козловський А.В., " Специфічні аспекти процесу автоматизованого тестування WEB-орієнтованого програмного забезпечення із застосуванням паралельності тестів на основі TestNG", Молодь науці: дослідження, проблеми, перспективи (МН-2024), 2024. Тип доступу: <https://jpvrnd.conf.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/20609/17035>
9. Морозов О.С., Яровий А.А., Козловський А.В., " Специфічні аспекти процесу автоматизованого тестування WEB-орієнтованого програмного забезпечення із застосуванням паралельності тестів на основі Selenium Grid", LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та

автоматизації (2024), 2024. Тип доступу:
<https://jpvrnd.conf.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20633/17062>

10. Морозов О.С., Яровий А.А., Козловський А.В. "Аналіз предметної області автоматизованого тестування WEB-ресурсів в контексті оптимізації процесів", Молодь в науці: дослідження, проблеми, перспективи (МН-2023), 2023. Тип доступу:
<https://d.conf.vntu.edu.ua/index.php/mn/mn2023/paper/viewFile/18706/15503>
11. Морозов О.С., Яровий А.А., "Інтегральна метрика оцінювання якості тестових селекторів у динамічних web-додатках", LV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2026), 2026. Тип доступу:
<https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2026/paper/view/28766/23649>

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	14
ВСТУП	15
РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ І ЗАСОБІВ, ПОСТАНОВКА ЗАДАЧІ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ОРІЄНТОВАНОГО ПЗ ІС НА ОСНОВІ ІНТЕЛЕКТУАЛЬНОГО ПІДХОДУ	21
1.1 Аналіз підходів до автоматизованого тестування web-ресурсів.....	21
1.2 Аналіз підходів до побудови DOM-селекторів.....	23
1.3 Аналіз проблемних аспектів підтримки селекторів та стабільності тестів	27
1.4 Постановка задач дослідження	32
1.5 Висновки до розділу 1	33
РОЗДІЛ 2 МЕТОДИ ТА МОДЕЛІ ІНТЕЛЕКТУАЛЬНОЇ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB- ОРІЄНТОВАНОГО ПЗ ІС	35
2.1 Аналіз моделі DOM як графової структури	35
2.2 Методи машинного навчання для роботи з графами	38
2.3 Ознаки DOM-елементів для GNN	41
2.4 Формалізація оцінювання якості селектора	45
2.5 Теоретичне обґрунтування підходу до побудови адаптивних селекторів.	48
2.6 Висновки до розділу 2	50
РОЗДІЛ 3 РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB- ОРІЄНТОВАНОГО ПЗ ІС	52
3.1 Основні етапи та загальна архітектура інтелектуальної інформаційної технології	52

3.1.1 Концептуальна модель інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого ПЗ ІС	53
3.1.2 Логічна модель інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого ПЗ ІС	55
3.1.3 Формування вхідних даних	56
3.2 Модуль побудови DOM-графа та екстракції ознак	57
3.3 Реалізація GNN-модуля	61
3.4 Алгоритм побудови селекторів	66
3.4.1 Характеристика основних принципів побудови селектора	66
3.4.2 Перевірка унікальності, скорочення довжини та складності селектора .	67
3.4.3 Інтеграція GNN.....	70
3.5 Інтелектуальний агент та його взаємодія з GNN-модулем.....	72
3.6 Підсистема оцінювання якості селектора.....	75
3.7 Обмеження та перспективи розвитку інтелектуальної інформаційної технології	77
3.8 Архітектура інтелектуальної інформаційної технології та інтеграція її модулів	80
3.9 Висновки до розділу 3	87
РОЗДІЛ 4 ЕКСПЕРЕМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ІНТЕЛЕКТУАЛЬНОЇ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ОРІЄНТОВАНОГО ПЗ ІС	91
4.1 Особливості організації експериментальних досліджень.....	91
4.2 Розробка методики проведення експериментальних досліджень.....	92
4.3 Налаштування та параметри експериментального середовища.....	95
4.4 Аналіз та підготовка вхідних даних для проведення експериментальних досліджень	98

4.5 Опис та порівняння підходів до формування тестових селекторів	100
4.6 Метрики оцінювання якості тестових селекторів.....	102
4.7. Аналіз результатів експериментальних досліджень	103
4.7.1 Програмна реалізація експериментальних досліджень	103
4.7.2 Результати експериментальних досліджень.....	106
4.8 Аналіз результатів впровадження	117
4.9 Висновки до розділу 4	118
ВИСНОВКИ.....	121
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	124
ДОДАТКИ.....	132
Додаток А Документи щодо впровадження результатів роботи	133
Додаток Б Основні лістинги програмного забезпечення.....	137
Додаток В Перелік експертів, які проводили оцінювання	150
Додаток Г Список публікацій за темою дисертації.....	151

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CI/CD – Continuous Integration/ Continuous Deployment

DOM – Document Object Model

ELU – Exponential Linear Unit

GAT – Graph Attention Network

GNN – Graph neural networks

HTML – HyperText Markup Language

QA – Quality Assurance

SPA – Single Page Application

ІС – Інформаційні системи

ПЗ – Програмне забезпечення

ВСТУП

Обґрунтування теми дослідження

Сучасна розробка програмного забезпечення (ПЗ), зокрема веб-ресурсів, характеризується зростаючою складністю, масштабованістю та високими вимогами до якості кінцевого продукту. Відповідно до цих викликів, забезпечення якості (Quality Assurance) стає все більш технологічно складним і вимагає впровадження сучасних інтелектуальних методів автоматизованого тестування. Межа між ручним і автоматизованим тестуванням поступово зникає, а інструменти та методики все активніше інтегруються у процеси розробки за принципами CI/CD та DevOps.

Попит на спеціалістів з автоматизованого тестування (Automation QA) щороку зростає [1]. За даними ресурсу DOU.ua, частка таких фахівців у структурі ІТ-ринку України стабільно зростає на 1-2% щорічно [2-4]. Це є наслідком стрімкого розвитку цифрових ІТ-проектів та інтеграції методів штучного інтелекту у процес створення інформаційних систем (ІС). За оцінками аналітиків, лише для економіки США перевитрати, пов'язані з дефектами у програмному забезпеченні інформаційних систем (ПЗ ІС), щороку сягають понад 60 мільярдів доларів. З метою мінімізації подібних втрат компанії впроваджують різні види тестування, комбінуючи ручні та автоматизовані підходи відповідно до потреб конкретної системи [5].

Автоматизоване тестування охоплює як серверну, так і клієнтську частини веб-додатків. У першому випадку – це модульне тестування, тестування API та баз даних; у другому – моделювання взаємодії з користувачем за допомогою спеціалізованих інструментів. У разі тестування графічного інтерфейсу веб-ресурсів особливе значення має побудова надійних селекторів DOM-елементів. Саме від якості побудови селекторів залежить коректність і стабільність автоматизованих тестів.

Основними проблемами при роботі з селекторами є значні часові витрати на їх створення, а також нестабільність при динамічних змінах DOM-структури. У більшості сучасних проєктів селектори створюються вручну, що знижує

масштабованість та стійкість тестів. Існують окремі утиліти для їх пошуку - від базового Chrome DevTools до розширень браузера (SelectorGadget, SelectorsHub), або інструментів, здатних записувати дії користувача (Katalon Recorder, Selenium IDE). Однак ці засоби не враховують структуру сторінки у цілому, не аналізують контекст DOM-елементів та не забезпечують побудову оптимізованих і стабільних селекторів.

Таким чином, актуальним є створення інтелектуальної інформаційної технології, здатної автоматично формувати стійкі селектори для елементів інтерфейсу з урахуванням особливостей їх позиціонування у структурі DOM. У якості інструменту розв'язання цієї задачі доцільним є застосування графових нейронних мереж (GNN), які дозволяють враховувати локальні та глобальні зв'язки у структурі сторінки. Цій задачі присвячена дана дисертаційна робота.

Зв'язок роботи з науковими програмами, планами, темами.

Дисертаційне дослідження проводилось згідно з планами науково-дослідних робіт кафедри комп'ютерних наук Вінницького національного технічного університету, в тому числі в межах кафедральних НДР 22 К1 «Розробка прикладних інтелектуальних інформаційних технологій та систем» (2022-2024) та 22 К2 «Моделювання складних систем та розробка прикладних інформаційних технологій» (2025-2029).

Мета і завдання дослідження. Мета дослідження - підвищення стабільності та якості DOM-селекторів в автоматизованому тестуванні web-орієнтованого програмного забезпечення інформаційних систем на основі формалізованих метрик з використанням GNN-моделей.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1. Провести аналіз існуючих аналогів та методів процесу автоматизованого тестування web-орієнтованого програмного забезпечення інформаційних систем, зокрема методів побудови DOM-селекторів.

2. Розробити метод автоматичного формування DOM-селекторів із використанням графового представлення структури HTML-документа та методів машинного навчання.
3. Реалізувати інтелектуальну інформаційну технологію автоматизованого тестування web-орієнтованого програмного забезпечення з можливістю побудови оптимізованих DOM-селекторів.
4. Провести експериментальне оцінювання ефективності запропонованого підходу на прикладі реальних web-ресурсів.

Об'єктом дослідження є процес інтелектуального аналізу інформації щодо структурної організації та автоматизованого тестування web-орієнтованого програмного забезпечення інформаційних систем.

Предметом дослідження є моделі, методи та інформаційна технологія інтелектуального аналізу DOM-структури для побудови та оцінювання селекторів в інформаційних системах автоматизованого тестування.

Методи дослідження: методи машинного навчання, методи графового моделювання структури DOM, експериментального моделювання та аналізу метрик якості, методи автоматизованого тестування ПЗ ІС, нейромережеві методи оброблення інформації.

Наукова новизна отриманих результатів. Під час розв'язання поставлених задач:

1. Вперше розроблено метод автоматичного формування DOM-селекторів, у якому реалізовано DOM-структуру у вигляді графа із подальшим вибором селектора на основі класифікації вузлів із використанням графових нейронних мереж, що забезпечило врахування структурного контексту елементів та підвищення стабільності селекторів.

2. Удосконалено модель оцінювання якості DOM-селекторів, яка відрізняється від існуючих інтеграцією різномірних метрик (точність, повнота, F1-міра, стабільність) в єдину інтегральну модель, що дозволяє комплексно оцінювати якість селекторів та підвищити ефективність їх автоматичного формування.

3. Удосконалено інтелектуальну інформаційну технологію автоматизованого тестування web-орієнтованого програмного забезпечення, яка відрізняється від існуючих інтеграцією GNN-модуля з модулем формування селекторів та удосконаленою моделлю оцінювання їх якості, що дозволяє здійснювати контекстно-залежну побудову селекторів і зменшити кількість нестабільних UI-тестів.

Практичне значення отриманих результатів роботи полягає у наступному:

- Розроблено алгоритм побудови DOM-графа, алгоритм екстракції семантичних та структурних ознак DOM-елементів, алгоритм формування векторних представлень вузлів DOM-графа на основі графової нейронної мережі, алгоритм автоматичного формування DOM-селекторів, застосування яких забезпечило досягнення середнього значення точності 88,0%, повноти 84,7%, F1-міри 86,3%, а також покращення інтегральної оцінки якості на 6,6% порівняно із ручним підходом (із суттєвим скороченням часу формування селектора) та на 18,9% порівняно із напівавтоматизованим підходом.
- Розроблено програмне забезпечення для формування DOM-селекторів у тестуванні веб-орієнтованих продуктів та інформаційних систем, використання якого дозволило після зміни DOM-структури зменшити кількість помилок, зокрема, у 4,4 рази порівняно із напівавтоматизованим підходом, а також забезпечити середній час формування селектора 60 мс, що є прийнятним з огляду на суттєве підвищення якості DOM-селекторів.

Окремі результати дисертаційної роботи впроваджено на підприємствах та закладах вищої освіти: ТОВ «Екзіп Текнолоджіз» (м. Вінниця, Україна, акт впровадження від 05.03.2026 р.) – моделі інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого ПЗ ІС; ТОВ «ВІН ІНТЕРАКТИВ» (м. Вінниця, Україна, акт впровадження від 06.04.2026 р.) – інтелектуальний агент та його взаємодія з GNN-модулем; у навчальний процес кафедри комп'ютерних наук Вінницького національного технічного

університету у межах дисципліни «Тестування програмного забезпечення» для студентів, які навчаються за освітньою програмою «Комп'ютерні науки» першого (бакалаврського) рівня вищої освіти спеціальності 122 «Комп'ютерні науки», що підтверджується актом впровадження від 10.04.2026 р. – метод автоматичного формування DOM-селекторів для web-орієнтованого ПЗ ІС.

Особистий внесок здобувача. Усі результати, які складають основний зміст дисертації, отримані здобувачем самостійно. У роботах в опублікованих у співавторстві, здобувачу належать: [1] – здійснено аналіз предметної області автоматизованого тестування веб-орієнтованого ПЗ ІС, виділено ключові компоненти та розглянуто доцільність їх графового подання; [5] - проведено аналіз особливостей процесу автоматизованого тестування веб-орієнтованого ПЗ ІС та обґрунтовано можливості підвищення стабільності та якості DOM-селекторів в автоматизованому тестуванні web-орієнтованого ПЗ ІС; [12] – досліджено підходи для підвищення ефективності автоматизованого тестування із застосуванням паралельного виконання тестів, що використано при удосконаленні архітектури інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого програмного забезпечення; [13, 14] – проаналізовано способи реалізації паралельного виконання автоматизованих тестів із використанням TestNG та Selenium Grid, результати чого враховано при удосконаленні архітектури інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого програмного забезпечення; [38] – розроблено модель генерації адаптивних веб-селекторів із формалізацією DOM-структури у вигляді графа та використанням графових нейронних мереж, а також вперше розроблено метод автоматичного формування DOM-селекторів; [50] – обґрунтовано гібридний підхід до вибору тестових селекторів у динамічних веб-додатках шляхом інтеграції графових нейронних мереж, класичних методів та статистичного оцінювання якості, що використано при удосконаленні архітектури інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого програмного забезпечення; [51] – проаналізовано підходи до застосування методів машинного навчання для

автоматизованого аналізу змін у веб-ресурсах, що обґрунтовує доцільність використання інтелектуальних методів автоматичного формування адаптивних DOM-селекторів; [65] - досліджено підходи, обґрунтовано доцільність застосування GNN для автоматичного формування DOM-селекторів на основі графового подання DOM-структури, удосконалено модель оцінювання якості DOM-селекторів; [66] - здійснено аналіз методів визначення довжини DOM-селекторів та запропоновано модель оцінювання як складової комплексного оцінювання якості тестових селекторів; [67] – удосконалено модель оцінювання якості DOM-селекторів шляхом інтеграції метрик точності, повноти, F1-міри та стабільності в єдину інтегральну модель.

Апробації матеріалів дисертації. Результати дисертаційної роботи доповідались та обговорювались на 7 науково-технічних конференціях: міжнародних наукових конференція «Молодь в науці: дослідження, проблеми, перспективи» (МН-2023, МН-2024, МН-2025), LV, LIV, LIII Всеукраїнських науково-технічних конференціях підрозділів ВНТУ (2024-2026), Cambridge Science and Education Conference : proceedings of the International scientific and practical conference (February 23-25, 2026).

Публікації. За темою дисертації опубліковано 11 праць, в тому числі 4 статті надруковано у наукових виданнях України, які входять до переліку фахових видань категорії Б, затверджених МОНУ, 7 робіт опубліковано у збірках матеріалів конференцій.

Структура та обсяг дисертації. Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. Основний зміст викладено на 128 сторінках друкованого тексту, містить 39 рисунків, 14 таблиць. Список використаних джерел містить 67 найменувань. Загальний обсяг 153 сторінок.

РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ І ЗАСОБІВ, ПОСТАНОВКА ЗАДАЧІ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ОРІЄНТОВАНОГО ПЗ ІС НА ОСНОВІ ІНТЕЛЕКТУАЛЬНОГО ПІДХОДУ

1.1 Аналіз підходів до автоматизованого тестування web-ресурсів

Автоматизоване тестування набуває дедалі більшого поширення в процесі розробки ПЗ ІС, особливо при розробці веб-ресурсів. Автоматизоване тестування має тривалу історію. На початкових етапах розвитку комп'ютерної індустрії функції тестування переважно виконували програмісти [6]. Та з часом професійна діяльність тестувальника сформувалася як окремий напрям у процесі розробки ПЗ ІС та почала розвиватись паралельно з розвитком розробки та масштабуванням продуктів.

Нині це стало значною сферою зі своїми правилами, технологіями, методами та власним інструментарієм для підвищення ефективності процесів тестування. Зараз можна виділити окремі напрямки спеціалістів з тестування: Manual QA Engineer, Automation QA Engineer, General QA Engineer, QA Analyst, Performance Engineer, Security Tester та інші. Кожен з них на різних рівнях оптимізує, пришвидшує та покращує свої процеси.

Якщо розглядати окремо спеціаліста з автоматизованого тестування, то основним їх інструментом є фреймворки для автоматичних тестів. Для розробки UI-тестів сучасних веб-застосунків можна виділити декілька з них: Selenium, Cypress, Playwright.

Selenium - це набір інструментів для автоматизованого тестування веб-інтерфейсів, що підтримує керування браузером через WebDriver та забезпечує кросбраузерне тестування [7, 8].

Cypress - інструмент для тестування веб-додатків, орієнтований на екосистему JavaScript; поєднує локальний раннер із хмарним дашбордом і поширюється за ліцензією MIT [9].

Playwright - фреймворк від Microsoft з відкритим кодом, який надає уніфікований API для керування основними браузерами й підтримує e2e-тестування веб-додатків [10].

Важливим завданням у сфері комп'ютерної інженерії є автоматизація процесу тестування [11]. Оскільки оптимізація є невід'ємною складовою розробки автоматизованих тестів та кожна команда хоче покращити свою якість продукту, розроблено низку методів, зокрема: паралелізація тестів, селективне виконання тестів, модульність тестів, використання стабільних тестових даних, інтеграція з CI/CD, моніторинг і аналітика, та інші [12-14]. Нижче узагальнено поширені підходи та інструменти:

Таблиця 1.1 - Огляд відомих підходів до оптимізації процесу автоматизованого тестування

Підхід	Опис	Інструменти
Паралелізація тестів	Запуск тестів одночасно в декількох потоках	Selenium Grid, TestNG, Docker, JUnit 5
Селективне виконання тестів	Запуск тільки тих тестів які змінились або відносяться до зміненого коду	Gradle Test Filtering, CircleCI Test Splitting, GitHub Actions Matrix.
Модульність тестів	Розбиття методів на незалежні частини	DRY, Page Object Model, Test Data Builders
Використання тестових даних	Використання підготовлених даних	Mocking, stubbing, тестові бази даних
Інтеграція з CI/CD	Автоматичний запуск тестів	Jenkins, GitLab CI, Azure DevOps
Моніторинг і аналітика	Аналіз пройдених тестів, їх тривалість, частоту падінь	ExtentReports, ReportPortal, Test Analytics dashboards.

Підходи постійно розвиваються і покращують свою стабільність та приносять багато користі для спеціалістів з автоматизованого тестування. Попри зазначені переваги залишається невирішеною проблема ефективної та стабільної взаємодії з DOM-елементами. Під час роботи з фреймворками для UI-тестів одним з головних процесів є використання селекторів для взаємодії з елементами браузера, але тестувальники постійно зіштовхуються з проблемами, пов'язаними з селекторами: нестабільність, динамічні об'єкти, важкі та довгі шляхи до них, багато витраченого часу на їх пошук.

Таким чином, аналіз існуючих фреймворків і підходів оптимізації показує, що незважаючи на високий рівень автоматизації процесів тестування, проблемні аспекти ефективної та стабільної взаємодії з DOM-елементами через селектори залишається актуальною. Жоден із існуючих підходів не забезпечує достатнє покращення процесу побудови селекторів з урахуванням їх структурного контексту, що знижує стабільність і ускладнює підтримку тестів.

1.2 Аналіз підходів до побудови DOM-селекторів

Селектор – це формалізований вираз (наприклад, CSS або XPath), який використовується для однозначної ідентифікації та пошуку одного або кількох DOM-елементів у структурі веб-сторінки з метою взаємодії з ними в процесі автоматизованого тестування [6]. Приклад простого селектору:

```
div[id='vntu-title'] h1
```

Побудова DOM-селекторів є ключовим етапом у створенні UI-тестів веб-застосунків, оскільки від їх якості залежить стабільність, точність та підтримуваність автоматизованих тестів. У цьому підрозділі наведено класифікацію підходів до створення селекторів, аналіз їх переваг і недоліків, а також огляд наявних інструментів для ручної та автоматизованого формування селекторів. Представлено аналіз основних підходів до побудови селекторів в таблиці 1.2.

Пошук за атрибутом id є одним із найбільш ефективних з точки зору швидкості й стабільності. Однак, на практиці такі елементи зустрічаються рідко, або мають динамічну природу. Найчастіше спеціалісти з автоматизованого тестування використовують CSS-селектори та XPath, оскільки ці селектори дозволяють комбінувати різні атрибути, зокрема id, class, text та інші.

Таблиця 1.2 - Огляд підходів пошуку елементів

Підхід	Переваги	Недоліки
Css-селектори	- Швидкі та ефективні - Підтримуються всіма браузерами - Гнучкий	- Неможливість пошуку по тексту - Іноді важко створити унікальний селектор
Xpath	- Підтримує навігацію по Dom-дереву - Працює з текстом або частинами тексту	- Повільний - Складніший у написанні - У деяких браузерах можуть виникати проблеми
Id	- Найшвидший - Легко читається та підтримується	Не завжди доступний або стабільний – може бути динамічним
Class	- Зручний у використанні - Легко читається та підтримується	- Неунікальні - Можуть бути зашифровані
Tag name	Зручний у використанні для пошуку всіх елементів одного типу	Дуже загальні - часто потребують додаткового уточнення
Text	Зручний для пошуку елементу за текстом або його частиною	- Не надійний - текст може змінюватись. - Не підтримується при локалізації

До основних характеристик якісного селектора належать унікальність, довжина та стабільність. Ефективний DOM-селектор має відповідати трьом основним критеріям: унікальність, мінімальна довжина та стабільність.

Унікальність селектору відповідає за точність знаходження його серед всіх елементів в DOM-дереві. При невідповідності цього параметру, елементів може бути декілька, що в свою чергу може вплинути на якість і правильність виконання тесту або тест може не пройти зовсім.

Довжина селектору впливає на два моменти: швидкість виконання тесту, читабельність та легкість підтримки цих тестів. Зі збільшенням довжини DOM-селектора зростає ймовірність того, що він втратить актуальність через зміни в DOM-структурі.

Стабільність селекторів впливає на якість проходження тестів. Якщо DOM-селектор є не стабільним, то збільшується шанс на flaky-тести – це тести, результати яких є нестабільними при повторних запусках без зміни програмного коду або тестових даних. Нестабільні тести не дають чітких результатів та не дають чітких вказівок на наявність програмних помилок [15].

Існують деякі підходи для формування селекторів: ручні, автоматичні, напівавтоматичні. Найпоширенішим є ручний підхід, оскільки він дозволяє створити селектор, що задовольняє всі вимоги щодо унікальності, стабільності та відповідності структурі DOM. Формування DOM-селекторів може займати значну частину часу при створенні автоматизованого тесту.

Для автоматичних і напівавтоматичних підходів використовуються додаткові інструменти або інструменти вже інтегровані браузером. Серед них можна виділити наступні: SelectorGadget, SelectorsHub, SeleniumIDE, Chrome DevTools.



Рисунок 1.1 – Інтерфейс та знайдений селектор в SelectorGadget

SelectorGadget - це інструмент з відкритим кодом, який спрощує створення та пошук селекторів CSS [16, 17]. Основним обмеженням інструменту є відсутність активної підтримки та адаптації до сучасних веб-технологій.

SelectorsHub - це інструмент для розробників, тестувальників та автоматизаторів, який допомагає швидко і точно створювати селектори (XPath, CSS) прямо в браузері. Він доступний як розширення для Chrome, Firefox та інших Chromium-браузерів [18]. Популярний інструмент, який має 400 тисяч завантажень. Перевагою можна виділити, що він будує декілька варіантів селектору: cssSelector, XPath, jQuery selector. Проте цей інструмент не враховує динамічні DOM-елементи, іноді формує некоректні або надмірно довгі селектори, що знижує їх стабільність.

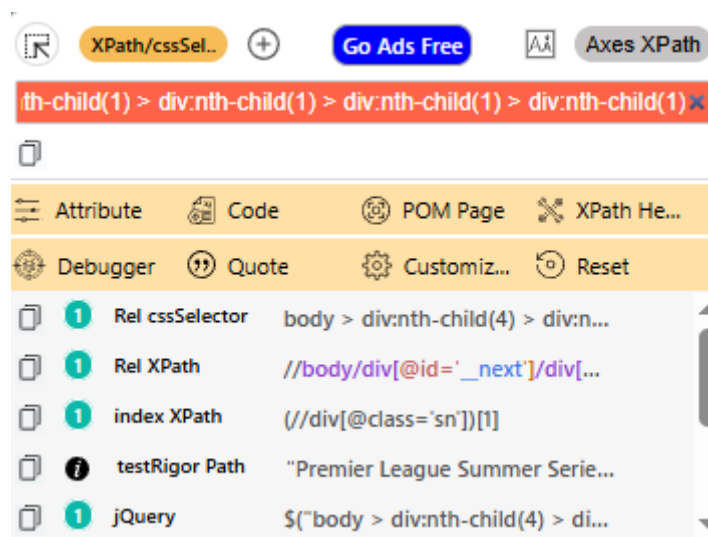


Рисунок 1.2 – Інтерфейс та знайдені селектори в SelectorHub

SeleniumIDE - це розширення браузера яке допомагає не просто знаходити селектори, а й створювати цілі тести та записувати всі дії, які виконує користувач [19]. Доцільний для невеликих, статичних сайтів, але якщо використовувати його для більш важких і сучасних сайтів, то створений тест не зможе бути успішним вже при наступному запуску, оскільки він не враховує динамічні об'єкти, та має обмежену ефективність при обробці складних та динамічних селекторів.

ChromeDevTools – внутрішній інструмент браузеру Google Chrome з розширеною базою різних інструментів для аналізу DOM, стилів, запитів, кешу

та ін. [20]. Для формування DOM-селектору можна використати звичайне копіювання елемента. Є можливість скопіювати: outerHTML, CSS-selector, JS path, styles, XPath, full XPath.

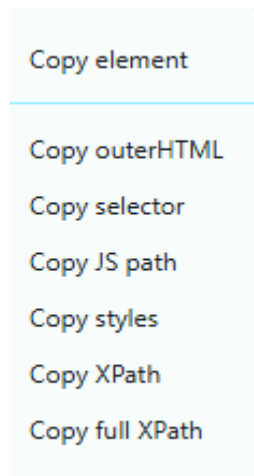


Рисунок 1.3 – Інтерфейс копіювання елементів ChromeDevTools

Таким чином, аналіз існуючих підходів та інструментів для побудови селекторів показав, що жоден із них не враховує глобальну структуру DOM, не забезпечує адаптацію до динамічних змін та не гарантує стабільність селекторів при зміні інтерфейсу. Зазначені обмеження створюють обґрунтовану потребу у розробці нової інформаційної технології, здатної враховувати контекстну структуру DOM, адаптуватися до динамічних змін та забезпечувати стабільність автоматизованих тестів шляхом інтеграції методів машинного навчання, зокрема графових нейронних мереж.

1.3 Аналіз проблемних аспектів підтримки селекторів та стабільності тестів

Проблема підтримки селекторів при розробці автоматизованих тестів веб-застосунків стає дедалі критичнішою, розширення веб-ресурсів потребує збільшення покриття тестами різного функціоналу. Основна проблематика полягає у втраті працездатності селекторів після оновлення DOM-моделі, наявності довгих або динамічних атрибутів. Це, у свою чергу, призводить до появи флаку-тестів, збільшення витрат на супровід і хибної діагностики дефектів.

Розглянемо два процеси: розробка автоматизованого тесту та підтримка тесту з помилкою. Розробка автоматизованого тесту має наступну структуру:

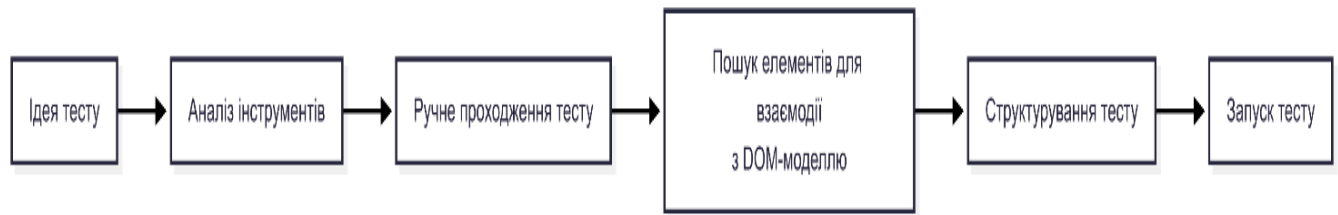


Рисунок 1.4 – Процес розробки автоматизованих тестів

Основні труднощі при автоматизації тестування пов'язані саме з коректною побудовою DOM-селекторів, оскільки на даний момент існуючі фреймворки забезпечують базову функціональність, однак жоден з них не дозволяє виконати дію без попередньо визначеного коректного селектора над веб-ресурсом без попередньо визначеного коретного шляху до елементу.

Окремим завданням є підтримка вже розроблених тестів, які перестали працювати через помилки, що передбачає зміну окремих рядків коду. Перш ніж розпочати роботу з помилковим тестом, необхідно визначити джерело проблеми - у самому тесті або у зміні функціоналу веб-ресурсу. Якщо проблема з тестом, то процес підтримки виглядає зобрачений на рисунку 1.5.

Як ми бачимо в кожному процесі є робота з DOM-моделлю. Не має прямих досліджень які б вказували на час пошуку селекторів. Але з практичної оцінки досвідчених тестувальників, то час на пошук складних селекторів може становити від кількох хвилин до години, при цьому стабільність знайдених елементів не гарантується [21, 22]. Все це витрачає час спеціалістів та зменшує якість продуктів через зменшення покриття тестами. Завдання з нестабільності тестів проблема всіх автоматизованих тестувальник, навіть компаній-гігантів по прикладу Google [23].

