

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та
інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістр

на тему: «Застосування великих мовних моделей штучного інтелекту для
рефакторингу програмного коду вебдодатків»

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи та
технології
спеціальності 126 Інформаційні
системи та технології
ступеня вищої освіти магістр
групи 126ІСТ_мд_2024
Кіріченко Сергій Русланович
Керівник: Копішинська Олена
Петрівна
Рецензент: Муравльов Володимир
В'ячеславович

Полтава – 2025 року

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та
інформаційних технологій
Кафедра інформаційних систем та технологій

Освітня програма Інформаційні управляючі системи та технології
Спеціальність 126 Інформаційні системи та технології
Рівень вищої освіти другий (магістерський)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Юрій УТКІН

«08» листопада 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Кіріченка Сергія Руслановича

1. Тема кваліфікаційної роботи:

«Застосування великих мовних моделей штучного інтелекту для рефакторингу програмного коду вебдодатків»,

керівник роботи: к. ф.-м. н., доцент, професор кафедри інформаційних систем та технологій Копішинська Олена Петрівна.

Затверджено наказом закладу вищої освіти від «31» жовтня 2025 року № 1332-ст

2. Строк подання здобувачем вищої освіти роботи «09» грудня 2025 р.

3. Вихідні дані до роботи: JavaScript-бібліотеки, JavaScript-фреймворки, закриті LLM штучного інтелекту GPT-4o та Gemini, інструменти розробки GitHub Copilot, платформа Microsoft .NET, статистичні дані аналітичних компаній Stackoverflow, дані офіційних вебсайтів комерційних компаній

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

Розділ 1. Програмні засоби розробки коду вебдодатків

Розділ 2. Дослідження ефективності мовних моделей штучного інтелекту щодо рефакторингу і генерації програмного коду

Розділ 3. Результати застосування окремих мовних моделей штучного інтелекту для оптимізації коду вебдодатку

5. Перелік графічного матеріалу: схеми, рисунки, діаграми за темою та об'єктом дослідження.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання отримав
Оцінювання економічної ефективності результатів дослідження	Калініченко О. В., к. е. н., доцент, доцент кафедри економіки та публічного управління	24.11.2025	04.12.2025

7. Дата видачі завдання «08» листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1.	Вибір і затвердження теми роботи	29.10.2024 р.	
2.	Складання та погодження розгорнутого плану та завдання на кваліфікаційну роботу	30.10.2024 р. – 08.11.2024 р.	
3.	Опрацювання джерел інформації	11.11.2024 р. – 27.12.2024 р.	
4.	Збір, вивчення і обробка інформації, необхідної для виконання роботи	30.12.2024 р.– 19.01.2025 р.	
5.	Виконання теоретико-методологічного розділу роботи	17.02.2025 р.– 16.05.2025 р.	
6.	Виконання дослідницько-аналітичного розділу роботи	02.06.2025 р.– 13.07.2025 р.	
7.	Виконання проектно-рекомендаційного розділу роботи	08.09.2025 р.– 14.11.2025 р.	
8.	Оцінювання економічної ефективності результатів дослідження	24.11.2025 р.– 04.12.2025 р.	
9.	Оформлення тексту роботи	05.12.2025 р.– 08.12.2025 р.	
10.	Попередній захист роботи на кафедрі	09.12.2025 р.	
11.	Доопрацювання роботи з урахуванням зауважень і пропозицій	10.12.2025 р.- 14.12.2025 р.	
12.	Нормоконтроль	15.12.2025 р. – 16.12.2025 р.	
13.	Захист кваліфікаційної роботи	18.12.2025 р.	

Здобувач вищої освіти

Сергій КІРІЧЕНКО

Керівник роботи

Олена КОПШИНСЬКА

**ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ,
ПРАВА ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ**

КІРІЧЕНКО СЕРГІЙ РУСЛАНОВИЧ

**«ЗАСТОСУВАННЯ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ШТУЧНОГО
ІНТЕЛЕКТУ ДЛЯ РЕФАКТОРИНГУ ПРОГРАМНОГО КОДУ
ВЕБДОДАТКІВ»**

Освітньо-професійна програма
Інформаційні управляючі системи та технології
Спеціальність 126 Інформаційні системи та технології
Ступінь вищої освіти магістр

РЕФЕРАТ
кваліфікаційної роботи на здобуття кваліфікації –
магістр з інформаційних систем та технологій

Полтава – 2025 року

Кваліфікаційна робота складається із вступу, 3 розділів, висновків, списку використаних джерел (57 найменувань), 2 додатків. Кваліфікаційна робота містить 6 таблиць, 13 рисунків, викладена на 64 сторінках.

Основний зміст роботи

У першому розділі «Програмні засоби розробки коду вебдодатків» представлено ґрунтовний огляд тенденцій розвитку базових технологій і мов програмування для розробки вебдодатків, а також сфери застосування різноманітних фреймворків, рейтингування.

У другому розділі «Дослідження ефективності мовних моделей штучного інтелекту щодо рефакторингу і генерації програмного коду» визначені характеристики GPT і великих мовних моделей та вплив на трансформацію генерації коду, оновні напрямки застосування ШІ для рефакторингу коду

У третьому розділі «Результати застосування окремих мовних моделей штучного інтелекту для оптимізації коду вебдодатку» наведено результати експерименту з розпізнання та оптимізації окремих скриптів JS коду з використанням LLM GPT-4o, Gemini та Codestral. Здійснене оцінювання економічної та технологічної ефективності розробки вебдодатку за участю штучного інтелекту.

Висновки

1. Огляд та узагальнення рейтингів популярності інструментарію веброзробників дозволяють стверджувати, що серед мов програмування JavaScript s TypeScript є лідерами вже не один рік, так само як Angular, React Vue.js - серед фреймворків та бібліотек.

2. AI-підтримуване генерування та рефакторинг коду трансформують веброзробку на платформі .NET, спрощують процеси кодування, покращують продуктивність програмного забезпечення та підвищують рівень безпеки

3. Інтеграція GPT та LLM у розробку ПЗ розглядається багатьма як парадигмальний зсув. Вони підвищують продуктивність, демократизують створення програм, але водночас ставлять нові етичні та освітні виклики.

4. Результати з досліджених LLM показують здатність до точного розпізнання коду, написаного на JavaScript, особливостей функціоналу, а також внесення пропозицій щодо оптимізації, раціоналізації коду без втрати функціоналу. Всі моделі працювали через завантаження досліджуваного коду безпосередньо в чат. Найбільш розлогі рекомендації щодо покращення отримано від Gemini. Швидку генерацію нового коду надав GPT-4o.

5. Результати з досліджених LLM показують здатність до точного розпізнання коду, написаного на JavaScript, особливостей функціоналу, а також внесення

пропозицій щодо оптимізації, рефакторингу коду. Найбільш розлогі рекомендації щодо покращення отримано від Gemini. Швидку генерацію нового коду надав GPT-4o.

Подальшими напрямками досліджень можуть бути нові версії open source LLM, а також спеціалізовані платформи, призначені для генерації програмних кодів на інших мовах програмування

Достовірність результатів роботи базується на використанні відомих фреймворків мов програмування, відкритих моделей штучного інтелекту (LLM), даних ІТ-компаній та аналітичних груп, застосуванні загально прийнятих методів дослідження та обробки результатів.

Список публікацій здобувача

1. Копішинська О. П., Дузенко О.О., Кіріченко С.Р. Тенденції використання штучного інтелекту в області програмної інженерії. *Modern technologies in education, work and science* : The VIII International scientific and practical conference, October 20-22, 2025, Krakow, Poland. P.179-181. URL: <https://eu-conf.com/wp-content/uploads/2025/08/modern-technologies-in-education-work-and-science.pdf>

2. Копішинська О. П., Кабак Д. І., Кіріченко С. Р. Можливості оптимізації та рефакторингу коду вебдодатків на основі великих мовних моделей штучного інтелекту. "Світ наукових досліджень. Випуск 45": матер. Міжнародної мультидисциплінарної наукової інтернет-конференції, 21-22 жовтня 2025 року, м. Ополь, Польща. [електронне видання] URL: <https://www.economy-confer.com.ua/full-article/6477/>

3. Кіріченко С.Р. Роль середовищ контролю версій при реалізації вебпроектів. *Матеріали науково-практичної конференції за підсумками проходження виробничих практик здобувачів вищої освіти спеціальності 126 Інформаційні системи та технології, кафедра інформаційних систем та технологій Полтавського державного аграрного університету, 22 жовтня 2025 р.* Вип. XI. Полтава: ПДАУ, 2025. С. 63-65

АНОТАЦІЯ

Кіріченко С.Р. «Застосування великих мовних моделей штучного інтелекту для рефакторингу програмного коду вебдодатків». Кваліфікаційна робота на правах рукопису.

Кваліфікаційна робота на здобуття ступеня вищої освіти магістр за освітньо-професійною програмою Інформаційні управляючі системи та технології спеціальності 126 Інформаційні системи та технології. Полтавський державний аграрний університет, Полтава, 2025.

Робота містить глибокий теоретичний аналіз основних досягнень у використанні мовних моделей штучного інтелекту в області програмної інженерії. Представлено оригінальні результати експериментального дослідження методів взаємодії великих мовних моделей штучного інтелекту для рефакторингу програмних кодів вебдодатків або ж безпосереднього створення таких.

Ключові слова: JavaScript, мовні моделі штучного інтелекту LLM, рефакторинг коду, GPT 4o, Gemini

ANNOTATION

Kirichenko S. R. «Application of Large Artificial Intelligence Language Models for Refactoring Web Application Source Code». Qualification work (manuscript).

Qualification work submitted for the degree of Master of Higher Education in the Educational and Professional Program Information Management Systems and Technologies, specialty 126 Information Systems and Technologies. Poltava State Agrarian University, Poltava, 2025.

The thesis provides an in-depth theoretical analysis of key achievements in the use of artificial intelligence language models in software engineering. It presents original experimental results on methods for interacting with large AI language models to refactor the source code of web applications or to directly generate such code.

Keywords: JavaScript, AI language models (LLM), code refactoring, GPT-4o, Gemini.

ЗМІСТ

ВСТУП	6
РОЗДІЛ 1 ПРОГРАМНІ ЗАСОБИ РОЗРОБКИ КОДУ ВЕБДОДАТКІВ.....	10
1.1 Сучасні тенденції застосування мов програмування для розробки вебдодатків з інтеграцією базових технологій.....	10
1.2 JavaScript і його фреймворки в забезпечені інтерактивності вебдодатків та мови програмування backend-частини	17
1.3 Призначення фреймворків мов програмування в розробці вебдодатків та нові тренди в програмній інженерії	22
РОЗДІЛ 2. ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ МОВНИХ МОДЕЛЕЙ ШТУЧНОГО ІНТЕЛЕКТУ ЩОДО РЕФАКТОРИНГУ І ГЕНЕРАЦІЇ ПРОГРАМНОГО КОДУ	29
2.1 Автоматизація на основі штучного інтелекту у веброзробці: підвищення продуктивності та якості коду	29
2.2 Характеристики GPT і великих мовних моделей та вплив на трансформацію генерації коду.....	36
2.3 Основні напрямки застосування AI для рефакторингу коду.....	42
РОЗДІЛ 3 РЕЗУЛЬТАТИ ЗАСТОСУВАННЯ ОКРЕМИХ МОВНИХ МОДЕЛЕЙ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ОПТИМІЗАЦІЇ КОДУ ВЕБДОДАТКУ	49
3.1 Використання LLM GPT-4o для розпізнання та рефакторингу JS коду	49
3.2 Використання мовних моделей AI Gemini та Codestral	54
3.3 Оцінювання ефективності розробки вебдодатку за участю AI.....	58
ВИСНОВКИ.....	63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	65
ДОДАТКИ.....	71

ВСТУП

Актуальність теми кваліфікаційної роботи пов'язується з інноваційними підходами щодо створення або рефакторингу коду на тлі потужних можливостей штучного інтелекту. Написання програм високої складності вимагає від сучасного програміста не лише досконалого знання мов програмування, але й володіння відповідними підходами до раціоналізації роботи, перекладання рутинних операцій на додаткові засоби підтримки розробки. Саме з такими проблемами пов'язані виникнення численних бібліотек у багатьох мовах високого рівня, фреймворків, репозиторіїв коду, технологій Dev-Ops розробки, тобто доставки коду й швидкого впровадження в готовий проєкт. Застосування великих мовних моделей штучного інтелекту, які навчені не лише на лінгвістичних словниках всіх мов світу, але й на мовах програмування, дають небезпідставне сподівання на успішне їх використання для створення валідних кодів за умови значного зменшення ручної праці програміста. Однак, як і завжди, при впровадженні всіх технологічних революційних методів, постає чимало нових питань щодо їх адекватності, ефективності, а також переосмислення ролі людини як фахівця у контролі та прийнятті нових потоків згенерованого матеріалу.

Розуміння та впровадження в роботу новітніх технологій, особливо доступних моделей штучного інтелекту, стає критично важливим для досягнення конкурентних переваг і в області веброзробки, яка має вирішальне значення для організації інформаційної діяльності всіх осіб або компаній, організацій в глобальному середовищі при веденні бізнесу, комунікацій, використанні інформаційних ресурсів мережі інтернет, створення і використання клієнтської частини інформаційних систем із розподіленою архітектурою. Все це підкреслює необхідність дослідження можливостей мовних моделей штучного інтелекту для розробки і вдосконалення функціональності вебдодатків на стадіях написання і тестування, програмного та семантичного коду у всіх виробничих, наукових та бізнес-аспектах.

Враховуючи широкий спектр спеціалізованих рішень на основі великих мовних моделей штучного інтелекту, які є у вільному доступі, і які можливо залучати до процесу роботи з програмним кодом, необхідно окреслити межі їхніх можливостей, зокрема, для роботи з кодами на мовах веброзробки, провести багатокритеріальне дослідження характеристик, особливостей раціонального вибору, майстерності використання і прогресу або відставання порівняно з тим, як формують рішення фахівці відповідної області.

Зв'язок роботи з науковими програмами, темами. Робота виконана у відповідності до науково–дослідної ініціативної теми «Організаційно–методологічні аспекти впровадження інформаційно–комунікаційних систем і технологій в управлінні діяльністю сучасних організацій та підприємств за умов переходу до цифрової економіки» ДРН 0123U105060, яка реалізується на кафедрі інформаційних систем та технологій Полтавського державного аграрного університету (ПДАУ).

Метою кваліфікаційної роботи є теоретичне й експериментальне дослідження методів взаємодії та ефективності використання мовних моделей штучного інтелекту для рефакторингу програмних кодів вебдодатків або ж безпосереднього створення таких.

Для досягнення мети були визначені та реалізовані такі наукові завдання:

- визначити ключові підходи і мови для створення сучасних вебдодатків на основі рейтингування, аналізу наукових джерел, практичного досвіду провідних компаній;
- провести збір та впорядкування інформаційних даних про методи застосування різних моделей штучного інтелекту, особливостей доступу та сучасної проблематики, на якій фокусують увагу дослідницькі наукові центри;
- визначити коло питань, які мають бути перевірені або порівняні в рамках експериментальної частини досліджень із використанням кількох мовних моделей штучного інтелекту та прикладів розробленого програмного коду із області веброзробки;

– провести оцінку та узагальнення результатів щодо використання можливостей великих мовних моделей штучного інтелекту для рефакторингу програмних кодів функціональних вебдодатків.

Об'єктом дослідження кваліфікаційної роботи є заходи технічного рефакторингу програмного коду вебдодатків із використанням штучного інтелекту, зокрема, великих мовних моделей.

Предметом дослідження є: порівняльні властивості й технічні аспекти найбільш відомих мовних моделей штучного інтелекту щодо рефакторингу коду на мовах вебпрограмування (JavaScript та інших), а також методології промпт-інжинірингу в процесі формування завдань та верифікації результатів.

Методи наукових досліджень: абстрактно-логічний, інформаційно-пошуковий, аналітичний, методи тестування, великі мовні моделі штучного інтелекту.

Інформаційну базу кваліфікаційної роботи складають: наукові статті відповідного тематичного спрямування, тематична література, матеріали міжнародних науково-практичних конференцій, статистичні дані аналітичних компаній, які розміщені на вебсайтах у вільному доступі, власні спостереження та фахові наукові дослідження з досліджуваної проблеми, а також середовища відкритих моделей штучного інтелекту різних розробників, що доступні через інтернет.

Елементи наукової новизни полягають у здійсненні нетривіального аналізу можливостей штучного інтелекту для рефакторингу та підвищення функціональності коду вебдодатків, отриманні оціночних показників ефективності та тестуванні спроможностей моделей штучного інтелекту до розпізнання й рефакторингу програмного коду та висвітленні неоднозначності вибору та ефективності кожного з них.

Практична значущість одержаних результатів: результати роботи являють собою оригінальну працю з можливістю практичного використання та сформованих напрямків подальших наукових досліджень. Застосування результатів кваліфікаційної роботи дає змогу розширити можливості

технологій розробки вебдодатків та їхнього вдосконалення на етапах тестування й реалізації. Проведення експерименту з використанням моделей штучного інтелекту дозволяє прогнозувати в недалекому майбутньому перехід на абсолютно нові методи роботи з програмним кодом, надання розробникам нових знань та перспектив, розуміння нових компетентностей фахівців з інформаційних систем та технологій.

