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: **«Оптимізація коду вебдодатків з використанням мовних моделей штучного інтелекту»**

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
Кабак Данило Ігоревич
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

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Кабака Данила Ігоревича

1. Тема кваліфікаційної роботи:
«Оптимізація коду вебдодатків з використанням мовних моделей штучного інтелекту»,
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: наукові публікації, дані інтернет-ресурсів, LLM CodeOpt, DeepCode, CodeOpt, CodeGuru, вебдодатки, побудовані на HTML, CSS і JavaScript, ChatGPT, GitHub Copilot

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

Розділ 1 Теоретичні основи принципів функціонування і застосування штучного інтелекту в розв’язанні інтелектуальних завдань діяльності людини
Розділ 2. Підготовка експериментальних досліджень із залученням мовних моделей штучного інтелекту для оптимізації програмного коду вебдодатків
Розділ 3. Результати випробування відомих мовних моделей штучного інтелекту для оптимізації коду вебдодатків

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 проєктно-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 **Данило КАБАК**

Керівник роботи **Олена КОПШИНСЬКА**

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

КАБАК ДАНИЛО ІГОРЕВИЧ

«ОПТИМІЗАЦІЯ КОДУ ВЕБДОДАТКІВ З ВИКОРИСТАННЯМ МОВНИХ МОДЕЛЕЙ ШТУЧНОГО ІНТЕЛЕКТУ»

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 (71 найменування), 3 додатків. Кваліфікаційна робота містить 8 таблиць, 18 рисунків, викладена на 65 сторінках.

Основний зміст роботи

У першому розділі «Теоретичні основи принципів функціонування і застосування штучного інтелекту в розв'язанні інтелектуальних завдань діяльності людини» представлено огляд наукових джерел та розкрито сутність понять машинне навчання, глибоке машинне навчання, мовні моделі штучного інтелекту, а також стану їх застосування в області програмної інженерії.

У другому розділі «Підготовка експериментальних досліджень із залученням мовних моделей штучного інтелекту для оптимізації програмного коду вебдодатків» узагальнено результати експериментальної перевірки ефективності окремих моделей ШІ для генерації й оптимізації коду, описано структуру і технології розробки додатків, використаних для власних досліджень.

У третьому розділі «Результати випробування відомих мовних моделей штучного інтелекту для оптимізації коду вебдодатків» описані результати експерименту при використанні ChatGPT і Copilot в ролі AI-асистента при розпізнанні вебкоду і його оптимізації. Здійснене оцінювання економічної та технологічної ефективності розробки вебдодатку за участю штучного інтелекту.

Висновки

1. Значна кількість публікацій у наукових базах даних підтверджує зростання інтересу до тематики ШІ, але перспективи їх використання в реальних умовах залишаються неоднозначними.

2. Конвергенція штучного інтелекту та розробки програмного забезпечення має потенціал для значного скорочення необхідних ресурсів, покращення якості та покращення взаємодії з користувачем за допомогою більш інтелектуальних програм, орієнтованих на користувача.

3. Провівши різнопланові дослідження варіантів залучення великих мовних моделей штучного інтелекту на прикладі ChatGPT та Copilot у процесі удосконалення і оптимізації кодів вебсайтів, отримали низку позитивних результатів. Обидві залучені в ході експерименту моделі можуть бути ефективними для програмістів у написанні, завершенні, поясненні та рефакторингу коду.

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.

5. З технічної точки зору були визначені або підтверджені аналогічні результати інших досліджень, що Copilot більш органічно працює для доповнення та завершення коду, надає підказки в IDE, завдяки інтегрованості в середовища розробки. Це потужний фактор, який дозволяє уникнути простих помилок на етапах написання коду. Обидві моделі є ефективними для аналізу готового коду і формування пропозицій щодо SEO-оптимізації та технічної оптимізації.

6. Проведено економічний аналіз, порахована орієнтовна вартість реалізації розробки вебдодатку для середнього бізнесу. На основі оцінок визначено, що застосування штучного інтелекту при оптимізаці й частковій генерації коду для вебсторінок може значно підвищити ефективність розробки, знизити витрати (за оцінками від 7360 грн або 30 %), прискорити вихід на ринок та покращити якість продукту.

Подальшими напрямками досліджень можуть бути цікавими експерименти з кодами, в яких застосований більш складний набір технологій, інші мови програмування, а також генерація графічних зображень для графічного дизайну.

Список публікацій здобувача

1. 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. Ополе, Польща. [електронне видання] URL: https://www.economy-confer.com.ua/full-article/6477/

2. Кабак Д. І. Особливості реалізації проєктної діяльності в ТОВ «Інфосвіт ІТ Сервіс» при впровадженні ПК «Універсал». *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. 60-62.

АНОТАЦІЯ

Кабак Д. І. «Оптимізація коду вебдодатків з використанням мовних моделей штучного інтелекту». 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 та перспективних напрямків у використанні мовних моделей штучного інтелекту в області програмної інженерії. Представлено оригінальні результати експериментального дослідження методів взаємодії великих мовних моделей штучного інтелекту для оптимізації програмних кодів вебдодатків або ж безпосереднього створення таких.