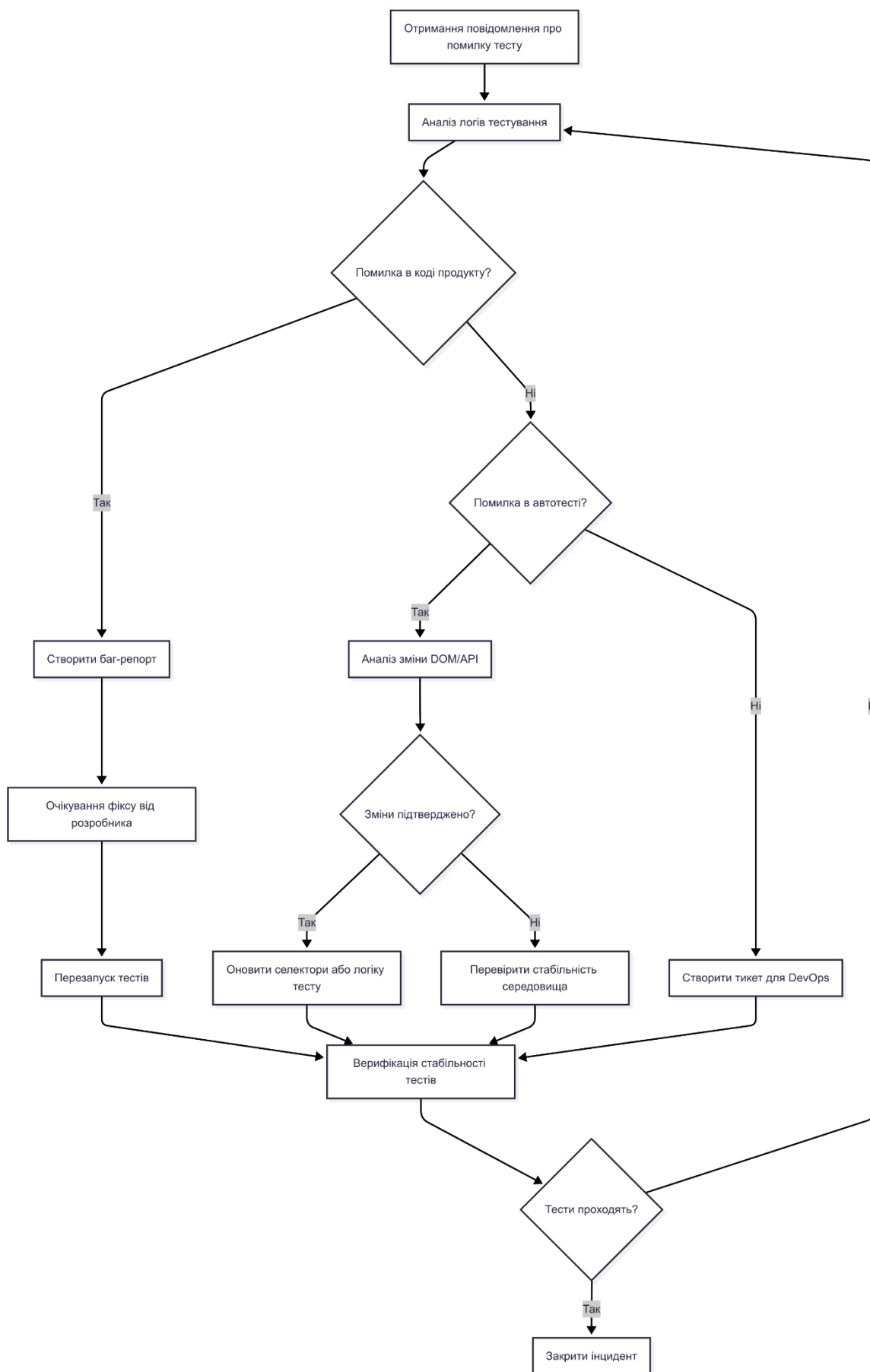


Рисунок 1.5 – Процес підтримки автоматизованих тестів

Розглянемо приклад нестабільного(flaky) тесту, у якому потрібно знайти елемент банеру, натиснути на нього та перевірити чи при переході відбувся перехід на наступну сторінку. Якщо елемент не буде знайдений, то весь тест не пройде успішно:

```
@Test() new *
public void openPromoBanner() {
    WebElement banner = driver.findElement(By.xpath( xpathExpression: "//div[@class='promo-banner_2351']"));
    banner.click();
    Assert.assertTrue(driver.getCurrentUrl().contains("promo"));
}
```

Рисунок 1.6 – Приклад flaky-тесту

Після оновлення інтерфейсу частина селектору «2351» стала динамічною і оновлюється при кожному завантаженні. Тест може падати з помилкою, що елемент не знайдено, хоча при деяких запусках тесту, він буде успішний. В такому випадку цей тест не можна назвати стабільним та потребує доопрацювання. Цей кейс демонструє типову ситуацію, коли нестабільність селектора призводить до flaky-тесту, що не лише затримує процес розробки, а й створює помилкове враження про наявність/відсутність дефекту в коді.

За емпіричними даними близько третини flaky UI-тестів виникають через неправильну взаємодію з DOM-подіями (Event-DOM interaction), що є безпосередньо пов'язано з нестабільністю DOM-селекторів. Події можуть змінювати структуру DOM, що призводить до нестабільності селекторів, повільного їх виправлення (до 153 днів) і значного втрачання ресурсів при підтримці тестів. Інші категорії помилок - Event operations та Response evaluations - теж часті і критично впливають на надійність UI-тестів [24].

Для пришвидшення процесу автоматизованого тестування важливо враховувати метрики стабільності, унікальності та точності селекторів. Сучасні інструменти не враховують контекст елементів та не враховують структуру DOM в цілому, недостатньо адаптуються до змін і не мінімізують втручання людини.

Для систематизації наслідків використання нестабільних селекторів було проведено аналіз типових помилок, з якими стикаються автоматизовані тестувальники. У таблиці 1.3 наведено основні типи помилок, їхні причини, вплив на якість тестування та рівень складності діагностики. Такий аналіз дозволяє визначити найбільш критичні випадки, на які варто звернути увагу при побудові інтелектуальної інформаційної технології процесу автоматизованого тестування.

Таблиця 1.3 – Порівняння типів помилок при нестабільних селекторах

Тип помилки	Причина виникнення	Наслідки для тестування	Частота появи	Складність діагностики
ElementNotFound	Зміна DOM або динамічний атрибут	Падіння тесту на етапі взаємодії з елементом	Висока	Низька
StaleElementReference	DOM оновлено після знаходження елемента	Неможливість виконати дію на елементом	Середня	Середня
TimeoutException	Селектор працює, але елемент не з'являється вчасно	Повільна робота UI або неправильне очікування	Висока	Висока
IncorrectElementMatch	Збіг селектора з кількома елементами або не тим, що очікується	Некоректне виконання дії (наприклад, клік не туди)	Середня	Висока

Таким чином, забезпечення стабільності селекторів потребує комплексного підходу, здатного враховувати глобальний контекст DOM-структури, виявляти потенційно нестабільні елементи та адаптуватися до змін без участі тестувальника. Одним із перспективних напрямів є застосування графових нейронних мереж, здатних моделювати DOM як граф для аналізу структури та виявлення оптимальних точок взаємодії.

1.4 Постановка задач дослідження

У процесі розробки автоматизованих UI-тестів стабільність та якість DOM-селекторів відіграють одну з ключових ролей, оскільки саме за їх допомогою забезпечується взаємодія тестів з веб-інтерфейсом. Проблема підтримки селекторів є особливо актуальна в умовах динамічних DOM-моделей та масштабування сучасних веб-ресурсів. Зміни атрибутів, оновлення структури або динамічне формування елементів призводять до втрати працездатності селекторів, що спричиняє flaky-тести, хибні результати та значні витрати на супровід.

На сьогодні існують ручні, автоматизовані та напіваавтоматизовані підходи до створення селекторів, проте жоден з них не забезпечує достатнього рівня адаптивності змін у DOM-моделі. Ручна побудова є трудомісткою, а автоматизовані та напіваавтоматизовані підходи працюють без урахування контексту та зв'язків між елементами, що призводить до формування довгих або нестабільних селекторів.

DOM має природну деревоподібну форму, фактично має графову структуру, що відкриває можливість застосування графових моделей для її аналізу. Зокрема, перспективним є використання графових нейронних мереж, які здатні враховувати топологію графа, контекстні зв'язки між вузлами та дозволяють моделювати поведінку складних структур, таких як DOM-дерево.

Зважаючи на вищенаведене, деталізуємо основні задачі дослідження:

- Формалізація DOM як графову структуру з відповідними ознаками для вузлів та ребер
- Аналіз контексту для побудови більш стійких та оптимальних селекторів
- Розробити метод оцінювання якості селектора на основі метрик стабільності, довжини та унікальності
- Застосувати GNN-модель для автоматичної побудови стійких селекторів
- Автоматизувати процес формування селекторів
- Зменшити час, необхідний для підтримки селекторів

1.5 Висновки до розділу 1

Проведено комплексний аналіз сучасних підходів до побудови DOM-селекторів, методів їх використання в автоматизованому тестуванні, а також проблем пов'язаних із їх стабільністю та підтримкою. Установлено, що від якості побудови селекторів напряду залежать ефективність, надійність та час підтримки UI-тестів.

Показано, що сучасні інструменти для ручного, автоматизованого та напівавтоматизованого формування селекторів (SelectorsHub, SeleniumIDE, Chrome DevTools) не враховують глобального контексту DOM і не здатні адаптуватись до динамічних змін інтерфейсів. Це призводить до частих помилок, зокрема `ElementNotFound`, `StaleElementReference`, `TimeoutException`, `IncorrectElementMatch`, які спричиняють flaky-тести та потребують додаткових витрат на підтримку.

Виявлено, що в переважній більшості випадків нестабільність автоматизованих UI-тестів переважно зумовлена нестійкістю селекторів. Особливої актуальності набуває проблема діагностики помилок, спричинених неправильним або нестабільним вибором елементів DOM.

Обґрунтовано доцільність формалізації DOM як графової структури для подальшого використання методів машинного навчання, зокрема графових нейронних мереж, які здатні враховувати топологічні зв'язки, контекстну інформацію та допомагати в побудові стійких, адаптивних селекторів.

Таким чином, на підставі проведеного аналізу сформульовано постановку задачі для створення інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого ПЗ ІС.

РОЗДІЛ 2 МЕТОДИ ТА МОДЕЛІ ІНТЕЛЕКТУАЛЬНОЇ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ОРІЄНТОВАНОГО ПЗ ІС

2.1 Аналіз моделі DOM як графової структури

DOM (Document object model) – це інтерфейс прикладного програмування (API) для валідних HTML-документів та правильно сформованих XML-документів. DOM-модель визначає логічну структуру документів та спосіб доступу до них і роботи з ними [25].

```
<body>
  <div class="dialog-off-canvas-main-canvas" data-off-canvas-main-canvas="">
    <div class="header-line">
      <div class="container-fluid horzlimiter">
        <a id="vntu-logo" href="https://vntu.edu.ua/"></a>
        <nav class="navbar transparent navbar-inverse">
          <div class="container-fluid horzlimiter">
            <div class="collapse navbar-collapse">
              <div id="vntu-title" class="navbar-form navbar-left">
                <h1>Вінницький Національний Технічний Університет</h1>
              </div>
              <div class="nav navbar-nav navbar-right buttons-box">
                <a href="/go/account" class="btn btn-primary" role="button">Особистий кабінет</a>
                <a href="/go/vstup_howto" class="btn btn-primary" role="button">Як вступити?</a>
              </div>
            </div>
          </nav>
        </div>
      </div>
    <div class="vntu-navbar" class="navbar navbar-default">
      <div class="container-fluid horzlimiter leftlimiter">
        <div class="navbar-header">
          <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-navbar-collapse-1" aria-expanded="false">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
        </div>
      </div>
    </div>
  </div>
</body>
```

Рисунок 2.1 – Приклад DOM-моделі

DOM-елемент – це окремий вузол у DOM-структурі, що відповідає конкретному HTML-тегу та містить його атрибути, текстовий вміст і зв'язки з іншими елементами (батьківськими, дочірніми або сусідніми).

```
<h1>Вінницький Національний Технічний Університет</h1>
```

Рисунок 2.2 – Приклад DOM-елемента

HTML-документ можна розглянути як граф де кожен тег відповідає окремому вузлу. Зв'язки між вузлами формували на підставі відношень батько-

нащадок (parent-child), а також – за потреби – сусідніх елементів (sibling). Математична модель DOM-графа має вигляд:

$$G = (V, E) \quad (2.1)$$

де V – множина вузлів(елементів), E – множина ребер (зв'язки між елементами). Розглянемо властивості графів у контекстів покращення пошуку оптимального селектору: глибину графа, ступінь вузла, зв'язність. Глибина вузла в графі визначається як довжина найкоротшого шляху від кореневого вузла (зазвичай це або `html`, або `body`) для поточного вузла. Максимальна глибина це найбільша з глибин усіх вузлів. Можна описати як:

$$depth(v_i) = d(v_{root}, v_i) \quad (2.2)$$

де v_i – довільний вузол у DOM-графі, v_{root} – кореневий вузол (`html` або `body`), $d(v_{root}, v_i)$ – кількість ребер, що з'єднують корінь вузлом v_i по найкоротшому шляху.

Ступінь вузла – це кількість його суміжних вузлів. У напрямленому графі, як у DOM-структурі розрізняються: вхідний ступінь – кількість батьківських елементів, вихідний ступінь – кількість дочірніх елементів.

$$deg^-(v_i) = |\{v_j \in V \mid (v_j, v_i) \in E\}| \quad (2.3)$$

$$deg^+(v_i) = |\{v_k \in V \mid (v_i, v_k) \in E\}| \quad (2.4)$$

$$deg(v_i) = deg^-(v_i) + deg^+(v_i) \quad (2.5)$$

де $deg^-(v_i)$ – кількість батьків вузла v_i , $deg^+(v_i)$ – кількість дочірніх вузлів, $deg(v_i)$ – загальна кількість суміжних елементів. Ступінь є важливим для аналізу структурної ролі вузла DOM, вузли з високим ступенем можуть бути ключовими при формуванні селектора.

Зв'язність графа характеризує наявність шляхів між вузлами. Для орієнтовних графів: сильна зв'язність – між будь-якими двома вузлами існують шляхи в обох напрямках, слабка зв'язність – існує шлях в прямому або зворотному напрямку. Для структури DOM, як дерева є притаманна слабка зв'язність, допомагає побудувати єдиний шлях від кореня до кожного елемента.

Давайте розглянемо частину структури DOM та візуалізуємо його як граф

```
<div id='root'>
```

```
<h1 class='title'></h1>
<div class='content'>
  <p></p>
</div> </div>
```

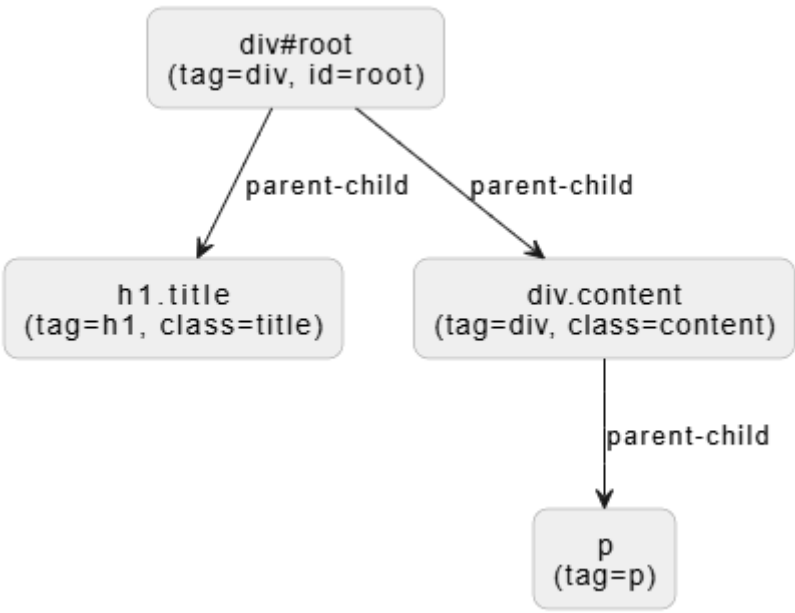


Рисунок 2.3 – Візуалізація структури DOM як графа

Формалізуємо ознаки вузлів та ребер для DOM-графа:

Таблиця 2.1 – Формалізація ознак вузлів та ребер для DOM-графа

Назва ознаки	Позначення	Опис
Тип тега	$\rightarrow t_i$	One-hot кодування HTML-тегу вузла
Глибина вузла	$depth(v_i)$	Відстань від кореневого вузла
Наявність атрибуту id	$id_i \in \{0, 1\}$	Ознака наявності id
Наявність атрибуту class	$class_i \in \{0, 1\}$	Ознака наявності class
Наявність атрибуту data-testid	$testid_i \in \{0, 1\}$	Ознака наявності data-testid
Наявність дочірніх елементів	$deg^+(v_i)$	Вихідний ступінь вузла

Ці ознаки є важливими для побудови селектора, тип тегу вказує конкретний тег для частини селектору, глибина вузла та наявність дочірніх елементів відповідає за стабільність селектору, id, class, data-testid – можуть бути унікальними ідентифікаторами для селектору.

У результаті проведеного аналізу DOM як графової структури, було визначено основні структурні характеристики (глибина, ступінь, зв'язність), що впливають на формування селекторів. Також було формалізовано набір ознак вузлів DOM-графа, які надалі використовуються для побудови матриці ознак вузлів - ключового елементу вхідних даних для GNN. Це закладає основу для реалізації методів графового навчання в задачах формування стабільних та оптимальних селекторів у веб-тестуванні.

2.2 Методи машинного навчання для роботи з графами

Графові нейронні мережі є класом моделей глибокого навчання, у складі сучасних методів штучного інтелекту, призначених для обробки структурованих у вигляді графів даних. Вони ефективно поєднують інформацію про структуру графа та ознаки його елементів, що робить їх придатними до розв'язання задач класифікації, кластеризації, побудови векторних представлень вузлів та передбачення зв'язків. У контексті даного дослідження, де модель DOM представляється як орієнтований граф, GNN дозволяють навчатися на структурі веб-сторінки та обирати найбільш релевантні вузли (елементи) для побудови стабільних та унікальних селекторів.

Розглянемо три найбільш вживані архітектури GNN, які використовуються для обробки графів: GCN (Graph Convolutional Network), GAT (Graph Attention Network) та GraphSAGE (Graph Sample and AggregatE). Охарактеризуємо існуючі архітектури графових нейронних мереж, які потенційно можуть бути застосовані для задачі побудови селекторів у структурі DOM-графа.

Найпростіша архітектура – модель GCN, запропонована Т. Кіпфом та М. Велінгом у 2017 році [26], є спектральною апроксимацією згортки на графах.

Основна ідея полягає в обчисленні нових представлень вузлів шляхом усереднення ознак їх сусідів з урахуванням нормалізації матриці суміжності. Враховуються також self-loop зв'язки, які дозволяють включити поточні ознаки вузла в обчислення [26-28].

Формалізація шару GCN:

$$H^{(l+1)} = \sigma(\hat{D}^{-1/2} \hat{A} \hat{D}^{-1/2} H^{(l)} W^{(l)}) \quad (2.6)$$

де $\hat{A} = A + I$ – модифікована матриця суміжності з одиничною діагоналлю (self-loop), $\hat{D}$ – ступенева матриця для $\hat{A}$, $H^{(l)}$ – матриця ознак на шарі l , $W^{(l)}$ – навчувана матриця ваг, σ – нелінійна функція активації.

GCN добре підходить для задач класифікації вузлів на малих та середніх графах. Однак, його агрегація не враховує важливість окремих сусідів, що може призводити до втрати інформації при високому ступені вузлів.

Модель GAT була запропонована П. Велічковичем у 2018 році [26] та вдосконалює GCN завдяки механізму уваги (attention). На відміну від GCN, GAT обчислює вагу кожного сусіда динамічно, залежно від ознак вузлів. [29, 30-34].

Агрегація у GAT виконується як:

$$h'_i = (\sum_{j \in N(i)} a_{ij} W h_j) \quad (2.7)$$

де a_{ij} – коефіцієнт уваги, який обчислюється як:

$$a_{ij} = \frac{\exp(\text{LeakyRelu}(a^T [W h_i || W h_j]))}{\sum_{k \in N(i)} \exp(\text{LeakyRelu}(a^T [W h_i || W h_k]))} \quad (2.8)$$

Цей механізм дозволяє моделі враховувати важливість сусідів, що є актуальним у контексті DOM, де значущість елементів залежить, зокрема, від атрибутів або позиції в структурі. Основний недолік - висока обчислювальна складність при великій кількості сусідів.

GraphSAGE, запропонована В. Гамільтоном у 2017 році [35], вводить принципово новий підхід до побудови графових представлень - індуктивне навчання. На відміну від GCN та GAT, ця модель дозволяє працювати з раніше не баченими вузлами та графами.

Для кожного вузла вибирається фіксована кількість сусідів, над якими виконується агрегація (mean, max, LSTM тощо). Агреговане значення поєднується з ознаками вузла та передається на наступний шар[32, 35].

Оновлення представлення має вигляд:

$$h_v^{(k)} = \sigma(W^{(k)} \cdot AGG^{(k)}(\{h_u^{(k-1)}, \forall u \in N(v)\})) \quad (2.9)$$

де AGG – агрегуюча функція.

GraphSAGE дозволяє масштабуватися до великих графів, таких як повноцінні HTML-документи, а також забезпечує добру узагальнюваність на нові сторінки. Можливість генералізації GraphSAGE особливо цінна при автоматизованому тестуванні, де кожна сторінка має унікальний DOM.

Порівняльна характеристика методів машинного навчання для графів подана у таблиці 2.2.

Таблиця 2.2 - Порівняльна характеристика методів машинного навчання для графів

Характеристика	GCN	GAT	GraphSAGE
Тип агрегації	Середнє значення	Зважене значення	Вибірка + агрегування
Обробка великих графів	Обмежена	Обмежена	Підтримується
Увага до сусідів	Ні	Так	Часткова
Індуктивна здатність	Ні	Ні	Так
Гнучкість	Середня	Висока	Висока

Розглянуті архітектури GNN мають свої переваги та обмеження. У рамках покращення процесу пошуку DOM-селекторів для задач автоматизованого тестування веб-ресурсів, доцільно використовувати GAT для врахування різної важливості сусідів у дереві DOM, або GraphSAGE - у випадку, коли необхідна індуктивна здатність моделі до генералізації на нові сторінки та структури.

Таким чином, огляд архітектур GNN дозволив виявити сильні та слабкі сторони кожної з моделей. Для завдань покращення процесу пошуку DOM-селекторів, критично важливою є здатність моделі враховувати структурну важливість вузлів, адаптуватися до нових сторінок та масштабуватись до складних деревоподібних структур. Відтак, у подальших експериментах було обрано архітектуру GAT, як таку, що найкраще відповідає поставленим задачам.

2.3 Ознаки DOM-елементів для GNN

Опишемо, які ознаки обираються для вузлів DOM-графа: one-hot кодування, бінарні ознаки атрибутів, структурні ознаки. One-hot кодування - це метод перетворення категоріальних даних у бінарне векторне представлення. Кожна категорія представлена вектором, довжина якого дорівнює загальній кількості унікальних категорій [36]. В задачі побудови селектору за допомогою GNN метод використовується для кодування тегів. Це дозволяє розрізняти типи елементів, наприклад `<div>`, `<span>`, `<a>` та інші. Розглянемо приклад:

Для кожного вузла DOM-графа визначається його тег (наприклад: `<div>`, `<a>`, `<span>`, `<ul>` тощо). Нехай існує множина можливих тегів:

$$\mathcal{T} = \{t_1, t_2, t_3, \dots, t_T\} \quad (2.10)$$

де T – кількість різних типів тегів, що зустрічаються в корпусі документів.

Тоді кожен v_i , який відповідає тегу t_k , кодується у вигляді one-hot вектора довжини T :

$$onehot(v_i) = [0, 0, \dots, 1, \dots, 0] \in \mathbb{R}^T \quad (2.11)$$

де одиниця стоїть на k -ій позиції, яка відповідає тегу t_k .

Нехай маємо множину тегів:

$$\mathcal{T} = \{div, span, a, ul\}, T=4$$

Для кожного кожного вузла маємо кодування:

$$\text{<div> - } x = [1, 0, 0, 0]$$

$$\text{ - } x = [0, 1, 0, 0]$$

$$\text{<a> - } x = [0, 0, 1, 0]$$

$$\langle \text{ul} \rangle - x = [0,0,0,1]$$

Таким чином, one-hot кодування забезпечує уніфіковане представлення типу тегу, яке може бути інтегроване у матрицю ознак $X \in \mathbb{R}^{N \times d}$ для подальшої обробки GNN. При цьому ознака тегу є категоріальною, і one-hot кодування дозволяє уникнути накладання «штучних» відстаней між різними тегами (на відміну від простого числового кодування).

Для атрибутів використовуються бінарні ознаки. Атрибутами виступають – id, class, data-testid та інші. Для кожного атрибуту додається ознака 1, якщо він присутній і 0 – якщо відсутній. Це критично важливо, оскільки в автоматизованих тестах найчастіше використовується пошук саме за допомогою атрибутів [37].

Також використовуються структурні ознаки, які дозволяють GNN краще враховувати структуру DOM-дерева. Серед них глибина вузла, кількість дочірніх елементів (ступінь вузла), позиція серед усіх сусідів. Такі структурні глобальні та локальні ознаки допомагають моделі диференціювати вузли навіть за ідентичними тегами, що важливо для контекстуального розуміння структури [38, 39]. Всі ознаки вузлів DOM описані в таблиці 2.3.

Таким чином, вектори ознак усіх вузлів об'єднуються у матрицю ознак:

$$X \in \mathbb{R}^{N \times d} \quad (2.12)$$

де N – кількість вузлів у DOM-графі, d – кількість ознак.

Ключовим елементом графового подання DOM-структури є визначення зв'язків між вузлами, які формалізуються у вигляді матриці суміжності або списку ребер. Нехай задано граф $G = (V, E)$, де $V = \{v_1, v_2, \dots, v_N\}$ – множина вузлів, а $E \subset V \times V$ – множина ребер, що відображають структурні відношення у DOM. Тоді матриця суміжності має вигляд:

$$A \in \{0, 1\}^{N \times N}, A_{ij} = \begin{cases} 1, \text{ якщо } (v_i, v_j) \in E \\ 0, \text{ інакше} \end{cases} \quad (2.13)$$

де ребра визначаються зв'язками parent-child, коли потрібно sibling.

Існують такі типи ребер у DOM-графі:

Таблиця 2.3 – Ознаки вузлів DOM

Категорія	Ознака	Тип/Значення	Пояснення
Семантичні	One-hot тегу	Вектор довжини T	Кодування типу елемента: <div>, , <a> тощо.
Атрибути	Наявність id	0/1	1 - Елемент має id, 0 - відсутній
	Наявність class	0/1	1 - Елемент має class, 0 - відсутній
	Наявність data-testid	0/1	1 - Елемент має data-testid, 0 - відсутній
Структурні	Глибина вузла	Ціле число	Відстань від кореня DOM-дерева
	Кількість дочірніх елементів	Ціле число	Скільки дочірніх елементів має вузол
	Позиція серед дочірніх елементів	Ціле число	Порядковий номер вузла у списку сусідніх елементів

- Parent-child – основні зв’язки, що відображають ієрархію DOM-дерева
- (наприклад →)
- Sibling – додаткові зв’язки між сусідніми елементами, які знаходяться на одному рівні (наприклад →)
- Self-loop – петля вузла на самого себе, яка дозволяє зберегти його власні ознаки у процесі агрегації.

Створимо DOM-граф з ознаками на прикладі частини HTML:

```
<div id="block" class="container">
  <span class="title">Example</span>
  <a href="#" data-testid="link-1">Click</a>
</div>
```

Для цього прикладу зробимо схему графу з ознаками а також матрицю суміжностей. DOM-структура цього фрагмента відображається у вигляді орієнтованого графа, де вузли відповідають HTML-елементам, а ребра – відношенню parent → child.

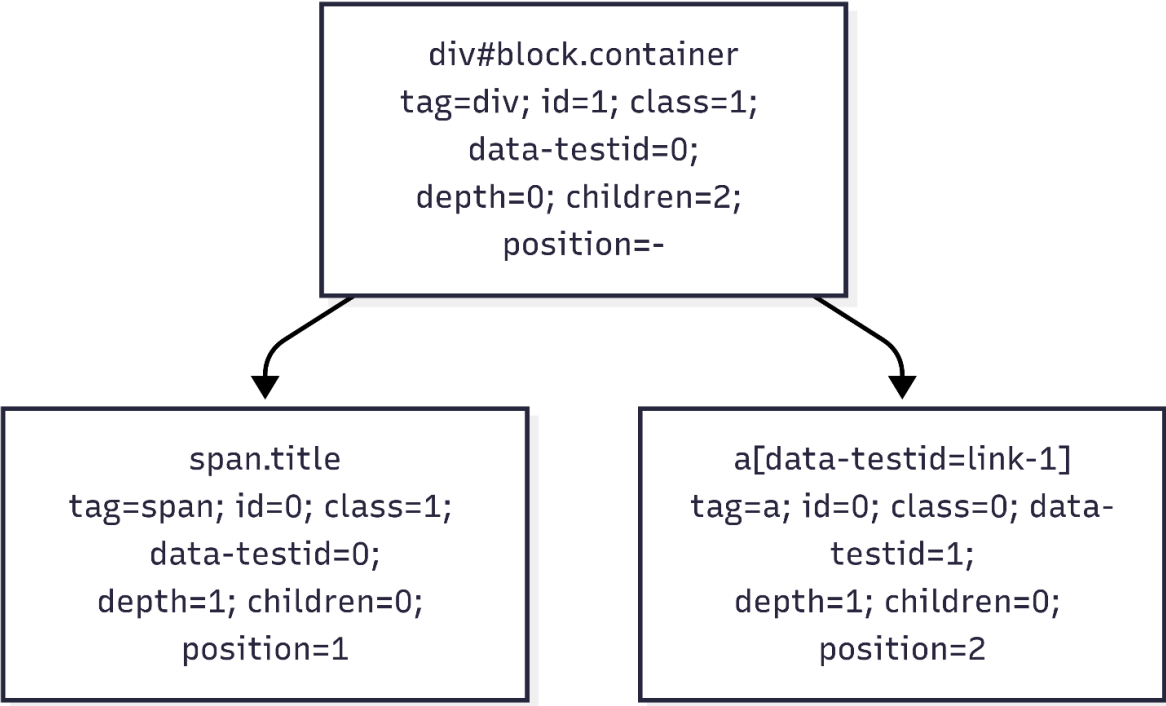


Рисунок 2.4 – Граф з ознаками вузлів

Кожен вузол v_i описується вектором ознак x_i , що включає як семантичні, так і структурні характеристики.

Таблиця 2.4 – Ознаки вузлів для запропонованого прикладу

Вузол	One-hot ter	Id	Class	Data-testid	Глибина	Дочірні	Позиція
<div>	[1,0,0]	1	1	0	0	2	-
	[0,1,0]	0	1	0	1	0	1
<a>	[0,0,1]	0	0	1	1	0	2

Таким чином формується матриця ознаки вузлів:

$$X = \begin{bmatrix} x_{div} \\ x_{span} \\ x_a \end{bmatrix} \in \mathbb{R}^{3 \times d}$$

де $d=T+5$, а $T=3$ – кількість тегів.

Для цього прикладу множина ребер:

$$E = \{(div, span), (div, a)\}$$

Матриця суміжності:

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Таким чином, GNN отримує на вхід не HTML-код напряму, а графове подання DOM у вигляді: Матриці ознак вузлів X , де відображені семантичні й структурні характеристики кожного елемента. Матриці суміжності A (або списку ребер E), що задає топологію DOM-дерева.

Це дозволяє моделі навчатися враховуючи ієрархічні зв'язки між елементами веб-сторінки, що є критичним для побудови стійких і унікальних селекторів у процесі автоматизованого тестування.

2.4 Формалізація оцінювання якості селектора

Одним з ключових аспектів процесу автоматизованого тестування є побудова селекторів, які одночасно є стабільними, унікальними та компактними. Для цього необхідна формалізація критеріїв якості селектора у вигляді математичних метрик, які дозволяють здійснювати кількісне порівняння різних підходів до їх формування. Такий підхід забезпечує можливість об'єктивної оцінки та використання алгоритмів машинного навчання, зокрема графових нейронних мереж, для їх автоматичного побудування [40].