Апробація результатів дослідження відбувалася шляхом оприлюднення доповідей на міжнародних наукових конференціях: *Modern Technologies in education, work and science: the VIII International scientific and practical conference*, October 20-22, 2025, Krakow, Poland; Міжнародна мультидисциплінарна наукова інтернет-конференція «*Світ наукових досліджень. Випуск 45*», 21-22 жовтня 2025 року, м. Ополь, Польща із публікацією результатів у вигляді тез наукових доповідей у збірниках матеріалів конференцій.

Публікації. За результатами проведеного дослідження опубліковано тези: «Можливості оптимізації та рефакторингу коду вебдодатків на основі великих мовних моделей штучного інтелекту» та «Тенденції використання штучного інтелекту в області програмної інженерії» у співавторстві з науковим керівником, які проіндексовані в науково-метричній базі Google Academia.

Структура та обсяг кваліфікаційної роботи. Пояснювальна записка кваліфікаційної роботи складається зі вступу, 3 розділів, висновків, списку використаних джерел (57 найменувань), 2 додатків. Кваліфікаційна робота містить 6 таблиць, 13 рисунків, викладена на 64 сторінках.

РОЗДІЛ 1

ПРОГРАМНІ ЗАСОБИ РОЗРОБКИ КОДУ ВЕБДОДАТКІВ

1.1 Сучасні тенденції застосування мов програмування для розробки вебдодатків з інтеграцією базових технологій

Інтернет в сучасному полі застосування відбувся завдяки появі технології World Wide Web (WWW), запропонованої Тімом Бернесом Лі в 1989 р. Його ж, разом із командою односторонців, вважають засновником перших вебтехнологій, які дозволили розвинути сучасну мережу вебсайтів та інтернет-додатків. Спершу були статичні HTML-сторінки (Web 1.0). На зміну Web 1.0 прийшов Web 2.0, а разом із ним динамічні сайти та змога взаємодіяти з користувачами, що дало великий простір для реалізації ідей розробників. Наразі повним ходом спостерігається третя ітерація рівня Web 3.0, яка базується на децентралізації, використанні штучного інтелекту, одноранговій системі комунікацій та обміну інформацією з використанням технології blockchain [1-2].

Першою базовою технологією розробки вебсайтів стала мова розмітки гіпертексту HTML (Hypertext Markup Language) і від початку створювалася як мова для обміну технічною та науковою документацією, яку б могли використовувати люди, які не є фахівцями в області верстання [3].

Стандартами та специфікаціями ключових технологій розроблення вебсайтів займається W3C-консорціум, який від початку й до сьогодні очолює легендарний британець Тім Бернес-Лі. Протягом тридцяти з лишнім років мова гіпертексту HTML розвивалася і удосконалювалася завдяки відкритості платформи W3C [4]. Всі специфікації, пояснення, приклади розробники можуть знайти на офіційному вебсайті консорціуму. Стандарти W3C визначають відкриту вебплатформу для розробки додатків, яка має безпрецедентний потенціал, щоб дозволити розробникам створювати насичений інтерактивний досвід на основі величезних сховищ даних,

доступних на будь-якому пристрої [5]. Хоча межі платформи продовжують розвиватися, лідери галузі говорять майже в унісон про те, що HTML5 стане наріжним каменем для цієї платформи. Але повна потужність платформи спирається на багато інших технологій, які створюють W3C та її партнери, включаючи CSS, SVG, WOFF, стек семантичного веб, XML API.

За допомогою нього описується сама структура сайту, елементи які на ньому будуть знаходитися [6]. Все інше, включаючи CSS, JavaScript та різні бібліотеки, фреймворки й препроцесори не є обов'язковим з точки зору браузера, але в сучасному світі без подібних інструментів не створюється жоден вебдодаток.

Загалом, все що стосується зовнішнього виду сайту на боці клієнта відноситься до частини, яка називається frontend. На мові HTML автори методом «ручної верстки» описують структуру сторінок за допомогою розмітки. Елементи мови , або теги, позначають частини вмісту, такі як параграф `<p>`, список ``, ``, ``, таблиця `<table>` тощо. Існує стандартизований набір тегів, який оновлюється. Оновлені стандарти публікує W3C [7]. 1995 р. W3C створив HTML версії 2.0 після року інтенсивної роботи [8]. Водночас, обговорювалися можливості версії 3.0. Хоча друга версія не дуже відрізнялася від першої, третя стала безперечним проривом. Вона включала такі основні нововведення:

- розмітку математичних формул;
- теги для створення сторінок;
- вставку малюнків, що обтікаються текстом;
- приміток і т. д.

Незабаром було створено HTML версію 3.2. Вона була спрямована вже на новостворений Microsoft Internet Explorer [9]. До 1999 р. ця версія HTML була єдиним мовним стандартом розробки інтернет-проектів. Тим не менш, розробки в напрямку йдуть дуже активно. За допомогою HTML можлива деяка впорядкованість елементів розмітки для всіх браузерів, але можливості й цієї версії згодом стає недостатньо.

У 1998 р. була прийнята нова версія HTML – 4.0, а в 1999 р. її наступник – версія 4.01. Вона пропонує більш кращу крос-браузерну та крос-платформну продуктивність. Але ж тільки на той час і для більш адаптивної розробки її не вистачало.

Актуальність візуального дизайну гіпертекстових сторінок зростала, і просто покладатися на HTML було недостатньо [10]. Тому W3C розробила незалежну систему, яка відповідала HTML та дозволяла відображати візуальний дизайн документів [11]. Результатом стало створення каскадних таблиць стилів (CSS), ієрархічних специфікацій стилів з унікальним синтаксисом, структурою та можливостями.

До того, W3C розпочав розробку мови HTML5, набір функцій для створення структури вебсайту все ще був досить невеликим і обмеженим. Наприклад для додавання мультимедії, потрібно було використовувати пропрієтарні компоненти (Flash і т.п.).

Оскільки потреба у візуально привабливих гіпертекстових сторінках зросла, HTML5, новий стандарт розмітки для сучасного вебконтенту був розроблений і, нарешті, випущений у 2014 р., став доступний для всіх розробників. Поступово багато сучасних вебсайтів прийняли цей інноваційний підхід через різні фактори.

Світ веброзробки зазнав радикальних змін з появою HTML5, оскільки він усунув усі обмеження попередньої версії. Крім того, тепер є безліч нових тегів, що дозволяють більш точно описувати елементи сторінки. Відмова від Flash і подібних технологій стала можлива завдяки вбудованій в HTML5 можливості вбудовування мультимедійних об'єктів. Разом з цим зникли супутні проблеми, такі як низька продуктивність та вразливість коду.

Сьогодні HTML5 дозволяє створювати складні вебсайти, що включають мультимедійні та інтерактивні функції, не покладаючись на інші рішення. Мова повністю оснащена вбудованими функціями, такими як відео плеєри, засоби перевірки форм, графічні дизайнери та багато іншого, які поєднані з аспектами CSS, XHTML, DHTML та Ajax.

Багато старих тегів, у яких більше немає необхідності, було знято з виробництва, щоб робота HTML5 не стала надто складною. Але водночас набуття фундаментальних технологічних навичок стало набагато простіше. З легко засвоюваними та зрозумілими етикетками було введено й нові теги.

Документ HTML 5.2 складається з трьох частин.

1. Декларація типу документа (англ. Document type declaration, Doctype), на початку документа, в якій визначається тип документа (DTD).

2. Шапка документа (знаходиться в межах елемента head), в якій записано загальні технічні відомості або додаткова інформація про документ, яка не відтворюється безпосередньо в браузері.

3. Тіло документа (може знаходитися в елементі body), в якому міститься основна інформація документа.

Оскільки HTML дозволяє лише розміщувати елементи на сторінці, постає необхідність в інструменті, який буде вказувати браузеру як ці об'єкти виглядають. Рішенням цієї проблеми є каскадні таблиці стилів CSS (Cascading Style Sheets). За допомогою таблиці стилів можна задавати, розмір, колір, положення одних елементів відносно інших, поведінку курсора при наведенні на елемент та багато інших. Препроцесори дозволяють розширити функціонал HTML при роботі з верстанням. Написаний за допомогою препроцесора код після інтерпретації сервером перетворюється в звичайний HTML. Прикладом таких препроцесорів може бути Pug або Haml.

До переваг використання HTML–препроцесорів відноситься:

- простіша робота з великим об'ємом коду;
- дотримання принципу DRY (Don't Repeat Yourself);
- зменшення загального часу необхідного на розробку та підтримка організації коду [4].

Популярність технологій розробки вебдодатків різної складності та призначення із застосуванням мов програмування Python, JavaScript, Ruby, PHP та інших у поєднанні з базовими технологіями HTML&CSS стрімко зростає, що підтверджується щорічними рейтингами мов програмування

відомих аналітичних груп [11]. Різноманітність завдань, еволюція технологій веброзробки та зростаючі вимоги до продуктивності, безпеки, масштабованості веб-додатків призвели до значної диверсифікації інструментарію розробника, а також появи засобів спеціалізації та спрощення при створенні окремих проєктів. Хокон Віум Лі, норвезький спеціаліст з інформаційних технологій, запропонував ідею каскадних таблиць стилів, працюючи у консорціумі W3C.

Каскадні таблиці стилів CSS (Cascading Style Sheets) – мова опису зовнішнього вигляду документів, створених за допомогою мови розмітки HTML [12]. CSS – це широкомасштабна технологія, схвалена W3C та названа «вебстандартом». У 1990 р. необхідно було стандартизувати їх у деякі конкретні єдині правила завдяки вебдизайнерам та програмістам, які розробляють вебсайти.

CSS3 – це остання версія популярної мови, яка використовується для оформлення вебсайтів, але до неї, звісно, було кілька попередніх версій. Перша версія CSS стала рекомендацією W3C 17 грудня 1996 року. Майже через два роки, 18 травня 1998 року, було прийнято CSS2. Оновлену версію CSS2, яка отримала назву CSS2.1, схвалили 8 вересня 2009 р.

Актуальною версією CSS на даний момент є CSS3, більш нова CSS4 знаходиться у розробці. CSS (рис. 1.1) [13] має порівняно простий і логічно зрозумілий синтаксис, використовує набір англійських слів для найменування складових стилю (рис. 1.1).

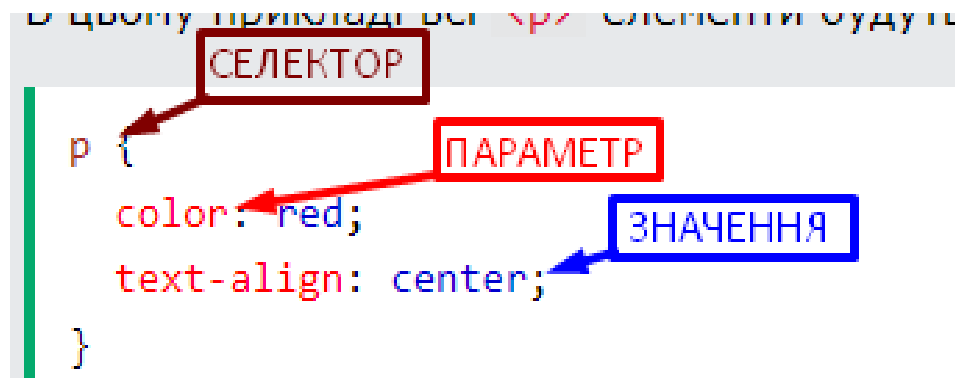


Рисунок 1.1 – Приклад синтаксису CSS та призначення обов’язкових знаків

По суті, таблиця стилів – це спеціальний розділ в головній частині сайту в тегах `<style>...</style>` або окремий файл стилів `style.css`, який створюється в одному з редакторів коду разом із HTML, має розширення `.css` для читання в браузері, в якому описується, як буде виглядати кожен із елементів на сайті [14]. При використанні окремого файлу стилів на нього необхідне спеціальне посилання з файлу розмітки `index.html` у вигляді тегу `<link>` і двох обов'язкових атрибутів: `<link rel="stylesheet" href="styles.css">`. Приклад запису частини вебсайту з посиланнями на зовнішній файл стилів наведено на рис. 1.2, приклад змісту файлу стилів – на рис. 1.3.

```

1  <html>
2  <head>
3    <title>вставка CSS в HTML сторінку</title>
4    <link rel="stylesheet" href="styles.css" />
5  </head>
6  <body>
7    <div>Текст параграфів цього документа лимонного кольору</div>
8    <div>Текст</div >
9    <div class="forexample">А тут текст оранжевий</div>
10   <div id="ident">Текст білого кольору, напівжирний</div>
11   <h2 class="forexample">Заголовок</h2>
12   <div>Текст</div>
13 </body>
14 </html>

```

Рисунок 1.2 – Фрагмент HTML-файлу із стандартним посиланням на стиль

```

index  styles.css  change.log  burger.svg
1  @charset "windows-1251";
2  /* CSS Document */
3
4  body {background-color:#0000cc}
5  div {font-size: 28; color:#F8F400}
6  .forexample {color:Orange}
7  #ident {color:#ffffff; font-weight:bold}

```

Рисунок 1.3 – Приклад запису CSS-селекторів в окремому файлі

Стилі складаються зі списку правил. Кожне правило має один або більше селектор (англ. *selector*) та блок визначення (англ. *declaration block*). Блок визначення складається з оточеного фігурними дужками списку властивостей.

1. Зовнішні таблиці стилів (англ. stylesheet), найчастіше окремий файл або файли .css.
2. Внутрішні таблиці стилів, включені як частина документу або блоку.
3. Стили для окремого елемента.
4. Стили користувача.
5. Локальний .CSS-файл, вказаний користувачем для використання на сторінках і вказаний в налаштуваннях браузера (наприклад Chrom).
6. Стили переглядача (браузера).
7. Стандартний стиль переглядача, наприклад стандартні стилі для елементів, визначені браузером, використовуються коли немає інформації про стиль елемента або вона неповна.

Стандарт CSS визначає порядок та діапазон застосування стилів, тобто, в якій послідовності і для яких елементів застосовуються стилі. Таким чином, використовується принцип каскадності, коли для елементів вказується лише та інформація про стилі, що змінилася або не визначена загальнішими стилями.

До переваг використання CSS відносять, зокрема, досягнення єдиного стильового оформлення однакових елементів, економія та раціоналізація коду, швидкість завантаження сторінки та інші. CSS – мова для опису представлення вебсторінок, включаючи кольори, макет і шрифти. Це дозволяє адаптувати презентацію до різних типів пристроїв, таких як великі екрани, маленькі екрани або принтери. CSS не залежить від HTML і може використовуватися з будь-якою мовою розмітки на основі XML. Відокремлення HTML від CSS забезпечує спільний доступ до таблиць стилів між сторінками та адаптацію сторінок до різних середовищ. Сторінки зменшуються в об'ємі, стають більш структурованими, оскільки інформація про стиль відділена від тексту та має певні правила застосування завдяки тому, що браузери можуть кешувати стилістичні дані кожної сторінки, а не завантажувати їх окремо для кожної.

Існують CSS-фреймворки, які ще більш спрощують роботу верстальника, серед яких найпоширеніші: Bootstrap, Foundation, Tailwind, а

також CSS-препроцесори – надбудови над CSS, які дозволяють писати спрощені синтаксичні конструкції, що підвищує простоту написання і редагування коду. Серед популярних препроцесорів можна виділити: Less, Sass (SCSS), Stylus [15]. Таким чином, основна робота з якістю коду сторінки переноситься в зміст написання властивостей таблиць стилів, уміння задавати необхідні функції та їх комбінування. За складністю та можливостями CSS давно визначена як одна з ключових технологій вебзробки. Тому володіння й виправлення є вимогою до кожного фронтенд розробника. Збільшення рутини потребує автоматизації вирішення за допомогою всіх можливих засобів, к тому числі й за рахунок залучення штучного інтелекту.

1.2 JavaScript і його фреймворки в забезпечені інтерактивності вебдодатків та мови програмування backend-частини

Мова програмування JavaScript використовується для створення динамічних інтерактивних вебсторінок. Мова JavaScript настільки популярна, що її використання сягає майже 98% [16]. JavaScript (JS) – динамічна, об'єктно-орієнтована прототипна мова програмування. Найбільш часто використовується для створення сценаріїв вебсторінок, які дозволяють користувачеві взаємодіяти з вебсторінкою, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та вигляд вебсторінки.

JavaScript є скриптовою мовою програмування з динамічною типізацією та є прототипом (підмножиною) об'єктно-орієнтованої. Окрім об'єктної на основі зразків, JavaScript частково підтримує інші парадигми програмування (імперативні та функціональні), а також деякі відповідні архітектурні властивості; це включає динамічну та слабку типізацію, автоматичне керування пам'яттю, наслідування на основі прототипів і функції як об'єкти першого класу. Мова JavaScript використовується для таких задач:

1. Написання сценаріїв вебсторінок для надання їм інтерактивності.

2. Створення односторінкових та прогресивних вебзастосунків (React, AngularJS, Vue.js).
3. Програмування на боці сервера (Node.js, Express.js).
4. Стаціонарних застосунків (Electron , NW.js).
5. Мобільних застосунків (React Native, Cordova).
6. Сценаріїв в прикладних програмах (наприклад, в програмах зі складу Adobe Creative Suite чи Apache JMeter).
7. Всередині PDF-документів.