Унікальність є базовою вимогою до селектора, оскільки лише у випадку однозначної ідентифікації DOM-елемента можна гарантувати коректність тесту. Введемо формальне визначення:

$$u(s) = \begin{cases} 1, \text{ якщо } |Match(s)| = 1, \\ 0, \text{ інакше} \end{cases}$$

де $Match(s)$ – множина елементів, які знаходить селектор.

Унікальність особливо важлива у випадках використання селекторів у масштабних регресійних тестах, де відсутність однозначності призводить до flaky-тестів.

При побудові складних селекторів, особливо коли вони базуються на класах чи вкладених елементах, часто виникають ситуації, коли:

- селектор повертає зайві елементи (FP, False Positive),
- не охоплює всі релевантні елементи (FN, False Negative).

Для оцінки таких випадків використовуються метрики інформаційного пошуку Precision(точність), Recall(повнота) [39]:

$$Precision(s) = \frac{TP}{TP+FP} \quad (2.14)$$

$$Recall(s) = \frac{TP}{TP+FN} \quad (2.15)$$

Для збалансованої оцінки застосовується гармонічне середнє:

$$F1(s) = 2 \cdot \frac{Precision(s) \cdot Recall(s)}{Precision(s) + Recall(s)} \quad (2.16)$$

Крім точності та унікальності, важливо враховувати довжину селектора, яка впливає на його зручність використання, стабільність та швидкість пошуку. Довжина селектора є важливим критерієм під час написання автоматизованих тестів, так як він виконує декілька функцій:

- Пришвидшує роботу тестів.
- Покращує стабільність тестів.
- Сприяє покращенню читабельності тестового коду та полегшує його супровід.

Для визначення довжини селектору запропоновано функцію з використанням вагових коефіцієнтів, що покращує визначення довжини селектора:

$$L(S) = \sum_{i=1}^k w(c_i) \quad (2.17)$$

де k – кількість компонентів, c_i – компонента селектора (теги, класи, id, оператори вкладеності тощо), $w \in (0; 1]$ – ваговий коефіцієнт. Розподілено вагові коефіцієнти можна наступним чином:

Таблиця 2.5 – Вагові коефіцієнти для компонентів селектору

Тип елементу	Ваговий коефіцієнт
Tag	0.5
Id	0.2
Class	0.5
Data-testid	0.4
nth-child	0.8
Вкладеність (>)	0.5

Для ілюстрації використання метрик наведемо умовний приклад:

Таблиця 2.6 – Приклад використання метрик

Тип селектора	Унікальність	Точність	Повнота	F1-score	Довжина
#menu (id)	1	1.0	1	1	0.2
.nav-item (class)	0	0.33	1	0.5	0.5
ul.na > li:nth-child(2)	1	1.0	1	1	2.8
li>a (tab based)	0	0.25	1	0.4	1.5
//li[@id='menu-item-5']/a (XPath)	1	1.0	1	1	1.2

Формалізація якості селекторів через систему метрик дозволяє: поєднувати бінарні показники (унікальність) з кількісними (Precision, Recall, F1), враховувати структурну складність (довжина), створювати збалансовану функцію оцінки, придатну для покращення процесу пошуку за допомогою GNN. У подальших розділах ці метрики будуть використані для порівняння традиційних підходів до побудови селекторів і методів, що ґрунтуються на графових нейронних мережах.

2.5 Теоретичне обґрунтування підходу до побудови адаптивних селекторів

Автоматизоване тестування веб-ресурсів значною мірою залежить від стабільності та якості селекторів, які застосовуються для ідентифікації елементів DOM-структури. Традиційні підходи до їх побудови мають низку обмежень. Ручне створення селекторів дає можливість тестувальнику повністю контролювати процес, проте воно вимагає значних часових витрат і високої кваліфікації спеціаліста. Будь-які зміни у верстці призводять до того, що побудований раніше селектор втрачає актуальність, а тест стає нестабільним. Автоматизовані інструменти, такі як Selenium IDE або SelectorHub, дозволяють швидко зформувати селектори, однак отримані результати часто виявляються надмірно довгими й крихкими: при найменших змінах DOM вони перестають працювати.

Враховуючи ці обмеження пропонується використання графових нейронних мереж (Graph Neural Networks, GNN) у поєднанні з агентом прийняття рішень. Такий підхід дозволяє врахувати як локальні ознаки вузлів (тег, атрибути, позицію серед сусідів), так і їхній контекст у DOM-дереві. На відміну від простих евристичних або текстових методів, GNN формує векторні репрезентації елементів, що містять інформацію не лише про сам вузол, а й про його зв'язки з батьківськими та дочірніми елементами. Це забезпечує можливість робити висновки про стабільність потенційного селектора навіть у випадку, коли атрибут, наприклад клас, виявляється повторюваним у багатьох елементах [41].

Агент, взаємодіючи з репрезентаціями, що формує GNN, здійснює вибір дій, спрямованих на побудову оптимального селектора. Ці дії можуть включати використання або ігнорування певних атрибутів, визначення глибини вкладеності, а також вибір стратегії формування шляху у вигляді CSS чи XPath. Метою агента є максимізація функції винагороди, яка базується на метриках якості селектора, а саме: унікальності, точності, повноті, F1-score та довжині.

Таким чином, агент навчається віддавати перевагу тим ознакам і стратегіям побудови, які забезпечують високу стабільність та компактність селекторів.

Формальна модель побудови адаптивних селекторів передбачає навчання агента таким чином, щоб він максимізував якість побудованих селекторів. Як уже встановлено, ключові метрики якості: унікальність $u(s)$, точність $Precision(s)$, повнота $Recall(s)$, F_1 -міра $F1(s)$ та довжина селектора $L(s)$. Ці показники формують основу для функції винагороди [41, 42].

Математично модель винагороди може бути виражена як:

$$R(s) = \alpha \cdot u(s) + \beta \cdot F1(s) - \gamma \cdot L(s) \quad (2.18)$$

де $u(s)$ – бінарний показник унікальності, $F1(s)$ – збалансована міра точності і повноти, $L(s)$ – нормалізована довжина селектору, α, β, γ – вагові коефіцієнти, які коригують вплив кожного компонента. Це дозволяє агенту оптимізувати баланс між стабільністю, точністю та компактністю.

Функція винагороди $R(s)$ дозволяє навчити агента переважати селектори, які є не лише коректними, тобто унікальними і точними, але й компактними й стабільними.

Особливо важливим є аспект адаптації. У разі зміни DOM-структури агент, спираючись на контекстні ознаки вузлів, здатний перебудувати селектор, замінивши нестабільний атрибут на інший або скориставшись альтернативним шляхом у графі. Це дозволяє суттєво знизити кількість нестабільних тестів і мінімізувати ручне втручання у підтримку автоматизованих сценаріїв.

Теоретичну модель роботи запропонованої системи можна описати так: на першому етапі HTML-документ перетворюється у граф, де вузли відповідають тегам та атрибутам, а ребра - відношенням parent-child чи sibling. Побудований граф подається на вхід GNN, яка формує векторні представлення вузлів. Далі агент, використовуючи ці репрезентації, послідовно формує селектор і перевіряє його якість за визначеними метриками. Отриманий зворотний зв'язок застосовується для оновлення стратегії агента, що підвищує ефективність подальшої побудови селекторів.

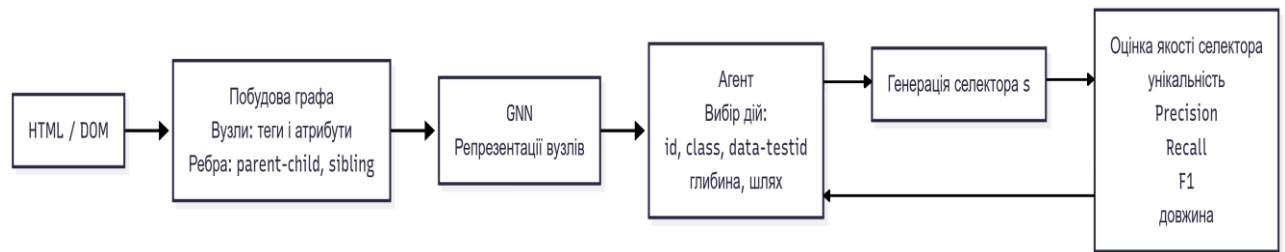


Рисунок 2.5 – Узагальнена схема функціонування системи

Таким чином, поєднання GNN та агента забезпечує теоретично обґрунтовану основу для створення адаптивних селекторів. На відміну від класичних методів, запропонований підхід дозволяє враховувати контекст, прогнозувати стабільність і реагувати на зміни DOM, що робить його перспективним інструментом у задачах процесу автоматизованого тестування веб-ресурсів.

2.6 Висновки до розділу 2

У другому розділі дисертації було сформовано теоретичну основу для побудови інформаційної технології автоматичного формування адаптивних селекторів DOM-елементів, яка дозволяє підвищити ефективність, стабільність та гнучкість процесу автоматизованого тестування веб-ресурсів.

Проведено критичний аналіз традиційних підходів до побудови селекторів: ручних, автоматизованих (Selenium IDE, SelectorHub). Встановлено їхні ключові обмеження – надмірна довжина, чутливість до змін у DOM-структурі, відсутність контекстної обробки елементів. Обґрунтовано доцільність застосування графових нейронних мереж (GNN) для врахування як локальних, так і контекстних ознак DOM-вузлів.

Здійснено формалізацію подання HTML-документа у вигляді графа, де вузли відображають DOM-елементи, а ребра - структурні зв'язки типу parent-child та sibling. Запропоновано систему ознак вузлів, які включають як атрибутивні характеристики (наявність id, class, data-testid), так і структурні (глибина, кількість нащадків). Побудовано матрицю ознак вузлів, яка формується шляхом

об'єднання векторів ознак для кожного вузла, з подальшим масштабуванням та подачею на вхід GNN.

Визначено систему метрик для оцінювання якості побудованого селектора: унікальність, точність (Precision), повнота (Recall), F1-міра та довжина. Запропоновано формулу оцінки довжини з урахуванням вагових коефіцієнтів для різних типів компонентів селектора (id, class, nth-child тощо), що дозволяє об'єктивно вимірювати компактність селектора. Такий підхід забезпечує математичну основу для навчання агента на послідовності дій, спрямованих на побудову оптимального селектора.

Описано теоретичну модель функціонування системи: побудова DOM-графа → формування ознак → GNN-репрезентації вузлів → робота агента над побудовою селектора → обчислення метрик → оновлення стратегії на основі винагороди. Особливої уваги надано здатності системи до адаптації — при зміні структури DOM агент може перебудувати селектор, використовуючи альтернативні шляхи у графі, що дозволяє мінімізувати флаку-тести й витрати на підтримку тестів.

Таким чином, запропонований підхід формує цілісне теоретичне підґрунтя для розробки інформаційної технології, здатної автоматично формувати адаптивні селектори з урахуванням контексту, стабільності та зміни DOM-структур, що є особливо актуальним для сучасного динамічного веб-середовища.

РОЗДІЛ 3 РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB- ОРІЄНТОВАНОГО ПЗ ІС

3.1 Основні етапи та загальна архітектура інтелектуальної інформаційної технології

Відповідно до проведеного аналізу та теоретичного обґрунтування, перед розроблюваною інтелектуальною інформаційною технологією автоматизованого тестування web-орієнтованого ПЗ ІС постають такі ключові завдання:

1. Автоматизоване подання DOM-структури у вигляді графа. Необхідно забезпечити перетворення HTML-документа у графову модель, де вузли відповідають елементам DOM, а ребра - відносинам «батько-дитина».
2. Формування векторних ознак вузлів. Для кожного елемента графа слід виділити ознаки, що характеризують його роль у структурі сторінки (тег, наявність атрибутів id, class, data-testid, глибина у дереві, кількість дочірніх елементів тощо).
3. Застосування графових нейронних мереж. Побудувати та навчити GNN-модуль для класифікації вузлів з метою оцінки їхньої важливості під час побудови селекторів.
4. Розробка агента формування селекторів. Створити агент, що, використовуючи вихідні дані GNN та евристичні правила, формує адаптивні та стійкі CSS-селектори.
5. Покращення селекторів. Забезпечити скорочення довжини селектора, уникнення надлишкових частин та вибір кращого варіанта за критерієм складності $L(S)$ при збереженні унікальності.

6. Оцінка якості селекторів. Реалізувати підсистему для обчислення метрик якості (Precision, Recall, F1-score) та часу пошуку елемента, що дозволяє кількісно оцінити ефективність зформованих селекторів.

3.1.1 Концептуальна модель інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого ПЗ ІС

Розробка концептуальної моделі інформаційної технології спрямована на узагальнене відображення структури та логіки її функціонування. На цьому етапі важливо було визначити основні елементи системи, характер їх взаємодії та послідовність обробки даних. Концептуальна модель відображає шлях, який проходить вхідний HTML-документ від моменту завантаження до формування оптимізованого селектора, що супроводжується обчисленням якісних характеристик.

Особливу увагу приділено принципам модульності та послідовності. Модель побудована так, щоб кожен етап процесу виконував окрему функцію: спочатку здійснюється парсинг HTML і перетворення його у DOM-граф, далі формується набір ознак для вузлів, які потім аналізуються графовою нейронною мережею. Результати цього аналізу використовуються агентом, що формує селектори з урахуванням як структурних властивостей документа, так і ймовірнісних прогнозів GNN. Після цього селектор оцінюванню за критеріями унікальності, довжини та якості (Precision, Recall, F1-score).

У результаті була сформована концептуальна модель інформаційної технології (рис. 3.1), яка містить такі основні модулі та етапи:

1. Вхідний модуль. Отримує HTML-документ як джерело даних.
2. Парсер та побудова DOM-графа. Перетворює HTML-структуру у графову модель, де вершини відповідають елементам, а ребра – відносинам між ними.
3. Модуль екстракції ознак. Формує векторні характеристики вузлів, що описують їхні властивості (тег, атрибути, глибина тощо).

4. GNN-модуль. Виконує класифікацію вузлів, визначаючи ймовірність їхньої важливості для побудови селектора.
5. Агент формування селекторів. На основі прогнозів GNN та евристик формує початковий селектор і оптимізує його.
6. Модуль оцінювання. Перевіряє унікальність селектора та обчислює метрики якості.
7. Вихідний модуль. Повертає оптимізований селектор разом з оцінкою його характеристик.



Рисунок 3.1 – Концептуальна модель інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого ПЗ ІС

Таким чином, концептуальна модель забезпечує цілісне бачення функціонування інформаційної технології. Вона не лише окреслює ключові етапи обробки, але й визначає принципи їх взаємодії: від послідовного перетворення даних до інтеграції машинного навчання з евристичними методами. Побудована модель закладає основу для подальшої логічної деталізації та програмної реалізації, що розглядається у наступних підрозділах.

3.1.2 Логічна модель інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого ПЗ ІС

Логічна модель деталізує внутрішню організацію інформаційної технології та відображає, яким чином відбувається взаємодія між її модулями. Якщо концептуальна модель окреслює загальну ідею переходу від HTML-документа до готового селектора, то логічна модель конкретизує послідовність операцій та структуру потоків даних у системі.

Робота інформаційної технології починається з обробки HTML-коду, який за допомогою парсера перетворюється у дерево елементів. Це дерево одразу структурується у вигляді графа, де вузли відповідають елементам сторінки, а ребра відображають ієрархічні відносини між ними. Далі формується набір ознак для кожного вузла, що включає як структурні характеристики (глибина розташування, кількість дочірніх елементів), так і атрибутивні (наявність id, class, data-testid). На цьому етапі відбувається кодування даних та їх нормалізація, що дозволяє передати їх у вигляді придатному для подальшого аналізу.

Отримані вектори ознак надходять у графову нейронну мережу, яка виконує класифікацію вузлів і визначає ймовірність того, що певний елемент може бути використаний для побудови стабільного селектора. На основі цих результатів працює агент, який комбінує прогнози GNN з евристичними правилами та формує початковий варіант селектора. Подальша перевірка та покращення

відбуваються у спеціалізованому модулі, де селектор оцінюється з точки зору унікальності, довжини та значень ключових метрик якості.

Логічна модель, таким чином, демонструє не лише послідовність виконання етапів, але й їхню тісну взаємодію. Кожен модуль передає результати наступному, формуючи єдиний потік обробки, що починається з вихідного HTML-документа і завершується отриманням оптимізованого селектора. Саме така організація забезпечує узгодженість роботи системи та створює підґрунтя для її програмної реалізації.

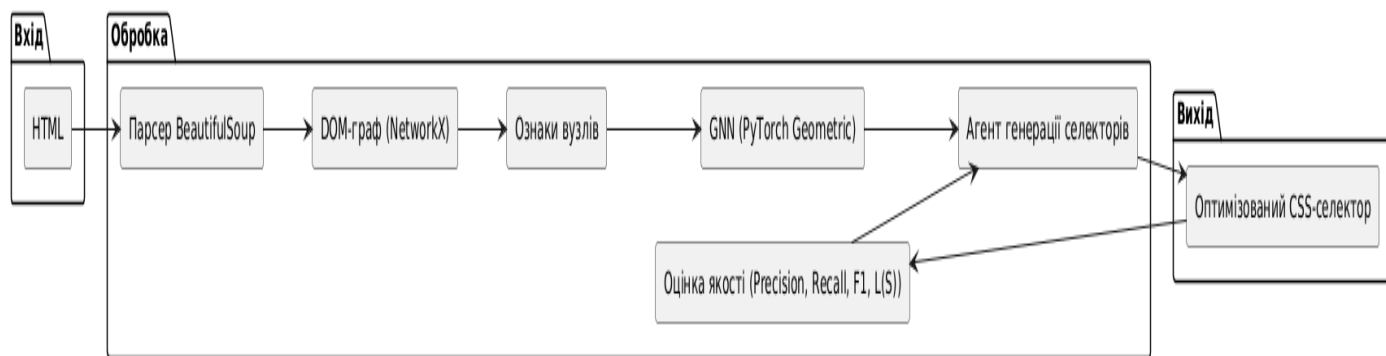


Рисунок 3.2 – Логічна модель інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого ПЗ ІС

3.1.3 Формування вхідних даних

Одним із базових етапів розробки інформаційної технології є формування якісних вхідних даних, оскільки саме від їхньої повноти та структурованості залежить коректність подальшої побудови DOM-графа та ефективність застосування графових нейронних мереж. У сучасних веб-ресурсах значна частина інформації генерується динамічно за допомогою JavaScript-фреймворків (React, Vue, Angular), що унеможлиблює використання традиційного підходу зі зчитуванням рядків коду HTML [43]. У такому випадку необхідно отримувати вже відрендерений документ, який повністю відповідає вигляду сторінки у браузері.

Для реалізації цього підходу в роботі застосовано інструмент Puppeteer – бібліотеку для автоматизації керування браузером Chrome/Chromium. Puppeteer

забезпечує завантаження веб-сторінки, виконання клієнтського JavaScript-коду, взаємодію з елементами інтерфейсу та подальше отримання актуального DOM. Використання цього інструменту дало змогу сформувати узгоджені набори даних для подальшої обробки, що критично важливо у випадках із динамічними інтерфейсами.

Отримані за допомогою Puppeteer HTML-сторінки зберігалися у вигляді локальних файлів та використовувалися для побудови графових структур. Аналіз цих даних показав, що у динамічних веб-ресурсах зростає середня глибина DOM-дерева, збільшується кількість вузлів, а також з'являється значна кількість атрибутів, які можуть змінюватися при кожному перезавантаженні сторінки. Зокрема, атрибути `id` та `class` у таких системах часто генеруються автоматично, що робить їх ненадійними для побудови селекторів.

Таким чином, використання Puppeteer дозволило вирішити одразу дві задачі: по-перше, забезпечити отримання достовірних структур даних, які відповідають фактичному стану сторінки після рендерингу; по-друге, виконати попередній аналіз особливостей сучасних DOM-структур, що створює основу для подальшої розробки механізмів формування та покращення селекторів.

3.2 Модуль побудови DOM-графа та екстракції ознак

Побудова DOM-графа є одним із ключових етапів розробки запропонованої інформаційної технології, оскільки саме ця структура виступає базовою моделлю для подальшого навчання графової нейронної мережі. DOM (Document Object Model) визначає ієрархічні відносини між елементами веб-сторінки, проте у звичайному вигляді це дерево має ряд обмежень для алгоритмічної обробки. Використання графового представлення дає змогу розширити модель DOM, перетворивши її на універсальну форму подання структурних даних, придатну для аналізу за допомогою GNN.

На відміну від звичайного DOM-дерева, побудований граф дозволяє не лише фіксувати вертикальні зв'язки, а й розглядати додаткові горизонтальні

залежності, наприклад, між сусідніми елементами одного рівня. Такий підхід робить модель більш наближеною до реальної логічної структури веб-документа, у якій поведінка елементів інтерфейсу часто визначається не лише їхнім розташуванням, а й контекстом сусідніх вузлів. Застосування бібліотеки NetworkX дало змогу зберігати цю інформацію у компактній формі, а також проводити операції пошуку, підрахунку глибини та аналізу топології графа.

Після побудови графа формується підсистема екстракції ознак, завдання якої полягає у перетворенні семантичної та структурної інформації про вузли у числовий формат, придатний для обробки нейронною мережею. На цьому етапі кожен вузол характеризується набором показників, які описують як його зовнішні властивості, так і позицію у структурі документа. До таких показників належать тип HTML-тегу (представлений у вигляді вектора one-hot кодування), глибина розташування у дереві, кількість дочірніх елементів, наявність атрибутів id, class та data-testid. Вибір саме цих ознак зумовлений тим, що вони мають найбільший вплив на ідентифікацію елемента у процесі побудови селектора.

Для підвищення стабільності навчання всі ознаки підлягають нормалізації за допомогою методу MinMaxScaler, який масштабує їх у діапазон $[0;1]$. Це дозволяє зменшити вплив окремих параметрів з великим числовим розкидом і забезпечити рівномірний внесок кожної характеристики у процес обчислень. Крім того, така нормалізація підвищує збіжність під час навчання GNN і дозволяє отримати більш стабільні результати при різних наборах вхідних даних.

Сформований у результаті DOM-граф із відповідною матрицею ознак є повноцінною моделлю структури веб-сторінки. Вона містить не лише опис елементів, але й топологічну інформацію, що відображає їх взаємне розташування та зв'язки. Така форма подання дозволяє застосовувати графові методи машинного навчання для виявлення закономірностей у структурі сторінки та визначення елементів, які найбільш підходять для побудови стабільних селекторів.

Таким чином, побудова DOM-графа та створення підсистеми екстракції ознак є фундаментальною складовою інформаційної технології. Вони забезпечують перехід від неструктурованого HTML-коду до формалізованої графової моделі, що слугує вхідними даними для графової нейронної мережі. Результатом цього етапу є підготовлений набір даних у вигляді пари $G = (V, E)$ та матриця ознак. Ця структура є основою для подальшого навчання моделі класифікації вузлів та формування оптимізованих селекторів.

Для реалізації графового представлення веб-документа розроблено алгоритм побудови DOM-графа, який забезпечує послідовне відтворення структури сторінки з урахуванням усіх елементів і вкладеностей. Основна ідея алгоритму полягає у перетворенні ієрархічної DOM-моделі у орієнтований граф $G = (V, E)$, де множина вузлів $V = \{v_1, v_2, \dots, v_n\}$ відповідає HTML-елементам, а множина ребер $E = \{(v_i, v_j)\}$ описує відносини «батько–дитина» між ними.

Процес побудови графа складається з кількох логічних етапів. На початковому етапі HTML-документ передається у модуль синтаксичного розбору, який формує базову DOM-структуру. Після цього алгоритм здійснює обхід елементів у ширину (метод Breadth-First Search, BFS), починаючи з кореневого елемента. Такий підхід дозволяє уникнути рекурсивного переповнення пам'яті при обробці великих сторінок та гарантує послідовне додавання вузлів у граф.

Для кожного елемента e_i створюється вузол v_i , до якого прив'язуються атрибути:

- назва тегу (tag);
- словник атрибутів (attrs);
- текстове наповнення (text);
- позиційні характеристики (порядковий номер, рівень глибини).

Якщо елемент має батьківський вузол, між ними встановлюється орієнтоване ребро (v_p, v_i) , що фіксує структурну ієрархію. У випадку, коли елемент має кілька дочірніх елементів, усі відповідні вузли додаються до черги

обробки. Для зберігання графа використовується структура даних DiGraph із бібліотеки NetworkX, яка дозволяє додавати атрибути як до вузлів, так і до ребер.

Формально алгоритм можна подати у такому вигляді:

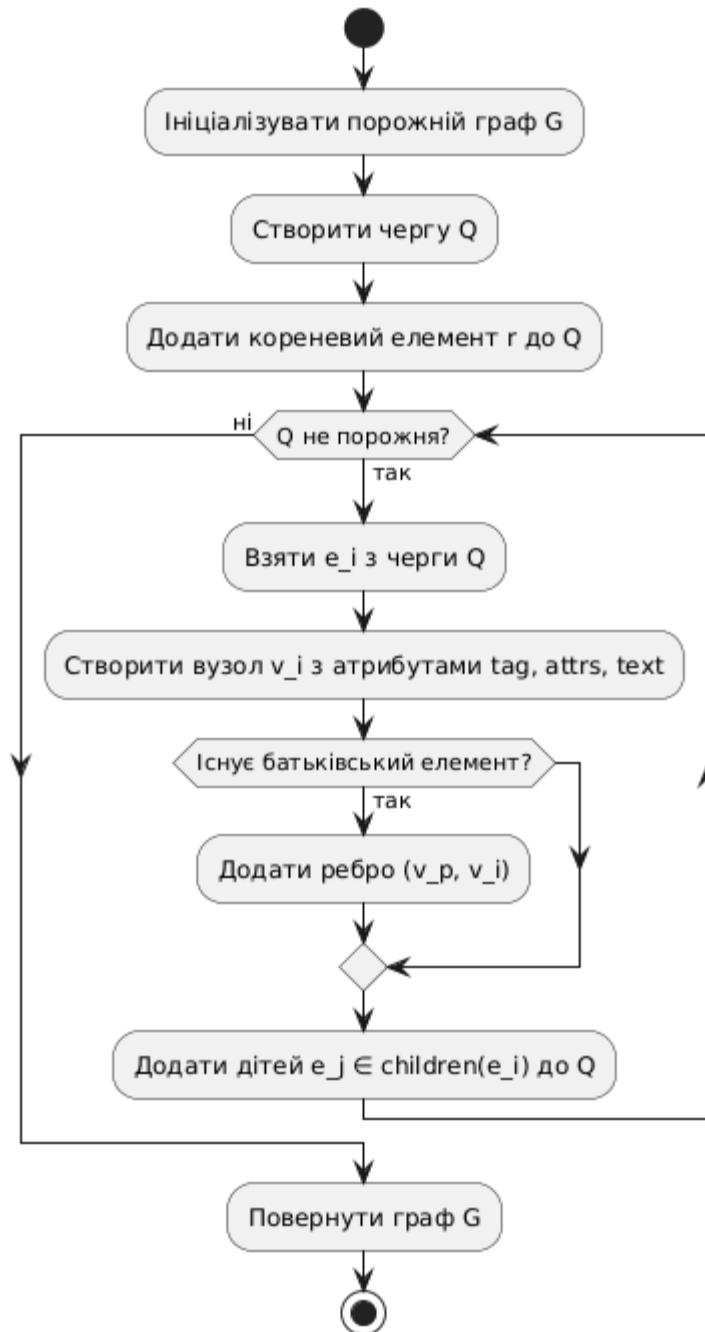


Рисунок 3.3 – Фрагмент алгоритму побудови дом-графа

Такий підхід дозволяє обробляти як статичні, так і динамічні веб-сторінки, що мають складну вкладену структуру. Використання обходу в ширину забезпечує контроль за порядком обробки елементів і дає змогу додатково

відстежувати рівень глибини кожного вузла, який у подальшому використовується як одна з ознак для графової нейронної мережі.

Отриманий DOM-граф є універсальною моделлю представлення веб-сторінки, яка зберігає як її семантичну, так і структурну інформацію. На наступному етапі він використовується підсистемою екстракції ознак для формування числового подання вузлів, що дозволяє здійснювати їх подальший аналіз у графових нейронних мережах.

Основні етапи роботи підсистеми екстракції ознак вузлів DOM-графа можна побачити на рисунку 3.4.

Отже, на даному етапі сформовано повну модель графового представлення веб-документа, що забезпечує зв'язок між HTML-структурою сторінки та вхідними даними для GNN. Це створює основу для наступного етапу – побудови та навчання графової нейронної мережі.

3.3 Реалізація GNN-модуля

Після формування графового представлення структури веб-документа та побудови матриці ознак вузлів наступним етапом розробки інформаційної технології є реалізація графової нейронної мережі (Graph Neural Network, GNN), призначеної для класифікації елементів DOM-графа та оцінки їхньої значущості у процесі побудови стабільних селекторів.

У даній роботі для моделювання структурних взаємозв'язків між елементами використано архітектуру Graph Attention Network (GAT), яка є однією з найефективніших модифікацій класичних GNN.

Її особливість полягає у застосуванні механізму уваги (attention mechanism) для вагового агрегування інформації від сусідніх вузлів, що дозволяє моделі навчатися розрізняти ступінь впливу кожного з них на поточний елемент. Це є критично важливим для задачі побудови селекторів, оскільки контекст елемента



Рисунок 3.4 – Основні етапи роботи підсистеми екстракції ознак вузлів DOM-графа

у DOM часто визначається не лише його безпосереднім батьком, а й сусідніми структурними компонентами [44].

Загальна архітектура GAT-модуля складається з двох послідовних шарів:

$$H^{(1)} = \text{ELU}(\text{GATConv}(X, A)) \quad (3.1)$$

$$H^{(2)} = \text{GATConv}(H^{(1)}, A), \quad (3.2)$$

де $X \in R^{N \times d}$ – матриця вхідних ознак, A – матриця суміжності графа, GATConv – шар уваги з коефіцієнтом a_{ij} , які визначають важливість вузла v_j для v_i . Кожен вузол на виході отримує вектор активації що інтерпретується як оцінка його значущості у структурі DOM. Функція активації ELU використовується для підвищення стабільності градієнтів і покращення збіжності [45]. Для запобігання перенавчанню застосовано dropout-регуляризацію ($p=0.3$), а на вихідному шарі -softmax-нормалізацію для отримання ймовірнісного розподілу класів.

Узагальнена структурна схема GNN-модуля показана на рисунку 3.5.

Метою GNN-модуля є класифікація кожного вузла v_i графа у два класи:

$y_i = 1$ – вузол є значущим елементом;

$y_i = 0$ – вузол є незначущим з точки зору унікальної ідентифікації елемента;

Формально, задача класифікації визначається як мінімізації крос-ентропійної функції втрат:

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^2 w_c \cdot y_{i,c} \log \hat{y}_{i,c}, \quad (3.3)$$

де w_c – вагові коефіцієнти класів, що компенсують дисбаланс даних, $y_{i,c}$ – реальна мітка класу, $\hat{y}_{i,c}$ – ймовірність, передбачена мережею.

Результатом роботи GNN є вектор вихідних логітів $\hat{Y} \in \mathbb{R}^{N \times 2}$ який після нормалізації визначає ступінь важливості кожного вузла у DOM-графі. Ці значення надалі використовуються агентом для вибору оптимального селектора.