При розробці сучасних вебсайтів, «ванільний» JS майже не використовують, за виключенням бажання досягти досконалої автентичності та відточити власну майстерність. Програмісти-практики віддають перевагу фреймворкам і бібліотекам. JavaScript-бібліотека – це колекція JS коду, який надає веброзробнику готові рішення для повсякденних задач. JavaScript-фреймворк також надає готові рішення, але також накладає певний стиль програмування [17]. Фреймворки (табл. 1.1) являють собою чітку структуру додатку і реалізуються с використанням так званих патернів проектування.

Таблиця 1.1– Порівняння фреймворків мови програмування JavaScript

Назва	Переваги	Недоліки
React	Ефективність, гнучкість, зручна система багаторазових компонентів, підтримка спільноти, просте налагодження	Погана документація, важкість пошукової оптимізації
Angular	Великий вибір функцій, ймовірність збоїв дуже мала, отримання даних напряму, підтримка Google та потужна екосистема	Високий поріг входу, важкість міграції за старої версії на нову
Vue.js	Простота, легкість інтеграції, дружній інтерфейс, хороша документація	Відсутність підтримки від великих проектів, відсутність плагінів
jQuery	Простота, прискорення роботи веброзробника , оптимізація під застарілі браузері, безліч готових рішення	Застарілий API, нема рівня даних, не підходить для комплексних задач

Найпопулярнішими фреймворками та бібліотеками є: React, Angular, Vue.js, jQuery. Ступінь їхньої популярності можна уявити за допомогою діаграми додатку А. Згідно аналітичних звітів компанії W3Tech [18], 17,8%

вебсайтів не використовують жодну з бібліотек JavaScript, які потрапили в зону опитування, jQuery використовується на 77,7 % усіх вебсайтів (верхня смуга), що складає 94,6 % частки ринку бібліотек JavaScript (нижня смуга).

Найбільш широко використовувані патерни це: MVC (Model-View-Controller), MVP (Model-View-Persistor), MVVW (Model-View-ViewModel). Додатків на фреймворках доволі багато, і цей сегмент набирає темп [19].

Технології, що описують і відповідають за функціонал вебсайту, об'єднують в частину backend-розробки. До них відносять: HTTP-запити на сервер для отримання інформації з бази даних, реєстрація та авторизація користувача, додавання та видалення контенту на сайті і т. ін. [20]. Переваги і недоліки різних популярних мов програмування, що використовують для backend частини наведено в табл. 1.2.

Таблиця 1.2 – Порівняльний аналіз мов програмування для backend

Назва	Переваги	Недоліки
PHP	Потужність і гнучкість, простота, велике ком'юніті, багато готових рішень	Менша захищеність порівняно з іншими мовами, неможливість створення десктопного застосунку
Ruby	Можливість використання лише необхідних бібліотек, швидкість розробки, надійний захист даних	Проблеми при роботі на Windows, мала кількість вакансій, використання великої кількості пам'яті
Python	Дуже легкий у вивченні, універсальність, дуже популярний, великий вибір готових рішень.	Повільність виконання, динамічна типізація, не сама зручна мова для мобільної розробки, місцями незручний синтаксис
Node.js	Швидкість виконання, кросплатформність, велика кількість інструментів	Важко писати великі проєкти, залежність від бібліотек
Java	Незалежність від платформи, запуск в «пісочниці», автоматичне управління пам'яттю застосунка	Низька швидкість, важкість написання коду, плата за комерційне використання

JavaScript можна використовувати, наприклад, для створення форми замовлення компонентів, перевірки достовірності даних, введених користувачем, та передачі даних на сервер у проєкті вебсайту зі збирання ПК. За допомогою заданих функцій JavaScript можна проводити обрахунки даних

та виводити результати на екран, не звертаючись до сервера. Крім того, JavaScript дозволяє взаємодіяти з API, що дає змогу отримувати найсвіжішу інформацію про компоненти безпосередньо з вебсайтів виробників.

Ще однією важливою особливістю JavaScript є можливість додавання на сайт динамічних елементів, таких як випадаючі меню, вкладки, повзунки тощо. Це підвищить естетичну привабливість сайту та покращить взаємодію з користувачами. Анімація, зокрема анімовані графіки, діаграми, графіки тощо, може бути створена за допомогою JavaScript. Це може допомогти у візуалізації даних і підвищити привабливість вебсайту для користувачів.

Існування JavaScript зробило можливим існування вебдодатків у повному розумінні – застосунків, в яких оновлення інформації відбувається без оновлення всієї сторінки [21]. У JavaScript фактично немає конкурентів: 98% вебсайтів використовують саме цю мову. Але альтернативи все ж є: Dart, CoffeeScript, Naxx, ClojureScript. Вважати їх повноцінними аналогами не можна, адже в результаті всі вони все рівно компілюються в JavaScript код. Рейтинг мов програмування за версією IEEE Spectrum [5] представлено на рис. 1.4.

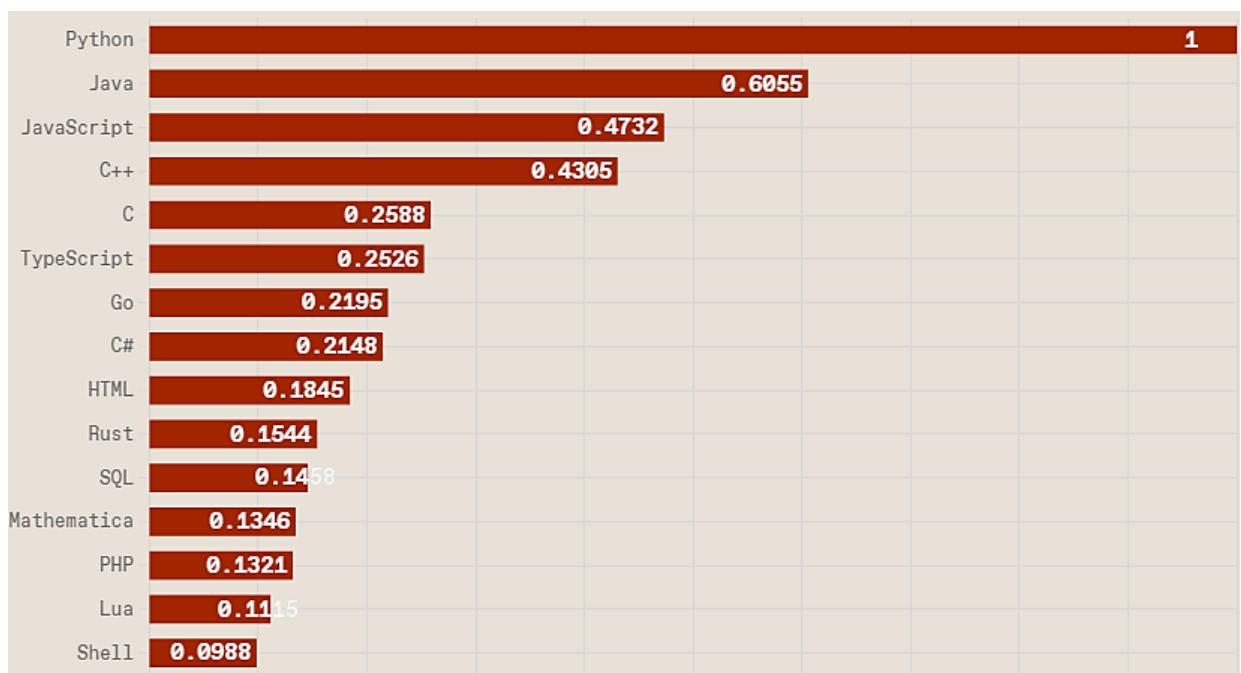


Рисунок 1.4 – Позиція JavaScript у рейтингу мов програмування (за даними [5])

Окремо варто розглядати TypeScript – мову програмування, що розширює JavaScript шляхом надання можливості статичної типізації (знаходиться в топі рейтингу мов вебпрограмування, див. рис. 1.4). TypeScript виник як надмножина JavaScript, яка додає статичну типізацію, що підвищує стабільність і зручність розробки великих проектів [21]. Використання TypeScript зазвичай не є обов'язковим (за винятком деяких фреймворків, наприклад – Angular), але його використання дуже бажано, особливо в проєктах з великою кількістю класів, функцій та змінних. Розробнику, який використовує TypeScript, не потрібно витрачати час, щоб дізнатися, які поля є у об'єкта, або що повертає та чи інша функція. В цьому йому допоможе IDE, яка повністю підтримує TypeScript, та буде показувати розробнику всю необхідну інформацію своєю функцією autocomplete (яка дуже погано працює з звичайним JavaScript).

Нові мови часто розробляються для полегшення специфічних задач або для надання більш зручних інструментів для певних типів проектів. Наприклад, Elm і Svelte були створені, щоб зробити розробку фронтенду більш передбачуваною та простою, з мінімальними помилками та високою продуктивністю. Різні компанії, організації та розробники експериментують з новими підходами до розробки вебдодатків. Це породжує конкуренцію та прагнення створити кращу мову програмування або фреймворк, що відповідає конкретним вимогам. На популярність мов програмування значний вплив мають також великі активні спільноти, що сприяють їх розвитку, створенню фреймворків та бібліотек. Наприклад, JavaScript став основною мовою для веброзробки завдяки широкій екосистемі бібліотек React, Angular, Vue.js, а також фреймворків.

Однак, на тлі еволюції та урізноманітнення технологій із стрімким зростанням кількості вебдодатків стали виявлятися й нові проблеми, які раніше не були такими очевидними. Наприклад, уніфікація та раціоналізація коду: кожен із представників масової професії веброзробника пише код у власному стилі. Раніше у кожній вебстудії, або навіть кожного окремого

розробника могла бути власна реалізація таких базових речей, як система авторизації, підключення та спілкування з базою даних тощо. Коли до проєкту долучається новий розробник, то йому потрібно довгий час вивчати підхід до написання коду, який використовувався попередніми програмістами. Внаслідок такого підходу утворилась велика кількість «spaghetti code», який вебпрограмістам до цього часу доводиться розбирати й переписувати на сучасний лад. Іншою проблемою стала велика кількість коду, який безпосередньо треба писати. Частковим рішенням цих проблем стали бібліотеки та фреймворки, а також інші системи контролю якості коду. Значний інтерес і перспективи має штучний інтелект, який навчений на десятках тисяч зразків програмного коду.

1.3 Призначення фреймворків мов програмування в розробці вебдодатків та нові тренди в програмній інженерії

Мова програмування JavaScript є інтегратором функцій сайту, керує його інтерактивністю та робить більш сприйнятним до дій користувача, дозволяє оновлювати зміст сайту залежно від запитів відвідувача. Сучасні сайти можливо розробляти й без використання фреймворків, але втім саме вони значно пришвидшують швидкість розробки і додають купу нових можливостей для розробника. Професіонали, які повсякчас стикаються з величезною кількістю рутинних або повторюваних операцій, оцінили інноваційні підходи до побудови кодів, які надали бібліотеки та фреймворки.

Фреймворки і бібліотеки містять код, який написав інший розробник. Цей код використовують, коли вирішують однотипні завдання чи поширені проблеми. Принцип раціональної розробки програмного забезпечення передбачає не дублювати код, який багато разів використовується. Фреймворк можна розглядати як певний набір інструментів, стандартів, підходів до написання коду. Саме він керує розробником та тим, як буде написаний код.

Фреймворки часто перетинаються з бібліотеками. Розробники використовують обидва компоненти, і новачки часто не розуміють, чим вони відрізняються. Стислі визначення вирізняють сутність двох понять. Фреймворк це платформа для створення вебсайтів та програм, яка полегшує розробку завдяки великій кількості реалізованих функцій. Програмістові не треба писати код із нуля. Він бере готове рішення і створює надбудову для реалізації продукту [22].

Бібліотеки – готові окремі компоненти, які вирішують певні завдання [22]. Наприклад, є бібліотеки для обробки файлів та виведення картинки на екран. За принципом дії відмінність полягає в наступному: фреймворк звертається до додатку, який містить основний код, та реалізує в ньому певні дії; до бібліотеки ж, навпаки, звертається додаток і включає її готові елементи, поєднуючи із основним планом, проектом. Основна відмінність фреймворку від бібліотеки пояснюється тим, що фреймворк ніби задає жорсткі рамки. Розробник інтегрує свій код у стороннє рішення, але не може вийти за межі стандартної логіки. Бібліотеки ж можна або використовувати будь-якої миті, або відключити зовсім, якщо є альтернативи [23]. Бібліотека – комплекс модулів, функцій та компонентів та іншого «готового» коду. По суті, це збірник коду, який можна повторно використовувати в своїх проєктах [24]. Завдяки ним веброзробнику не потрібно писати з нуля такі речі як наприклад модальні вікна, каруселі зображень, слайдери і багато інших. Це значно спрощує процес розробки.

Вказані відмінності бажано розуміти, щоб правильно використовувати обидві технології. Коли замовник просить створити сайт на базі готової CMS, PHP-фреймворки вже не знадобляться, а ось CSS можуть стати в нагоді. Можна підключити їх до проєкту та не писати код стандартних функцій. У бібліотек та фреймворків спільна мета – звільнити програміста від вирішення однотипних завдань.

Фреймворк обмежує розробника, не даючи вийти за свої рамки. Додатки написані на одному фреймворкові будуть мати майже однакоvu структуру

файлів, ідентичні патерни розробки та підхід до вирішення задач. Окрім цього, більшість сучасних фреймворків мають великий арсенал вбудованого функціоналу, що вберігає розробників від «винайдення велосипедів», а також надає розробникам можливість сфокусуватися на більш важливих аспектах розробки, наприклад на бізнес-логіці [25].

З часом кількість готових рішень стала збільшуватись зі значною швидкістю. Існує жарт, що якщо відкрити словник англійської мови і обрати звідти будь-яке слово, то з такою назвою вже буде існувати JavaScript фреймворк. Кожен фреймворк має свої особливості, переваги та недоліки. Деякі з них краще підходять для розробки маленьких і простих проєктів, а деякі розроблені спеціально для використання в масштабних проєктах, які пишуться не один рік.

Існують PHP, CSS та JS-фреймворки. Кожен вид програмних інструментів вирішує свої завдання, але вони об'єднані спільною метою – допомогти розробнику позбутися рутини. Замість писати, наприклад, систему виведення шаблонів сторінки кожного проєкту, програміст може займатися нестандартними завданнями. Фреймворки та бібліотеки не є легким вирішенням проблем. Вони лише пропонують «фундамент» з урахуванням якого можна створити проєкт. Якщо розробник сподівається, що з нього зроблять всю роботу, ці інструменти не вміють.

PHP-фреймворки використовуються на бекенд для формування базової структури. Для фронтенду існує велика кількість своїх фреймворків, які допомагають швидко зібрати каркас і змінити його зовнішній вигляд.

Кількість бібліотек та фреймворків постійно збільшується. З'являються гнучкіші рішення, творці яких обіцяють високу продуктивність та швидкодію. Розробникам треба стежити за трендами, щоб через кілька років не виявилось, що вони користуються застарілим стеком і він перестав бути затребуваним. Відомі аналітичні компанії шляхом різних методів досліджень складають рейтинги популярності фреймворків та бібліотек. Перелік найбільш відомих фреймворків наведено на рис. 1.5.

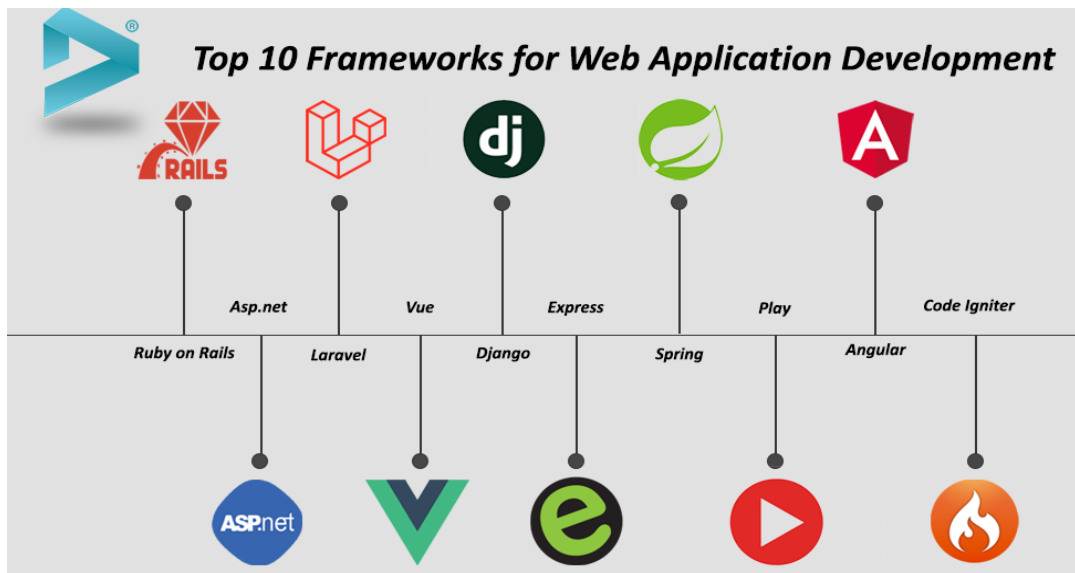


Рисунок 1.5 – Схематичний розподіл відомих фреймворків за призначенням

Фахівці компанії Stack Owerflow у 2024 р. провели щорічне опитування серед професійних девелоперів, кількість яких для статистичної вибірки склала 46261 респондента [25]. Опитування включало оцінювання різних технологій за певною шкалою відповідей на питання про різні фреймворки. Потрібно було класифікувати, наприклад, за категоріями: найулюбленіший, найважчий і бажаний; найлегший для навчання та вирішення проблем; найбільш високооплачувані технології та ін.