Процес навчання моделі реалізовано у модулі GraphNeuralNetworkClassifier, який використовує два шари GATConv бібліотеки PyTorch Geometric [46, 47]. Навчання проводиться у супроводі циклу оптимізації, що включає пряме поширення сигналу (forward pass), обчислення функції втрат, зворотне поширення градієнта (backpropagation) та оновлення параметрів оптимізатором Adam з коефіцієнтом навчання $lr=0.0005$ [48]. Для оцінки точності навчання використовувалися три базові метрики: точність Recall та F1. Дані метрики дозволяють оцінити, наскільки точно GNN визначає вузли, що мають високу унікальність у структурі сторінки. У результаті навчання модель

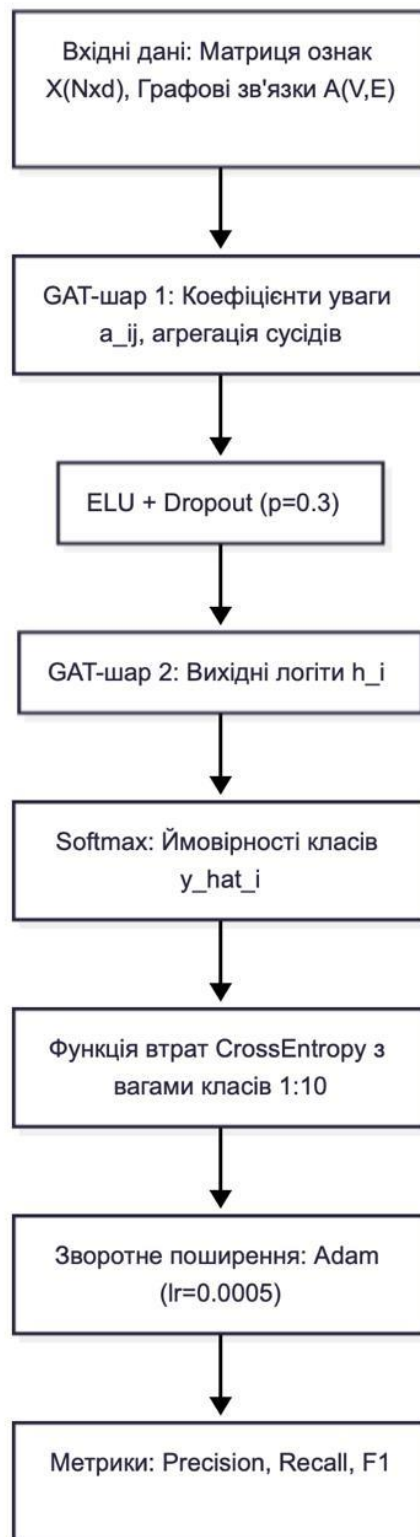


Рисунок 3.5 – Узагальнена структурна схема GNN-модуля

продемонструвала стабільну збіжність (рисунок 3.6), де показано зміну значень F1-score і функції втрат протягом 100 епох.

Для невеликих веб-сторінок (до 200 вузлів) середня точність класифікації досягала 92–94%, що свідчить про ефективне узагальнення GNN навіть на невеликих наборах даних.

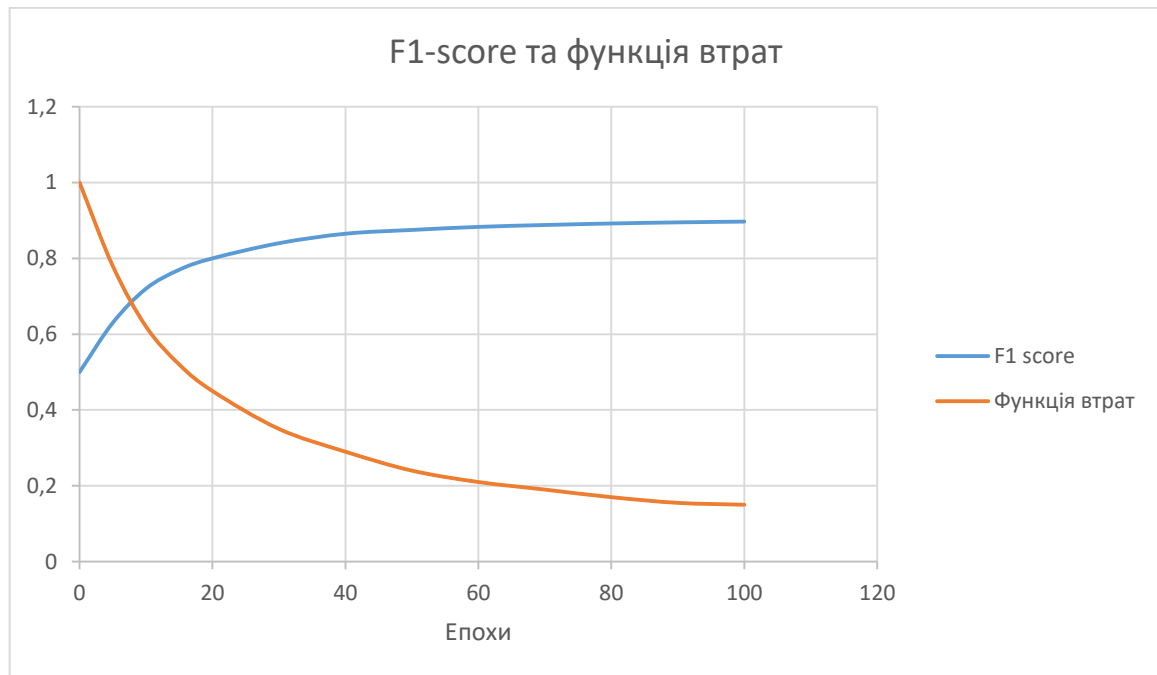


Рисунок 3.6 – графік F1-score та функції втрат

Розроблений GNN-модуль забезпечує:

- автоматичну ідентифікацію структурно значущих вузлів DOM-графа;
- формування векторів важливості, що передаються агенту генерації селекторів;
- зменшення фрагментації при зміні DOM та підвищення стабільності побудованих селекторів.

Таким чином, розроблена модель є центральним компонентом запропонованої інформаційної технології, оскільки саме вона формує інтелектуальну основу для адаптивної побудови CSS/XPath селекторів на основі контексту сторінки.

3.4 Алгоритм побудови селекторів

Побудова CSS-селекторів є ключовим компонентом запропонованої інформаційної технології, оскільки саме селектор визначає здатність автоматизованого тесту стабільно взаємодіяти з потрібним елементом веб-інтерфейсу. У межах розробленої системи запропоновано гібридний підхід, у якому поєднуються результати класифікації графовою нейронною мережею (GNN), структурні особливості DOM-дерева та набір доменних евристик. Такий підхід дозволяє адаптивно формувати селектори, які є одночасно унікальними, компактними та стійкими до змін інтерфейсу.

3.4.1 Характеристика основних принципів побудови селектора

Алгоритм побудови селекторів базується на п'яти ключових принципах.

1. Використання структурного контексту DOM. DOM-граф містить інформацію про ієрархічні зв'язки між елементами, їхню глибину, кількість дочірніх вузлів, приналежність до шаблонних блоків тощо. Ці параметри впливають на вибір оптимального шляху селектора, особливо для елементів у динамічних веб-додатках із повторюваними структурами.

2. Інтеграція прогнозів GNN-модуля. GAT-модель, повертає для кожного вузла ймовірність важливості p_i . В алгоритмі побудови селектора ця величина використовується для вибору стійкіших вузлів та обмеження необхідної глибини. Високі значення GNN-передбачень свідчать про наявність стабільних атрибутів або унікальної структурної позиції.

3. Пріоритетні евристики побудови селектора. Система використовує строгий порядок застосування атрибутів елемента:

- id
- data-testid
- name
- type

- class
- nth-child() (лише за необхідності)

Такий порядок забезпечує баланс між стабільністю та довжиною селектора.

4. Перевірка унікальності на кожному кроці побудови. Після формування кожної проміжної версії селектора виконується перевірка за DOM-парсером:

$$u(s) = 1, \text{ якщо } |Match(s)| = 1 \quad (3.4)$$

Якщо селектор унікальний, алгоритм переходить до етапу скорочення, а не продовжує нарощувати шлях.

5. Скорочення селектора після досягнення унікальності. Якщо селектор унікальний, далі виконується послідовне видалення верхніх рівнів шляху. Це мінімізує довжину селектора та зменшує ризик його нестабільності при оновленнях верстки.

Ці принципи забезпечують адаптивність, компактність і стійкість селекторів. Побудова селектора починається з аналізу атрибутів цільового елемента. Евристика має вигляд:

$$s_0 = \begin{cases} \text{tag, [id=...], якщо існує id} \\ \text{tag, [data-testid=...], якщо існує data-testid} \\ \text{tag, [name=...], якщо існує name} \\ \text{tag, [type=...], якщо існує type} \\ \text{tag.class}_1\text{class}_2\text{... , якщо існує class} \\ \text{tag : nth-child(k), у випадку наявності сусідніх елементів одного типу} \end{cases} \quad (3.5)$$

Сформований селектор розширюється до максимального рівня `max_depth`, додаючи батьківські вузли. На кожному кроці перевіряється унікальність селектора. Якщо селектор стає унікальним раніше – пошук зупиняється. Фрагмент алгоритму побудови селекторів зображений на рисунку 3.7.

3.4.2 Перевірка унікальності, скорочення довжини та складності селектора

Унікальність селектора є критичним критерієм, який визначає, чи може селектор однозначно ідентифікувати потрібний елемент. Перевірка виконується за допомогою пошуку елементів у DOM.

Якщо селектор знаходить 0 або більше одного елемента, він вважається неунікальним, і алгоритм продовжує пошук більш деталізованої структури – або через поглиблення, або через застосування додаткових атрибутів.

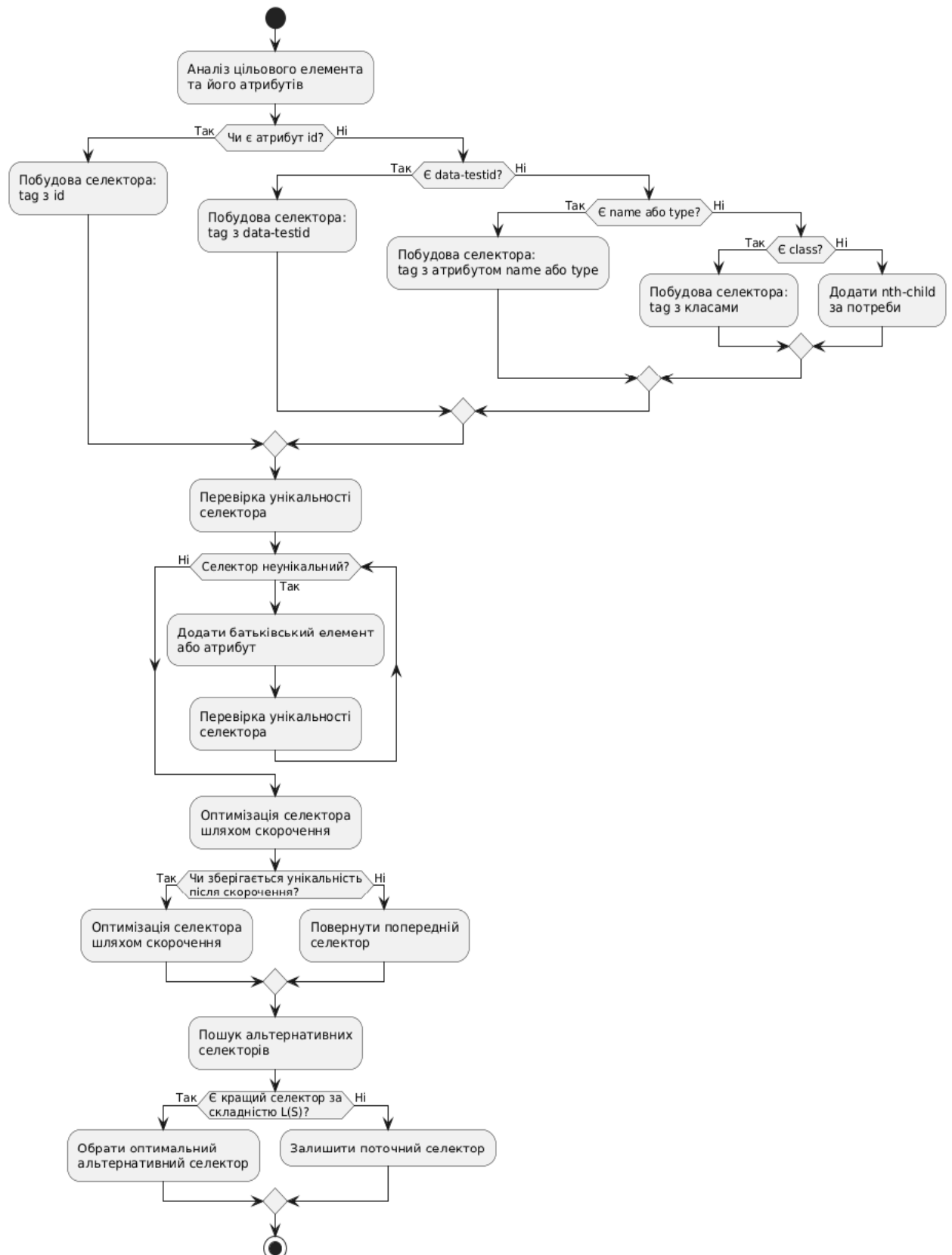


Рисунок 3.7 – Фрагмент алгоритму побудови селекторів

Скорочення довжини та структурної складності селектора є ключовим етапом у підвищенні стабільності автоматизованих тестів і забезпеченні їхньої ефективності. Надмірно довгі або структурно складні селектори збільшують час пошуку елементів у DOM, підвищують вразливість тестів до змін інтерфейсу та погіршують читабельність тестового коду. Тому в межах запропонованої інформаційної технології застосовано комбінований підхід, що об'єднує:

- Евристичні правила скорочення селектора;
- Метрику складності на основі вагових коефіцієнтів;
- Аналіз альтернативних варіантів селекторів;
- Обрізання ієрархії (top-down trimming) для усунення зайвих предків.

Таким чином, селектор оптимізується не лише на етапі формування, але й після нього, що забезпечує мінімальну складність при збереженні унікальності.

Після того як селектор став унікальним, виконується його скорочення за правилом зверху вниз:

```
div.container > section > ul > li > a
      ↓
section > ul > li > a
      ↓
ul > li > a
```

Скорочення припиняється, коли подальше видалення призводить до втрати унікальності.

Для кожного компоненту селектора визначено коефіцієнт ваги, що відображає його стабільність та передбачуваність при оновленнях DOM.

Таблиця 3.1 – Вагові коефіцієнти для компонентів селектору

Тег	Коефіцієнт
Id	0.2
Class	0.5
Data-testid	0.4

Продовження таблиці 3.1

nth-child	0.8
Вкладеність (>)	0.5

Використано зважену формулу:

$$L(S) = \sum_{i=1}^k w(c_i) \quad (3.6)$$

де k – кількість компонентів, $w(c_i)$ – вага кожного компонента залежно від його типу. Це дозволяє мінімізувати селектори, схильні до нестабільності (особливо з nth-child).

Для підвищення якості селекції після побудови основного селектора виконується пошук альтернатив. Для кожного можливого варіанта з множини:

- #id
- tag[data-testid="..."]
- tag[name="..."]
- tag.class1.class2
- спрощені структурні варіанти

перевіряється унікальність та складність $L(S)$. Найкращим вважається селектор з мінімальною складністю серед усіх унікальних.

Скорочення довжини селектора також дозволяє зменшити середній час пошуку елементів у DOM, що на практиці скорочує тривалість виконання UI-тестів. Фрагмент алгоритму скорочення довжини та складності селектора на рисунку 3.8.

3.4.3 Інтеграція GNN

Передбачення GNN використовується у процесі побудови селектора для:

- пріоритезації вузлів – якщо вузол має високий p_i , він частіше обирається як основа селектора;
- адаптивного обмеження глибини – глибина зменшується для вузлів з високою важливістю;



Рисунок 3.8 – Фрагмент алгоритму скорочення довжини та складності селектора

- оцінювання альтернатив – селектори, що включають високовагові вузли, отримують меншу штрафну складність.

Таким чином, GNN сприяє побудові стійкіших селекторів у динамічних DOM.

Модель GNN повертає для вузла v_i ймовірність важливості $p_i \in [0,1]$.

Таким чином, розроблений алгоритм формує основу інтелектуальної підсистеми формування селекторів запропонованої інформаційної технології.

3.5 Інтелектуальний агент та його взаємодія з GNN-модулем

У розробленій інформаційній технології ключову роль відіграє інтелектуальний агент, який відповідає за покрокове формування та скорочення селектора. Його робота не зводиться лише до перебору можливих варіантів, а ґрунтується на раціональному прийнятті рішень, сформованому на підставі двох джерел інформації: структурного контексту DOM-графа та оцінок важливості вузлів, отриманих із графової нейронної мережі (GNN). Таке поєднання дозволяє моделювати побудову селектора як процес адаптивного прийняття рішень у динамічному середовищі, що характерне для сучасних веб-інтерфейсів.

Агент працює у циклічній взаємодії з середовищем, що концептуально відповідає замкненій петлі «Середовище $\rightarrow$ Агент $\rightarrow$ Дія $\rightarrow$ Винагорода $\rightarrow$ Середовище». У ролі середовища виступає DOM-граф із його вузлами, атрибутами, глибиною вкладеності, структурними залежностями та значеннями важливості, які повертає GNN [49]. Стан середовища для агента репрезентується множиною:

$$s = \{v_i, f(v_i), gnn_score_i, depth_i, S_t\} \quad (3.7)$$

де v_i – поточний вузол, $f(v_i)$ – вектор його ознак, gnn_score_i – оцінка важливості GNN, $depth_i$ – глибина вузла у графі, а S_t – проміжний селектор на кроці t .

На основі цього стану агент обирає одну дію a_t , що модифікує селектор. Набір можливих дій охоплює як додавання атрибутивної інформації (id, data-testid, name, class), так і розширення шляху за рахунок батьківського вузла. У разі надмірної деталізації агент може виконати операцію скорочення селектора,

усуваючи структурні компоненти, які не впливають на його унікальність. Завершальний крок – підтвердження готовності селектора, коли він уже є унікальним й оптимізованим.

Поведінка агента формується через функцію винагороди, яка оцінює якість проміжного селектора. Функція винагороди відображає ключові критерії стабільності, компактності та структурної чистоти:

$$R = \alpha \cdot U(S) - \beta \cdot L(S) + \gamma \cdot gnn_score_i - \delta \cdot P_{frag}(S) \quad (3.8)$$

де $U(S)$ – індикатор унікальності (1 – селектор унікальний, 0 – ні), $L(S)$ – довжина селектора у вагових одиницях (див. розділ 2.4), gnn_score_i – важливість вузла, $P_{frag}(S)$ – штраф за «крихкі» патерни (наприклад, nth-child), $\alpha, \beta, \gamma, \delta$ – коефіцієнти вагомості критеріїв.

Довжина селектора враховується не за кількістю символів, а за ваговою структурною моделлю:

$$L(S) = \sum_{i=1}^k w_l \quad (3.9)$$

де k – кількість компонентів селектора, а w_l – ваговий коефіцієнт компонента (id, class, data-testid, тег, nth-child тощо). Завдяки цьому агенту вигідно уникати конструкцій із великою кількістю вкладених тегів або позиційних обмежень, оскільки вони суттєво збільшують $L(S)$ і знижують винагороду.

Прийняття рішень агентом можна формально подати як задачу вибору дії з максимальним прогнозованим значенням винагороди:

$$\pi(a_t | s_t) = \operatorname{argmax} R(s_t, a) \quad (3.10)$$

що відповідає детермінованій політиці, сформованій на основі евристик та GNN-оцінок. Таким чином агент спрямовується на ті операції, які забезпечують найкращий баланс між унікальністю та компактністю.

Інтеграція GNN у роботу агента має виняткове значення. Завдяки структурному аналізу графа GNN повертає для кожного вузла значення важливості, що відображає його потенційну стабільність і унікальність. Вузли з високим gnn_score зазвичай мають надійні атрибути (id, data-testid) або розташовані в характерних структурних контекстах (наприклад, кореневі блоки компонентів у SPA-додатках). Агент, орієнтуючись на такі вузли, мінімізує

кількість зайвих кроків і уникає побудови селекторів, що спираються на нестійкі або повторювані елементи DOM.

Узгоджена робота GNN та агента утворює єдиний процес побудови селектора. Спочатку агент формує атрибутивне ядро селектора, ґрунтуючись на найбільш стабільних характеристиках елемента. Якщо цього недостатньо для досягнення унікальності, він розширює селектор за рахунок батьківських вузлів, одночасно отримуючи від GNN додаткові оцінки щодо їхньої важливості. Після досягнення унікальності запускається етап наступний етап, у якому агент моделює зворотне скорочення шляху:

$$S_{t+1} = Trim(S_t) \quad (3.11)$$

доки селектор не стане мінімальним за довжиною без втрати унікальності. Такий підхід дозволяє отримувати селектори, які за своїми властивостями є порівняними з тими, що створює досвідчений інженер з автоматизованого тестування, однак мають додаткову перевагу в системності та відтворюваності.

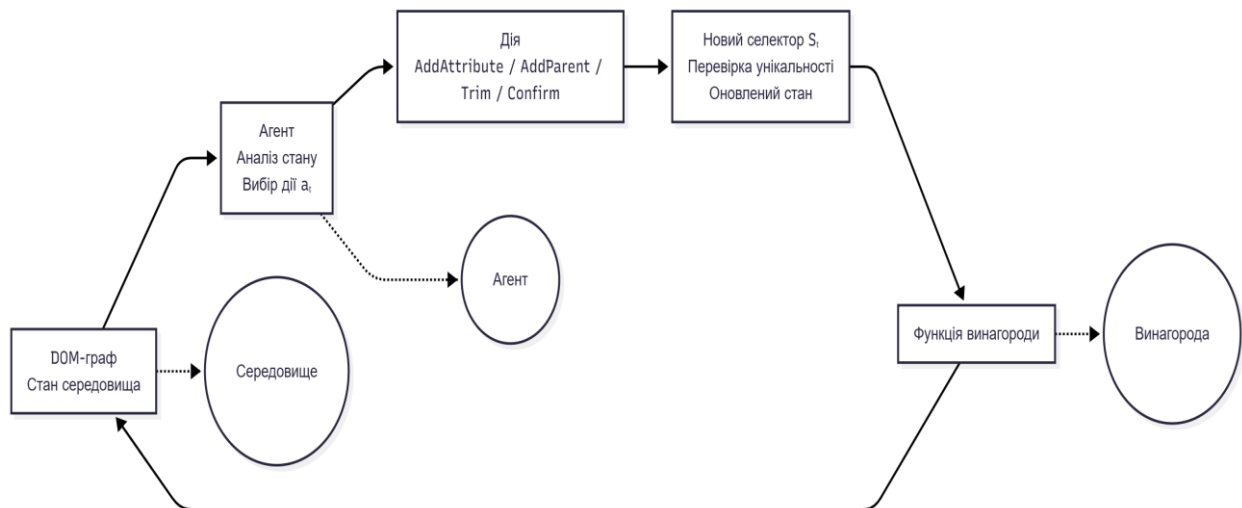


Рисунок 3.9 – Схема роботи інтелектуального агента в контексті DOM-середовища

Таким чином, інтелектуальний агент у взаємодії з GNN забезпечує адаптивне, контекстно залежне формування селекторів, що дозволяє значно підвищити їхню стабільність у динамічних веб-інтерфейсах та зменшити кількість помилок, пов'язаних зі змінами DOM. Поєднання структурної

аналітики, формальної моделі оцінювання та механізму прийняття рішень створює основу для гнучкої й ефективної побудови тестових селекторів у рамках запропонованої інформаційної технології.

3.6 Підсистема оцінювання якості селектора

Підсистема оцінювання якості селекторів у розробленій інформаційній технології виконує функцію формального контролю відповідності зформованих селекторів вимогам стабільності, точності та ефективності. На відміну від традиційних підходів, де коректність селектора визначається виключно його здатністю знайти цільовий елемент, запропонована підсистема забезпечує багатовимірний аналіз, що враховує як структурні характеристики селектора, так і його поведінку у зміненому DOM. Такий підхід дозволяє кількісно оцінювати ступінь його надійності, а також обґрунтовано обирати найкращий варіант серед декількох кандидатів.

У межах підсистеми оцінювання використовується узагальнена модель, що включає три групи характеристик. Перша група стосується унікальності селектора. Унікальність визначається кількістю елементів у DOM-дереві, які відповідають конкретному селектору, і формально виражається співвідношенням:

$$u(s) = 1, \text{ якщо } |Match(s)| = 1 \quad (3.12)$$

тобто селектор вважається коректним, якщо він однозначно ідентифікує лише один елемент. Саме ця властивість виступає базовою умовою для оцінювання решти параметрів, оскільки селектор, що не є унікальним, втрачає будь-яку практичну цінність для автоматизованого тестування [50].

Друга група характеристик пов'язана з якістю селекції та відображає здатність селектора правильно визначати свій цільовий елемент серед множини можливих вузлів DOM. Для цього використовується підхід, аналогічний оцінюванню моделей класифікації. Нехай для елемента з індексом i у множині всіх вузлів сформовано вектор істинних міток

$$y_i = \begin{cases} 1, j = i \\ 0, j \neq i \end{cases} \quad (3.13)$$

а селектор повернув множину передбачених елементів P , на основі якої формується вектор передбачень $\hat{y}$. На основі цих величин обчислюються метрики точності, повноти та F1-міри [51]:

$$Precision = \frac{TP}{TP+FP}, \quad Recall = \frac{TP}{TP+FN}, \quad F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision+Recall} \quad (3.14)$$

Ці параметри дозволяють визначити, наскільки саме селектор «узгоджується» з очікуваною поведінкою та чи не вказує він на зайві елементи.

Окреме місце у підсистемі займає оцінювання структурної складності селектора. Складність визначається на основі вагової моделі, сформованої в попередньому дослідженні автора. Кожному компоненту селектора (тег, клас, ідентифікатор, атрибут data-testid, псевдоклас nth-child, а також операція вкладеності) призначається ваговий коефіцієнт, що характеризує його стабільність та схильність до змін. Сумарна складність селектора визначається виразом (3.6). Така оцінка дає змогу порівнювати між собою декілька унікальних селекторів та обирати той, який є структурно простішим і, відповідно, потенційно більш стабільним у перспективі багатьох релізів програмного продукту.

Ще однією важливою характеристикою є стабільність селектора щодо змін у DOM-структурі. Для цього селектор перевіряється не лише на початковій версії сторінки, але й на модифікованих варіантах, що можуть виникати внаслідок незначних змін стилізації, додавання службових елементів або перегруповання внутрішніх блоків. Стабільність визначається як частка успішних ідентифікацій серед усіх сценаріїв перевірки:

$$Stability(S) = \frac{m}{T} \quad (3.15)$$

де m – кількість випадків, у яких селектор зберіг унікальність, а T – загальна кількість тестових сценаріїв. Високе значення цього показника свідчить про низьку залежність селектора від конкретної структури DOM.

Підсистема оцінювання виконує послідовну перевірку селектора. Насамперед визначається його унікальність; у разі невідповідності вимога

унікальності подальше оцінювання припиняється. Для унікального селектора формується вектор істинних і передбачених міток, на основі яких обчислюються метрики точності, повноти та F1-міри. Далі визначається структурна складність за ваговою моделлю $L(S)$. Після цього селектор перевіряється на кількох сценаріях модифікованого DOM для визначення стабільності. Завершальним етапом є розрахунок інтегральної оцінки якості, яка дозволяє порівнювати селектори між собою. Основні етапи роботи підсистеми оцінювання якості селекторів зображені на рисунку 3.10.

Усі перелічені характеристики інтегруються у єдину систему оцінювання, яка дає змогу зіставляти селектори як у межах одного елемента, так і в ширшому контексті тестового набору. Програмна реалізація підсистеми забезпечує автоматичне обчислення відповідних метрик.

Підсистема оцінювання таким чином відіграє роль формального механізму контролю якості селекторів, забезпечуючи їх об'єктивне порівняння та надаючи кількісне підтвердження ефективності розробленого алгоритму побудови та скорочення селекторів.

3.7 Обмеження та перспективи розвитку інтелектуальної інформаційної технології

Розроблена інтелектуальна інформаційна технологія побудови CSS-селекторів на основі графових нейронних мереж і інтелектуального агента демонструє високу ефективність під час роботи з різними типами DOM-структур, проте, як і будь-яке комплексне програмне рішення, вона має низку обмежень. Аналіз цих обмежень є необхідним як з погляду вдосконалення моделі, так і для визначення напрямів майбутніх досліджень.

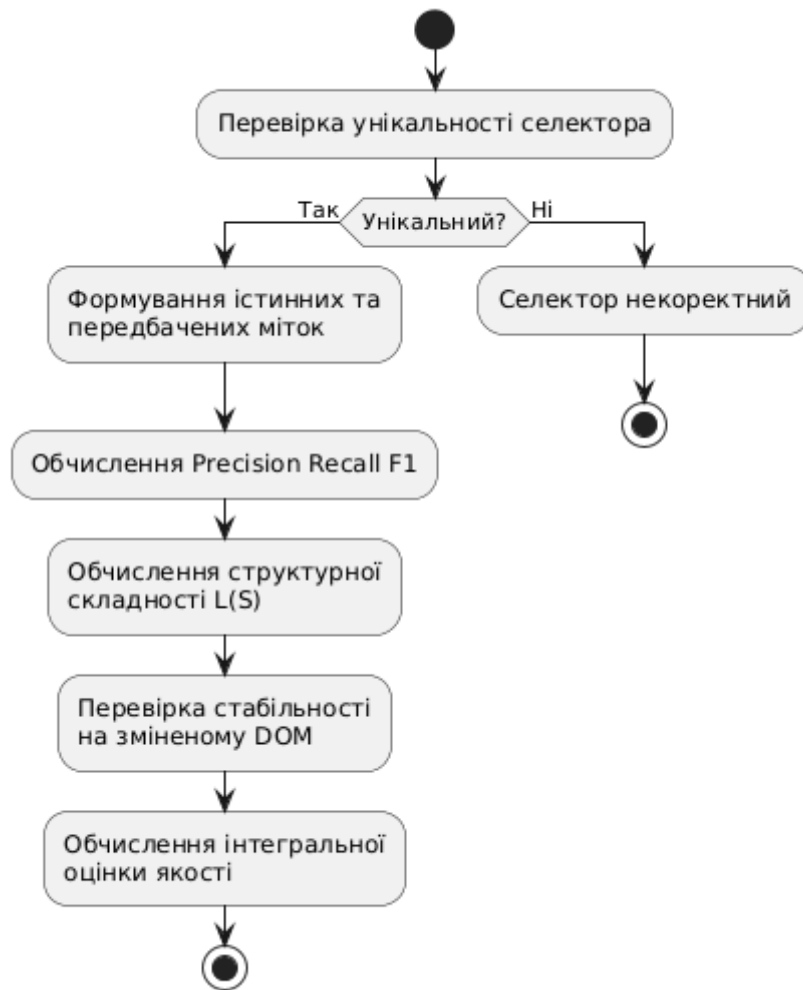


Рисунок 3.10 – Основні етапи роботи підсистеми оцінювання якості селекторів

Одним із ключових обмежень є залежність якості побудованих селекторів від повноти та різноманітності тренувальних даних. DOM-структури різних веб-ресурсів мають суттєві відмінності у глибинах вкладеності, принципах компонування та підходах до використання атрибутів. Для деяких сайтів характерне активне використання динамічної генерації елементів, що може суттєво змінювати ієрархію під час роботи інтерфейсу. У таких випадках GNN-модуль може мати недостатньо контекстної інформації, особливо якщо подібні патерни не були представлені у тренувальному наборі. Це зумовлює потребу у розширенні датасету за рахунок веб-ресурсів із різними архітектурними підходами – від класичних серверних рендерингів до сучасних SPA-фреймворків.

Важливим обмеженням є також часові витрати на попередню обробку даних. Побудова DOM-графа, нормалізація ознак та створення матриці суміжності для великих сторінок із сотнями або тисячами вузлів може вимагати значних обчислювальних ресурсів. Хоча сам процес побудови селектора є досить швидким, початкова обробка впливає на масштабованість системи при інтеграції з реальними тестовими фреймворками або CI/CD-пайплайнами.

Обмеження стосуються і агента. Хоча реалізований агент використовує евристичні принципи та GNN-оцінки, його політика залишається детермінованою. Це означає, що система виконує оптимізацію за обмеженими правилами, які не завжди охоплюють усю різноманітність змін у сучасних інтерфейсах. Зокрема, алгоритм не розглядає стохастичні варіанти дій або альтернативні траєкторії побудови селектора, які могли б привести до кращого балансу між стабільністю та компактністю. Це відкриває перспективу використання методів підкріплювального навчання (Reinforcement Learning), зокрема таких алгоритмів як PPO, DDPG або A3C, які здатні адаптивно вивчати політику побудови оптимальних селекторів у змінному середовищі [52-55].

Окремим напрямом розвитку є дослідження альтернативних архітектур графових моделей. Хоча використання Graph Attention Network дало змогу ефективно моделювати контекст кожного вузла, інші архітектури – такі як GraphSAGE або Graph Isomorphism Networks (GIN) – можуть покращити здатність моделі узагальнювати знання для раніше невідомих DOM-структур [56, 57]. Наприклад, GraphSAGE дозволяє формувати ембеддинги на основі індуктивного навчання, що особливо важливо при обробці нових сайтів, які не входили до тренувального набору [35]. У свою чергу, GIN забезпечує кращу дискримінаційну здатність графової моделі, що може бути корисним для аналізу складних компонентів SPA-додатків [58].

Перспективним також є впровадження компонентів self-supervised learning для поліпшення якості GNN-ембеддингів [59]. Методики на кшталт node masking, graph contrastive learning або structural prediction можуть дозволити

моделі отримувати глибші структурні уявлення навіть за відсутності повної розмітки [60, 61].

Ще одним напрямом удосконалення є розширення підсистеми оцінювання якості селекторів. У поточній версії вона охоплює такі характеристики, як точність, повнота, F1-міра, унікальність, стабільність та структурна складність. Однак для практичного застосування в промислових CI/CD-середовищах доцільним є включення додаткових метрик, таких як швидкість знаходження елемента браузером, кількість помилкових спрацьовувань у довготривалих тестових сесіях, а також відмовостійкість селектора при одночасній появі декількох схожих елементів.

Отже, попри очевидні переваги застосування GNN та інтелектуального агента для побудови тестових селекторів, система не позбавлена обмежень, що пов'язані з різноманітністю DOM-структур, вартістю попередньої обробки та детермінованістю політики агента. Перспективи розвитку пов'язані з поглибленням навчальних можливостей моделі, інтеграцією сучасних RL-алгоритмів, розширенням датасету та удосконаленням архітектури GNN. Сукупність цих напрямів може значно підвищити адаптивність і практичну цінність інформаційної технології в реальних умовах автоматизованого тестування.

3.8 Архітектура інтелектуальної інформаційної технології та інтеграція її модулів

Архітектура розробленої інформаційної технології побудована на принципах багаторівневої модульності, що забезпечує узгоджену взаємодію між усіма етапами аналізу, побудови та оцінювання CSS-селекторів. Такий підхід дозволяє не лише формально відокремити функціональні компоненти, а й забезпечити їхню гнучкість, розширюваність та можливість подальшого вдосконалення відповідно до потреб автоматизованого тестування веб-ресурсів. Інтеграція модулів здійснюється таким чином, щоб обробка веб-сторінки перетворювалась

на єдиний конвеєр, починаючи від отримання DOM-структури й завершуючи формуванням оптимального селектора, що проходить валідацію якісними метриками та може бути інтегрований у рядки коду автоматизованого тесту для подальшої взаємодії з відповідним DOM-елементом.

Формування вхідних даних забезпечується модулем збору веб-інформації, який відповідає за отримання фінального вигляду DOM-дерева. Використовуючи браузерний рушій Puppeteer, система зчитує не лише статичний HTML-код, але й результати рендерингу, ініційовані JavaScript, що особливо важливо для аналізу сучасних SPA-компонентів та динамічних інтерфейсів [62]. Отримана структура перетворюється на внутрішнє графове представлення, у якому вузли відповідають елементам, а ребра – структурним зв'язкам і вкладеності. У цій же підсистемі відбувається нормалізація даних та формування набору ознак, що включає як атрибутивні, так і структурні характеристики елементів.

Наступним рівнем архітектури є модуль графового аналізу, побудований на основі графової нейронної мережі. Використання GNN дає змогу моделювати DOM не як плоску HTML-структуру, а як семантичний граф, у якому враховується контекст кожного вузла та його зв'язки з іншими елементами. GNN обчислює для кожного вузла узагальнену оцінку важливості, яка використовується агентом як сигнальна інформація щодо того, наскільки перспективним є включення певного елемента або його предка у формований селектор. Фактично GNN формує вектор ймовірностей, що відображає структурну стабільність та інформаційну значущість елемента при потенційних змінах DOM.

У центральній частині архітектури розміщується інтелектуальний агент – ключовий компонент інтелектуальної системи, що виконує адаптивну побудову селектора. Його взаємодія з середовищем описується циклічною петлею, у якій кожен крок включає формування нового стану, прийняття дії, оновлення селектора та оцінку його якості. Агент використовує дані, отримані з GNN-модуля, а також локальний контекст вузла, з якого розпочинається побудова

селектора. Завдяки функції винагороди агент не лише досягає унікальності селектора, але й прагне мінімізувати його складність, що дозволяє уникати надмірно вкладених або крихких конструкцій. Коли селектор досягає унікальності, агент переходить до етапу структурного спрощення, поступово видаляючи зайві компоненти до моменту, коли подальше скорочення спричинить втрату однозначності.

Після побудови селектора він передається у підсистему оцінювання якості, яка здійснює кількісний аналіз відповідно до критеріїв, визначених у розділі 2.4. Ця підсистема перевіряє унікальність селектора, порівнює його очікувану поведінку з фактичними елементами, які він знаходить у DOM, обчислює метрики точності, повноти та F1-міри, а також визначає структурну складність на основі вагових коефіцієнтів. Важливою частиною цієї підсистеми є оцінювання стабільності селектора при різних модифікаціях DOM, що дозволяє визначити його надійність у випадку зміни верстки або еволюції інтерфейсу. Результатом роботи підсистеми є інтегральна оцінка, яка дає змогу об'єктивно порівняти кілька кандидатів та обрати найкращий.

Завершальний рівень архітектури забезпечує інтеграцію отриманих результатів у зовнішні системи автоматизованого тестування, такі як Selenium, TestNG або інші фреймворки, що використовуються у промислових процесах розробки [63, 64]. Селектори експортуються у стандартизованому форматі, що дозволяє безпосередньо включати їх у тестові сценарії. Такий підхід створює замкнений цикл – від аналізу веб-сторінки до автоматизованого тесту – і забезпечує практичну застосовність запропонованої технології.

У результаті інтеграції всіх функціональних модулів формується цілісна інформаційна технологія, здатна не лише автоматично формувати селектори, але й кількісно оцінювати їхню якість та стійкість, адаптуючись до різномірних та змінних веб-інтерфейсів. Архітектура забезпечує гнучкість, формалізованість і можливість подальшого розширення, що робить її придатною як для наукових досліджень, так і для практичного використання у сучасних системах автоматизованого тестування.

Компонентна діаграма відображає архітектуру розробленої інформаційної технології та інтеграцію її основних модулів (рис.3.11). На діаграмі виділено кілька логічних рівнів, що відповідають етапам обробки веб-ресурсу: рівень доступу до сторінки, рівень представлення DOM-структури, аналітичний рівень та рівень інтеграції з зовнішніми системами тестування. На початковому етапі браузерний рушій (Puppeteer) забезпечує отримання фінального стану HTML/DOM-структури, яка далі передається до модулів парсингу та побудови DOM-графа. Сформований граф разом із векторизованими ознаками вузлів використовується GNN-модулем для обчислення оцінок важливості елементів.

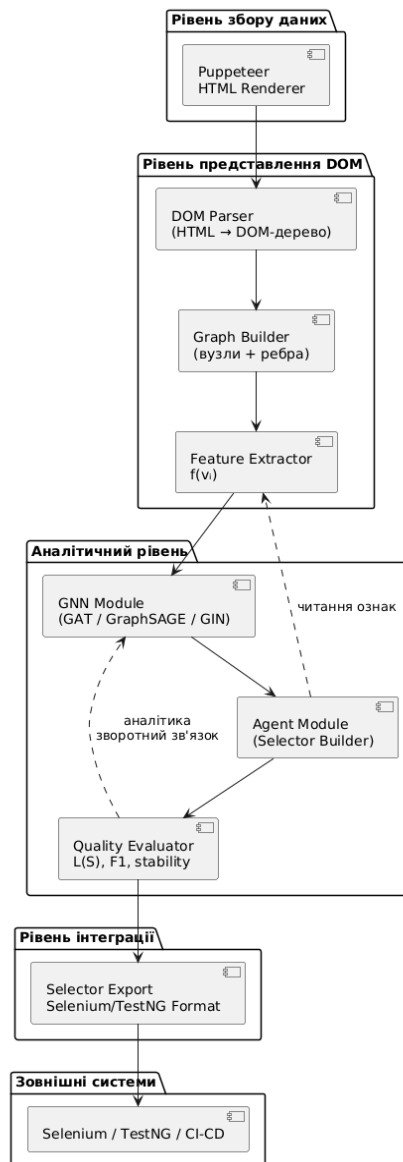


Рисунок 3.11 – Компонентна діаграма інтелектуальної інформаційної технології

У центральній частині діаграми розташовано інтелектуальний агент та підсистему оцінювання якості селекторів, які утворюють ядро інформаційної технології. Агент, спираючись на результати GNN-аналізу, формує та оптимізує селектори, тоді як підсистема оцінювання визначає їхню унікальність, структурну складність, точність, повноту та стабільність. Завершальний компонент архітектури відповідає за інтеграцію отриманих селекторів у зовнішні тестові фреймворки (Selenium, TestNG, CI/CD-середовища), що забезпечує практичне використання результатів роботи системи. Таким чином, компонентна діаграма демонструє, як окремі модулі, описані в попередніх підрозділах, об'єднуються в єдину інформаційну технологію, орієнтовану на підвищення стабільності та ефективності автоматизованого тестування веб-ресурсів.

Діаграма потоків даних відображає послідовність перетворення інформації у межах розробленої інформаційної технології – від моменту отримання веб-сторінки до формування готових селекторів для використання в автоматизованих тестах. На ній показано, як HTML/DOM-структура, отримана з веб-ресурсу за допомогою інструмента Puppeteer, послідовно передається до модулів парсингу, побудови DOM-графа та екстракції ознак, після чого потрапляє у GNN-модуль для визначення важливості вузлів. Подальша обробка здійснюється інтелектуальним агентом, який на основі отриманих даних формує селектори, а підсистема оцінювання якості перевіряє їхню коректність та стабільність. Завершальним етапом є передавання відібраних селекторів до модуля експорту, який інтегрує їх у зовнішні тестові фреймворки, що забезпечує практичне застосування результатів роботи інформаційної технології.



Рисунок 3.12 – Діаграма потоків даних інтелектуальної інформаційної технології

Конвеєр обробки DOM-графа ілюструє внутрішню логіку взаємодії між графовою нейронною мережею, інтелектуальним агентом та підсистемою оцінювання якості селекторів. Спочатку DOM-структура веб-сторінки перетворюється на граф із відповідними ознаками вузлів, які подаються на вхід GNN-модулю. На основі обробки графа формується оцінка важливості вузлів, що використовується як вхідна інформація для агента під час побудови селектора. Агент у кілька ітерацій оновлює селектор, поки не буде досягнуто його унікальності та прийнятної структурної складності. Після цього селектор

спрямовується до підсистеми оцінювання якості, де розраховуються метрики точності, повноти, F1-міри, довжини та стабільності. Таким чином, конвеєр відображає послідовний, але адаптивний процес, у якому поєднуються глибокий графовий аналіз, агентне прийняття рішень та формалізована оцінка результату.



Рисунок 3.13 - Конвеєр обробки DOM-графа GNN-модулем, агентом та підсистемою оцінювання

Окремим елементом архітектури є динамічна взаємодія між модулями, що відбувається під час тренування моделі та подальшого формування селектора. Повна UML-діаграма послідовностей відображає процеси завантаження DOM,

побудови графа, формування ознак, проходження даних через GNN, обчислення функції втрат, оновлення ваг моделі, а також використання логітів для роботи агентного модуля. Така діаграма дозволяє побачити взаємозв'язки між компонентами в реальному часовому порядку, що підкреслює інтегрованість системи та завершеність запропонованої інформаційної технології. Зображено на рисунку 3.14.

3.9 Висновки до розділу 3

У даному розділі розроблено повну інтелектуальну інформаційну технологію автоматизованої побудови та оцінювання CSS-селекторів для веб-ресурсів на основі графових нейронних мереж та інтелектуального агента. Запропонована технологія складається з низки взаємопов'язаних модулів, кожен з яких виконує окрему функціональну роль у загальному конвеєрі формування стабільних селекторів.

На першому етапі сформовано графове представлення DOM-структури, що дало змогу перейти від плоскої HTML-моделі до формального графа з ознаками вузлів та ребер. Побудована підсистема екстракції ознак формує векторне подання елементів, включаючи їхні атрибутивні характеристики, локальний контекст та структурні параметри. Такий підхід забезпечує можливість подальшого застосування графових методів машинного навчання.

У підрозділі 3.3 реалізовано GNN-модуль на основі архітектури Graph Attention Network (GAT). Модель успішно формує оцінки важливості вузлів та дозволяє визначати структурно значущі елементи DOM, що слугують опорними точками для побудови селектора. Навчання GNN забезпечило отримання стійких ембеддингів, які коректно узагальнюють інформацію про різні типи веб-сторінок.

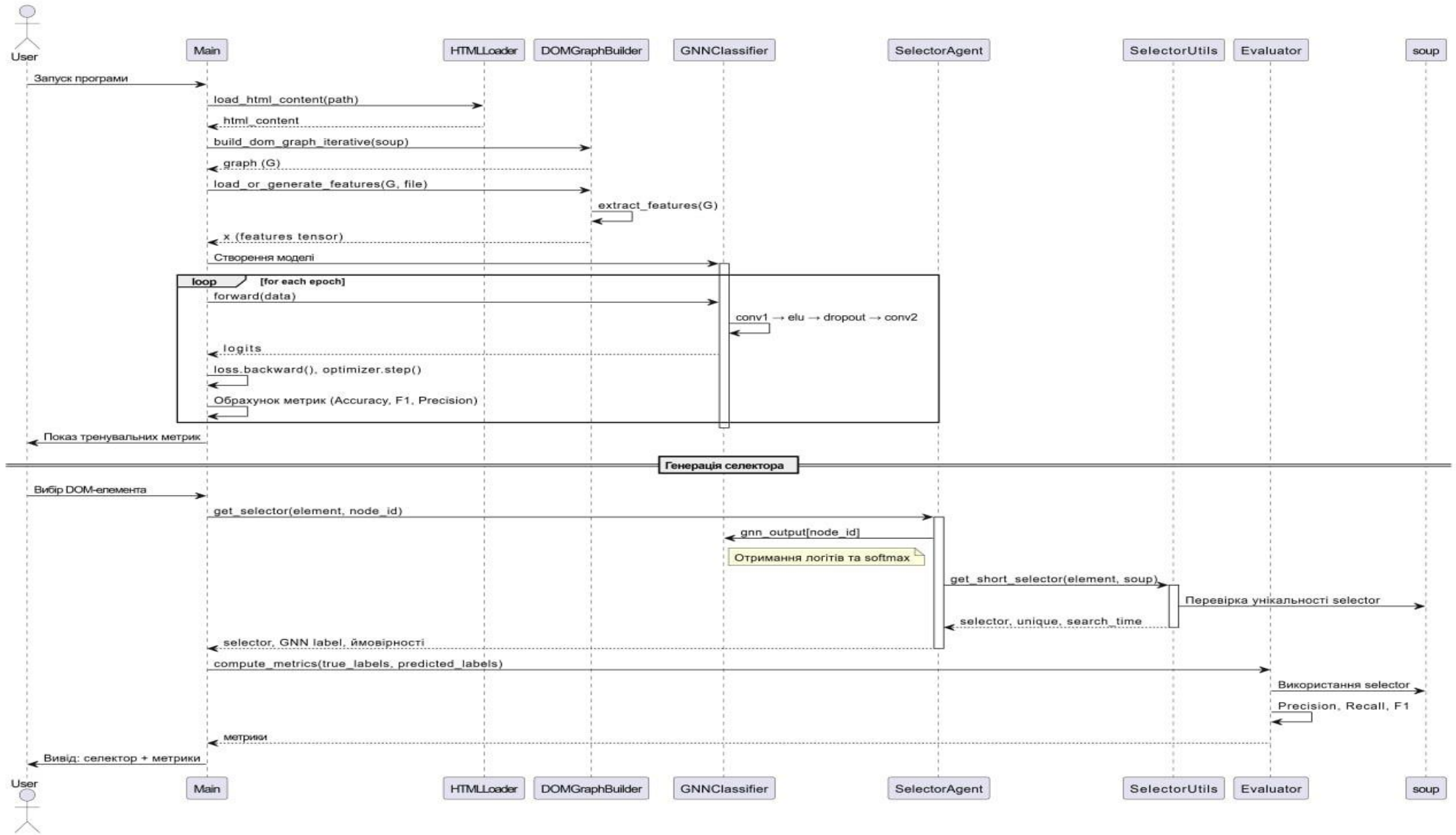


Рисунок 3.14 – Повна UML-діаграма послідовностей з тренуванням

Підрозділ 3.4 присвячено розробці алгоритму побудови CSS-селекторів, який поєднує евристики, інформацію з DOM-контексту та прогнози GNN. Було запропоновано механізми перевірки унікальності селектора, його скорочення зверху вниз та оцінювання структурної складності на основі вагових коефіцієнтів для компонентів. Такий підхід забезпечує компактність та стабільність селекторів при мінімальному ризику втрати унікальності.

У підрозділі 3.5 розроблено інтелектуальний агент, що реалізує адаптивний процес побудови селектора в циклічній взаємодії зі середовищем. Він виконує впорядковані дії над селектором, керуючись формальною функцією винагороди, яка включає унікальність, структурну довжину, оцінки важливості вузлів та штрафи за крихкі конструкції. Інтеграція агента з GNN створює замкнену систему, здатну формувати оптимальні селектори навіть у складних та динамічних DOM-структурах.

Підсистема оцінювання якості (підрозділ 3.6) забезпечує формальний контроль правильності та стабільності селектора, включаючи метрики точності, повноти, F1-міри, структурної складності та стійкості до змін DOM. Це дозволило створити універсальний механізм порівняння кандидатів та забезпечити об'єктивність результатів.

У підрозділі 3.7 проаналізовано обмеження технології та визначено перспективні напрями розвитку, серед яких використання RL-методів (PPO, DDPG), застосування GIN або GraphSAGE для підвищення узагальнюваності, а також використання self-supervised learning для побудови структурних ембеддингів.

Архітектура розробленої інформаційної технології (підрозділ 3.8) продемонструвала поєднання всіх модулів у єдиний конвеєр – від отримання DOM, побудови графа та роботи GNN до селекторного агента й підсистеми оцінювання. Розроблені схеми підтвердили логічність, формальну завершеність та узгоджену взаємодію модулів.

У результаті виконаної роботи створено повноцінний функціональний прототип інформаційної технології, який:

- формує графове подання DOM;
- виконує класифікацію вузлів за допомогою GNN;
- будує оптимальний CSS-селектор за допомогою агента;
- оцінює якість і стабільність селектора за формальними метриками;
- забезпечує інтеграцію результатів у системи автоматизованого тестування.

Запропонована технологія закладає основу для подальших експериментальних досліджень, описаних у наступному розділі, та має значний потенціал до практичного застосування у промислових процесах UI-тестування.

РОЗДІЛ 4 ЕКСПЕРЕМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ІНТЕЛЕКТУАЛЬНОЇ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ WEB-ОРІЄНТОВАНОГО ПЗ ІС

4.1 Особливості організації експериментальних досліджень

Експериментальне дослідження є завершальним етапом перевірки ефективності розробленої інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого ПЗ ІС, що базується на застосуванні графових нейронних мереж та інтелектуального агента для побудови стабільних тестових селекторів. Необхідність проведення експерименту зумовлена потребою кількісного підтвердження переваг запропонованого підходу порівняно з традиційними методами формування селекторів, а також оцінювання його стійкості до змін структури DOM у динамічних і статичних web-орієнтованого ПЗ ІС.

Метою експериментального дослідження є перевірка ефективності розробленої інформаційної технології шляхом порівняльного аналізу якості тестових селекторів, зформованих із використанням графової нейронної мережі та інтелектуального агента, з селекторами, отриманими традиційними підходами автоматизованого тестування.

Для досягнення поставленої мети в межах експериментального дослідження були сформульовані та послідовно реалізовані такі завдання:

- дослідити роботу розробленої технології на різних типах веб-ресурсів, що відрізняються складністю та динамічністю DOM-структур;
- оцінити якість зформованих селекторів за формалізованими метриками унікальності, точності, повноти, F1-міри, довжини, структурної складності та стабільності;
- виконати порівняння запропонованого підходу з традиційними методами побудови селекторів;

- проаналізувати вплив використання графової нейронної мережі на процес прийняття рішень інтелектуальним агентом;
- визначити характер зміни якості селекторів у разі модифікації DOM-структури веб-ресурсу.

Організація експериментального дослідження ґрунтується на поетапному виконанні серії контрольованих експериментів, у межах яких розроблена інформаційна технологія застосовувалася до набору веб-ресурсів із різною архітектурою та характером динамічних змін. Для кожного веб-ресурсу формувалося графове подання DOM-структури, здійснювалася екстракція ознак вузлів, після чого виконувалася класифікація елементів за допомогою графової нейронної мережі. Отримані оцінки важливості вузлів використовувалися інтелектуальним агентом у процесі побудови тестових селекторів.

Експериментальне дослідження проводилося у декількох режимах, що відрізнялися підходами до формування селекторів. Це дозволило виконати коректний порівняльний аналіз та виключити вплив сторонніх факторів на результати експерименту. Для забезпечення відтворюваності результатів усі експерименти виконувалися в однаковому програмно-апаратному середовищі з фіксованими параметрами моделі та агента.

Отримані результати експериментального дослідження слугують основою для подальшого кількісного аналізу ефективності запропонованої інформаційної технології, виявлення її переваг та обмежень, а також формування узагальнених висновків щодо доцільності застосування графових нейронних мереж для процесу автоматизованого тестування веб-ресурсів.

4.2 Розробка методики проведення експериментальних досліджень

Методика проведення експериментальних досліджень визначає формалізовану процедуру оцінювання ефективності інтелектуальних інформаційної технології автоматизованого тестування web-орієнтованого ПЗ ІС та забезпечує відтворюваність і об'єктивність отриманих результатів.

Методика ґрунтується на принципі контрольованого порівняння та включає послідовність взаємопов'язаних етапів.

1. Формування вхідних даних для експерименту

1.1. Вибір веб-ресурсів для дослідження відповідно до їх архітектурних особливостей, рівня динамічності та складності DOM-структур.

1.2. Отримання фінального стану DOM-структури кожного веб-ресурсу після повного виконання клієнтського JavaScript-коду.

1.3. Фіксація версії DOM-дерева для усунення впливу випадкових змін інтерфейсу на результати експерименту.

2. Побудова графового представлення DOM

2.1. Перетворення DOM-дерева у графову модель, де вершини відповідають HTML-елементам, а ребра відображають ієрархічні та структурні зв'язки.

2.2. Формування матриці суміжності або списку ребер для графового представлення DOM.

2.3. Перевірка коректності побудованого графа шляхом контролю зв'язності та відповідності структури вихідному DOM.

3. Екстракція та нормалізація ознак вузлів

3.1. Формування векторів ознак для кожного вузла DOM-графа, що включають атрибутивні, структурні та контекстні характеристики.

3.2. Нормалізація числових ознак та кодування категоріальних атрибутів для подальшої обробки графовою нейронною мережею.

3.3. Формування єдиного вхідного простору ознак для всіх веб-ресурсів, що забезпечує узгодженість експериментів.

4. Обчислення важливості вузлів за допомогою GNN

4.1. Подання DOM-графа та векторів ознак на вхід графової нейронної мережі.

4.2. Обчислення узагальнених представлень вузлів та значень їхньої важливості.

4.3. Збереження результатів GNN-інференсу для подальшого використання інтелектуальним агентом.

5. Формування цільових елементів тестування

5.1. Вибір множини цільових елементів веб-інтерфейсу, що типово використовуються в автоматизованих UI-тестах.

5.2. Закріплення цільових елементів для всіх порівнюваних методів побудови селекторів.

5.3. Перевірка наявності кожного цільового елемента у зафіксованій DOM-структурі.

6. Формування тестових селекторів

6.1. Побудова селекторів із використанням розробленої інформаційної технології на основі GNN та інтелектуального агента.

6.2. Формування селекторів традиційними методами автоматизованого тестування для забезпечення порівняльного аналізу.

6.3. Фіксація зформованих селекторів для кожного методу та кожного цільового елемента.

7. Первинна перевірка коректності селекторів

7.1. Перевірка унікальності кожного селектора у вихідному DOM.

7.2. Відкидання неунікальних селекторів з подальшого аналізу.

7.3. Формування множини коректних селекторів для подальшого оцінювання.

8. Оцінювання якості селекторів

8.1. Обчислення метрик точності, повноти та F1-міри для кожного селектора.

8.2. Визначення структурної складності селектора на основі вагової моделі.

8.3. Агрегація значень метрик для кожного методу побудови селекторів.

9. Аналіз стабільності селекторів при зміні DOM

9.1. Виконання контрольованих модифікацій DOM-структури веб-ресурсу.

9.2. Повторна перевірка селекторів у зміненому DOM-середовищі.

9.3. Визначення стабільності селекторів як частки успішних ідентифікацій цільового елемента.

10. Узагальнення та інтерпретація результатів

10.1. Усереднення значень метрик для всіх веб-ресурсів та цільових елементів.

10.2. Побудова таблиць і графіків для наочного порівняння методів.

10.3. Формування висновків щодо ефективності та практичної доцільності запропонованої інформаційної технології.

4.3 Налаштування та параметри експериментального середовища

Для забезпечення коректності, відтворюваності та об'єктивності експериментального дослідження всі експерименти проводилися в уніфікованому програмно-апаратному середовищі з фіксованими параметрами. Такий підхід дозволяє мінімізувати вплив зовнішніх факторів на результати дослідження та гарантує коректність порівняльного аналізу між різними методами побудови тестових селекторів.

Експериментальне середовище було розгорнуте на персональній обчислювальній системі з сучасною багатоядерною архітектурою процесора та достатнім обсягом оперативної пам'яті для обробки великих DOM-структур.

Апаратна конфігурація забезпечувала стабільне виконання операцій побудови графів, навчання графової нейронної мережі та ітеративної роботи інтелектуального агента без виникнення обчислювальних обмежень, що могли б вплинути на результати експерименту.

Програмна частина експериментального середовища включала інтерпретатор мови Python та набір спеціалізованих бібліотек для обробки веб-даних і машинного навчання. Для отримання фінального стану DOM-структури веб-ресурсів використовувалась бібліотека Puppeteer, який забезпечує коректне виконання клієнтського JavaScript-коду та формування динамічного DOM, характерного для сучасних веб-додатків [65]. Отриманий HTML-документ підлягав подальшому парсингу та перетворенню у графове представлення.

Для побудови та навчання графової нейронної мережі використовувався фреймворк PyTorch у поєднанні з бібліотекою PyTorch Geometric, що дозволяє ефективно працювати з графовими структурами та реалізовувати сучасні архітектури GNN. У межах дослідження застосовувалася модель Graph Attention Network, параметри якої були зафіксовані для всіх експериментів. Кількість шарів мережі, розмірність прихованих представлень, кількість шарів уваги та функція активації обиралися таким чином, щоб забезпечити баланс між якістю класифікації вузлів та обчислювальною складністю моделі.

Для забезпечення відтворюваності експериментального дослідження та виключення впливу варіації гіперпараметрів на результати, усі ключові параметри графової нейронної мережі були зафіксовані. Значення параметрів наведено в таблиці 4.1.

Таблиця 4.1 – Параметри графової нейронної мережі

Параметр	Значення
Тип архітектури	GAT
Кількість шарів	2
Розмірність вхідного вектора ознак	Визначається набором ознак DOM-вузла
Розмірність прихованого шару	64

Продовження таблиці 4.1

Функція активації	ELU
Функція втрат	Cross-entropy
Оптимізатор	Adam
Швидкість навчання	0.001
Кількість епох навчання	100

Навчання GNN-модуля здійснювалося на попередньо сформованому наборі DOM-графів із відповідною розміткою, після чого модель використовувалася в режимі прямого проходу під час проведення експериментів. Параметри навчання, зокрема кількість епох, швидкість навчання та функція втрат, залишалися незмінними для всіх запусків, що дозволяє розглядати отримані результати як порівнювані.

Інтелектуальний агент, який реалізує процес побудови селекторів, працював у детермінованому режимі з фіксованими евристичними правилами та коефіцієнтами функції винагороди. Значення вагових коефіцієнтів, що враховують унікальність селектора, його структурну складність, важливість вузлів за оцінками GNN та штрафи за крихкі конструкції, були визначені експериментально на етапі налаштування системи та не змінювалися під час основного експериментального дослідження.

Для забезпечення коректного порівняння результатів усі методи побудови селекторів застосовувалися до однакових веб-ресурсів, однакових цільових елементів та однакових версій DOM-структур. Кожен експеримент повторювався в ідентичних умовах, що виключає вплив випадкових факторів та дозволяє інтерпретувати отримані результати як наслідок використання відповідного підходу до формування селекторів.

4.4 Аналіз та підготовка вхідних даних для проведення експериментальних досліджень

Для проведення експериментального дослідження ефективності розробленої інформаційної технології було сформовано репрезентативний набір веб-ресурсів, що відрізняються за архітектурою, складністю DOM-структур та характером динамічних змін інтерфейсу. Такий підхід дозволяє оцінити універсальність запропонованого методу та його здатність працювати в умовах, наближених до реального використання автоматизованого тестування.

До експериментального набору увійшли DOM-моделі з різних сайтів, які містили різноманітні веб-сторінки з різними структурами (25 реальних сайтів) – інтернет-магазини, статичні сайти, новинні ресурси, медіа-ресурси, стрімінгові сервіси. Всі їх можна поділити на такі категорії:

- веб-сторінки з незмінною структурою DOM – ці сторінки використовуються для перевірки точності та стабільності селекторів у відсутності динамічних змін.
- інтерфейси, що змінюються в реальному часі, включаючи елементи, які можуть бути додані або видалені без перезавантаження сторінки,
- веб-ресурси, де DOM-структура змінюється без повного перезавантаження сторінки, що є складною ситуацією для традиційних методів вибору селекторів, оскільки динамічні зміни інтерфейсу ускладнюють стабільність селекторів.