Виявили, що React, Angular не втрачають своїх позицій. Усі вони входять до першої п'ятірки, а React виграє з результатом у 41,6 %. Vue знизив позиції до 16,6 % у порівнянні з 2023 р. [25]. Серед фреймворків є вдалі рішення і недоліки, тому їх не можна назвати ідеальним інструментом. При правильному використанні вони економлять час, але, якщо у розробника недостатньо досвіду роботи, то у взаємодії з новим фреймворком виникає багато проблем. Завдання, які вирішують фреймворки для посилення якості коду і розробки, які можна узагальнити.

1. Покращують швидкість розробки. У програміста буде відкладене «ядро», яке можна використовувати як основу проекту.

2. Звільняють від рутинних завдань, займатися реалізацією нестандартних функцій.

3. Допомагають залишитися конкурентоспроможним на ринку. Якщо програміст в досконалої освоїв кілька популярних фреймворків, він не залишиться без роботи.

4. Зменшують вартість задач. Якщо програміст не створює сайт з нуля, а використовує фреймворк, то вебсайт може коштувати дешевше.

У фреймворків відкритий вихідний код, тому будь-який розробник може внести зміни в логіку та адаптувати програмний продукт під свої завдання. Переважна більшість backend-розробників в своїй роботі також використовують фреймворки. Найпопулярніші backend-фреймворки станом на 2024 р. наведено на рис. 1.6.

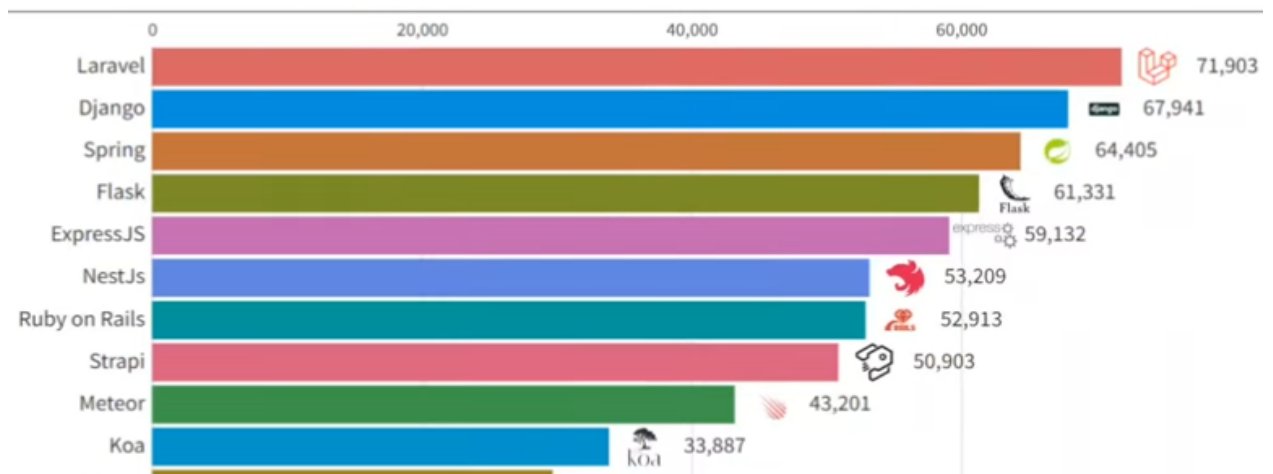


Рисунок 1.6 – Діаграма популярності backend-фреймворків [26]

На діаграмі видно, що найбільш популярними фреймворками є: Laravel (PHP), Django (Python), Spring (Java) та ExpressJS (Node.js). Backend-частину вебсайту зазвичай пишуть на таких мовах програмування як Java, Python, Ruby, Node.js, PHP. Вибір є досить суб'єктивним та розрізняється підходами до реалізації при розробці, але всі інструменти для самостійної розробки сайту мають конкурентоспроможні ресурси.

Програмна інженерія значно розвивалася протягом років завдяки поступовим змінам, які призвели до появи монолітних структур та їх заміни модульними та масштабованими структурами. Оскільки автоматизація та попит на гнучкість, швидкість та здатність реагувати на зміни зростають,

оптимізація кодів та рефакторинг стали важливими практиками. Зі зростанням складності програмних систем стало важче підтримувати якісний код з високою продуктивністю та гнучкістю. Більшість старих методів, що використовуються для оптимізації коду, а також його рефакторингу, які фактично передбачають аналіз коду за допомогою статичних інструментів, більше не є достатніми для відповідності вимогам сучасного програмного забезпечення. Ці підходи не є ефективними для управління великою складною базою коду додатків, і вони часто вимагають значного втручання людини, що забирає як багато часу, так і схильне до помилок. Крім того, оскільки організації хочуть, щоб кожен цикл розробки був коротшим за попередній, написання ефективних програм, зберігаючи при цьому їхню якість та читабельність, є досить складним завданням.

Автоматизація завдяки новому елементу штучного інтелекту стала ключовим елементом цієї зміни в традиційних підходах до оптимізації та рефакторингу коду [27].

Штучний інтелект (Artificial Intelligence, AI) став корисним у розробці програмного забезпечення, допомагаючи покращити оптимізацію/рефакторинг коду, тим самим підвищуючи продуктивність та стійку підтримку. Інструменти AI, включаючи CodeT5, Codex, Intel Neural Compressor та Refactoring Miner, допомагають розробникам аналізувати код, мінімізувати його та просувати рефакторинг. Повний набір напрямків застосування AI в програмній інженерії наведено в додатку А.

У наступному розділі досліджується використання AI в рефакторингу коду та його ефективність в оптимізації кодів, що використовуються в різних галузях, на реальних прикладах, підкреслюючи вплив AI на підвищення продуктивності системи, можливості читання коду та зменшення надмірно обтяжливого та проблемного технічного боргу. Необхідно дослідити дієві моделі AI, горизонти використання для розробки програмного забезпечення; тестування та забезпечення якості; самоадаптивного коду; синтезу програм, що може повністю змінити цикл розробки та методи кодування.

Висновки до розділу 1

В теоретичному розділі було проаналізовано сучасний стан вебтехнологій та тенденції їх розвитку. На основі аналізу численних наукових праць та відгуків самих програмістів можна стверджувати, що наразі відбувається стрімке зростання вимог до ефективності різноманітних програмних засобів для розробки вебдодатків.

Огляд та узагальнення рейтингів популярності інструментарію веброзробників дозволяють стверджувати, що серед мов програмування JavaScript і TypeScript є лідерами вже не один рік, так само як Angular, React Vue.js – серед фреймворків та бібліотек. Однак, вони також мають окремі недоліки, що не дає можливості вважати їх універсальним інструментом, тим більше, в доволі конкурентному середовищі програмних засобів проєктування.

Backend-частину вебдодатків найчастіше пишуть на таких мовах програмування, як Java, Python, Ruby, Node.js, PHP. Для цих мов також розробляються фреймворки, найбільш популярними з яких є: Laravel (PHP), Django (Python), Spring (Java) та ExpressJS (Node.js).

Одним із головних завдань сучасного веброзробника є аналіз та підбір найкращого рішення саме для потреб його проєкту, пошук нових підходів та технологій. Впровадження методів на основі штучного інтелекту стало основним сучасним трендом програмної інженерії. Однак, про тотальне застосування поки що не йдеться, оскільки інновація створює й нові проблеми: як розробляти моделі, які могли б працювати в різних репозиторіях, і як зробити їхні результати більш зрозумілими та надійними. Подальший аналіз має зосередитись на короткому описі основних питань, що стосуються застосування методів штучного інтелекту для рефакторингу коду, а також способів пом'якшення проблем та неузгодженостей для підвищення ефективності та якості коду.

РОЗДІЛ 2

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ МОВНИХ МОДЕЛЕЙ ШТУЧНОГО ІНТЕЛЕКТУ ЩОДО РЕФАКТОРИНГУ І ГЕНЕРАЦІЇ ПРОГРАМНОГО КОДУ

2.1 Автоматизація на основі штучного інтелекту у веброзробці: підвищення продуктивності та якості коду

Для досягнення мети кваліфікаційної роботи було сформульовано низку завдань, основною ідеєю яких є дослідження можливостей кількох найбільш відомих моделей AI для досягнення вищої продуктивності у веброзробці при генерації і рефакторингу коду. На першому етапі проведено аналіз досягнень дослідників у цій області та визначення для майбутніх експериментів відповідних інструментів.

Інтеграція штучного інтелекту у розробку програмного забезпечення змінює підхід до написання, оптимізації та підтримки коду. Традиційні процеси кодування, які часто передбачають значні ручні зусилля, дедалі частіше доповнюються інструментами на основі AI, що допомагають у генеруванні коду, налагодженні (debugging) й оптимізації [28].

Ця трансформація особливо помітна у веброзробці на платформі .NET (Microsoft .NET), де складні застосунки потребують ефективного й масштабованого коду для задоволення сучасних вимог до продуктивності та безпеки. Зазвичай застосунки пишуть на мовах мовах C#, інколи F# або VB.NET. Генерування й оптимізація коду за допомогою AI не лише підвищує продуктивність розробників, а й сприяє створенню більш надійних і високопродуктивних програмних систем.

AI-підтримуване генерування коду (AI-assisted code generation) передбачає використання моделей машинного навчання (machine learning models) та технологій обробки природної мови (Natural Language Processing, NLP) для автоматичного створення або пропонування фрагментів коду на основі введення розробником. Інструменти, такі як GitHub Copilot, OpenAI

Codex і Microsoft IntelliCode, аналізують великі репозиторії коду, щоб надавати пропозиції в режимі реального часу, зменшуючи потребу у ручному написанні повторюваних конструкцій. Для веброзробки на платформі .NET це означає швидшу реалізацію типових шаблонів, таких як контролери API, рівні доступу до даних та модулі автентифікації. Автодоповнення коду (code completion), кероване AI, пришвидшує робочі процеси розробки, мінімізує синтаксичні помилки та забезпечує дотримання найкращих практик програмування, що веде до створення більш підтримуваних і ефективних застосунків [29].

Крім генерування коду, AI відіграє ключову роль в оптимізації коду. Інструменти профілювання продуктивності, що працюють на основі AI, аналізують шаблони виконання, виявляють «вузькі місця» (bottlenecks) і пропонують оптимізації для підвищення ефективності роботи застосунку. У веброзробці на платформі .NET, де високопродуктивність має критичне значення, інструменти на кшталт «Performance Profiler» і «Application Insights» у Visual Studio допомагають розробникам удосконалювати запити до баз даних, оптимізувати використання пам'яті та усувати надлишкові обчислення.

Алгоритми AI можуть прогнозувати потенційні проблеми з продуктивністю до того, як вони вплинуть на кінцевих користувачів, що дає змогу розробникам проактивно усувати неефективності.

Ще один важливий аспект AI-підтримуваної розробки – інтелектуальне налагодження й автоматичне виявлення помилок. Налагодження традиційно є тривалим процесом, який вимагає від розробників ручного відстеження помилок і визначення їхніх причин. AI-підтримувані інструменти налагодження застосовують моделі машинного навчання для виявлення аномалій, пропонування виправлень і навіть автоматичного рефакторингу (refactoring) неефективного коду [30].

Такі функції, як передбачуване виявлення помилок («Predictive bug detection» у Visual Studio) та класифікація помилок, керована AI, у Azure DevOps дають змогу швидше усувати проблеми, підвищувати надійність коду.

Безпека – ще одна сфера, де AI робить вагомий внесок у веброзробку на .NET. Інструменти статичного аналізу коду, такі як SonarQube і CodeQL, перевіряють кодову базу на наявність вразливостей і надають рекомендації в реальному часі щодо безпечного програмування. В умовах, коли загрози кібербезпеці стають дедалі складнішими, AI-підтримуваний аналіз безпеки допомагає виявляти потенційні ризики, такі як SQL injection, міжсайтове скриптування (XSS) та небезпечні механізми автентифікації ще до розгортання застосунків у продакшн.

Інтеграція AI у веброзробку на платформі .NET суттєво змінила спосіб, у який розробники пишуть, оптимізують і підтримують код. Із зростанням складності сучасних вебзастосунків розробники часто стикаються з проблемами масштабованості, оптимізації продуктивності та налагодження. Автоматизація на основі AI вирішує ці проблеми, спрощуючи робочі процеси кодування, підвищуючи ефективність і забезпечуючи якість коду за допомогою інтелектуальних рекомендацій, автоматизованого налагодження та профілювання продуктивності.

Використовуючи інструменти розробки, керовані AI, такі як GitHub Copilot, OpenAI Codex і Microsoft IntelliCode, розробники можуть пришвидшити цикл розробки програмного забезпечення, водночас підтримуючи високу якість коду.

Одним із найвагоміших внесків AI у веброзробку на платформі .NET є його здатність допомагати в генеруванні коду. Традиційне програмування вимагає значних ручних зусиль – розробникам потрібно писати шаблонний код, впроваджувати стандартні шаблони проектування і дотримуватися найкращих практик. Асистенти на основі AI аналізують великі репозиторії коду, щоб створювати контекстно-залежні фрагменти коду на основі введення розробника [31].

Наприклад, GitHub Copilot може пропонувати цілі функції або класи на основі кількох рядків коментарів, що дозволяє швидко реалізовувати API, рівні доступу до даних і модулі автентифікації. Це скорочує час розробки та

дає змогу командам зосередитися на складніших аспектах архітектури застосунку.

Поза межами генерування коду автоматизація на основі AI відіграє ключову роль у налагодженні та оптимізації коду. Традиційно налагодження є трудомістким процесом, що вимагає від розробників відстеження помилок, визначення вузьких місць і ручного усунення проблем. Інструменти налагодження на базі AI, такі як IntelliCode у Visual Studio та діагностичні засоби Azure, автоматизують виявлення помилок, аналізуючи історичні шаблони коду та типові сценарії налагодження.

Такі інструменти виділяють потенційні вразливості, пропонують виправлення та навіть прогнозують появу помилок до їх виникнення [32]. Автоматизація процесів налагодження підвищує надійність коду, мінімізує дефекти програмного забезпечення та прискорює цикли розгортання.

Оптимізація продуктивності – ще одна сфера, де автоматизація на базі AI істотно впливає на веброзробку .NET. Інструменти профілювання, керовані AI, аналізують роботу застосунку в реальному часі, виявляючи повільні запити до бази даних, неефективні алгоритми та надмірне використання пам'яті. Visual Studio Performance Profiler і Azure Application Insights надають практичні рекомендації для оптимізації виконання коду, зниження затримок і підвищення загальної швидкодії застосунку. Завдяки алгоритмам машинного навчання ці інструменти можуть прогнозувати погіршення продуктивності та рекомендувати оптимізації ще до того, як проблеми вплинуть на кінцевих користувачів.

Крім підвищення продуктивності, автоматизація на базі AI зміцнює безпеку застосунків .NET. У зв'язку з швидкою еволюцією кіберзагроз, інструменти безпеки, що працюють на основі AI, допомагають зменшувати ризики, аналізуючи код на наявність вразливостей. Засоби статичного аналізу коду, такі як SonarQube і CodeQL, застосовують методи машинного навчання для виявлення таких вразливостей, як SQL injection, міжсайтове скриптування і помилки автентифікації [33]. AI-підтримувані системи сканування безпеки

надають рекомендації в реальному часі, забезпечуючи дотримання принципів безпечного кодування на всіх етапах життєвого циклу ПЗ.

Попри численні переваги, автоматизація на базі AI у веброботці .NET має й певні обмеження. Одне з головних занепокоєнь стосується надійності згенерованого коду: хоча моделі AI навчаються на великих наборах даних, вони не завжди створюють оптимальний або безпечний код, тому потребують людського контролю. Розробники повинні перевіряти код, згенерований AI, щоб переконатися, що він відповідає галузевим стандартам і специфічним вимогам застосунку.

Впроваджуючи автоматизацію, керовану AI, розробники .NET можуть оптимізувати робочі процеси, створювати високоякісні застосунки та підтримувати конкурентоспроможність у сучасному технологічному середовищі. Рис. 2.1 ілюструє, як AI підсилює різні етапи веброботки, підвищуючи продуктивність і якість коду.

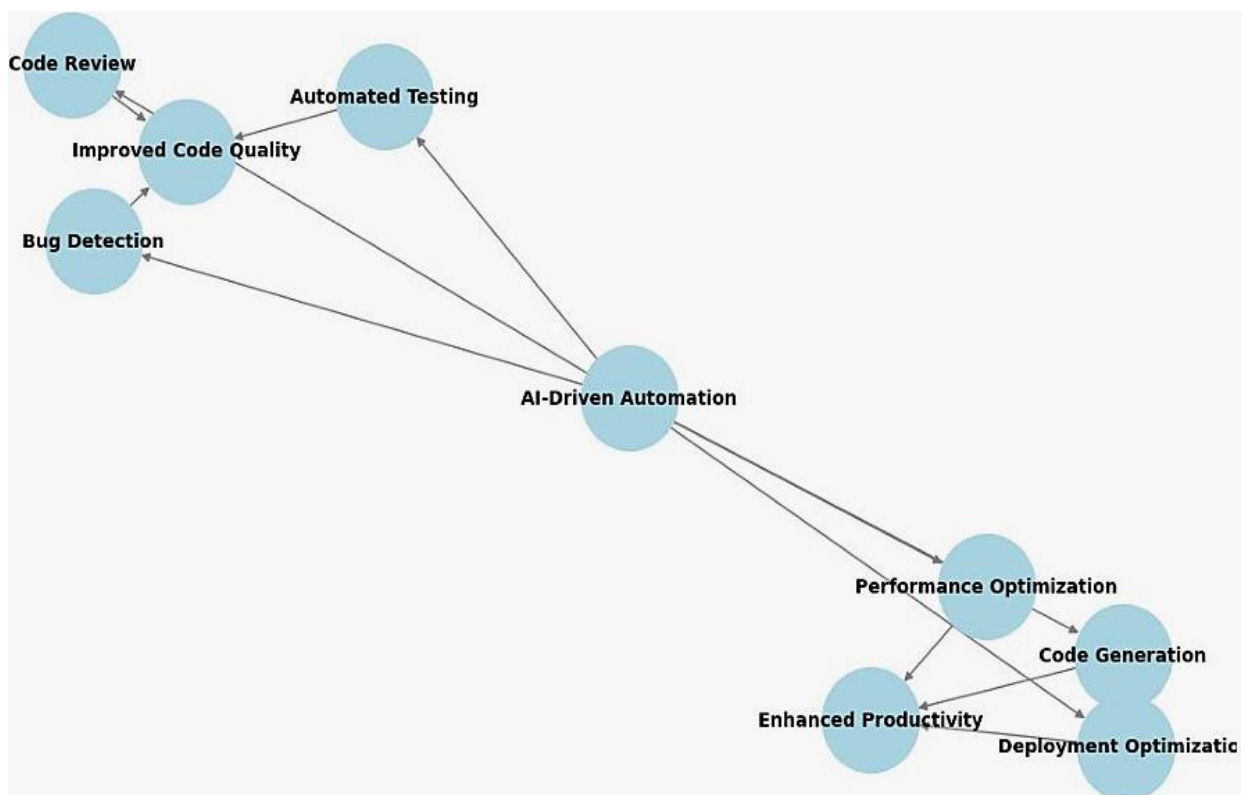


Рисунок 2.1 – Автоматизація на основі штучного інтелекту і методи, що підвищують ефективність та якість веброботки .NET [34]

Крім того, етичні аспекти, зокрема упередженість у моделях AI і питання інтелектуальної власності, також мають бути враховані, щоб уникнути небажаних наслідків у розробці ПЗ. Дивлячись у майбутнє, автоматизація на основі AI продовжить розвиватися, удосконалюючи веброзробку .NET завдяки більш інтелектуальній і контекстно-обізнаній підтримці програмування. У міру вдосконалення моделей AI розробники отримуватимуть ще ширші AI можливості автоматизації, включно з колаборацією в реальному часі, передбачуваним технічним обслуговуванням та автоматизованим тестуванням [35].

Завдяки AI-керованим технологіям, таким як автоматизоване профілювання продуктивності, передбачуване налагодження і розумне кешування, розробники .NET можуть підвищити відгук системи та забезпечити безперервний користувацький досвід.

Одним із ключових компонентів AI-підтримуваної оптимізації коду є профілювання продуктивності. Традиційні методи профілювання вимагають від розробників ручного аналізу шаблонів виконання, пошуку повільних процесів і оптимізації розподілу ресурсів. Інструменти профілювання, що працюють на основі AI, такі як Visual Studio Performance Profiler і Azure Monitor, автоматизують цей процес, збираючи дані про продуктивність у реальному часі та надаючи практичні рекомендації. Ці інструменти використовують алгоритми машинного навчання для виявлення неефективних SQL-запитів, надмірного використання пам'яті та ресурсомістких операцій центрального процесора (CPU-intensive operations). Аналізуючи історичні дані продуктивності, AI може передбачати потенційне погіршення роботи системи та пропонувати оптимізації до того, як проблеми вплинуть на користувачів.

Передбачуване налагодження – ще один аспект оптимізації коду на основі AI. Традиційно цей процес є тривалим і передбачає аналіз журналів помилок, відстеження залежностей і виправлення багів вручну. Інструменти налагодження, керовані AI, автоматично визначають потенційні проблеми й пропонують рішення, спираючись на попередні шаблони налагодження.

Асистент налагодження, керований AI, у Microsoft Visual Studio аналізує шляхи виконання коду та виділяє можливі помилки до моменту їх виникнення під час виконання [36]. Такий проактивний підхід зменшує простої, підвищує надійність застосунку та прискорює цикл розробки.

AI також відіграє важливу роль в оптимізації роботи баз даних у застосунках .NET. Повільні SQL-запити можуть призвести до вузьких місць у продуктивності, що знижує швидкість відгуку застосунку.

Інструменти оптимізації запитів на основі AI аналізують плани виконання SQL-запитів (SQL execution plans), виявляють неефективні запити та пропонують стратегії індексування для покращення продуктивності бази даних.

Функція Intelligent Query Processing в Azure SQL Database використовує машинне навчання для адаптації стратегій виконання запитів залежно від навантаження. Динамічно регулюючи плани виконання, AI забезпечує ефективну обробку великих обсягів даних без зниження продуктивності.

Інтелектуальне кешування – це сфера, в якій AI також підвищує масштабованість застосунків. Кешування є ключовим механізмом зменшення надлишкових обчислень і покращення часу відгуку, однак управління термінами зберігання кешу (cache expiration) може бути складним. AI-підтримувані системи кешування аналізують шаблони використання та прогнозують, які дані слід зберігати для досягнення оптимальної продуктивності [37].

AI-підтримуваний рефакторинг також змінює підхід до оптимізації коду в .NET. Рефакторинг є необхідним для підтримання чистої, ефективною та масштабованою кодовою бази. Інструменти рефакторингу, що використовують машинне навчання, аналізують структуру коду та пропонують покращення, наприклад, усунення дублювання, спрощення складних функцій і оптимізацію використання пам'яті.

2.2 Характеристики GPT і великих мовних моделей та вплив на трансформацію генерації коду

Поява генеративних попередньо натренованих трансформерів (GPT) та великих мовних моделей (LLM) започаткувала нову епоху у сфері розробки програмного забезпечення, докорінно змінивши спосіб написання, рецензування та підтримки коду [38]. Ці технології, керовані штучним інтелектом, здатні розуміти та генерувати тексти й код, подібно до людини, і вже зараз швидко трансформують традиційні методи програмної інженерії. Від автоматизації рутинних завдань до надання пропозицій у реальному часі чи пояснення складних кодових баз – GPT і LLM формують нову парадигму створення ПЗ. Цей зсув парадигми не лише обіцяє підвищення продуктивності та ефективності, але й ставить питання про майбутню роль людини-програміста та потенційну демократизацію процесу створення програмного забезпечення [39]. По мірі розвитку цих моделей їхня інтеграція в робочі процеси розробки відкриває безпрецедентні можливості, але водночас породжує і низку унікальних викликів. У рамках даного дослідження важливо провести аналіз багатовимірного впливу GPT та LLM на процес розробки ПЗ й трансформації коду, вивчення їхніх поточних застосувань, потенційних напрямів розвитку та наслідків для програмної індустрії в цілому.

Генеративні попередньо натреновані трансформери GPT та LLM є значним досягненням у сфері штучного інтелекту, особливо в галузі обробки та генерації природної мови. Це нейронні мережі, натреновані на величезних обсягах текстових даних, що дозволяє їм розуміти й створювати текст, подібний до людського, у широкому спектрі завдань і контекстів, у т.ч. й код.

В основі GPT та LLM лежить архітектура трансформера, яка використовує механізми самоуваги для обробки та генерації послідовних даних. Завдяки цьому моделі здатні враховувати довготривалі залежності в тексті, що робить їх особливо ефективними для завдань, де важливий широкий контекст. Технічна база цих моделей полягає у процесі навчання та

архітектурі. Вони зазвичай тренуються методом неконтрольованого навчання на великих корпусах текстів, що дозволяє виявляти закономірності та зв'язки без ручного маркування даних. Ключові аспекти трансформерів:

1. Архітектура трансформера – обробка послідовностей паралельно завдяки механізму самоуваги, що підвищує ефективність.
2. Масштабне попереднє навчання – моделі тренуються на мільярдах токенів, формуючи широке розуміння мови.
3. Донавчання (fine-tuning) – адаптація попередньо натренованих моделей до конкретних завдань за допомогою спеціалізованих даних.
4. Закони масштабування – продуктивність моделей передбачувано зростає зі збільшенням їхнього розміру та обсягів даних для навчання [40].

Сфера великих мовних моделей швидко розвивається, і серед найпомітніших прикладів:

1. OpenAI GPT (GPT 3, GPT 4), відомі своїми потужними можливостями розуміння та генерації тексту.
2. Google BERT і T5 – орієнтовані на двонаправлене розуміння та багатозадачне навчання.
3. Megatron-Turing NLG (Microsoft + Nvidia) – одна з найбільших мовних моделей, створених на сьогодні.
4. Facebook RoBERTa – оптимізована версія BERT із вдосконаленою методологією навчання.
5. Anthropic Constitutional AI – моделі, спрямовані на узгодження роботи AI з людськими цінностями.

LLM призвели революцію в автоматизації рутинних завдань програмування, значно підвищивши продуктивність розробників. Вони здатні генерувати шаблонний код, стандартні функції та реалізовувати поширені патерни проєктування з мінімальним втручанням людини. Наприклад, операції CRUD (створення, читання, оновлення, видалення), базові структури даних чи початкове налаштування проєкту тепер можуть бути автоматизовані. LLM значно прискорили створення прототипів і мінімально життєздатних

продуктів (MVP). Розробник може описати бажану функціональність природною мовою, а модель згенерує відповідний код. Це дозволяє швидко перевіряти ідеї та демонструвати їх замовникам [41].

GitHub Copilot, розроблений GitHub у співпраці з OpenAI, служить яскравим прикладом, як LLM трансформує генерацію коду. Це «AI-парний програміст», який пропонує фрагменти коду та цілі функції у реальному часі. Він був навчений на мільярдах рядків публічного коду, що дозволяє йому генерувати контекстуально релевантні пропозиції для різних мов програмування та фреймворків. Дослідження показало, що розробники з Copilot виконували завдання на 55% швидше, ніж без нього [42]. Інструмент продемонстрував особливу ефективність у допомозі розробникам у орієнтуванні в незнайомих бібліотеках або API, пропонуючи відповідні виклики методів та шаблони використання. Водночас, виникають питання щодо якості коду, безпеки та нових навичок, які потрібні програмістам (наприклад, prompt engineering). Узагальнені результати з урахуванням досліджень [43-44] показані в табл. 2.1.

Таблиця 2.1 – Вплив штучного інтелекту на етапи розробки ПЗ

Сфера застосування	Переваги	Виклики / ризики
Генерація коду	Автоматизація шаблонних завдань, швидке прототипування, пришвидшення роботи	Можливе надмірне покладання на AI, ризик втрати базових навичок
Налагодження та ревію	Виявлення багів і вразливостей, оптимізація продуктивності, пояснення складних кодових баз	Питання прозорості рішень AI, складність перевірки правильності
Продуктивність команд	Стандартизація практик, швидший онбординг, ефективніший код-ревію	Необхідність нових навичок (prompt engineering), зміна ролей у команді
Доступність для нефахівців	Зниження бар'єрів входу, можливість створювати прості застосунки без глибоких знань	Ризик поверхневого розуміння програмування, потреба контролю якості
Освіта та навички	Нові можливості навчання, інтеграція AI в освітні програми	Втрата акценту на фундаментальних алгоритмах і структурах даних
Майбутні перспективи	Повна підтримка full-stack, автоматизація тестування та управління проєктами	Етичні та правові питання: авторство, відповідальність, безпека даних

Хоча переваги AI в розробці ПЗ є значними, але одним із найреволюційніших аспектів LLM є поява інтерфейсів програмування природною мовою. Наприклад, користувач може ввести запит на кшталт «Створити вебсторінку, яка відображає список імен користувачів з бази даних», і AI генеруватиме необхідний код HTML, CSS та бекенд-код для виконання цього завдання. Цей підхід усуває розрив між людським мисленням та машинним виконанням, роблячи програмування більш інтуїтивно зрозумілим та доступним для ширшої аудиторії. Важливо зазначити, що хоча ці інтерфейси роблять основні завдання доступнішими, вони все ще вимагають розуміння користувачами фундаментальних концепцій програмування, щоб ефективно використовувати та змінювати згенерований код.

Зростає акцент на навчанні майбутніх фахівців тому, як формулювати проблеми та рішення таким чином, щоб ефективно використовувати можливості AI. Це включає такі навички, як швидка інженерія, де розробники вчаться створювати точні інструкції природною мовою, щоб отримати бажаний результат від моделей AI. Оскільки інструменти AI стають більш поширеними, зростає потреба в розробниках, які можуть створювати, підтримувати та вдосконалювати ці системи. Це призводить до зростаючого попиту на навички на перетині програмної інженерії та машинного навчання.

Однак, вони також викликають важливі етичні міркування. Однією з головних проблем є потенційне збереження упереджень, присутніх у навчальних даних. Якщо непідготовлені користувачі навчаються на репозиторіях коду, які містять упереджені або небезпечні практики кодування, вони можуть поширювати ці проблеми у своїх пропозиціях, потенційно вносячи вразливості або посилюючи проблемні шаблони кодування [45]. Ще одним етичним міркуванням є вплив на зайнятість та розвиток навичок у індустрії програмного забезпечення. Оскільки AI бере на себе більше рутинних завдань кодування, існує необхідність переосмислити роль розробників та забезпечити, щоб залежність від AI не призводила до втрати кваліфікації.

Як зазначають Хіндл та ін. у своєму дослідженні про природність ПЗ, існує сильна кореляція між природною мовою та мовою програмування [46]. Цей висновок підтверджує тенденцію до інтерфейсів програмування природною мовою та свідчить про те, що навички, необхідні для ефективної розробки програмного забезпечення, можуть розвиватися, включаючи глибше розуміння лінгвістичних шаблонів та структур (рис. 2.2).

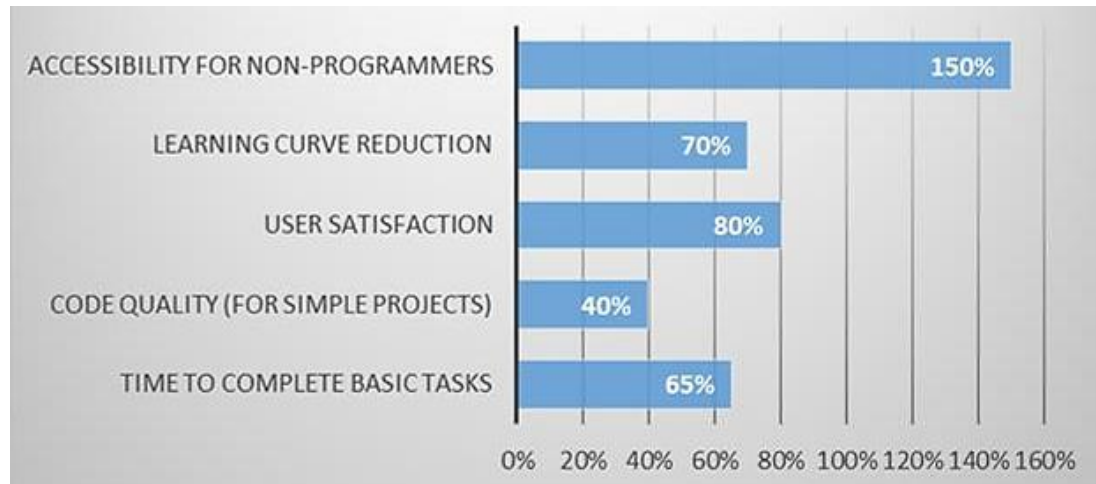


Рисунок 2.2 - Ефективність інтерфейсів програмування природною мовою [46]

Також виникають проблеми конфіденційності та інтелектуальної власності, особливо під час використання хмарних сервісів AI, які можуть мати доступ до власного коду. Команди розробників повинні ретельно враховувати питання захисту даних та відповідності під час інтеграції цих інструментів у свої робочі процеси. Нарешті, є питання підзвітності. Коли AI значно сприяє генерації коду, стає важливим встановити чіткі правила відповідальності у разі збоїв програмного забезпечення або порушень безпеки.

Отже, незважаючи на багатообіцяючі досягнення, для реалізації повного потенціалу AI в розробці програмного забезпечення необхідно вирішити кілька проблем та обмежень:

1. Оскільки системи AI стають складнішими, забезпечення прозорості та пояснень їхніх процесів прийняття рішень для розробників стає критично важливим, особливо в додатках, де підзвітність є обов'язковою.

2. Упередженість та справедливість: фахівці, навчені на існуючих базах коду, можуть ненавмисно підтримувати обмеження, присутні в цьому коді. Розробка методів для виявлення та пом'якшення цих упереджень буде постійною проблемою.