Для кожного веб-ресурсу аналізувався фінальний стан DOM-структури, отриманий після повного виконання клієнтських сценаріїв JavaScript. Це є принципово важливим для коректного дослідження, оскільки значна частина сучасних веб-додатків формує або модифікує DOM-дерево динамічно під час виконання. Отриманий DOM використовувався як вхідні дані для побудови графового представлення, у якому вершини відповідають елементам сторінки, а ребра відображають ієрархічні та структурні зв'язки між ними.

```

<body>

  <div class="dialog-off-canvas-main-canvas" data-off-canvas-main-canvas="">
    <div class="header-line">
      <div class="container-fluid horzlimiter">
        <a id="vntu-logo" href="https://vntu.edu.ua/"></a>
        <nav class="navbar transparent navbar-inverse">
          <div class="container-fluid horzlimiter">
            <div class="collapse navbar-collapse">
              <div id="vntu-title" class="navbar-form navbar-left">
                <h1>Вінницький Національний Технічний Університет</h1>
              </div>
              <div class="nav navbar-nav navbar-right buttons-box">
                <a href="/go/account" class="btn btn-primary" role="button">Особистий кабінет</a>
                <a href="/go/vstup_howto" class="btn btn-primary" role="button">Як вступити?</a>
              </div>
            </div>
          </div>
        </nav>
      </div>
    </div>
    <nav id="vntu-navbar" class="navbar navbar-default">
      <div class="container-fluid horzlimiter leftlimiter">
        <div class="navbar-header">
          <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-navbar-collapse-1" aria-expanded="false">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
        </div>
        <div class="collapse navbar-collapse" id="bs-navbar-collapse-1">
          <div class="region region-navigation-collapsible">
            <nav role="navigation" aria-labelledby="block-vntu-main-menu-menu" id="block-vntu-main-menu">
              <h2 class="sr-only" id="block-vntu-main-menu-menu">Main navigation</h2>
              <ul class="nav navbar-nav navbar-hoverenable">
                <li class="dropdown first">
                  <a href="/" class="dropdown-toggle" aria-haspopup="true" aria-expanded="false">Університет</a>
                  <ul class="dropdown-menu">
                    <li class="first">
                      <a href="/">Головна</a>
                    </li>
                    <li>
                      <a href="/uk/about-university/vntu-today.html">Про університет</a>
                    </li>
                    <li>
                      <a href="/uk/about-university/administration.html">Адміністрація університету</a>
                    </li>
                    <li>
                      <a href="/uk/topic/rejtingi-1505.html">ВНТУ

```

Рисунок 4.1 – Приклад DOM-структури

Кількісні характеристики веб-ресурсів варіювалися у широкому діапазоні. Середня кількість вузлів DOM-дерева становила від кількох сотень до кількох тисяч елементів залежно від типу сайту. Особлива увага приділялася сторінкам із повторюваними шаблонними блоками, вкладеними компонентами та динамічно згенерованими елементами керування, оскільки саме такі структури створюють найбільші труднощі для побудови стабільних тестових селекторів традиційними методами.

У межах кожного веб-ресурсу було відібрано набір цільових елементів, що типово використовуються в автоматизованих UI-тестах. До них належали кнопки, поля введення, елементи навігації, списки та компоненти форм. Для кожного такого елемента виконувалася побудова селекторів різними методами, що дозволило отримати порівнювані результати для подальшого аналізу.

Таким чином, сформований експериментальний набір веб-ресурсів забезпечує достатню різноманітність DOM-структур і сценаріїв використання, що дозволяє об'єктивно оцінити ефективність розробленої інформаційної технології, її стійкість до змін інтерфейсу та переваги порівняно з традиційними підходами до формування тестових селекторів.

4.5 Опис та порівняння підходів до формування тестових селекторів

Для об'єктивного оцінювання ефективності розробленої інформаційної технології в межах експериментального дослідження було визначено кілька підходів до формування тестових селекторів, які відрізняються рівнем автоматизації та способом використання інформації про структуру DOM. Залучення різних підходів дозволяє здійснити коректний порівняльний аналіз і кількісно оцінити переваги запропонованого методу відносно традиційних практик автоматизованого тестування.

Базовим для порівняння виступає ручний підхід до побудови тестових селекторів, за якого селектори формуються тестувальником шляхом аналізу DOM-структури веб-сторінки з використанням інструментів розробника браузера. У цьому випадку вибір атрибутів, глибини вкладеності та ієрархічних зв'язків між елементами здійснюється на основі досвіду та інтуїції спеціаліста. Незважаючи на широку поширеність такого підходу в практиці автоматизованого тестування, він характеризується високою суб'єктивністю та значною чутливістю до змін верстки, особливо у випадку динамічних веб-додатків, де структура DOM може змінюватися між запусками тестів [65].

Другим порівнюваним підходом є напівавтоматизований метод формування селекторів, що базується на використанні інструментів автоматизованого запису дій користувача. У межах цього підходу селектори формуються автоматично на основі зафіксованих взаємодій із веб-інтерфейсом, після чого можуть бути відредаговані вручну. Такий підхід дозволяє скоротити час початкового створення селекторів, однак він, як правило, не враховує структурний і семантичний контекст елементів DOM. У результаті зформовані селектори часто мають надмірну довжину, включають позиційні обмеження та демонструють низьку стабільність при навіть незначних змінах DOM-структури.

Запропонований у дисертаційній роботі підхід реалізує повністю автоматизоване формування тестових селекторів на основі графових нейронних мереж та інтелектуального агента. На відміну від традиційних методів, DOM-структура веб-сторінки розглядається не як плоске дерево, а як граф, що дозволяє враховувати контекст кожного елемента та його взаємозв'язки з іншими вузлами. Графова нейронна мережа обчислює оцінки важливості вузлів DOM, які відображають їхню структурну значущість і потенційну стабільність у разі змін інтерфейсу. Ці оцінки використовуються інтелектуальним агентом у процесі покрокової побудови селектора.

Процес формування селектора в запропонованому підході формалізується як задача прийняття рішень у динамічному середовищі, де агент прагне досягти унікальності селектора за мінімальної структурної складності. На відміну від ручних і напів-автоматизованих методів, агент цілеспрямовано уникає крихких конструкцій, таких як позиційні обмеження або надмірна вкладеність, і віддає перевагу атрибутам та структурним патернам, що демонструють більшу стійкість до змін DOM. Це дозволяє формувати селектори, які є компактними, інтерпретованими та придатними для використання в умовах еволюції веб-інтерфейсу.

Таким чином, у межах експериментального дослідження порівнюються підходи, що репрезентують різні рівні автоматизації процесу формування

тестових селекторів – від повністю ручного до інтелектуально автоматизованого. Такий вибір порівнюваних методів створює надійну основу для подальшого кількісного аналізу якості селекторів і дозволяє обґрунтовано оцінити практичну ефективність розробленої інформаційної технології.

4.6 Метрики оцінювання якості тестових селекторів

Оцінювання якості тестових селекторів у межах експериментального дослідження здійснюється на основі формалізованої системи метрик, що дозволяє кількісно охарактеризувати коректність селекції, структурну ефективність і стійкість селекторів у змінному DOM-середовищі. Такий підхід забезпечує об'єктивне порівняння різних методів формування селекторів і унеможливорює суб'єктивну інтерпретацію результатів.

Базовою вимогою до тестового селектора є унікальність, тобто його здатність однозначно ідентифікувати цільовий елемент веб-інтерфейсу:

$$u(s) = \begin{cases} 1, \text{ якщо } |Match(s)| = 1 \\ 0, \text{ інакше} \end{cases} \quad (4.1)$$

Для кількісної оцінки коректності селекції використовується підхід, аналогічний оцінюванню задач класифікації. Нехай TP – кількість коректно ідентифікованих цільових елементів, FP – кількість помилково ідентифікованих елементів, а FN – кількість випадків, коли цільовий елемент не був знайдений. Тоді точність та повнота селектора визначаються відповідно як у формулах (2.14) та (2.15).

Для узагальненого оцінювання коректності селекції використовується F1-міра, яка дозволяє збалансовано врахувати як точність, так і повноту у формулі (2.16).

Окрему увагу в дослідженні приділено структурній складності селектора, яка безпосередньо впливає на його стабільність та читабельність. На відміну від простого підрахунку кількості символів, у роботі використовується вагова модель довжини селектора, відповідно до якої кожному компоненту селектора призначається коефіцієнт стабільності [66]. Зображено у формулі (2.17).

Важливою характеристикою є стабільність селектора щодо змін DOM-структури веб-ресурсу. Для її оцінювання селектор перевіряється на множині модифікованих версій DOM, що імітують типові зміни інтерфейсу. Стабільність визначається формулою (3.15).

Для забезпечення комплексного оцінювання якості тестових селекторів у межах експериментального дослідження запропоновано узагальнену інтегральну метрику, яка поєднує коректність селекції, структурну ефективність та стабільність [67]. Інтегральна оцінка якості селектора визначається виразом:

$$Q(S) = \sqrt[3]{F_1(S) \cdot Stability(S) \cdot (1 - \frac{L(S)}{L_{max}})}, \quad (4.2)$$

де L_{max} – максимальна довжина селектора в межах експериментального набору, використана для нормалізації значення $L(S)$.

Запропонована інтегральна метрика дозволяє порівнювати різні селектори не лише за окремими показниками, а й за їх узагальненою якістю, що є особливо важливим при аналізі результатів експериментального дослідження. Значення $Q(S)$ використовується в наступних підрозділах для побудови порівняльних таблиць і графіків, а також для формування обґрунтованих висновків щодо ефективності розробленої інформаційної технології.

4.7. Аналіз результатів експериментальних досліджень

4.7.1 Програмна реалізація експериментальних досліджень

Для проведення експериментального дослідження DOM-структура кожного веб-ресурсу перетворювалась у графове подання, у якому вершини відповідають HTML-елементам, а ребра – ієрархічним зв'язкам між ними. Побудова графа реалізована програмно та застосовувалась однаково для всіх порівнюваних підходів.

```

def build_dom_graph_iteratively(html_soup):
    dom_graph = nx.DiGraph()
    root_element = html_soup.find()
    elements_processing_queue = deque([(root_element, None)])
    current_node_index = 0

    while elements_processing_queue:
        current_element, parent_node_id = elements_processing_queue.popleft()
        if current_element is None:
            continue

        node_identifier = current_node_index
        current_node_index += 1

        dom_graph.add_node(node_identifier,
                           tag=current_element.name,
                           attrs=current_element.attrs,
                           text=current_element.string or "")

        if parent_node_id is not None:
            dom_graph.add_edge(parent_node_id, node_identifier)

        for child_element in current_element.children:
            if hasattr(child_element, 'name') and child_element.name is not None:
                elements_processing_queue.append((child_element, node_identifier))

    return dom_graph

```

Рисунок 4.2 – Побудова DOM-графу

Для кожного вузла DOM-графа формувався вектор ознак, що включав атрибутивні та структурні характеристики елементів. Саме ці ознаки використовувалися GNN-моделлю для оцінювання важливості вузлів і подальшого формування тестових селекторів.

```

def extract_node_features_from_graph(network_graph):
    extracted_features_list = []

    for node_id in network_graph.nodes():
        current_node_data = network_graph.nodes[node_id]

        # One-hot encoding для тегу
        tag_onehot_vector = tag_onehot_encoder.transform([[current_node_data['tag']]])[0]

        # Атрибути елемента
        element_attributes = current_node_data.get('attrs', {})

        # Глибина в DOM дереві
        try:
            node_depth_in_dom = nx.shortest_path_length(network_graph, source=0, target=node_id)
        except nx.NetworkXNoPath:
            node_depth_in_dom = 0

        # Кількість дочірніх елементів
        children_count = len(list(network_graph.successors(node_id)))

        # Бінарні ознаки наявності атрибутів
        has_class_attribute = 1.0 if 'class' in element_attributes else 0.0
        has_id_attribute = 1.0 if 'id' in element_attributes else 0.0
        has_data_testid_attribute = 1.0 if 'data-testid' in element_attributes else 0.0

        # Об'єднання всіх ознак
        combined_features = (list(tag_onehot_vector) +
                             [node_depth_in_dom, has_class_attribute, has_id_attribute,
                              has_data_testid_attribute, children_count])
        extracted_features_list.append(combined_features)

```

Рисунок 4.3 – Формування ознак DOM-вузлів

Після формування графового представлення та векторів ознак здійснювався прямий прохід графової нейронної мережі, результатом якого були числові оцінки важливості кожного DOM-вузла.

```

gnn_classification_model.eval()
with torch.no_grad():
    final_model_output = gnn_classification_model(graph_data_structure)
    final_predictions = final_model_output.argmax(dim=1)

```

Рисунок 4.4 – Прямий прохід графової нейронної мережі

Оцінки важливості вузлів, отримані з GNN, використовувалися інтелектуальним агентом для покрокового формування тестового селектора з урахуванням стабільності та мінімальної структурної складності.

```
def generate_css_selector_for_element(self, target_html_element, element_node_id):
    optimized_selector, is_selector_unique, selector_search_time = generate_optimized_css_selector(
        target_html_element, parsed_html_soup
    )

    gnn_prediction_logits = self.neural_network_output[element_node_id]
    gnn_prediction_probabilities = torch.softmax(gnn_prediction_logits, dim=0)
    predicted_element_label = torch.argmax(gnn_prediction_logits).item()
```

Рисунок 4.5 – Формування селектора на основі оцінок GNN

4.7.2 Результати експериментальних досліджень

Перед проведенням безпосереднього експериментального порівняння підходів до формування тестових селекторів доцільно оцінити коректність і збіжність процесу навчання графової нейронної мережі, що використовується в запропонованій інформаційній технології. Для цього було проаналізовано динаміку значення функції втрат та показника точності під час навчання моделі.

На рисунку 4.6 наведено зміну значення функції втрат у процесі навчання графової нейронної мережі. Як видно з графіка, функція втрат монотонно зменшується та стабілізується після певної кількості епох, що свідчить про збіжність процесу оптимізації та відсутність ознак перенавчання. Отримана динаміка підтверджує коректність обраної архітектури моделі та параметрів навчання.

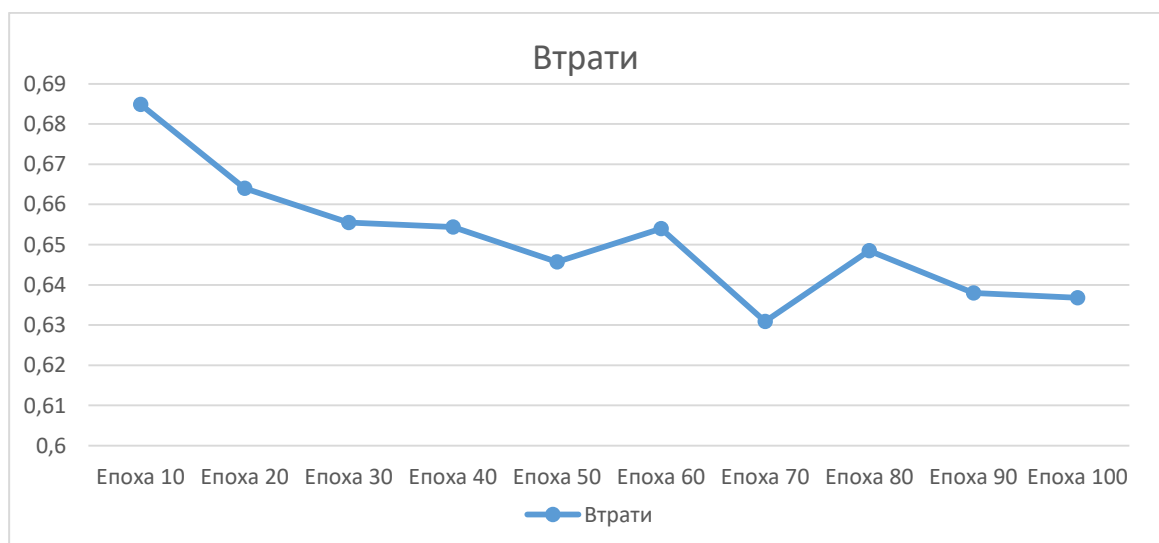


Рисунок 4.6 – Втрати під час навчання графової нейронної мережі

На рисунку 4.7 показано зміну показника точності класифікації вузлів DOM-графа під час навчання. Зростання точності та її стабілізація на фінальних етапах навчання підтверджують здатність графової нейронної мережі ефективно виділяти структурно значущі елементи DOM-дерева. Таким чином, результати, наведені на рисунках 4.6 та 4.7, підтверджують адекватність навченої моделі та доцільність її подальшого використання в експериментальному дослідженні.

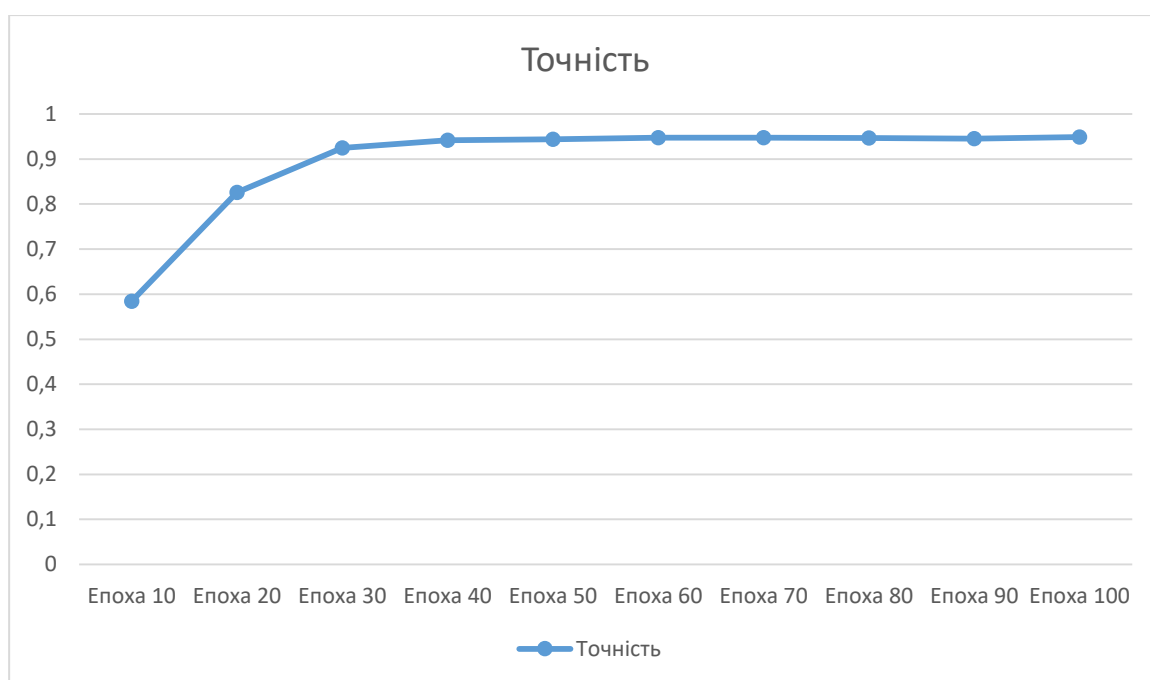


Рисунок 4.7 – Точність класифікації під час навчання графової нейронної мережі

На рисунку 4.8 наведено результати навчання графової нейронної мережі у вигляді логів зміни значень функції втрат та точності класифікації протягом епох. Зменшення значення функції втрат та зростання точності свідчать про збіжність процесу навчання та коректність обраної архітектури моделі. Стабілізація показників на пізніх епохах підтверджує відсутність ознак перенавчання та доцільність використання навченої моделі на етапі формування та оцінювання тестових селекторів.

```

Training GNN:
Epoch 0, Loss: 0.6849, Accuracy: 0.5840,
Epoch 10, Loss: 0.6640, Accuracy: 0.8257
Epoch 20, Loss: 0.6555, Accuracy: 0.9249
Epoch 30, Loss: 0.6544, Accuracy: 0.9415
Epoch 40, Loss: 0.6457, Accuracy: 0.9440
Epoch 50, Loss: 0.6540, Accuracy: 0.9478
Epoch 60, Loss: 0.6309, Accuracy: 0.9478
Epoch 70, Loss: 0.6485, Accuracy: 0.9466
Epoch 80, Loss: 0.6380, Accuracy: 0.9453
Epoch 90, Loss: 0.6368, Accuracy: 0.9491

```

Рисунок 4.8 – Динаміка функції втрат та точності під час навчання графової нейронної мережі

На рисунку 4.9 наведено приклад роботи запропонованої інформаційної технології на етапі застосування навченої графової нейронної мережі. Для вибраного DOM-елемента продемонстровано процес формування CSS-селектора, його перевірку на унікальність та обчислення показників якості. Отримані значення точності, повноти та F1-міри дорівнюють 1,0, що свідчить про коректну ідентифікацію цільового елемента. Додатково наведено значення структурної складності селектора з урахуванням вагових коефіцієнтів, що дозволяє оцінити компактність сформованого селектора. Наведений приклад ілюструє механізм інтеграції результатів інференсу GNN у процес формування та оцінювання тестових селекторів.

```

 Оцінка якості селектора:
Precision: 1.0000
Recall: 1.0000
F1-score: 1.0000

 Готовий селектор:
div[data-test-id="virtuoso-item-list"]:nth-child(1)
Складність (з вагами): 2.20
Знайдено елементів: 1
 Селектор унікальний!

```

Рисунок 4.9 - Приклад формування та оцінювання тестового селектора з використанням навченої GNN-моделі

Загальну логіку проведення експериментального дослідження, включаючи етапи підготовки DOM-графів, формування селекторів різними підходами та їх оцінювання за формалізованими метриками, узагальнено на рисунку 4.10. Наведена схема відображає послідовність дій, яка застосовувалася для всіх досліджуваних веб-ресурсів, що забезпечує коректність і відтворюваність експериментального порівняння.

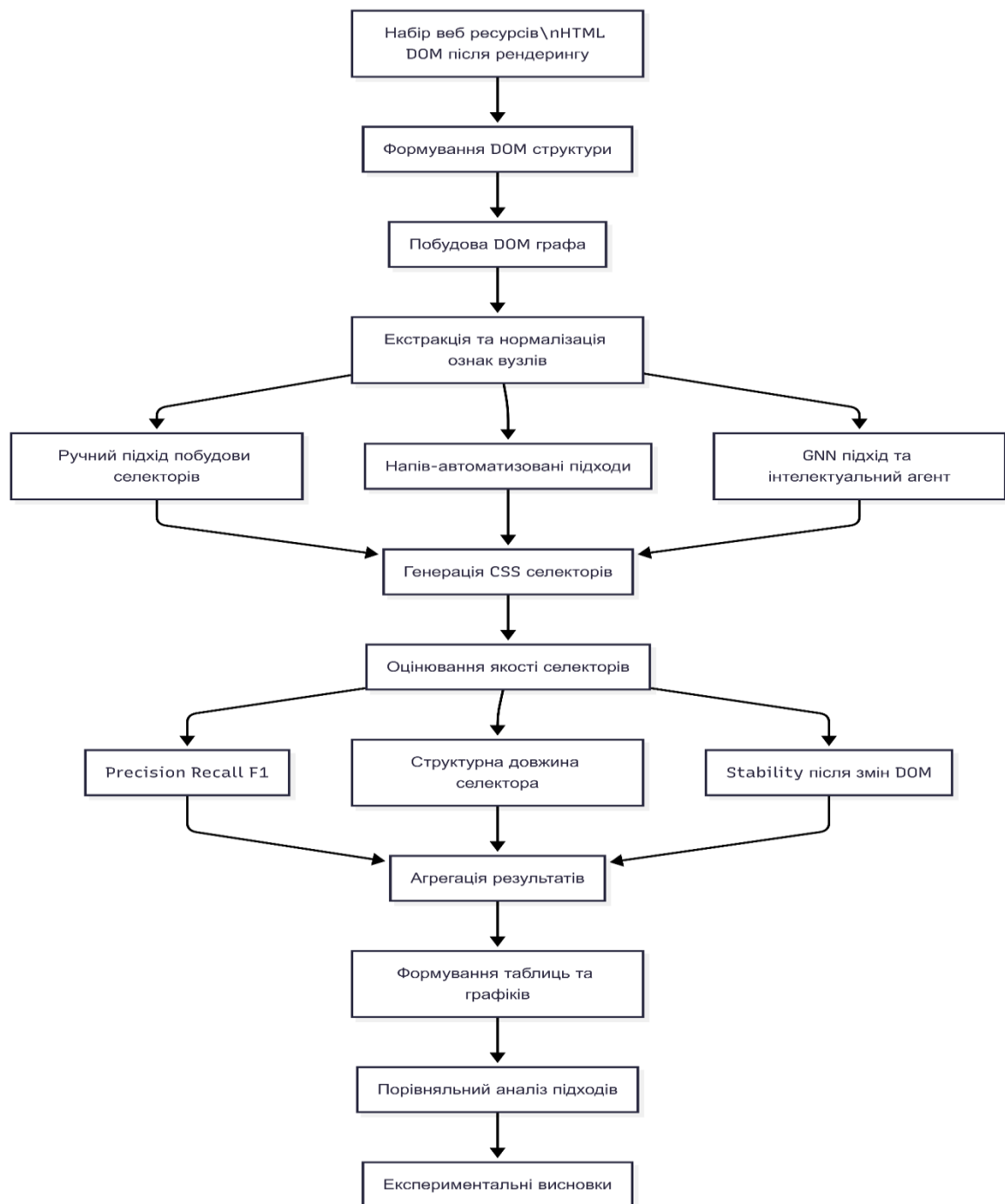


Рисунок 4.10 - Узагальнена схема експериментального порівняння підходів до формування тестових селекторів

Кількісне порівняння основних метрик якості селекторів, а саме точності, повноти, F1-міри та часу формування, наведено в таблиці 4.2. З отриманих даних видно, що GNN-підхід забезпечує найвищі значення точності (88,0%), повноти (84,7%) та F1-score (86,3%) у порівнянні з ручним і напіваавтоматизованим підходами. При цьому час формування селектора для запропонованого методу залишається прийнятним і суттєво меншим за ручний варіант, що свідчить про практичну придатність інтелектуального підходу.

Для наочного підтвердження результатів, наведених у таблиці 4.2, на рисунку 4.11 подано графічне порівняння показників точності, повноти та F1-score для трьох підходів формування селекторів. Візуальний аналіз графіка дозволяє чітко простежити перевагу GNN-підхід за всіма показниками коректності ідентифікації DOM-елементів, а також виявити обмеження напіваавтоматизованих інструментів, які демонструють нижчу повноту через надмірну специфічність селекторів.

Окремо проаналізовано часові характеристики формування селекторів, оскільки вони безпосередньо впливають на інтерактивність і масштабованість автоматизованого тестування. Порівняння часу формування тестових селекторів для всіх підходів наведено на рисунку 4.12. Отримані результати показують, що запропонований GNN-підхід забезпечує значне скорочення часу у порівнянні з ручним формуванням селекторів та залишається близьким за швидкодією до напіваавтоматизованих інструментів, водночас перевершуючи їх за якістю.

Таблиця 4.2 - Порівняння точності, повноти, F1-score, часу у трьох підходах формування селекторів

Підхід	Точність (%)	Повнота (%)	F1-score (%)	Час формування (мс)
Ручний	76,5	56,9	65	1200
Напів-автоматизований	78,5	63	69,8	45
GNN	88	84,7	86,3	60

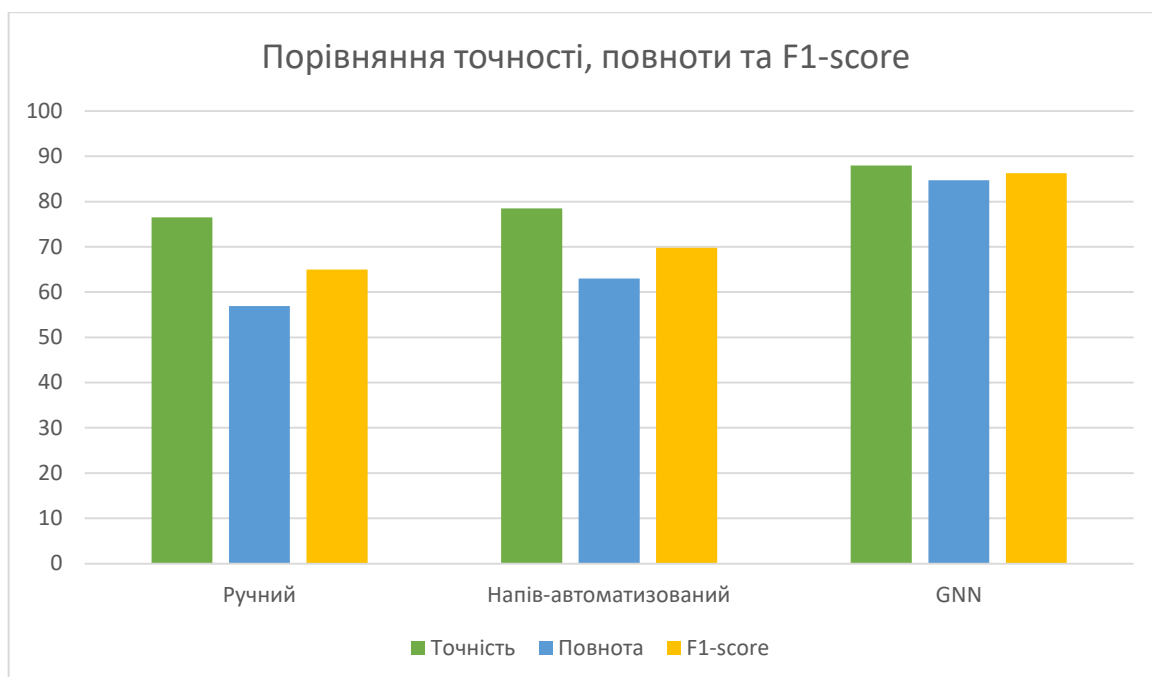


Рисунок 4.11 – Графічне порівняння показників точності (Precision), повноти (Recall) та F1-score для трьох підходів формування селекторів



Рисунок 4.12– Порівняння часу формування тестових селекторів

Аналіз метрик коректності ідентифікації показав, що використання GNN-підходу дозволяє досягти середнього значення точності 88,0%, що на 11,5% перевищує показник ручного підходу (76,5%) та на 9,5% – показник напівавтоматизованого підходу (78,5%). Повнота ідентифікації для GNN становить 84,7%, що на 27,8% більше у порівнянні з ручним підходом та на 21,7% більше у порівнянні з напівавтоматизованим методом.

Значення F1-міри для GNN-підходу становить 86,3%, що відповідає зростанню на 21,3% відносно ручного підходу (65,0%) та на 16,5% відносно напівавтоматизованого підходу (69,8%). Таким чином, використання графової нейронної мережі дозволяє підвищити збалансовану якість ідентифікації DOM-елементів більш ніж на 20% у порівнянні з традиційними методами.

Оцінка часових характеристик показала, що середній час формування селектора для GNN-підходу становить 60 мс, що є у 20 разів менше, ніж для ручного формування селекторів (1200 мс), та лише на 15 мс більше, ніж для напівавтоматизованих інструментів (45 мс). Отримані результати свідчать про прийнятну обчислювальну складність запропонованого підходу з огляду на суттєве підвищення якості селекції.