3. Проблеми безпеки та конфіденційності: системи AI отримують доступ до більш конфіденційних баз коду та даних проєктів, забезпечення безпеки та конфіденційності цієї інформації ставатиме дедалі важливішим.

4. Надмірна залежність та атрофія навичок: існує ризик того, що надмірна залежність від інструментів AI може призвести до зниження фундаментальних навичок програмування серед розробників. Досягнення балансу між допомогою AI та розвитком людських навичок буде вирішальним.

Відсоток узагальнених результатів опитування думки фахівців наведено на рис. 2.3 за даними [45-47].

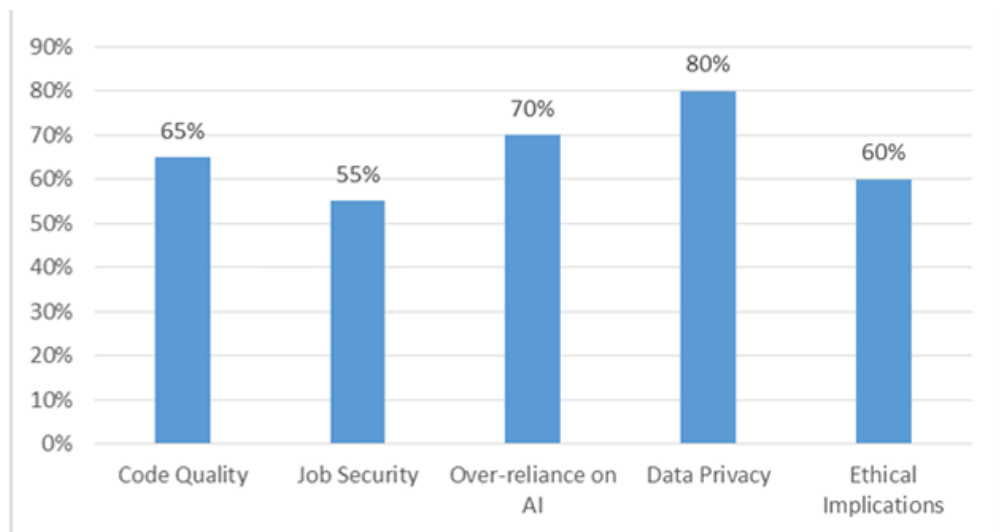


Рисунок 2.3 – Рівень стурбованості з головних питань щодо впливу поширення AI в розробці програмного забезпечення

Як видно з рис. 2.3, думки програмістів розподілені з невеликими коливаннями серед найбільш часто виявлених напрямків і стосуються якості коду (65%), перенавчання AI (70%), збереження приватності даних (80%). Однак, збереження робочих місць хвилює лише 55% опитаних професіоналів.

2.3 Основні напрямки застосування AI для рефакторингу коду

Рефакторинг та оптимізація коду були традиційними проблемами в програмній інженерії, фундаментальною метою яких було створення програмного забезпечення з вищою ефективністю та легшою модифікацією. У минулому всі ці завдання виконувалися вручну, з метою розробника переглянути код та розпізнати закономірності того, як і де його може знадобитися оптимізувати. Аналізатори вихідного коду, які, як впливає з їхньої назви, сканують дерева вихідного коду, не виконуючи його фактично, були одними з перших, хто підтримував цей процес, виявляючи такі проблеми, як мертвий код, невикористані змінні та основні порушення стандартів кодування. Статичний аналіз, з іншого боку, означав, що код виконувався. Це давало розробникам можливість помітити проблеми під час роботи, наприклад, витoki пам'яті та вузькі місця [48].

AI пропонує функції, що перевершують традиційні підходи: застосування машинного навчання для прогнозування продуктивності, підхід навчання з підкріпленням для стимулювання оптимізації коду або алгоритми NLP для розбору коду та розуміння його синтаксичного та семантичного контекстів. Зокрема, AI може використовуватися для пришвидшення виявлення неефективності та мінімізації реструктуризації коду та покращення здатності читати код з мінімальною ручною роботою.

Крім того, адаптації штучного інтелекту можуть легко бути включені в поточні процеси розробки для підвищення ефективності без тригодинних марафонів та зниження якості. Ці досягнення показують важливість AI для підтримки ефективних практик розробки програмного забезпечення у великих масштабах у дедалі складніших обставинах [49]. Значна частина досліджень сьогодні полягає у визначенні того, як можна використовувати методи AI для покращення, а також ефективної реархітектури коду, з акцентом на машинне навчання; навчання з підкріпленням; та НЛП у використанні коду для покращення.

У процесах оптимізації коду та рефакторингу набирають популярності підходи AI для аналізу та прогнозування продуктивності коду, виявлення шаблонів, що вказують на неефективні частини, та пропонування модифікації коду, щоб зробити його більш читабельним та зручним у підтримці. Наприклад, моделі, такі як алгоритми машинного навчання, можна навчати на даних, щоб прогнозувати ті сегменти коду, які можуть призвести до проблем із продуктивністю; з іншого боку, моделі навчання можуть розуміти складні структури коду, що допомагає у великомасштабному рефакторингу.

Таким чином, процес еволюції від простої автоматизації до складних інструментів на основі штучного інтелекту є новим кроком у зростаючому вдосконаленні життєвого циклу розробки програмного забезпечення, що робить управління кодом більш точним та продуктивним.

Існує кілька фреймворків, які працюють із AI у сфері аналізу та рефакторингу коду, і всі вони сприяють розвитку дисципліни. «Code BERT» – це модель обробки природної мови, навчена на великих кодових даних; вона може виявляти, коли створюється акреційний код, і надавати пропозиції щодо реорганізації, оскільки розуміє семантику коду.

Ще одним відомим фреймворком є LLVM (Low-Level Virtual Machine), який застосовується для оптимізації коду на рівні компілятора, щоб забезпечити інструменти, що трансформують код з метою оптимізації продуктивності виконання незалежно від мови програмування.

Модель трансформатора під назвою CodeT5, використовується для завдань, пов'язаних з кодом, таких як генерація та удосконалення, і допомагає розробникам генерувати пропозиції щодо коду в режимі реального часу. Ці та інші інструменти втілюють перетин AI та програмної інженерії, оскільки все більше і більше моделей машинного навчання або агентів навчання з підкріпленням та NLP-технологій використовуються для оптимізації коду та підтримки керованості кодових баз. У табл. 2.2 наведено інструменти на основі AI, необхідні в цій галузі, з описом їхніх основних функцій та сфер використання.

Таблиця 2.2 – Огляд інструментів та фреймворків на основі AI

Інструмент/ Фреймворк	Основна функція	Переваги	Застосування
BERT	Розуміння коду за допомогою NLP-моделі	Висока точність у синтаксичному аналізі коду	Аналіз коду
LLVM	Виявлення надлишковості, допомога у рефакторингу	Компільований фреймворк з точним аналізом семантики	Оптимізація коду
Модель Transformer	Покращення продуктивності виконання	Гнучкість у мовах, широке застосування	Генерація / уточнення коду
Модель Transformer (в реальному часі)	Пропозиції коду в реальному часі	Ефективність у доповненні та вдосконаленні коду	Автозавершення та уточнення коду

Автоматизовані системи рецензування інженерного коду широко використовують штучний інтелект для реалізації важливого процесу розробки – рецензії коду. Такі системи створюються з метою виявлення антипатернів, недбалого стилю кодування та проблемної архітектури шляхом порівняння коду з набором стандартів і найкращих практик [50].

Антипатерни – це поширені шаблони кодування, які на перший погляд здаються ефективними, але з часом призводять до зниження продуктивності, поганої масштабованості та високих витрат на підтримку. Автоматизовані системи рецензії коду навчаються на різноманітних репозиторіях, щоб розпізнавати такі помилкові конструкції, як:

- «God Objects» (клас, що має занадто багато відповідальностей);
- «Copy-Paste Code» (копійовані фрагменти коду);
- недостатня обробка помилок тощо.

Системи рецензії на основі AI аналізують ці антипатерни, допомагаючи розробникам писати чистіший і оптимізований код.

Окрім виявлення проблем з кодом, інструменти автоматизації на основі AI здатні виявляти й архітектурні недоліки – наприклад, неправильні інтерфейси модулів або надмірно зв'язані багаторазові компоненти. Такі системи аналізують структуру коду, щоб надати поради щодо покращення

модульності та масштабованості – двох критично важливих аспектів підтримки та розвитку застосунків.

Наприклад, великі технологічні компанії, такі як Google і Microsoft, розробили централізовані додатки на основі AI для рецензії коду, які можуть консультувати розробників у режимі реального часу. Також сервіси DeepCode і Codacy реалізують рецензію коду з використанням AI, здатну виявляти численні помилки ще на етапі розробки. Ці інструменти допомагають оптимізувати процеси рецензування коду, покращують зворотний зв'язок, якість та скорочують час, витрачений розробниками на перевірки. Розглянемо найбільш традиційні прийоми рефакторингу коду.

Движки рекомендацій для рефакторингу - автоматизовані інструменти рецензії коду сканують написаний код і пропонують способи його покращення, підвищення читабельності та підтримуваності.

Аналіз і рекомендації. Хоча це поки не є масовим стандартом, вже існують рефакторингові системи, що базуються на технологіях AI і здатні виявляти частини коду, які можна спростити, виділити в окремі методи або зробити легшими для читання та підтримки. Наприклад, якщо функція стала занадто великою чи заплутаною, система може запропонувати розділити її на менші, спеціалізовані частини. Такі системи також виявляють «code smells» (ознаки потреби в рефакторингу), як-от повторювані шаблони коду, надто тісно пов'язані фрагменти, велика кількість параметрів тощо.

Сценарії використання. Під час роботи рефакторингових систем моделі машинного навчання використовуються для виявлення ефективних структур коду та надання найкращих варіантів змін. Наприклад, при оцінці змін у кодї з часом модель може визначити, які зміни покращують читабельність або знижують складність. IntelliJ IDEA від JetBrains містить інтегроване рішення для рефакторингу, яке використовує AI для надання підказок і виправлень. Також Eclipse IDE уже підтримує підказки з рефакторингу на основі AI, що покращують загальну структуру коду. шляхом вибору коду, щоб витягти логіку; усунути дублікати; або ввести модулюючі інтерфейси. Аналіз реальних

випадків використання обох інструментів показав, що ці інструменти допомагають зменшити занепокоєння щодо технічного боргу та покращити процес рефакторингу, щоб підвищити проблеми з підтримкою програмного забезпечення. Наприклад, IntelliJ IDEA може надавати опції вибору коду, такі як видалення логіки, усунення дублікатів або впровадження модулюючих інтерфейсів. Аналіз реальних випадків використання обох інструментів показав, що ці інструменти допомагають зменшити занепокоєння щодо технічного боргу та покращити процес рефакторингу, щоб підвищити рівень підтримки програмного забезпечення.

Розуміння та реструктуризація складних залежностей є доволі новим напрямком рефакторингу. Складні залежності є важливими для завдань на кшталт аналізу ієрархії викликів або потоку даних у програмі, тому GNN відіграють ключову роль. Вони здатні аналізувати код з точки зору його залежностей та пропонувати можливості для рефакторингу, наприклад, зменшення зв'язності (coupling) і підвищення ступеня інкапсуляції.

Підсумовуючи досягнення, не можна не відмітити наявність проблем, які потребують вирішення. Перша група стосується можливостей отримання маркованих (розмічених) даних для навчання моделей AI, здатних виявляти неефективність, дублювання чи структурні проблеми.

На сьогодні більшість доступних наборів даних існують лише для найпоширеніших мов, таких як Python, Java чи JavaScript. Проте, ці мови охоплюють лише частину всіх програмних завдань. Моделі AI, натреновані переважно на цих мовах, не здатні узагальнювати результати на менш розповсюджені або спеціалізовані мови, як-от Rust, Go, мови для вбудованих систем чи фінансових застосунків. Це обмежує можливість застосування таких моделей у нових сферах.

Така проблема, як розмітка коду, наприклад, для виявлення поганих циклів, використання «мертвих» змінних або антипатернів дизайну, потребує глибокої технічної експертизи, і її складно автоматизувати. На відміну від сфер, де можна розмічати зображення або тексти (наприклад, позначати

об'єкти на фото або встановлювати зв'язки між фразами), у програмному кодї не завжди очевидно, які фрагменти є проблемними, особливо без спеціалізованих знань.

Щоб впоратися з браком даних, деякі вчені застосовують напівавторське навчання, що означає, що моделі навчаються як на позначених, так і на немаркованих даних, мінімізуючи залежність від великих обсягів позначених даних (табл. 2.3). Третій підхід передбачає отримання позначених даних від розробників з більшим досвідом праці на платформах з відкритим кодом; це означає, що можна опрацьовувати багато різних стилів кодування та мов.

Таблиця 2.3 – Проблеми з даними в оптимізації коду штучного інтелекту

Виклик даних	Вплив	Потенційне рішення
Обмежена різноманітність мов	Моделі важко адаптуються до різних мов і доменів	Розширення джерел даних, включення різних open-source проєктів
Складність розмітки даних	Високі витрати, повільний процес розмітки	Використання напівнаглядного навчання, краудсорсинг для розмічених даних
Нестача розмічених прикладів	Знижує точність навчання	Застосування трансферного навчання та методів адаптації домену

Такі підходи, як адаптація до предметної області, можуть бути використані для точного налаштування моделей для нових розроблених репозиторії коду, уникаючи при цьому перенавчання. Крім того, ідея агностичних моделей архітектури навчання, тобто моделей, які налаштовані не на мову, що використовується, а на «структуру» коду, може підвищити узагальнюваність моделей. Існує також розуміння навчання (мета-навчання), яке передбачає, що моделі повинні бути здатні вивчати нові завдання в новій області якомога швидше з мінімальними даними.

Проведення власних експериментів дозволяє здійснити додаткову перевірку окремих тверджень та мати план власних дій щодо застосування AI в рефакторингу коду на прикладі вебдодатків.

Висновки до розділу 2

AI-підтримуване генерування та оптимізація коду трансформують веброзробку на платформі .NET, спрощують процеси кодування, покращують продуктивність програмного забезпечення та підвищують рівень безпеки. Інструменти, керовані AI, такі як GitHub Copilot, IntelliCode і Performance Profiler у Visual Studio, забезпечують рекомендації в реальному часі, виявляють неефективності та автоматизують налагодження, істотно скорочують час розробки. Аналіз безпеки на основі AI допомагає виявляти вразливості та забезпечувати створення надійних і захищених вебзастосунків.

Дослідження застосування різних моделей AI виявило низку обмежень, які включали труднощі з документаційними рядками, які описують довгі ланцюжки операцій, і з прив'язкою операцій до змінних. Ідея генерації складних кодів показала перспективність і потребує подальших досліджень.

Інтеграція GPT та LLM у розробку ПЗ розглядається багатьма як парадигмальний зсув. Вони підвищують продуктивність, демократизують створення програм, але водночас ставлять нові етичні та освітні виклики.

AI бере на себе рутинні завдання – шаблонний код, CRUD-операції, автодоповнення функцій. Розробники можуть описати задачу природною мовою, а модель генерує робочий код. Це пришвидшує створення MVP та перевірку ідей. Інтеграція з IDE перетворює їх на «AI-парних програмістів», що підвищує командну продуктивність і стандартизує практики.

Зростає роль prompt engineering, але важливо не втратити фундаментальні знання алгоритмів і структур даних. очікується повна підтримка full-stack, автоматизація тестування та управління проектами, а також перенос знань між різними сферами.

Серед значних викликів: прозорість, безпека, уникнення надмірної залежності від AI, питання авторства та відповідальності. Майбутнє розробки визначатиме симбіоз людини та AI, де технології допомагають, але критичне мислення й творчість залишаються за людьми.

РОЗДІЛ 3

РЕЗУЛЬТАТИ ЗАСТОСУВАННЯ ОКРЕМИХ МОВНИХ МОДЕЛЕЙ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ОПТИМІЗАЦІЇ КОДУ ВЕБДОДАТКУ

3.1 Використання LLM GPT-4o для розпізнання та рефакторингу JS коду

Кінцевим продуктом при веброзробці, з яким працюють як розробники, так і різноманітні аналітичні додатки, є програмний код, який написаний на обраній мові програмування. У практичній частині представлені результати проведеного експерименту, основною метою якого було дослідити можливості популярних великих мовних моделей LLM AI та перевірити гіпотези про можливість розпізнання, коригування й оптимізації програмного коду, а також можливість створення абсолютно оригінального коду із заданими параметрами як продукту штучного інтелекту.

Для тестування обрані закриті LLM (доступ через інтернет) – GPT4o, Gemini, а також локальні відкриті (open source) LLM, що встановлені на платформу LM Studio. Для експерименту був використаний js-файл з пакету завершеного вебдодатку, який розміщено на хості, без вказання URL-адреси. Файл містив 148 рядків, у тому числі коментарі щодо призначення окремих функцій та методів обробки даних. Фрагмент коду представлено на рис. 3.1.

```

8  /*****/ // Check if module is in cache
9  /*****/ if(installedModules[moduleId]) {
10 /*****/   return installedModules[moduleId].exports;
11 /*****/ }
12 /*****/ // Create a new module (and put it into the cache)
13 /*****/ var module = installedModules[moduleId] = {
14 /*****/   i: moduleId,
15 /*****/   l: false,
16 /*****/   exports: {}
17 /*****/ };
18 /*****/
19 /*****/ // Execute the module function
20 /*****/ modules[moduleId].call(module.exports, module, module.exports, __webpack_require__)
21 /*****/

```

Рисунок 3.1 – Фрагмент коду скрипта на мові JS з працюючого вебдодатку

Вебдодаток складався з класичного набору файлів, створених за технологіями HTML&CSS, а також мовою програмування JavaScript на фреймворку React. Інтерфейс користувача передбачав наявність окремих динамічних візуальних ефектів, наприклад, слайдер, масштабування зображень, внесення та елементарну обробку даних, а також забезпечення функціональності та адаптивності елементів сайту, наприклад, гамбургер-меню та інше.

Зазначений файл оброблявся в GPT-4o [51] від компанії OpenAI [52] з метою розпізнання та подальшої оптимізації. Код завантажувався безпосередньо в чат (ChatGPT) й аналізувався шляхом ведення діалогу, в ході якого використовувався сформований промпт-запитів, наприклад:

1. Що це за код?
 2. Цей код можна оптимізувати без втрати функціоналу?
 3. Напиши власний код замість цього, щоб виконувався той же самий функціонал.
 4. У твоєму коді є помилки?
 5. Питання щодо довжини коду: твій коротший за початковий?
- У ході експерименту отримані такі результати спілкування з GPT-4o.

Відповідь 1. Цей код є частиною проекту, який використовує Webpack для збирання модулів JavaScript. Він містить кілька скриптів для вебсторінки, включаючи функціонал для анімації елементів, роботи з якірними посиланнями, управління бургер-меню та надсилання форми. Давайте розглянемо основні компоненти:

1. Webpack Bootstrap: функція `_webpack_require_` використовується для завантаження модулів; кешування модулів: запобігає повторному завантаженню модулів, які вже були завантажені.
2. Модуль `anchors.js` – обробляє кліки на якірні посилання та плавно прокручує сторінку до відповідного елемента.
3. Модуль `animate.js` – додає клас `_active` до анімованих елементів при прокручуванні сторінки, щоб запускати анімацію.

4. Модуль `burger-menu.js` – керує відображенням бургер-меню та основного екрана, включаючи зміну `overflow` для блокування/розблокування прокручування сторінки.

5. Модуль `successful.js` – керує поведінкою форми та відображенням повідомлення про успішне відправлення, включаючи приховування форми та скидання значень полів.

6. Цей код використовується для створення інтерактивного та анімованого інтерфейсу користувача на вебсторінці.

На рис. 3.2 наведено фрагмент висхідного коду, який описує бургер-меню, що точно був розпізнаний AI.

```

3  const burgerTrigger = document.querySelector('.burger'),
4      burgerMenu = document.querySelector('.burger-menu'),
5      mainScreen = document.querySelector('.main-screen'),
6      header = document.querySelector('.header'),
7      burgerMenuLink = document.querySelectorAll('.burger-menu__link');
8
9  function hide () {
10     mainScreen.classList.toggle('hide');
11     header.classList.toggle('hide');
12 }
13
14 burgerTrigger.addEventListener('click', () => {
15     burgerTrigger.classList.toggle('active');
16
17     if (burgerTrigger.classList.contains('active')) {
18         document.body.style.overflow = "hidden";
19     } else {
20         document.body.style.overflow = "";
21     }
22     hide();
23
24 });
25
26 for (let item of burgerMenuLink) {
27     item.addEventListener('click', (e) => {
28         if(item) {
29             burgerTrigger.classList.toggle('active');
30             document.body.style.overflow = "";
31             hide();
32         }
33     });
34 }

```

Рисунок 3.2 – Висхідний код бургер-меню головної сторінки вебдодатку

Відповідь 2. Оптимізація коду без втрати функціоналу може включати покращення продуктивності та читабельності. Ось деякі можливі покращення для представленого коду:

1. Використання `forEach` замість `for` для перебору елементів: `forEach` часто читається легше і більш декларативно.
2. Видалення непотрібних перевірок та операцій, наприклад, перевірка наявності елемента всередині циклу може бути виконана один раз до циклу.
3. Об'єднання загальних операцій на функції: створення допоміжних функцій для завдань, що повторюються.
4. Усунення дублювання коду: якщо та сама дія виконується у кількох місцях, його можна винести на окрему функцію.

Під час відповіді запропонований код (рис. 3.3), який можна порівняти з початковим (див. рис. 3.2).

```
// burger-menu.js
const burgerTrigger = document.querySelector('.burger'),
      burgerMenuLinks = document.querySelectorAll('.burger-menu__link');

function toggleBurgerMenu() {
  document.body.style.overflow = burgerTrigger.classList.contains('active') ? "hidden" : "";
  document.querySelector('.main-screen').classList.toggle('hide');
  document.querySelector('.header').classList.toggle('hide');
}

burgerTrigger.addEventListener('click', () => {
  burgerTrigger.classList.toggle('active');
  toggleBurgerMenu();
});

burgerMenuLinks.forEach(link => {
  link.addEventListener('click', () => {
    burgerTrigger.classList.toggle('active');
    document.body.style.overflow = "";
    toggleBurgerMenu();
  });
});
```

Рисунок 3.3 – Код бургер-меню, який був оптимізований GPT-4o

Основні зміни, які виявлені при порівнянні:

1. Використання `forEach` для перебору елементів: покращує читабельність і зручність коду.

2. Об'єднання загальних операцій: функція `toggleBurgerMenu` використовується для зміни стану бургер-меню.

3. Видалення надлишкових перевірок: спрощення умов більшої ясності.

Проведений рефакторинг покращує читабельність, підтримуваність та підвищує продуктивність за рахунок зменшення дублювання коду та більш ефективного використання вбудованих методів.

Відповідь 3: Ось оптимізована версія вашого коду, яка виконує той самий функціонал (видано код, але не повний).

Відповідь 4 (про помилки): Мій попередній код не містить явних помилок, але є кілька уточнень та покращень, які можна внести для підвищення надійності та читання. Ось виправлений та доповнений варіант коду (рис. 3.4).

```
// burger-menu.js
const burgerTrigger = document.querySelector('.burger');
const burgerMenuLinks = document.querySelectorAll('.burger-menu__link');
const mainScreen = document.querySelector('.main-screen');
const header = document.querySelector('.header');

function toggleBurgerMenu() {
  document.body.style.overflow = burgerTrigger.classList.contains('active') ? "hidden" : "";
  mainScreen.classList.toggle('hide');
  header.classList.toggle('hide');
}

burgerTrigger.addEventListener('click', () => {
  burgerTrigger.classList.toggle('active');
  toggleBurgerMenu();
});

burgerMenuLinks.forEach(link => {
  link.addEventListener('click', () => {
    burgerTrigger.classList.toggle('active');
    document.body.style.overflow = "";
    toggleBurgerMenu();
  });
});
```

Рисунок 3.4 – Доповнений варіант коду після перевірки самим GPT-4o

Відповідь 5: Мій оптимізований код має ту ж функціональність, але за кількістю рядків він майже такий самий, як і вихідний. Основне покращення пов'язане з читабельністю та структуруванням коду. Оптимізація не завжди

означає скорочення кількості рядків, а скоріше покращення якості коду та його підтримки.

Таким чином, було проведено рефакторинг фрагменту коду на JavaScript, отримано покращену версію коду зі змінами, наприклад, вилучення зайвих перевірок та операцій, поєднання схожих операцій у функції, усунення дублювання коду та інші. Як результат, підвищено читабельність коду та його продуктивність, швидкість завантаження сторінки. Використання AI виправдане з технічної сторони.

3.2 Використання мовних моделей AI Gemini та Codestral

Для розширення меж експерименту аналогічне дослідження відбувалось з Gemini від компанії Google [53], який можна назвати відповіддю на створення ChatGPT. Серед широкого спектру можливостей, пов'язаних із обробкою, генерацією текстів будь-якого змісту і складності, перекладу з різних мов та інших завдань, розробники декларують вміння писати код на мовах програмування Python, JavaScript, Java, C++, C#, PHP, Ruby, Swift. Діалог відбувався так само, як і в попередньому варіанті. Далі наведено зміст діалогу в чаті, де відбувалося завантаження коду та «обговорення» із AI.

Питання 1: «Ти можеш проаналізувати код JavaScript?»

Відповідь 1: «Так, я можу проаналізувати код JavaScript декількома способами:

1. Статичний аналіз:

– Лістинг: можу перевірити ваш код на наявність поширених помилок та стилістичних проблем за допомогою таких інструментів, як ESLint та JSHint.

– Типова перевірка: можу перевірити типи даних ваших змінних та функцій, щоб переконатися, що вони використовуються правильно за допомогою таких інструментів, як TypeScript.

- Аналіз залежностей: можу проаналізувати ваш код, щоб побачити, які модулі він використовує та як вони пов'язані між собою.

- Покриття коду: можу підрахувати, скільки рядків коду виконується вашими тестами.

2. Динамічний аналіз:

- Налаштування: можу допомогти вам налагоджувати JavaScript, встановлюючи точки зупинки, переглядаючи значення змінних і виконуючи код по крокам.

- Профіль: можу допомогти вам визначити, які частини вашого коду виконуються найдовше, щоб ви могли їх оптимізувати.

- Аналіз продуктивності: можу допомогти вам піддаватися вашому JavaScript-коду, який виконується в термінах використання пам'яті, CPU usage, і network requests.

Крім того, я можу:

- Пояснити код: можу надати докладні пояснення того, що робить ваш код.

- Запропонувати рефакторинг: можу запропонувати способи покращення вашого коду, зробивши його більш читаним, підтримуваним та ефективним.

- Згенерувати код: можу генерувати JavaScript code, що базується на ваших поданнях або examples.

Надалі було введено той самий код (див. рис. 3.1-3.2) для аналізу в Chat Gemini.

Завдання 1: «Проаналізуй код з погляду проблем ефективності та потенційних покращень».

Відповідь:

Наданий код є пакетом веб-пакета, що містить кілька файлів JavaScript з проекту. Він не написаний на чистому JavaScript, але використовує збирач, такий як webpack, для керування залежностями та їхнього об'єднання. Далі слідує опис розпізнання коду та можливості його оптимізації. AI визначає

Webpack Bootstrap як «систему модулів, яку використовує веб-пакет для керування залежностями між різними файлами JavaScript у проєкті».

Окремі файли JavaScript були правильно класифіковані: кожен із перерахованих файлів (`anchors.js`, `animate.js`, `burger-menu.js`, `Success.js`), ймовірно, буде містити функціональні можливості для певних частин веб-сайту, такі як плавне прокручування для прив'язок, анімація при прокручуванні, взаємодія з бургер-меню, та обробка успішного відправлення форми.

Завдання 2: «Вдосконалъ вихідний код».

Відповідь 2: «Оскільки я не маю доступу до окремих файлів JavaScript (`anchors.js`, `animate.js`, `burger-menu.js`, `successful.js`), я не можу безпосередньо удосконалити наданий код. Однак, я можу надати загальні рекомендації щодо рефакторингу коду JavaScript».

Загальні методи оптимізації, які запропоновані Gemini:

1. Зменшення: зменшіть код, щоб видалити непотрібні пробіли, коментарі та форматування. Це може значно зменшити розмір пачки та скоротити час завантаження.

2. Розділення коду: розділіть великі пакети на менші блоки залежно від використання. Це може покращити початковий час завантаження сторінки та зменшити вплив окремих сценаріїв на продуктивність.

3. Літинг: використовуйте лінтер, як-от ESLint, для виявлення та виправлення потенційних помилок, забезпечення узгодженості стилю коду та покращення читабельності.

4. Маніпуляції DOM: мінімізуйте маніпуляції DOM за допомогою ефективних методів, таких як пакетне оновлення та віртуальні бібліотеки DOM.

5. Прослуховувачі подій: видаляйте прослуховувачі подій, коли вони більше не потрібні, щоб запобігти витокам пам'яті та підвищити продуктивність.

6. Сторонні бібліотеки: вибирайте добре оптимізовані сторонні бібліотеки та уникайте непотрібних залежностей.

7. Профілювання продуктивності: використовуйте інструменти профілювання, щоб виявити вузькі місця продуктивності та оптимізувати відповідно.

Окрім загальних, Gemini видав конкретні поради для удосконалення кожного сценарію.

Для `anchors.js`:

- оптимізуйте поведінку плавного прокручування, щоб мінімізувати вплив на продуктивність;
- розгляньте можливість використання бібліотеки, як-от `smooth-scroll`, для ефективного прокручування.

Для `animate.js`:

- використовуйте `IntersectionObserver` API, щоб спостерігати за елементами, що входять у вікно перегляду, і запускати анімацію лише за необхідності;
- оптимізуйте обчислення анімації та використовуйте апаратне прискорення, якщо воно доступне.

Для `burger-menu.js`:

- використовуйте переходи або анімацію CSS для плавного відкриття/закриття меню замість того, щоб покладатися виключно на JavaScript;
- оптимізуйте обробку подій, щоб мінімізувати вплив на продуктивність.

Додаткові поради:

- використовуйте AJAX для надсилання форм асинхронно без оновлення сторінки;
- використовуйте переходи або анімацію CSS для повідомлення про успішне надсилання;

- використовуйте CDN (Мережа доставки вмісту) для обслуговування файлів JavaScript із різних місць, зменшуючи затримку для користувачів у всьому світі;
- увімкніть кешування файлів JavaScript, щоб зменшити кількість повторних завантажень;
- використовуйте інструмент моніторингу продуктивності, щоб відстежувати продуктивність вебсайту та визначати області, які потрібно покращити з часом.

Резюме від AI: дотримуючись цих вказівок і оптимізуючи кожен сценарій окремо, ви можете значно покращити продуктивність і загальну взаємодію з користувачем вашого вебдодатку.

Враховуючі зростання інтересу до локальних версій LLM, аналогічні дослідження проводились з Codestral-22B-v0.1-GGUF [54]. На відміну від розглянутих варіантів, зазначений підхід має низку переваг, що пов'язано з кількома чинниками: конфіденційність даних; низька затримка та висока доступність; налаштування та адаптація під конкретні завдання; економічна ефективність; незалежність від зовнішніх послуг; підтримка локальних мов та особливостей. Для цього використовувалась платформа LM Studio [55]. Це інструмент із відкритим вихідним кодом, який оптимізує процес навчання, оцінки та використання сучасних мовних моделей локально. Як і GPT-4o, Codestral-22B-v0.1-GGUF видала прийнятний результат.

3.3 Оцінювання ефективності розробки вебдодатку за участю AI

В процесі підготовки кваліфікаційної роботи була проведена серія експериментів з рефакторингу програмного коду вебдодатків, а також протестовані моделі щодо написання ними програмного коду. Напрямок досліджень формує нові ідеї, однак, на основі отриманих результатів можна висунути припущення, що для різних випадків використання AI може бути

економічно обґрунтованою альтернативою для створення та рефакторингу коду вебдодатків на етапах завершення, тестування порівняно з послугами сторонніх комерційних компаній, які проводять спеціальну оптимізацію вебдодатків.

За даними різних джерел, наприклад [57], ціна на розробку вебдодатку для середнього бізнесу стартує від 35,62 тис. грн (850\$ за курсом НБУ станом на 20.10.2025). При цьому розробка займає від 1 до 3 місяців. Розробка вебдодатку включає комплекс операцій та забезпечення. Формула для розрахунку сумарної вартості вебдодатку в загальному вигляді:

$$S_w = \sum_{k=1}^n w_k t, \quad (3.1)$$

де S_w – сумарна вартість робіт;

w_k – вартість окремого виду роботи за одиницю часу;

t – тривалість робіт;

n – кількість видів робіт.

Тривалість і вартість передбачених робіт і витрат наведена в табл. 3.1 на основі даних, взятих з актуальних інтернет-джерел. Загальна вартість розробки вебдодатку для середнього бізнесу розрахована за формулою (3.1).

Таблиця 3.1 – Розцінки на окремі види робіт при розробленні вебдодатку

Вид роботи/витрати	Вартість за од. часу, грн	Затрати часу, одиниць	Сумарні витрати, грн
Доменне ім'я [57]	615 грн/рік	1 рік	615
Хостинг	1200 грн/рік	1 рік	1200
CMS для сайту (безкоштовні версії)	0 грн/рік	1 рік	0
Технічне завдання (одноразово)	4950 грн	По факту	4950
Дизайн сайту (одноразово)	12000 грн	По факту	12000
Верстання сайту (за 5 сторінок)	9500 грн	По факту	9500
Програмування сайту	650 грн/год	40 год	26000
Наповнення сайту	350 грн/год	10 год	3500
Тестування*			7100
Сумарна вартість вебдодатку для бізнесу під ключ із хостингом			64865

Час та вартість на тестування вебдодатку перед хостингом знаходиться в межах 10-20 % часу, відведеного на верстку і програмування [57].

Використання AI для генерації коду вебдодатків стає все більш затребуваним завдяки здатності оптимізувати процеси розробки. Основні економічні переваги застосування AI в розробці вебдодатків можна розглянути в кількох ключових аспектах.

1. Зниження витрат на розробку: AI здатний автоматизувати процес написання коду, що зменшує обсяг ручної праці та скорочує час, необхідний для виконання рутинних завдань. Це дозволяє компаніям економити на оплаті праці за тривалістю часу робіт. З огляду на проведену оцінку, економічний ефект на розробці може скласти від 20% на програмуванні, тобто порядку 12970 грн.