Наступним етапом експериментального аналізу стала оцінка стабільності селекторів після змін DOM-структури веб-ресурсів. Узагальнені результати стабільності та кількості помилок після внесення змін наведено в таблиці 4.3. Аналіз показує, що селектори, зформовані за допомогою графової нейронної мережі, характеризуються високою стійкістю до змін DOM і демонструють значно меншу кількість відмов у порівнянні з напівавтоматизованим підходом.

Аналіз стабільності селекторів після змін DOM-структури показав, що кількість помилок для GNN-підходу становить у середньому 5 помилок, що відповідає рівню ручного підходу та є у 4,4 рази меншим у порівнянні з напівавтоматизованим підходом, для якого зафіксовано 22 помилки. Це підтверджує здатність графової нейронної мережі формувати селектори, менш чутливі до змін структури веб-сторінок.

Таблиця 4.3 - Порівняння стабільності та кількості помилок після змін у трьох підходах формування селекторів

Підхід	Помилки після змін (шт)	Стабільність
Ручний	5	0.9
Напів-автоматизований	22	0.44
GNN	5	0.9

У таблиці 4.4 наведено результати порівняння структурної довжини тестових селекторів, сформованих трьома підходами: ручним, напівавтоматизованим та запропонованим GNN-підходу. Для кожного підходу подано середнє, мінімальне та максимальне значення довжини селектора, що дозволяє оцінити як типову складність селекторів, так і межі її варіації.

Аналіз наведених даних показує, що селектори, сформовані за допомогою графової нейронної мережі, характеризуються найменшим середнім значенням структурної довжини, яке становить 4,3, що є меншим у порівнянні з ручним підходом (4,5) та суттєво меншим у порівнянні з напівавтоматизованим підходом (5,3). Це свідчить про здатність запропонованого методу формувати компактні селектори без надмірної деталізації шляху до цільового DOM-елемента.

Мінімальне значення довжини селектора для всіх підходів є однаковим і становить 1,2, що пояснюється наявністю простих випадків ідентифікації елементів із використанням унікальних атрибутів. Водночас максимальна довжина селекторів для GNN-підходу є меншою (6,8) у порівнянні з ручним (7,0) та напівавтоматизованим (8,0) підходами, що вказує на обмеження росту складності селекторів навіть у складних DOM-структурах.

Оцінка структурної складності селекторів показала, що середня зважена довжина селекторів, зформованими GNN-підходом, становить 4,3, що на 4,4% менше у порівнянні з ручним підходом (4,5) та на 18,9% менше у порівнянні з напівавтоматизованим підходом (5,3). Максимальна довжина селекторів для

GNN також є меншою (6,8 проти 7,0 і 8,0 відповідно), що додатково знижує їхню крихкість до змін DOM.

Таблиця 4.4 - Порівняння структурної довжини тестових селекторів

Підхід	Середня довжина селектора (у.о.)	Мінімальне значення (у.о.)	Максимальне значення (у.о.)
Ручний	4,5	1,2	7
Напів-автоматизований	5,3	1,2	8
GNN	4,3	1,2	6,8

Оцінка структурної складності селекторів показала, що середня зважена довжина селекторів, зформованими GNN-підходом, становить 4,3, що на 4,4% менше у порівнянні з ручним підходом (4,5) та на 18,9% менше у порівнянні з напівавтоматизованим підходом (5,3). Максимальна довжина селекторів для GNN також є меншою (6,8 проти 7,0 і 8,0 відповідно), що додатково знижує їхню крихкість до змін DOM.

Для детальнішого аналізу узагальнюваності запропонованого підходу було розглянуто результати роботи GNN для кожного веб-ресурсу окремо. На рисунку 4.13 наведено значення точності, повноти, F1-міри та кількості помилок для всіх досліджуваних сайтів. Отримані дані свідчать про стабільну поведінку моделі на різних веб-ресурсах незалежно від складності та глибини DOM-структури, що підтверджує її здатність до узагальнення.



Рисунок 4.13 – Результати точності, повноти, F1 та помилок для кожного сайту

Поряд із аналізом коректності та стабільності селекторів було додатково досліджено їх структурну складність у межах запропонованого GNN-підходу. На рисунку 4.14 наведено результати аналізу довжини селекторів, зформовані графовою нейронною мережею, для кожного досліджуваного веб-ресурсу.

Подане графічне представлення дозволяє оцінити варіативність структурної довжини селекторів залежно від особливостей DOM-структури конкретного сайту.

Аналіз наведених даних свідчить, що довжина селекторів, сформованих GNN-підходом, залишається в обмеженому діапазоні значень для більшості веб-ресурсів, без різких зростань навіть для сайтів зі складною або глибоко вкладеною DOM-структурою. Це підтверджує здатність моделі адаптивно формувати компактні селектори з урахуванням структурних властивостей DOM-графа, не вдаючись до надмірної специфікації шляху до елемента.

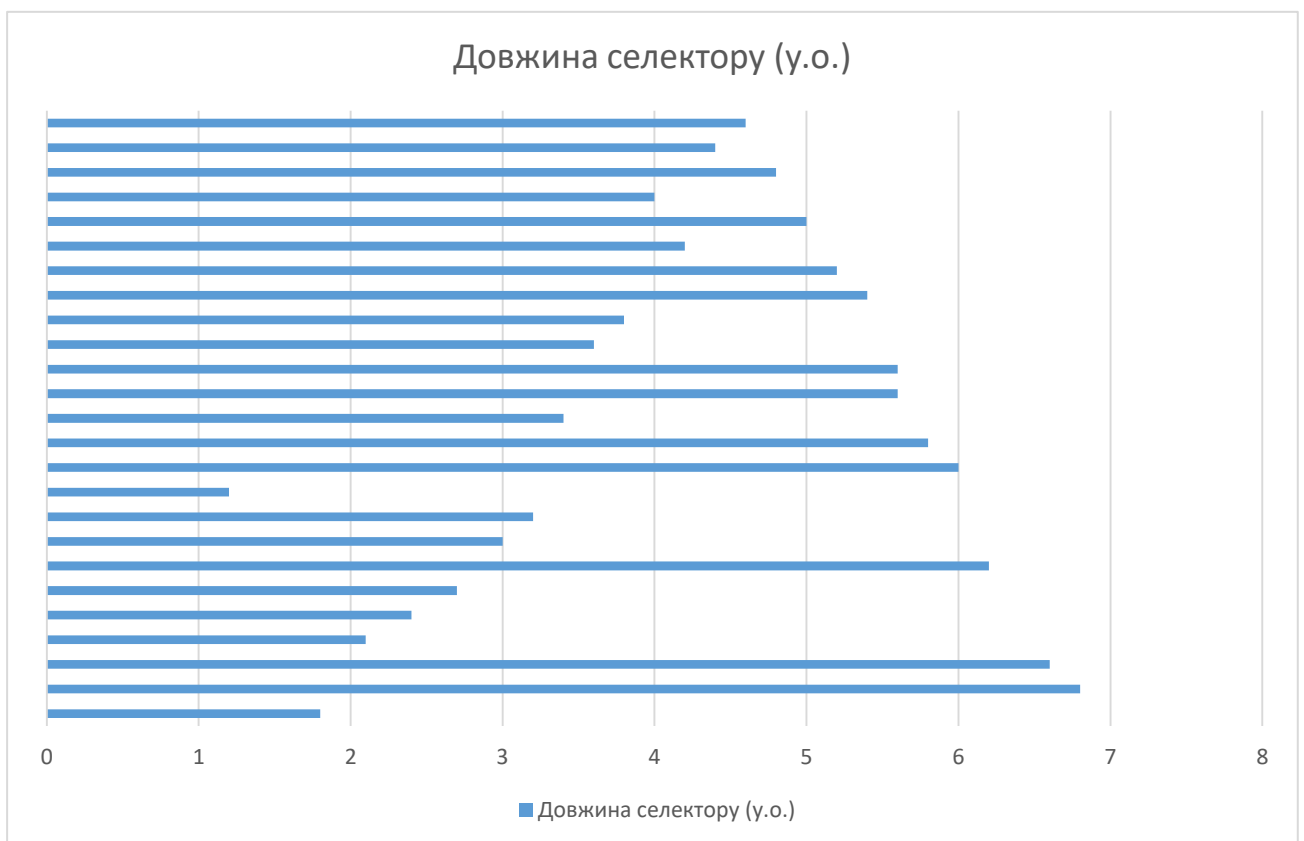


Рисунок 4.14 – Результати довжини селектору для кожного сайту

Для узагальненої кількісної оцінки ефективності підходів використано інтегральну метрику якості, яка поєднує F1-міру, показник стабільності та структурну довжину селектора. Результати обчислення інтегральної оцінки наведено в таблиці 4.5. Отримані значення підтверджують, що GNN-підхід забезпечує найвищий рівень сумарної якості, поєднуючи високу коректність

ідентифікації, стабільність до змін DOM та помірну структурну складність селекторів.

Таблиця 4.5 – Інтегральна оцінка якості селекторів

Підхід	F1-score (%)	Стабільність	Середня довжина	Інтегральна оцінка Q(S)
Ручний	65	0.9	4,5	0,593
Напів-автоматизований	69.8	0.44	5,3	0,47
GNN	86.3	0.9	4,3	0,659

Застосування інтегральної метрики якості, яка поєднує значення F1-міри, показник стабільності та структурну довжину селектора, дозволило виконати комплексне порівняння ефективності підходів. Значення інтегральної оцінки для GNN-підходу становить 0,659, що на 0,066 перевищує значення для ручного підходу (0,593) та на 0,189 перевищує відповідний показник напівавтоматизованого підходу (0,470). Отримані значення підтверджують, що запропонований метод забезпечує найкращий баланс між коректністю ідентифікації, стабільністю та структурною компактністю селекторів.

Таким чином, числові результати експериментального дослідження підтверджують, що використання графових нейронних мереж дозволяє одночасно підвищити F1-міру ідентифікації більш ніж на 20%, зменшити кількість помилок після змін DOM у понад чотири рази порівняно з напівавтоматизованими інструментами та забезпечити найвище значення інтегральної оцінки якості серед усіх розглянутих підходів.

4.8 Аналіз результатів впровадження

Ефективне впровадження розробленої інтелектуальної інформаційної технології автоматизованого тестування веб-орієнтованого програмного

забезпечення передбачає її адаптацію до реальних умов використання, зокрема до різних типів веб-ресурсів, динамічних DOM-структур та існуючих процесів тестування. Ключовим елементом впровадження є підготовка вхідних даних у вигляді DOM-структур, їх трансформація у графове представлення та формування ознак вузлів для подальшої обробки графовою нейронною мережею. Практична реалізація показала можливість використання технології як окремого інструменту для генерації селекторів або як допоміжного модуля у процесі розробки автоматизованих тестів. При цьому зменшуються витрати часу на створення та підтримку тестів, а також знижується кількість нестабільних (flaky) тестів.

Таким чином, прототип даної інформаційної технології було запропоновано до використання в двох комерційних компаніях Він Інтерактив та Екзип Текнолоджіх, що безпосередньо займаються розробкою web-орієнтованого програмного забезпечення у різних сферах та активно впроваджують автоматизацію тестування. У зв'язку із комерційною таємницею, результати використання не підлягають розповсюдженню у відкритих джерелах інформації. Але представники компанії позитивно охарактеризували вплив використання інформаційної технології, що відображено у документах щодо впровадження.

4.9 Висновки до розділу 4

У четвертому розділі дисертаційної роботи проведено експериментальне дослідження розробленої інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого ПЗ ІС на основі графових нейронних мереж. Метою експериментів було підтвердження ефективності запропонованого підходу до побудови стабільних та оптимізованих DOM-селекторів у порівнянні з традиційними методами.

У ході експериментальних досліджень сформовано набір тестових веб-ресурсів із різною структурною складністю DOM, включно з динамічними елементами та вкладеними ієрархіями. Для кожного ресурсу проведено серію

експериментів із побудови селекторів із використанням запропонованої GNN-орієнтованої інформаційної технології та базових підходів (ручних і евристичних методів).

Отримані результати показали, що застосування графових нейронних мереж дозволяє суттєво підвищити якість побудови селекторів за ключовими метриками. Зокрема, експериментально підтверджено зростання значень показників точності (Precision), повноти (Recall) та F1-міри для селекторів, зформованих із використанням GNN, у порівнянні з традиційними підходами. Це свідчить про більш точну та однозначну ідентифікацію DOM-елементів у складних веб-інтерфейсах.

Встановлено, що запропонований підхід забезпечує зменшення довжини селекторів при збереженні їх унікальності, що позитивно впливає на стабільність автоматизованих UI-тестів та знижує їх чутливість до змін DOM-структури. Експерименти з модифікацією DOM підтвердили, що селектори, побудовані із застосуванням GNN-моделі та інтелектуального агента, демонструють меншу кількість відмов і значно рідше призводять до появи flaky-тестів.

Порівняльний аналіз показав, що запропонована інформаційна технологія перевершує базові методи не лише за окремими показниками якості селекторів, а й за інтегральною оцінкою, яка враховує унікальність, точність, повноту, F1-міру та довжину селектора. Отримані результати є стабільними для різних типів веб-ресурсів і не залежать від конкретної реалізації DOM-структури.

Таким чином, експериментальне дослідження підтвердило доцільність і ефективність використання графових нейронних мереж у задачі процесу автоматизованого тестування веб-ресурсів. Запропонована інформаційна технологія забезпечує підвищення стабільності автоматизованих тестів, зменшення витрат часу на підтримку селекторів та створює передумови для подальшого розвитку інтелектуальних засобів автоматизованого тестування веб-додатків.

Результати дисертаційного дослідження впроваджені на підприємствах:

- ТОВ «ВІН ІНТЕРАКТИВ» - інтелектуальний агент та його взаємодія з GNN-модулем, метод автоматичної генерації DOM-селекторів, удосконалена архітектура інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого ПЗ.

- «ТОВ «Екзип Текнолоджіз» - моделі інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого ПЗ ІС, модуль побудови DOM-графа та екстракції ознак, моделі та підсистема оцінювання якості селектора.

Результати проведених досліджень опубліковані в [1, 5, 12-14, 38, 51, 52, 65, 66, 67].

ВИСНОВКИ

У дисертації розв'язано науково-прикладну задачу створення інтелектуальної інформаційної технології автоматизованого тестування web-орієнтованого ПЗ ІС шляхом підвищення стабільності та якості DOM-селекторів із використанням методів графового моделювання та графових нейронних мереж. Отримані результати мають теоретичне й практичне значення та відповідають поставленій меті дослідження.

1. Проведено комплексний аналіз сучасних підходів до автоматизованого тестування веб-ресурсів, методів побудови DOM-селекторів і проблем, пов'язаних із їх нестабільністю та підтримкою. Установлено, що традиційні ручні, автоматизовані та напівавтоматизовані підходи не враховують глобальну структуру DOM і є чутливими до динамічних змін інтерфейсу, що призводить до появи flaky-тестів і збільшення витрат на супровід автоматизованих сценаріїв.
2. Обґрунтовано доцільність формалізації HTML-документа у вигляді графової структури, де DOM-елементи представлені у вигляді вузлів, а їхні ієрархічні та контекстні зв'язки – у вигляді ребер. Запропоновано набір семантичних, атрибутивних і структурних ознак DOM-вузлів, що дозволяє коректно відобразити контекст елемента та використати його для подальшої обробки методами графового машинного навчання.
3. Виконано аналіз і порівняння основних архітектур графових нейронних мереж (GCN, GAT, GraphSAGE) у контексті задачі покращення DOM-селекторів. Обґрунтовано вибір архітектури GAT як найбільш придатної для врахування різної важливості сусідніх елементів у DOM-дереві та формування контекстно-залежних представлень вузлів.
4. Запропоновано модель формалізації якості DOM-селекторів на основі системи кількісних метрик, що включає унікальність, точність (Precision), повноту (Recall), F1-міру та зважену довжину селектора. Розроблено

інтегральну функцію оцінювання, яка дозволяє об'єктивно порівнювати різні підходи до побудови селекторів.

5. Розроблено інтелектуальну інформаційну технологію побудови адаптивних DOM-селекторів, яка поєднує графове подання структури веб-сторінки, графову нейронну мережу та інтелектуальний агент прийняття рішень. Запропонований підхід забезпечує автоматичне формування селекторів з урахуванням контексту DOM-елементів і дозволяє адаптуватися до змін структури веб-ресурсів.
6. Реалізовано програмні модулі запропонованої інформаційної технології, що включають побудову DOM-графа, екстракцію ознак вузлів, навчання GNN-моделі, алгоритм формування селекторів і підсистему оцінювання їх якості. Архітектура рішення є модульною та може бути інтегрована в існуючі процеси автоматизованого тестування веб-додатків.
7. Проведено експериментальне дослідження ефективності запропонованої інформаційної технології на реальних веб-ресурсах із різною структурною складністю DOM. Експериментально підтверджено, що застосування графових нейронних мереж забезпечує підвищення точності та стабільності селекторів, зменшення їх довжини та зниження кількості відмов автоматизованих тестів у порівнянні з традиційними підходами.
8. Показано, що запропонований підхід дозволяє суттєво скоротити витрати часу на підтримку автоматизованих UI-тестів і зменшити ймовірність появи flaky-тестів при зміні DOM-структури. Отримані результати є стабільними для різних типів веб-ресурсів і підтверджують практичну доцільність використання запропонованої інформаційної технології в реальних умовах розробки ПЗ ІС.
9. Отримані в дисертаційній роботі результати узгоджуються з поставленою метою та завданнями дослідження, а також підтверджують наукову новизну. Розроблена інформаційна технологія може бути використана як основа для створення інтелектуальних засобів автоматизованого тестування веб-

ресурсів, зокрема у вигляді окремого програмного інструменту або розширення для браузера.

10. Перспективними напрямками подальших досліджень є розширення запропонованого підходу для підтримки сучасних SPA-фреймворків, удосконалення механізмів навчання інтелектуального агента, а також інтеграція запропонованої інформаційної технології у CI/CD-процеси IT-проектів з метою повної автоматизації підтримки автоматизованих тестів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Морозов О.С., Яровий А.А., Козловський А.В. "Аналіз предметної області автоматизованого тестування WEB-ресурсів в контексті оптимізації процесів", Молодь в науці: дослідження, проблеми, перспективи (МН-2023), 2023. Тип доступу: <https://d.conf.vntu.edu.ua/index.php/mn/mn2023/paper/viewFile/18706/15503>
2. Dou.ua. Статистика українських тестувальників – 2023. [Online] Available: <https://dou.ua/lenta/articles/salary-report-qa-summer-2023/>
3. Dou.ua. Статистика українських тестувальників – 2024. [Online] Available: <https://dou.ua/lenta/articles/salary-report-qa-summer-2024/>
4. Dou.ua. Статистика українських тестувальників – 2025. [Online] Available: <https://dou.ua/lenta/articles/salary-report-qa-summer-2025/>
5. Морозов О.С., Яровий А.А., “Аналіз особливостей процесу автоматизованого тестування веб-орієнтованих програмних продуктів”, Наука і техніка сьогодні, № 3(44), 2025. С. 1330-1339, DOI: [https://doi.org/10.52058/2786-6025-2025-3\(44\)-1330-1339](https://doi.org/10.52058/2786-6025-2025-3(44)-1330-1339) (ISSN 2786-6025)
6. Sharma M., Angmo R. Web based automation testing and tools //International Journal of Computer Science and Information Technologies. – 2014. – Т. 5. – №. 1. – С. 908-912. ISSN : 0975-9646.
7. Gundecha U., Cocchiaro C. Learn Selenium: build data-driven test frameworks for mobile and web applications with Selenium Web Driver 3. – Packt Publishing Ltd, 2019.
8. Holmes A., Kellogg M. Automating functional tests using selenium //AGILE 2006 (AGILE'06). – IEEE, 2006. – С. 6 pp.-275. DOI: [10.1109/AGILE.2006.19](https://doi.org/10.1109/AGILE.2006.19)
9. M. T. Taky, Automated Testing With Cypress. PhD thesis, 2021.
10. Utomo A., Wijaya G., Setiawan Y. Implementation of crowdfunding web application using AWS Amplify architecture with end-to-end testing using Playwright //Indonesian Journal of Multidisciplinary Science. – 2023. – Т. 2. – №. 11. – С. 3888-3904. DOI: <https://doi.org/10.55324/ijoms.v2i11.620>

11. O. Kernytsky, A. Kernytsky and V. Teslyuk, "The Synthesis Method for Specifications and Requirements in the Process of IT Project Reengineering," 2023 IEEE 18th International Conference on Computer Science and Information Technologies (CSIT), Lviv, Ukraine, 2023, pp. 1-4, DOI: [10.1109/CSIT61576.2023.10324175](https://doi.org/10.1109/CSIT61576.2023.10324175).
12. Морозов О.С., Яровий А.А., "Аналіз підходів та експериментальні дослідження процесу оптимізації автоматизованого тестування програмного забезпечення із застосуванням паралелізму тестів". Наука і техніка сьогодні, №9(50), 2025. С. 1361-1374, DOI: [https://doi.org/10.52058/2786-6025-2025-9\(50\)-1361-1374](https://doi.org/10.52058/2786-6025-2025-9(50)-1361-1374) (ISSN 2786-6025)
13. Морозов О.С., Яровий А.А., Козловський А.В., "Специфічні аспекти процесу автоматизованого тестування WEB-орієнтованого програмного забезпечення із застосуванням паралельності тестів на основі TestNG", Молодь нації: дослідження, проблеми, перспективи (МН-2024), 2024. Тип доступу: <https://jpvrrnd.conf.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/20609/17035>
14. Морозов О.С., Яровий А.А., Козловський А.В., " Специфічні аспекти процесу автоматизованого тестування WEB-орієнтованого програмного забезпечення із застосуванням паралельності тестів на основі Selenium Grid", ЛІІІ Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024), 2024. Тип доступу: <https://jpvrrnd.conf.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20633/17062>
15. Parry O. et al. A survey of flaky tests //ACM Transactions on Software Engineering and Methodology (TOSEM). – 2021. – Т. 31. – №. 1. – С. 1-74. DOI: <https://doi.org/10.1145/3476105>
16. Selectorgadget.com. SelectorGadget: point and click CSS selectors. [Online] Available: <https://selectorgadget.com/>

17. Dogucu, M., Çetinkaya-Rundel, M. (2021). Web scraping in the statistics and data science curriculum: Challenges and opportunities. *Journal of Statistics and Data Science Education*, 29(sup1), S112-S122.
18. Selectorshub.com. Selectorshub: Booster Tools for Testers [Online]. Available: <https://selectorshub.com/>
19. N. Uppal, & V. Chopra, "Design and implementation in selenium ide with web driver," *International Journal of Computer Application*, vol.46, no.12, pp.8-11, May 2012.
20. Google.com. Open Chrome DevTools. [Online] Available: <https://developer.chrome.com/docs/devtools/open>
21. Medium.com. Building Trust One Test at a Time. [Online] Available: <https://medium.com/%40seedyjahateh31/week-4-blog-post-building-trust-one-test-at-a-time-b712e6ceaf2a>
22. Stackoverflow.com. What is the difference between a CSS and XPath selector? And which is better with respect to performance for cross-browser testing? [Online]. Available <https://stackoverflow.com/questions/16788310/what-is-the-difference-between-a-css-and-xpath-selector-and-which-is-better-wit>
23. Google.com. Flaky Tests at Google and How We Mitigate Them [Online] Available: <https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html>
24. Y. Pei, J. Sohn, M. Papadakis, An Empirical Study of Web Flaky Tests: Understanding and Unveiling DOM Event Interaction Challenges, in: *Proceedings of the Conference on Software Testing, Verification and Validation (ICST)*, (2025) 92-102. DOI: 10.1109/ICST62969.2025.10989030.
25. Mellati, P. ., Galuh Wilujeng Saraswati, Wildan Mahmud, Lutfina, E. ., & Resha Meiranadi Caturkusuma. (2026). Optimization of Web-Based Printing Order Management System Using Redis Database for Efficient Data Handling. *Sinkron : Jurnal Dan Penelitian Teknik Informatika*, 10(1), 220-231. DOI: <https://doi.org/10.33395/sinkron.v10i1.15502>

26. Wood L. et al. Document object model (dom) level 1 specification //W3C recommendation. – 1998. – T. 1. – C. 1-212.
27. Jiang B. et al. Semi-supervised learning with graph learning-convolutional networks //Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. – 2019. – C. 11313-11320.
28. Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. AI Open 1 (2020), 57–81. DOI: <https://doi.org/10.1016/j.aiopen.2021.01.001>
29. Medium.com. GNN vs GCN vs GAN (Graph networks). [Online]. Available: <https://vtiya.medium.com/gnn-vs-gcn-vs-gan-grap-cf7a790848ad>
30. Veličković, P. et al. Graph attention networks. In International Conference on Learning Representations Vol. 6 (2018). DOI: <https://doi.org/10.48550/arXiv.1710.10903>
31. Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020b. Gpt-gnn: Generative pre-training of graph neural networks. In KDD. 1857--1867. DOI: <https://doi.org/10.1145/3394486.340323>
32. Brody, S., Alon, U. & Yahav, E. How attentive are graph attention networks? International Conference on Learning Representations (2021). DOI: <https://doi.org/10.48550/arXiv.2105.14491>
33. Коротенко, С. (2025). Порівняння GNN-архітектур для прогнозу активності у бізнес-процесах: Logs-only та BPMN-підходи. Information Technology: Computer Science, Software Engineering and Cyber Security, 36–47, doi: <https://doi.org/10.32782/IT/2025-2-5>
34. Vrahatis, A.G.; Lazaros, K.; Kotsiantis, S. Graph Attention Networks: A Comprehensive Review of Methods and Applications. Future Internet 2024, 16, 318. DOI: <https://doi.org/10.3390/fi16090318>
35. Hamilton W, Ying Z, Leskovec J. Inductive representation learning on large graphs. Advances in neural information processing systems. 2017; 30. DOI: <https://doi.org/10.48550/arXiv.1706.02216>

36. Medium.com. Graph Neural Networks (GraphSAGE). [Online]. Available: <https://medium.com/@gc.iitr/graph-neural-networks-graphsage-5987af9face7>
37. Medium.com. What Is One-Hot Encoding in Machine Learning? A Comprehensive Guide with Examples. [Online]. Available: https://medium.com/@sanjay_dutta/what-is-one-hot-encoding-in-machine-learning-a-comprehensive-guide-with-examples-090f037a6bbe
38. Морозов О.С., Яровий А.А., “Інтелектуальна модель генерації адаптивних веб-селекторів на основі GNN”. Опт-ел. інф-енерг. техн., вип. 50, вип. 2, с. 142–149, Січ 2026, DOI: <https://doi.org/10.31649/1681-7893-2025-50-2-142-149> (ISSN 2311-2662)
39. B. Yeoh and H. Wang, “Grown+ up: A “graph representation of a webpage ” network utilizing pre-training,” in Proceedings of the 31st ACM International Conference on Information & Knowledge Management, pp. 2372–2382, 2022. DOI: <https://doi.org/10.1145/3511808.355734>
40. Prykhodko S., Makarova L., Pukhalevych A. Statistical Analysis of the Three-Dimensional Data of Software Metrics RFC, CBO, and WMC that are not Normally Distributed. Proceedings of International Conference on Applied Innovation in IT. 2025. Vol. 13, issue 1. P. 127–132. DOI: <https://doi.org/10.25673/119254>
41. Yuanhong Lan, Yifei Lu, Zhong Li, Minxue Pan, Wenhua Yang, Tian Zhang, and Xuandong Li. 2024. Deeply reinforcing android GUI testing with deep reinforcement learning. In Proceedings of the 46th IEEE/ACM International Conference on Software Engineering. 1–13. DOI: <https://doi.org/10.1145/3597503.3623344>
42. A Srinivasan, R Raja, JO Glass, MM Hudson, ND Sabin, KR Krull, WE Reddick, Graph Neural Network Learning on the Pediatric Structural Connectome, Tomography 2025, 11(2), 14; <https://doi.org/10.3390/tomography11020014>

43. Ali Hassaan Mughal. 2025. An Autonomous RL Agent Methodology for Dynamic Web UI Testing in a BDD Framework. DOI: <https://doi.org/10.48550/arXiv.2503.08464>
44. Jelica Cincović, Marija Punt (2020). Comparison: Angular vs. React vs. Vue. Which framework is the best choice? University of Belgrade, School of Electrical Engineering, Belgrade, Serbia. Eventiotic.
45. Rönnerberg, J. et al. (2022) The cognitive hearing science perspective on perceiving, understanding, and remembering language: the ELU model. Front. Psychol. 13, 967260. DOI: <https://doi.org/10.3389/fpsyg.2022.967260>
46. Songgaojun Deng, Shusen Wang, Huzefa Rangwala, Lijing Wang, and Yue Ning. 2020. Cola-GNN: Cross-location Attention based Graph Neural Networks for Long-term ILI Prediction. In Proceedings of the 29th ACM International Conference on Information & Knowledge Management. ACM, 245--254. DOI: <https://doi.org/10.1145/3340531.3411975>
47. Fey, M. & Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. (2019). PyTorch Geometric is the most widely used library to develop GNNs. DOI: <https://doi.org/10.48550/arXiv.1903.02428>
48. Kajla, N.I., Missen, M.M.S., Luqman, M.M., Coustaty, M., Mehmood, A., Choi, G.S.: Additive angular margin loss in deep graph neural network classifier for learning graph edit distance. IEEE Access 8, 201752–201761 (2020). DOI: [10.1109/ACCESS.2020.3035886](https://doi.org/10.1109/ACCESS.2020.3035886)
49. Adam K. D. P. B. J. et al. A method for stochastic optimization //arXiv preprint arXiv:1412.6980. – 2014. – Т. 1412. – №. 6. DOI: <https://doi.org/10.48550/arXiv.1412.6980>
50. Морозов О.С., Яровий А.А., Петришин С.І., " Гібридні методи для вибору тестових селекторів у динамічних веб додатках", Молодь науці: дослідження, проблеми, перспективи (МН-2025), 2025. Тип доступу: <https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/23706/19578>

51. Морозов О.С., Яровий А.А., Козловський А.В., "Застосування машинного навчання для аналізу змін у веб-ресурса", LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2025), 2025. Тип доступу: <https://jpvrnd.conf.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23685/19566>
52. Yacouby R., Axman D. Probabilistic extension of precision, recall, and f1 score for more thorough evaluation of classification models //Proceedings of the first workshop on evaluation and comparison of NLP systems. – 2020. – С. 79-91. DOI: [10.18653/v1/2020.eval4nlp-1.9](https://doi.org/10.18653/v1/2020.eval4nlp-1.9)
53. Kaelbling L. P., Littman M. L., Moore A. W. Reinforcement learning: A survey //Journal of artificial intelligence research. – 1996. – Т. 4. – С. 237-285. DOI: <https://doi.org/10.1613/jair.301>
54. Yu C. et al. The surprising effectiveness of ppo in cooperative multi-agent games //Advances in neural information processing systems. – 2022. – Т. 35. – С. 24611-24624. DOI:
55. Hou Y. et al. A novel DDPG method with prioritized experience replay //2017 IEEE international conference on systems, man, and cybernetics (SMC). – IEEE, 2017. – С. 316-321. DOI: [10.1109/SMC.2017.8122622](https://doi.org/10.1109/SMC.2017.8122622)
56. Babaeizadeh M. et al. GA3C: GPU-based A3C for deep reinforcement learning //CoRR abs/1611.06256. – 2016.
57. Liu J., Ong G. P., Chen X. GraphSAGE-based traffic speed forecasting for segment network with sparse data //IEEE Transactions on Intelligent Transportation Systems. – 2020. – Т. 23. – №. 3. – С. 1755-1766. DOI: [10.1109/TITS.2020.3026025](https://doi.org/10.1109/TITS.2020.3026025)
58. Xu K. et al. How powerful are graph neural networks? //arXiv preprint arXiv:1810.00826. – 2018. DOI: <https://doi.org/10.48550/arXiv.1810.00826>
59. Kim B. H., Ye J. C. Understanding graph isomorphism network for rs-fMRI functional connectivity analysis //Frontiers in neuroscience. – 2020. – Т. 14. – С. 630. DOI: <https://doi.org/10.3389/fnins.2020.00630>

60. Jaiswal A. et al. A survey on contrastive self-supervised learning //Technologies. – 2020. – Т. 9. – №. 1. – С. 2. DOI: <https://doi.org/10.3390/technologies9010002>
61. Mishra P. et al. Node masking: Making graph neural networks generalize and scale better. arXiv 2020 //arXiv preprint arXiv:2001.07524. DOI: <https://doi.org/10.48550/arXiv.2001.07524>
62. You Y. et al. Graph contrastive learning with augmentations //Advances in neural information processing systems. – 2020. – Т. 33. – С. 5812-5823.
63. Beust C., Suleiman H. Next generation Java testing: TestNG and advanced concepts. – Pearson Education, 2007.
64. Khder, Moaiad. 2021. Web scraping or web crawling: State of art, techniques, approaches and application. International Journal of Advances in Soft Computing and its Applications 13(3):145–168. <https://doi.org/10.15849/ijasca.211128.11>.
65. Морозов О.С., Яровий А.А., “Експериментальні дослідження підходів до генерації тестових селекторів із застосуванням GNN у процесі автоматизованого тестування веб-додатків”. Український журнал інформаційних технологій, Том 7, №2, 2025. С. 44-49, DOI: <https://doi.org/10.23939/ujit2025.02.044> (ISSN 2707-1898)
66. Morozov Oleksandr, Yarovyi Andrii., METHODS OF EVALUATING SELECTOR LENGTH FOR AUTOMATED TESTING // Cambridge Science and Education Conference : proceedings of the International scientific and practical conference (February 23-25, 2026). – Cambridge, United Kingdom : naukainfo.com, 2026. - Pp. 108-113. - DOI: <https://doi.org/10.64828/conf-100-2026>
67. Морозов О.С., Яровий А.А., " Інтегральна метрика оцінювання якості тестових селекторів у динамічних web-додатках", LV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2026), 2026. Тип доступу: <https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2026/paper/view/28766/23649>

ДОДАТКИ

Додаток А Документи щодо впровадження результатів роботи

“ЗАТВЕРДЖУЮ”
Директор Товариства
з обмеженою відповідальністю
“ВІН ІНТЕРАКТИВ”
Томашпольський О.С.

«06» квітня 2026 р.

АКТ про впровадження результатів дисертаційної роботи Морозова Олександра Сергійовича

Комісія у складі: голови комісії директора Томашпольського Олександра Самойловича, керівника проєктів Комісаренко Дмитра Юрійовича, технічного керівника Іванова Павла Володимировича,

склали цей акт про те, що результати наукових досліджень дисертації Морозова Олександра Сергійовича впроваджено в контексті практично-прикладної реалізації розробленої інтелектуальної інформаційної технології автоматизованого тестування WEB-орієнтованого програмного забезпечення (ПЗ).

Комісія підтверджує, що використані такі наукові та практичні результати:

- 1) інтелектуальний агент та його взаємодія з GNN-модулем;
- 2) метод автоматичної генерації DOM-селекторів;
- 3) удосконалена архітектура інтелектуальної інформаційної технології автоматизованого тестування WEB-орієнтованого ПЗ.

Ефект від впровадження полягає у пришвидшенні виправлення флаку-тестів за рахунок додаткових можливостей для ручних тестувальників та інженерів з автоматизації у підтримці тестів, полегшуючи процес оновлення автотестів без потреби вручну аналізувати складну структуру DOM.

Голова комісії:

Члени комісії:



Томашпольський О.С.

Комісаренко Д.Ю.

Іванов П.В.

"ЗАТВЕРДЖУЮ"

Директор Товариства
з обмеженою відповідальністю
"Екзіп Текнолоджіз"
Томашпольський О.С.

"05" березня 2026 р.

АКТ

про впровадження результатів
дисертаційної роботи Морозова Олександра Сергійовича

Комісія у складі: голови комісії директора Томашпольського Олександра Самойловича, технічного директора Денисенка Юрія Станіславовича, головного керівника проектів Муращенка Олександра Геннадійовича,

склали цей акт про те, що результати дисертаційної роботи Морозова Олександра Сергійовича отримали практичну реалізацію у вигляді впровадження створеної в роботі інтелектуальної інформаційної технології автоматизованого тестування WEB-орієнтованого програмного забезпечення.

Комісія підтверджує, що використані наступні результати дисертаційної роботи:

1. Моделі інтелектуальної інформаційної технології автоматизованого тестування WEB-орієнтованого програмного забезпечення.
2. Модуль побудови DOM-графа та екстракції ознак.
3. Моделі та підсистема оцінювання якості селектора.

Ефект від впровадження полягає у забезпеченні підвищення стабільності автоматизованих UI-тестів, а також скороченні витрат часу на підтримку та створення нових автоматизованих тестів.

Голова комісії

Члени комісії



Томашпольський О.С.

Денисенко Ю.С.

Муращенко О.Г.



Проректор з науково-педагогічної роботи
та організації освітнього процесу ВНТУ

Олександр ПЕТРОВ

«10» 04 2026 р.

АКТ

про впровадження результатів в рамках виконання
дисертаційних досліджень

МОРОЗОВА ОЛЕКСАНДРА СЕРГІЙОВИЧА

за темою: «Інтелектуальна інформаційна технологія автоматизованого
тестування WEB-орієнтованого програмного забезпечення»,
яка подана на здобуття ступеня доктора філософії
з галузі знань 12 «Інформаційні технології»
за спеціальністю 122 «Комп'ютерні науки»

Комісія Вінницького національного технічного університету у складі: голова комісії – декан факультету інтелектуальних інформаційних технологій та автоматизації (ФІТА), к.т.н., доц. Севастьянов В.М. та члени комісії – заступник декана з навчально-методичної роботи ФІТА, ст. викладач кафедри системного аналізу та інформаційних технологій Присяжнюк В.В., заступник завідувача кафедри комп'ютерних наук ФІТА, д.т.н., проф. Іванчук Я.В. склали цей акт про те, що на базі кафедри комп'ютерних наук Вінницького національного технічного університету впроваджено результати дисертаційної роботи Морозова Олександра Сергійовича у освітньому процесі, зокрема, впроваджено методи та засоби побудови та оптимізації DOM-селекторів у процесі автоматизованого тестування WEB-орієнтованого програмного забезпечення, методику оцінювання якості селекторів на основі інтегральних метрик, що враховують точність, повноту, F1-міру та довжину селектора, а також підходи до зменшення кількості нестабільних (flaky) тестів у процесі автоматизованого тестування в межах освітнього компонента «Тестування програмного забезпечення» освітньо-професійної програми «Комп'ютерні науки».

Зазначені розробки мають важливе значення для підвищення ефективності процесу автоматизованого тестування WEB-додатків та забезпечення стабільності виконання тестів в умовах динамічних змін DOM-структури.

Отримані результати сприяли розширенню функціональних можливостей інформаційної технології автоматизованого тестування, зокрема за рахунок використання графових моделей та методів машинного навчання для аналізу структури WEB-ресурсів і побудови адаптивних селекторів.

Голова комісії:



Володимир СЕВАСТЬЯНОВ

Члени комісії:



Василь ПРИСЯЖНІУК



Ярослав ІВАНЧУК


```

        text=current_element.string or "")

    if parent_node_id is not None:
        dom_graph.add_edge(parent_node_id, node_identifier)

    for child_element in current_element.children:
        if hasattr(child_element, 'name') and child_element.name is not None:
            elements_processing_queue.append((child_element, node_identifier))

    return dom_graph

dom_network_graph = build_dom_graph_iteratively(parsed_html_soup)

all_html_tags = [dom_network_graph.nodes[node]['tag'] for node in
dom_network_graph.nodes()]
tag_onehot_encoder = OneHotEncoder(sparse_output=False)
encoded_tag_features = tag_onehot_encoder.fit_transform([[tag] for tag in
all_html_tags])

def extract_node_features_from_graph(network_graph):
    extracted_features_list = []

    for node_id in network_graph.nodes():
        current_node_data = network_graph.nodes[node_id]

        # One-hot encoding для тегу
        tag_onehot_vector =
tag_onehot_encoder.transform([[current_node_data['tag']]])[0]

        element_attributes = current_node_data.get('attrs', {})

        try:
            node_depth_in_dom = nx.shortest_path_length(network_graph, source=0,
target=node_id)
        except nx.NetworkXNoPath:
            node_depth_in_dom = 0

        children_count = len(list(network_graph.successors(node_id)))

        has_class_attribute = 1.0 if 'class' in element_attributes else 0.0
        has_id_attribute = 1.0 if 'id' in element_attributes else 0.0
        has_data_testid_attribute = 1.0 if 'data-test-id' in element_attributes else 0.0

        combined_features = (list(tag_onehot_vector) +
            [node_depth_in_dom, has_class_attribute, has_id_attribute,

```

```

        has_data_testid_attribute, children_count])
    extracted_features_list.append(combined_features)

    feature_scaler = MinMaxScaler()
    normalized_features = feature_scaler.fit_transform(extracted_features_list)
    return torch.tensor(normalized_features, dtype=torch.float)

cached_features_file_path = "features.pt"

def load_cached_features_or_generate_new(graph, features_cache_file):
    total_nodes_count = len(graph.nodes())

    if os.path.exists(features_cache_file):
        cached_node_features = torch.load(features_cache_file)
        if cached_node_features.shape[0] != total_nodes_count:
            print(f"[INFO] Розмір features.pt ({cached_node_features.shape[0]}) не
відповідає кількості вузлів ({total_nodes_count}). Видаляємо файл кешу.")
            os.remove(features_cache_file)
            generated_node_features = extract_node_features_from_graph(graph)
            torch.save(generated_node_features, features_cache_file)
        else:
            print("[INFO] Завантажено features.pt з кешу.")
            generated_node_features = cached_node_features
    else:
        generated_node_features = extract_node_features_from_graph(graph)
        torch.save(generated_node_features, features_cache_file)

    return generated_node_features

node_features_tensor =
load_cached_features_or_generate_new(dom_network_graph,
cached_features_file_path)

graph_edge_connections =
torch.tensor(list(dom_network_graph.edges())).t().contiguous()
graph_data_structure = Data(x=node_features_tensor,
edge_index=graph_edge_connections)

class GraphNeuralNetworkClassifier(nn.Module):
    def __init__(self, input_feature_channels, hidden_layer_channels,
output_class_channels):
        super(GraphNeuralNetworkClassifier, self).__init__()
        self.first_gat_convolution = GATConv(input_feature_channels,
hidden_layer_channels)
        self.dropout_layer = nn.Dropout(p=0.3)

```

```

        self.second_gat_convolution = GATConv(hidden_layer_channels,
        output_class_channels)

    def forward(self, input_graph_data):
        node_features, edge_connections = input_graph_data.x,
        input_graph_data.edge_index

        processed_features = self.first_gat_convolution(node_features,
        edge_connections)
        processed_features = self.dropout_layer(F.elu(processed_features))

        final_output = self.second_gat_convolution(processed_features,
        edge_connections)
        return final_output

node_classification_labels = torch.zeros(len(dom_network_graph.nodes()),
dtype=torch.long)
for label_index in range(0, len(node_classification_labels), 20):
    node_classification_labels[label_index] = 1

class_importance_weights = torch.tensor([1.0, 10.0])
gnn_classification_model = GraphNeuralNetworkClassifier(
    input_feature_channels=node_features_tensor.shape[1],
    hidden_layer_channels=32,
    output_class_channels=2
)

loss_calculation_function =
nn.CrossEntropyLoss(weight=class_importance_weights)
model_optimizer = optim.Adam(gnn_classification_model.parameters(), lr=0.0005)

print("\nTraining GNN:")
training_start_time = time.time()

model_training_logs = []

for training_epoch in range(100):
    gnn_classification_model.train()
    model_optimizer.zero_grad()

    model_predictions = gnn_classification_model(graph_data_structure)
    training_loss = loss_calculation_function(model_predictions,
    node_classification_labels)

    training_loss.backward()

```

```

model_optimizer.step()

if training_epoch % 10 == 0:
    predicted_classes = model_predictions.argmax(dim=1)
    classification_accuracy = (predicted_classes ==
node_classification_labels).sum().item() / len(node_classification_labels)

    predictions_numpy = predicted_classes.cpu().numpy()
    true_labels_numpy = node_classification_labels.cpu().numpy()

    calculated_f1_score = f1_score(true_labels_numpy, predictions_numpy,
average='binary', zero_division=0)
    calculated_precision = precision_score(true_labels_numpy, predictions_numpy,
average='binary', zero_division=0)
    calculated_recall = recall_score(true_labels_numpy, predictions_numpy,
average='binary', zero_division=0)

    training_log_message = f'Epoch {training_epoch}, Loss:
{training_loss.item():.4f}, Accuracy: {classification_accuracy:.4f}, F1:
{calculated_f1_score:.4f}, Precision: {calculated_precision:.4f}, Recall:
{calculated_recall:.4f}'
    print(training_log_message)
    model_training_logs.append(training_log_message)

training_end_time = time.time()

print("\nTraining finished.")
print(f"Total training time: {(training_end_time - training_start_time):.2f} seconds")

gnn_classification_model.eval()
with torch.no_grad():
    final_model_output = gnn_classification_model(graph_data_structure)
    final_predictions = final_model_output.argmax(dim=1)
    final_accuracy = (final_predictions == node_classification_labels).sum().item() /
len(node_classification_labels)
    print(f'Final Accuracy: {final_accuracy:.4f}')
    print(f'Trained for {training_epoch+1} epochs')

class CSSelectorGenerationAgent:
    def __init__(self, available_actions, gnn_model_output,
training_history_logs=None):
        self.possible_actions = available_actions
        self.q_learning_table = {}
        self.neural_network_output = gnn_model_output
        self.training_history = training_history_logs or []

```

```

def get_state_identification_key(self, element_node_id):
    return f"node_{element_node_id}"

def choose_optimal_action(self, element_node_id):
    state_key = self.get_state_identification_key(element_node_id)
    if state_key not in self.q_learning_table:
        self.q_learning_table[state_key] = [0.0 for _ in self.possible_actions]

    # Epsilon-greedy вибір дії
    if random.random() > 0.2:
        return
    self.possible_actions[int(torch.argmax(torch.tensor(self.q_learning_table[state_key]))
    )]
    return random.choice(self.possible_actions)

def generate_css_selector_for_element(self, target_html_element,
element_node_id):
    optimized_selector, is_selector_unique, selector_search_time =
generate_optimized_css_selector(
    target_html_element, parsed_html_soup
    )

    gnn_prediction_logits = self.neural_network_output[element_node_id]
    gnn_prediction_probabilities = torch.softmax(gnn_prediction_logits, dim=0)
    predicted_element_label = torch.argmax(gnn_prediction_logits).item()

    print(f"\nСкорочений селектор: {optimized_selector}")
    print(f"Унікальність: {'так' if is_selector_unique else 'ні'}")
    print(f"Час пошуку селектора: {selector_search_time * 1000:.2f} ms")
    print(f"GNN передбачення (label): {predicted_element_label}")
    print(f"GNN ймовірності: {gnn_prediction_probabilities.tolist()}")

    alternative_selectors_list = []

    if 'id' in target_html_element.attrs:
        alternative_selectors_list.append(f"#{target_html_element.attrs['id']}")
    if 'data-test-id' in target_html_element.attrs:
        alternative_selectors_list.append(f"{target_html_element.name}[data-test-
id=\"{target_html_element.attrs['data-test-id']}\"")
    if 'name' in target_html_element.attrs:
        alternative_selectors_list.append(f"{target_html_element.name}[name=\"{tar
get_html_element.attrs['name']}\"")
    if 'class' in target_html_element.attrs:

```

```

        escaped_css_classes = ['\\:'.join(class_name.split(':')) for class_name in
target_html_element.attrs['class']]
        alternative_selectors_list.append(f"{target_html_element.name}." +
'.'.join(escaped_css_classes))

        best_found_selector = optimized_selector
        best_selector_score =
calculate_css_selector_complexity_score(optimized_selector) if is_selector_unique
else float('inf')

        for alternative_selector in alternative_selectors_list:
            try:
                found_elements_with_selector =
parsed_html_soup.select(alternative_selector)
                if len(found_elements_with_selector) == 1:
                    alternative_complexity_score =
calculate_css_selector_complexity_score(alternative_selector)
                    if alternative_complexity_score < best_selector_score:
                        best_found_selector = alternative_selector
                        best_selector_score = alternative_complexity_score
                        print(f"[INFO] Знайдено кращий селектор: {best_found_selector} з
довжиною {best_selector_score:.2f}")
            except Exception as selector_error:
                print(f"[WARN] Альтернативний селектор не валідний:
{alternative_selector} -> {selector_error}")

        selector_hierarchy_parts = optimized_selector.split(' > ')
        best_found_selector = optimized_selector
        best_selector_score =
calculate_css_selector_complexity_score(optimized_selector)

        for hierarchy_level in range(1, len(selector_hierarchy_parts)):
            shortened_selector = ' > '.join(selector_hierarchy_parts[hierarchy_level:])
            try:
                elements_found_by_shortened_selector =
parsed_html_soup.select(shortened_selector)
                if (len(elements_found_by_shortened_selector) == 1 and
target_html_element in elements_found_by_shortened_selector):
                    shortened_complexity_score =
calculate_css_selector_complexity_score(shortened_selector)
                    if shortened_complexity_score < best_selector_score:
                        best_found_selector = shortened_selector
                        best_selector_score = shortened_complexity_score
            except Exception as selector_validation_error:

```

```

        print(f"[WARN] Неможливо перевірити селектор: {shortened_selector}
-> {selector_validation_error}")

```

```

    return best_found_selector

```

```

def generate_optimized_css_selector(target_element, html_soup,
maximum_hierarchy_depth=4):

```

```

    css_selector_parts = []

```

```

    current_element = target_element

```

```

    current_depth = 0

```

```

while current_element and current_depth < maximum_hierarchy_depth:

```

```

    element_selector_part = current_element.name or '*'

```

```

    if 'id' in current_element.attrs:

```

```

        element_selector_part += f"#{current_element.attrs['id']}"

```

```

        css_selector_parts.append(element_selector_part)

```

```

        break

```

```

    elif 'data-test-id' in current_element.attrs:

```

```

        element_selector_part += f"[data-test-id='{current_element.attrs['data-test-id']}']"

```

```

    elif 'name' in current_element.attrs:

```

```

        element_selector_part += f"[name='{current_element.attrs['name']}']"

```

```

    elif 'type' in current_element.attrs:

```

```

        element_selector_part += f"[type='{current_element.attrs['type']}']"

```

```

    if 'class' in current_element.attrs:

```

```

        escaped_class_names = ['\\'.join(class_name.split(':')) for class_name in
current_element.attrs['class']]

```

```

        element_selector_part += f".{'.'.join(escaped_class_names)}"

```

```

    parent_element = current_element.find_parent()

```

```

    if parent_element:

```

```

        sibling_elements = parent_element.find_all(current_element.name)

```

```

        if len(sibling_elements) > 1:

```

```

            element_position_index = sibling_elements.index(current_element) + 1

```

```

            element_selector_part += f":nth-child({element_position_index})"

```

```

    css_selector_parts.append(element_selector_part)

```

```

    current_element = parent_element

```

```

    current_depth += 1

```

```

complete_css_selector = ' > '.join(css_selector_parts[::-1])

selector_validation_start_time = time.time()
try:
    elements_found_by_selector = html_soup.select(complete_css_selector)
    is_selector_unique = (len(elements_found_by_selector) == 1)
except Exception as selector_error:
    print(f"Помилка селектора: {selector_error}")
    is_selector_unique = False
selector_validation_duration = time.time() - selector_validation_start_time

return complete_css_selector, is_selector_unique, selector_validation_duration

def calculate_css_selector_complexity_score(css_selector_string: str):
    # Ваги для різних компонентів селектора
    tag_weight = 0.5
    class_weight = 0.5
    id_weight = 0.2
    data_testid_weight = 0.4
    nth_child_weight = 0.8
    nesting_hierarchy_weight = 0.5

    total_complexity_score = 0.0
    selector_hierarchy_parts = css_selector_string.split('>')

    for selector_part in selector_hierarchy_parts:
        total_complexity_score += nesting_hierarchy_weight # за вкладеність
        cleaned_selector_part = selector_part.strip()

        if not cleaned_selector_part:
            continue

        total_complexity_score += tag_weight

        if '#' in cleaned_selector_part:
            total_complexity_score += id_weight
        if '.' in cleaned_selector_part:
            total_complexity_score += class_weight
        if '[data-test-id=' in cleaned_selector_part:
            total_complexity_score += data_testid_weight
        if ':nth-child' in cleaned_selector_part:
            total_complexity_score += nth_child_weight

    return total_complexity_score

```

```

def evaluate_selector_quality(generated_selector, selected_element_index,
all_elements):
    """
    Оцінює якість згенерованого селектора
    """
    # Справжня мітка: лише один правильний елемент (наш обраний)
    true_element_labels = [0] * len(all_elements)
    true_element_labels[selected_element_index] = 1

    predicted_element_labels = [0] * len(all_elements)
    try:
        elements_found_by_generated_selector =
parsing_html_soup.select(generated_selector)
        for found_element in elements_found_by_generated_selector:
            if found_element in all_elements:
                found_element_index = all_elements.index(found_element)
                predicted_element_labels[found_element_index] = 1
    except Exception as selector_evaluation_error:
        print(f"Помилка селектора під час обрахунку метрик:
{selector_evaluation_error}")

    calculated_f1_score = f1_score(true_element_labels, predicted_element_labels,
average='binary', zero_division=0)
    calculated_precision_score = precision_score(true_element_labels,
predicted_element_labels, average='binary', zero_division=0)
    calculated_recall_score = recall_score(true_element_labels,
predicted_element_labels, average='binary', zero_division=0)

    return calculated_f1_score, calculated_precision_score, calculated_recall_score

# ===== 8. Інтерактивний цикл для генерації селекторів =====
def interactive_selector_generation():
    """
    Інтерактивний цикл для генерації кількох селекторів підряд
    """
    all_html_elements = list(parsing_html_soup.find_all(True))

    # Створюємо агент один раз
    selector_generation_agent = CSSSelectorGenerationAgent(
        available_actions=['add_class', 'add_id', 'add_nth_child', 'stop'],
        gnn_model_output=final_model_output,
        training_history_logs=model_training_logs
    )

    print("=" * 60)

```

```

print("🔧 ІНТЕРАКТИВНИЙ ГЕНЕРАТОР CSS СЕЛЕКТОРІВ")
print("=" * 60)
print("Доступні команди:")
print(" - Введіть номер елемента (0-{}) для генерації
селектора".format(len(all_html_elements)-1))
print(" - 'list' або 'l' - показати всі елементи")
print(" - 'quit' або 'q' - вийти з програми")
print("=" * 60)

while True:
    try:
        user_input = input(f"\n👉 Введіть команду або номер елемента (0-
{len(all_html_elements)-1}): ").strip().lower()

        if user_input in ['quit', 'q', 'exit']:
            print("👋 До побачення!")
            break

        elif user_input in ['list', 'l']:
            print(f"\n📋 Список всіх HTML елементів ({len(all_html_elements)}
елементів):")
            print("-" * 80)
            for element_index, html_element in enumerate(all_html_elements):
                # Обмежуємо вивід атрибутів для кращої читабельності
                attrs_str = str(html_element.attrs)
                if len(attrs_str) > 60:
                    attrs_str = attrs_str[:57] + "..."
                print(f"{element_index:3d}: {html_element.name:10s} -> {attrs_str}")
            print("-" * 80)
            continue

        try:
            user_selected_element_index = int(user_input)

            if not (0 <= user_selected_element_index < len(all_html_elements)):
                print(f"❌ Помилка: Номер елемента повинен бути від 0 до
{len(all_html_elements)-1}")
                continue

            user_chosen_element = all_html_elements[user_selected_element_index]

            print(f"\n👁️ Обраний елемент #{user_selected_element_index}:")
            print(f"    Тер: {user_chosen_element.name}")
            print(f"    Атрибути: {user_chosen_element.attrs}")

```

```

print(f" Текст: {repr((user_chosen_element.string or '').strip()[:50])}")

print("\n❌ Генерую CSS селектор...")

generated_css_selector =
selector_generation_agent.generate_css_selector_for_element(
    user_chosen_element, user_selected_element_index
)

print("\n📊 Оцінка якості селектора:")
final_f1_score, final_precision_score, final_recall_score =
evaluate_selector_quality(
    generated_css_selector, user_selected_element_index, all_html_elements
)

print(f" Precision: {final_precision_score:.4f}")
print(f" Recall: {final_recall_score:.4f}")
print(f" F1-score: {final_f1_score:.4f}")

selector_complexity_score =
calculate_css_selector_complexity_score(generated_css_selector)

print(f"\n✅ Готовий селектор:")
print(f" {generated_css_selector}")
print(f" Складність (з вагами): {selector_complexity_score:.2f}")

# Перевірка селектора
try:
    found_elements = parsed_html_soup.select(generated_css_selector)
    print(f" Знайдено елементів: {len(found_elements)}")
    if len(found_elements) == 1:
        print(" ✅ Селектор унікальний!")
    elif len(found_elements) > 1:
        print(" ⚠️ Селектор знаходить кілька елементів")
    else:
        print(" ❌ Селектор не знаходить жодного елемента")
except Exception as e:
    print(f" ❌ Помилка перевірки селектора: {e}")

except ValueError:
    print(f"❌ Помилка: '{user_input}' не є валідним номером елемента")
    print("💡 Використовуйте 'list' для перегляду елементів або 'quit' для
виходу")

```

```
except KeyboardInterrupt:
    print("\n\n👋 Програма перервана користувачем. До побачення!")
    break
except Exception as unexpected_error:
    print(f"❌ Неочікувана помилка: {unexpected_error}")
    print("🔄 Продовжуємо роботу...")

interactive_selector_generation()
```

Додаток В Перелік експертів, які проводили оцінювання

Деякі коефіцієнти було визначено на основі експертного оцінювання. Більшість експертів надавали свою думку щодо розробленої інформаційної технології. Експертами виступили тестувальники та технічні лідери ІТ-компаній Екзип Текнолоджіз, ВІН ІНТЕРАКТИВ, Ria.com:

1. Ozerova Kateryna, Software Engineer, Ria.com
2. Tetiana Litovko, Manual QA Engineer, Екзип Текнолоджіз
3. Evelina Hirbachova, Manual QA Engineer, ВІН ІНТЕРАКТИВ
4. Nazarii Dzhevaha, Automation QA Engineer, ВІН ІНТЕРАКТИВ
5. Denys Gasenko, Manual QA Engineer, Екзип Текнолоджіз
6. Denys Chaika, Manual QA Engineer, ВІН ІНТЕРАКТИВ
7. Ivan Alekseyvych, Tech Lead QA Engineer, ВІН ІНТЕРАКТИВ
8. Dmytro Hurba, Tech Lead QA Engineer, Екзип Текнолоджіз

Додаток Г Список публікацій за темою дисертації

Наукові праці, в яких опубліковані основні наукові результати дисертації:

1. Морозов О. С., Яровий А. А. Експериментальні дослідження підходів до генерації тестових селекторів із застосуванням GNN у процесі автоматизованого тестування вебдодатків. Український журнал інформаційних технологій. 2025, т. 7, № 2. С. 44-49, DOI: <https://doi.org/10.23939/ujit2025.02.044> (ISSN 2707-1898). *Фахове видання категорії Б*
2. Морозов О.С., Яровий А.А., “Аналіз підходів та експериментальні дослідження процесу оптимізації автоматизованого тестування програмного забезпечення із застосуванням паралелізму тестів”. Наука і техніка сьогодні, №9(50), 2025. С. 1361-1374, DOI: [https://doi.org/10.52058/2786-6025-2025-9\(50\)-1361-1374](https://doi.org/10.52058/2786-6025-2025-9(50)-1361-1374) (ISSN 2786-6025). *Фахове видання категорії Б*
3. О. Морозов і А. Яровий, «Інтелектуальна модель генерації адаптивних веб-селекторів на основі GNN», Оптико-електронні інформаційно-енергетичні технології, вип. 50, вип. 2, с. 142–149, Січ 2026. DOI: <https://doi.org/10.31649/1681-7893-2025-50-2-142-149> (ISSN 2311-2662). *Фахове видання категорії Б*
4. Морозов О.С., Яровий А.А., “Аналіз особливостей процесу автоматизованого тестування веб-орієнтованих програмних продуктів”, Наука і техніка сьогодні, № 3(44), 2025. С. 1330-1339, DOI: [https://doi.org/10.52058/2786-6025-2025-3\(44\)-1330-1339](https://doi.org/10.52058/2786-6025-2025-3(44)-1330-1339) (ISSN 2786-6025). *Фахове видання категорії Б*

Наукові праці, що засвідчують апробацію матеріалів дисертації:

5. Morozov Oleksandr, Yarovyi Andrii., METHODS OF EVALUATING SELECTOR LENGTH FOR AUTOMATED TESTING // Cambridge

- Science and Education Conference : proceedings of the International scientific and practical conference (February 23-25, 2026). – Cambridge, United Kingdom : naukainfo.com, 2026. - Pp. 108-113. - DOI: <https://doi.org/10.64828/conf-100-2026>
6. Морозов О.С., Яровий А.А., Петришин С.І., " Гібридні методи для вибору тестових селекторів у динамічних веб додатках", Молодь науці: дослідження, проблеми, перспективи (МН-2025), 2025. Тип доступу: <https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/23706/19578>
 7. Морозов О.С., Яровий А.А., Козловський А.В., "Застосування машинного навчання для аналізу змін у веб-ресурса", LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2025), 2025. Тип доступу: <https://jpvrnd.conf.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23685/19566>
 8. Морозов О.С., Яровий А.А., Козловський А.В., " Специфічні аспекти процесу автоматизованого тестування WEB-орієнтованого програмного забезпечення із застосуванням паралельності тестів на основі TestNG", Молодь науці: дослідження, проблеми, перспективи (МН-2024), 2024. Тип доступу: <https://jpvrnd.conf.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/20609/17035>
 9. Морозов О.С., Яровий А.А., Козловський А.В., " Специфічні аспекти процесу автоматизованого тестування WEB-орієнтованого програмного забезпечення із застосуванням паралельності тестів на основі Selenium Grid", LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024), 2024. Тип доступу: <https://jpvrnd.conf.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20633/17062>

10. Морозов О.С., Яровий А.А., Козловський А.В. "Аналіз предметної області автоматизованого тестування WEB-ресурсів в контексті оптимізації процесів", Молодь в науці: дослідження, проблеми, перспективи (МН-2023), 2023. Тип доступу: <https://d.conf.vntu.edu.ua/index.php/mn/mn2023/paper/viewFile/18706/15503>
11. Морозов О.С., Яровий А.А., "Інтегральна метрика оцінювання якості тестових селекторів у динамічних web-додатках", LV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2026), 2026. Тип доступу: <https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2026/paper/view/28766/23649>