2. Підвищення продуктивності та швидкості розробки. Інструменти AI дозволяють розробникам працювати швидше, оскільки вони можуть генерувати готові фрагменти коду на основі коротких запитів або описів функцій. Наприклад, AI може генерувати HTML, CSS або JavaScript код для типових компонентів, таких як кнопки, форми чи інтерактивні елементи, що суттєво зменшує робоче навантаження. Це означає, що підвищується швидкість розробки, яка особливо критична в галузях, де нові продукти і функції повинні швидко відповідати на зміни в потребах користувачів та ринковим тенденціям.

3. Оптимізація якості коду: AI може використовуватися для перевірки коду на помилки, дотримання стандартів або оптимізацію продуктивності, що знижує витрати на тестування та покращує якість кінцевого продукту. Наприклад, інструменти AI можуть автоматично визначати можливі помилки, пропонувати виправлення або покращення, що зменшує ризики, пов'язані з багами або низькою продуктивністю веб-сторінок. Це особливо важливо в умовах, де стабільність та швидкість роботи вебресурсу впливають на конверсію та утримання клієнтів. Оптимізація займає від 10 % від вартості розробки, або 6500 грн у нашому розрахунку.

Таким чином, навіть часткове залучення AI дозволяє отримати економію коштів $S = 12973 + 6500 = 19460$ грн.

По відношенню до вартості цілого проєкту економія складе:

$$19460 \div 64865 \times 100 = 30 \%$$

Окрім фінансової вигоди розглядаються також і організаційні переваги.

4. Економія часу на навчання та адаптацію нових співробітників: використання AI для генерації коду дозволяє менше покладатися на досвідчених програмістів для виконання базових завдань. Компанії можуть швидше інтегрувати нових співробітників у робочий процес. Однак, необхідно враховувати зміну вимог до компетентностей програміста в частині промпт-інжинірингу.

5. Інновації та конкурентна перевага: використання AI дозволяє компаніям тестувати нові ідеї без значних інвестицій в ресурси, а також бути більш гнучкими до змін. AI може швидко адаптувати існуючий код або генерувати нові ідеї на основі поточних трендів, що допомагає компаніям залишатися на передових позиціях на ринку. Це особливо важливо в висококонкурентних галузях, таких як електронна комерція та онлайн-сервіси, де постійні інновації є запорукою успіху.

6. Ефективне використання хмарних технологій та масштабованість: інтеграція AI для генерації коду в хмарних платформах дозволяє компаніям автоматизувати процес розробки у великому масштабі. Наприклад, для стартапів або невеликих компаній AI може забезпечити доступ до високотехнологічних рішень без необхідності великих інвестицій в інфраструктуру або кадрові ресурси.

Таким чином, застосування штучного інтелекту при генерації коду для вебдодатків може значно підвищити ефективність розробки, знизити витрати, прискорити вихід на ринок та покращити якість продукту. В умовах високої конкуренції та швидких технологічних змін ці економічні переваги стають вирішальними для компаній, які прагнуть досягти успіху в цифровому світі.

Висновки до розділу 3

У результаті роботи над останнім розділом були описані результати експерименту з використання мовних моделей AI для оптимізації і написання коду згідно зразка або визначеного завдання. Отримані результати з досліджених мовних моделей AI показують здатність до точного розпізнання коду, написаного на JavaScript, особливостей функціоналу, а також внесення пропозицій щодо оптимізації, рефакторингу коду. Всі моделі працювали через завантаження досліджуваного коду безпосередньо в чат.

Найбільш розлогі рекомендації щодо оптимізації й рефакторингу видала система Gemini. Разом із тим, при спілкуванні з Gemini не вдалось отримати генерацію коду, тобто все звелось до узагальнених положень. Швидко генерацію нового коду за сформульованим завданням надав GPT 4o.

Проведено економічний аналіз, порахована орієнтовна вартість реалізації розробки вебдодатку для середнього бізнесу. За оцінками вартість такого проєкту 64865 грн. Застосування штучного інтелекту при генерації коду для вебсторінок може значно підвищити ефективність розробки, знизити витрати (за оцінками від 19460 грн або 23,08 %), прискорити вихід на ринок та покращити якість продукту.

Результати дослідження пройшли апробацію на двох міжнародних конференціях і опубліковані у вигляді тез доповідей на міжнародних конференціях. Копії оригіналів текстів публікацій представлено в додатку Б.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проаналізовано значну кількість різноманітного інструментарію веброзробника, який впливає на якість та швидкість розробки вебдодатків. Підсумовуючи результати проведеного дослідження, яке стосувалося пошуку можливих шляхів удосконалення процесів розробки або рефакторингу програмних кодів, які постають перед вебпрограмістами при розробці вебдодатків, зокрема на JavaScript, можна сформулювати наступні висновки.

1. Огляд та узагальнення рейтингів популярності інструментарію веброзробників дозволяють стверджувати, що серед мов програмування JavaScript s TypeScript є лідерами вже не один рік, так само як Angular, React Vue.js - серед фреймворків та бібліотек.

2. AI-підтримуване генерування та рефакторинг коду трансформують веброзробку на платформі .NET, спрощуючи процеси кодування, покращуючи продуктивність програмного забезпечення та підвищуючи рівень безпеки.

3. Інтеграція GPT та LLM у розробку ПЗ розглядається багатьма як парадигмальний зсув. Вони підвищують продуктивність, демократизують створення програм, але водночас ставлять нові етичні та освітні виклики.

4. Результати з досліджених LLM показують здатність до точного розпізнання коду, написаного на JavaScript, особливостей функціоналу, а також внесення пропозицій щодо оптимізації, рефакторингу коду. Найбільш розлогі рекомендації щодо покращення отримано від Gemini. Швидку генерацію нового коду надав GPT-4o.

5. Проведено економічний аналіз, порахована орієнтовна вартість реалізації розробки вебдодатку для середнього бізнесу. На основі оцінок визначено, що застосування штучного інтелекту при генерації коду для вебсторінок може значно підвищити ефективність розробки, знизити витрати (за оцінками від 12700 грн або 23,08 %), прискорити вихід на ринок та покращити якість продукту.

6. AI бере на себе рутинні завдання – шаблонний код, CRUD-операції, автодоповнення функцій. Ідея генерації складних кодів за допомогою AI показала безумовну перспективність і потребує подальших досліджень, враховуючи низку обмежень, які включали труднощі з документаційними рядками, які описують довгі ланцюжки операцій, і з прив'язкою операцій до змінних.

Запропонована методика досліджень може розширити спектр завдань і прикладів для підготовки вебпрограмістів у навчальному процесі та формування актуальних компетентностей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A. K. Goel, R. Bakshi, K. K. Agrawal. Web 3.0 and Decentralized Applications. *Proc. 2022 the 2nd Int. Conf. on Innovative Research in Renewable Energy Technologies*. 2022. P. 1–8.
2. Гнатишин М. Аналіз сучасних тенденцій розвитку технологій веброзробки. *Матеріали VI Міжнародної студентської науково-технічної конференції «Природничі та гуманітарні науки. Актуальні питання»* (м. Тернопіль, 27-28 квітня 2023 року). Тернопіль: ТНТУ ім. І. Пулюя, 2023. С. 132-133.
3. Історія розвитку вебсайтів. URL: <https://webcraft.by/blog/istoria-razvitiya-saitov.html> (дата звернення: 20.11.2025).
4. What is HTML Preprocessor? *GeeksforGeek*. URL: <https://cutt.ly/BwT5iubA> (дата звернення: 20.11.2025).
5. The Top Programming Languages. *IEEE Spectrum*. 2024. URL: <https://spectrum.ieee.org/top-programming-languages-2024> (дата звернення: 20.11.2025).
6. Chuck Musciano, Bill Kennedy. HTML & XHTML: The Defenitive Guide. 6-th Edition. O'Reilly, 2006. 680 p.
7. About W3C. *W3C: вебсайт*. URL: <https://www.w3.org/Consortium/> (дата звернення 03.09.2025)
8. HTML довідка. URL: <http://xn--80adth0aefm3i.xn--j1amh/HTML/> (дата звернення: 20.10.2025).
9. DuRocher D. HTML & CSS QuickStart Guide: The Simplified Beginners Guide to Developing. ClydeBank Media LLC, 2021. 361p. ISBN 9781636100005.
10. Mike McGrath. HTML, CSS & JavaScript in easy steps. In Easy Steps Limited, 2020. ISBN 9781840789218. Retrieved from <https://www.perlego.com/book/3016276/html-css-javascript-in-easy-steps-pdf> (дата звернення: 20.10.2025).

11. W3C study of practices and tooling for Web data standardisation. URL: <https://www.w3.org/2017/12/odi-study/> (дата звернення: 20.11.2025).

12. HTML Стили CSS. URL: https://w3schoolsua.github.io/html/html_css.html (дата звернення: 20.11.2025).

13. Користувацькі сценарії: що це, як і навіщо їх потрібно будувати. *Сценарії взаємодії користувача з промо-сайтом* вебсайт URL : <https://laba.ua/blog/2651-polzovatelskie-scenarii-hto-eto-kak-i-dlya-chego-ih-nuzhno-stroit> (дата звернення: 20.11.2025).

14. Након Wium Lie, Bert Bos. Cascading Style Sheets: Designing for the Web. 3rd Edition. R.R. Doneley and Sons Company in Crowfordswill, Indiana, US. 2005. 416 p. ISBN-13: 978-0-321-19312-4.

15. CSS фреймоврки. URL: <https://html-plus.in.ua/css-frameworks/> (дата звернення: 20.11.2025).

16. W3Techs. URL: <https://w3techs.com/technologies/details/cp-javascript> (дата звернення: 20.11.2025).

17. 10 JavaScript фреймворків URL: <https://uaspectr.com/2021/03/30/10-javascript-frejmvorkiv-2021/> (дата звернення: 20.11.2025).

18. Usage statistics of JavaScript libraries for websites. W3Techs: *вебсайт*. URL: https://w3techs.com/technologies/overview/javascript_library (дата звернення: 20.11.2025).

19. JavaScript фреймворки. *Reclamare:* вебсайт. URL: <https://www.reclamare.ua/blog/javascript-frejmvorki/> (дата звернення: 20.11.2025).

20. Бекенд сайту. URL: <https://kitweb.pro/bekend-sayta/> (дата звернення 03.05.2025).

21. TypeScript vs JavaScript: Know The Difference. URL: <https://radixweb.com/blog/typescript-vs-javascript#vs>

22. Різниця між фреймворком та бібліотекою. URL: <https://difference.in.ua/riznicya-mizh-frejmvorkom-ta-bibliotekoju/>. (дата звернення: 20.11.2025).

23. Developer Survey: Web frameworks. Stackoverflow: вебсайт. URL: <https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks> (дата звернення: 23.11.2025).
24. Framework vs Library: Full Comparison. *InterviewBit*. URL: <https://cutt.ly/ywT5tTHe> (дата звернення: 20.11.2025).
25. Web frameworks and technologies. *Stackoverflow*: вебсайт. URL: <https://survey.stackoverflow.co/2023#most-popular-technologies-webframe-prof> (дата звернення: 20.11.2025).
26. Most Popular Backend Frameworks – 2012/2023. *Statistics & Data*. URL: <https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2023/> (дата звернення: 20.11.2025).
27. Sandeep Konakanchi. 2025. “Artificial Intelligence in Code Optimization and Refactoring”. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*. 11 (2): 1197-1211. <https://doi.org/10.32628/CSEIT25112463>.
28. S. Chinamanagonda. AI-driven Performance Testing AI tools enhancing the accuracy and efficiency of performance testing. *Advances in Computer Sciences*. 2021. Vol. 4. No. 1.
29. Yella, A., & Kondam, A. *Integrating AI with Big Data: Strategies for Optimizing Data-Driven Insights*. *Innovative Engineering Sciences Journal*, 2023, vol. 9, no. 1.
30. Javaid, H. A. *Revolutionizing AML: How AI is leading the Charge in Detection and Prevention*. *Journal of Innovative Technologies*, 2024, vol. 7, no. 1.
31. Naseer, I. *Machine Learning Algorithms for Predicting and Mitigating DDoS Attacks*. *International Journal of Intelligent Systems and Applications in Engineering*, 2024, vol. 12, no. 22s, p. 4.
32. Dhoni, P., Chirra, D., & Sarker, I. *Integrating Generative AI and Cybersecurity: The Contributions of Generative AI Entities, Companies, Agencies, and Government in Strengthening Cybersecurity*. [s.l.], [s.n.], [n.d.].

33. Azmat, H., & Huma, Z. *Analog Computing for Energy-Efficient Machine Learning Systems*. Aitoz Multidisciplinary Review, 2024, vol. 3, no. 1, pp. 33–39.
34. Aditya S Shethiya. AI-Assisted Code Generation and Optimization in .NET Web Development. *Annal. App. Sci*, 2025. Vol 6(1). P.1-7.
35. Floridi, L. *AI as Agency without Intelligence: On ChatGPT, Large Language Models, and Other Generative Models*. *Philosophy & Technology*, 2023, vol. 36, no. 1, p. 15.
36. Goriparthi, R. G. *AI and Machine Learning Approaches to Autonomous Vehicle Route Optimization*. *International Journal of Machine Learning Research in Cybersecurity and Artificial Intelligence*, 2021, vol. 12, no. 1, pp. 455–479.
37. Damaraju, A. *The Role of AI in Detecting and Responding to Phishing Attacks*. *Revista Espanola de Documentacion Cientifica*, 2022, vol. 16, no. 4, pp. 146–179.
38. A. Vaswani et al., "Attention is All You Need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998-6008. Available: <https://papers.nips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>
39. M. Chen et al. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*, 2021. Available: <https://arxiv.org/abs/2107.03374>
40. J. Kaplan et al. Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361*, 2020. <https://arxiv.org/abs/2001.08361>.
41. M. Allamanis, et al. A Survey of Machine Learning for Big Code and Naturalness. *ACM Computing Surveys (CSUR)*. Vol. 51. Issue. 4. Article No.: 81. pp. 1-37, 2018. <https://doi.org/10.1145/321269>.
42. N. Ziegler, et al. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. *arXiv preprint arXiv:2302.06590*, 2023. <https://arxiv.org/abs/2302.06590>.
43. M. Tufano, et al. An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation. *ACM Transactions on Software*

Engineering and Methodology. Vol. 28. No. 4. P. 1-29. 2019.
<https://doi.org/10.1145/3340544>.

44. A. Svyatkovskiy, et al. IntelliCode Compose: Code Generation Using Transformer. *ESEC/FSE 2020: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2020. P. 1433-1443.
<https://doi.org/10.1145/3368089.341705>.

45. N. Papernot, P. McDaniel, A. Sinha and M. P. Wellman. SoK: Security and Privacy in Machine Learning. *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, London, UK. 2018. pp. 399-414. doi: 10.1109/EuroSP.2018.00035.

46. Harman, M., McMinn, P., de Souza, J.T., Yoo, S. Search Based Software Engineering: Techniques, Taxonomy, Tutorial. In: Meyer, B., Nordio, M. (eds) *Empirical Software Engineering and Verification*. LASER LASER LASER 2010 2009 2008. Lecture Notes in Computer Science. 2012. Vol.7007. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-25231-0_1

47. S. Amershi, et al. Software Engineering for Machine Learning: A Case Study. In *IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, Montreal, QC, Canada, 2019. pp. 291-300. doi: 10.1109/ICSE-SEIP.2019.00042.

48. Sandeep Konakanchi. 2025. “Artificial Intelligence in Code Optimization and Refactoring”. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*. 11 (2): 1197-1211.
<https://doi.org/10.32628/CSEIT25112463>

49. Rasmita Panigrahi, Sanjay K. Kuanar, Lov Kumar, and Neelamadhab Padhy. A systematic approach for software refactoring based on class and method level for AI application. *International Journal of Powertrains*. 2021. Vol. 10. No. 2. p. 143-174. <https://doi.org/10.1504/IJPT.2021.117461>.

50. A. Almogahed, M. Omar, N. H. Zakaria, G. Muhammad and S. A. AlQahtani . Revisiting scenarios of using refactoring techniques to improve software

systems quality. *IEEE Access*. 2023. Vol. 11. pp.28800-28819. doi:10.1109/ACCESS.2022.3218007.

51. GPT-4o. URL: <https://chatgpt.com> (дата звернення: 19.11.2025).

52. OpenAI. URL: <https://openai.com/index/gpt-4-research/> (дата звернення: 19.11.2025).

53. Gemini. URL: <https://gemini.google.com/app?hl=ru> (дата звернення: 19.11.2025).

54. Codestral-22B-v0.1-GGUF. URL: <https://huggingface.co/bartowski/Codestral-22B-v0.1-GGUF> (дата звернення: 23.11.2025).

55. LM Studio. URL: <https://lmstudio.ai> (дата звернення: 20.11.2025).

56. Ціни на створення сайту 2025. *STUDIO IFISH: вебсайт*. URL: <https://ifish.com.ua/ua/tsini-na-sajt/> (дата звернення: 23.11.2025).

57. Домени COM.UA. *Імена.Ua: вебсайт*. URL: <https://www.imena.ua/ua/domains/zones/com.ua> (дата звернення: 23.11.2025).