Ключові слова: JavaScript, мовні моделі штучного інтелекту LLM, оптимізація коду, GPT 4o, ChatGPT, GitHub Copilot

ANNOTATION

Kabak D. I. "Optimization of web application code using language models of artificial intelligence". Qualification work in the form of a manuscript.

Qualification work for the degree of Master of higher education Master in the educational and professional program Information Management Systems and Technologies, specialty 126 Information Systems and Technologies. Poltava State Agrarian University, Poltava, 2025.

The work contains a deep theoretical analysis of the main achievements and promising directions in the use of language models of artificial intelligence in the field of software engineering. The original results of an experimental study of methods for the interaction of large language models of artificial intelligence for optimizing the program codes of web applications or directly creating them are presented.

Keywords: JavaScript, language models of artificial intelligence LLM, code refactoring, GPT 4o, ChatGPT, GitHub Copilot

ЗМІСТ

ВСТУП

Актуальність теми кваліфікаційної роботи пов’язана з питаннями оптимізації та раціоналізації програмного коду вебдодатків різної складності та призначення із залученням мовних моделей штучного інтелекту. Галузь розробки вебдодатків невпинно вдосконалює інструментарій та підходи до роботи з мовами програмування, мовами розмітки та структуризації коду, завдяки появі таких засобів, як фреймворки й бібліотеки, інтегровані середовища розробки та інші засоби. Все більша кількість професійних розробників прагне автоматизувати роботу з кодом, у тому числі пошук і виправлення помилок, вразливостей, використання готового коду, доставку коду DevOps та інше. Поява штучного інтелекту, а саме мовних моделей, великих мовних моделей, які працюють з неструктурованими даними, відкриває нові можливості в програмній інженерії. Кожний новий метод потребує всебічного дослідження, проведення Strengths-Weaknesses аналізу (сильні-слабкі сторони), тому перспективність обраної теми в області оптимізації коду вебдодатків та можливості залучення штучного інтелекту не викликає заперечень.

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 та експериментальне дослідження можливостей сучасних версій великих мовних моделей штучного інтелекту та ефективності їх використання для технічної оптимізації коду вебдодатків на етапах розробки та верифікації. Для досягнення мети були визначенні та реалізовані такі наукові *завдання*:

– дослідити тенденції розвитку мовних моделей штучного інтелекту;

– провести різнопланове дослідження стану застосування штучного інтелекту в різних напрямках програмної інженерії;

– структурувати перелік вимог щодо оптимальності та функціональності написання кодів у вебдодатках;

– провести експериментальне дослідження за допомогою різних моделей штучного інтелекту з оптимізації програмного коду, написаного на мові JavaScript, який є частиною вебдодатку;

– узагальнити результати та розробити корисні рекомендації щодо використання можливостей великих мовних моделей штучного інтелекту для удосконалення й оптимізації програмного коду вебдодатку, а також сформувати перспективні напрямки досліджень в обраній області.

Об'єктом дослідження кваліфікаційної роботи є заходи технічної та семантичної оптимізації програмного коду вебдодатків із використанням популярних мовних моделей штучного інтелекту.

Предметом дослідження є: порівняльні властивості й технічні аспекти застосування найбільш відомих мовних моделей штучного інтелекту щодо удосконалення, оптимізації програмного коду вебдодатків, а також методології промпт інжинірингу в процесі роботи зі штучним інтелектом.

Методи наукових досліджень: абстрактно–логічний, інформаційно–пошуковий, аналітичний, Strengths-Weaknesses аналіз, економіко-статистичний, графічний, методи усереднених статистичних оцінок, мовні моделі штучного інтелекту.

Інформаційну базу кваліфікаційної роботи складають: наукові статті, широкий огляд тематичної літератури, статистичні дані аналітичних компаній, які розміщені на вебсайтах у вільному доступі, власні спостереження та фахові наукові дослідження з обраної проблеми, також відкриті моделі штучного інтелекту різних розробників, що є вільно доступні в мережі інтернет.

Елементи наукової новизни полягають у здійсненні дослідницько-експериментального аналізу можливостей штучного інтелекту для оптимізації

та підвищення функціональності коду вебдодатків, тестуванні спроможностей моделей штучного інтелекту до розпізнання й оптимізації програмного коду та узагальненні доказової бази ефективності кожного з них.

Практична значущість одержаних результатів: результати роботи являють собою оригінальну працю з можливістю практичного використання та сформованими напрямками подальших наукових досліджень. Застосування результатів кваліфікаційної роботи дає змогу розширити можливості технологій розробки вебдодатків та їхнього вдосконалення на етапах тестування, оптимізації й реалізації. Залучення до експерименту моделей штучного інтелекту дозволяє прогнозувати в недалекому майбутньому перехід на абсолютно нові методи роботи з програмним кодом, надання розробникам нових знань та перспектив.

Апробація результатів дослідження відбувалася шляхом оприлюднення доповідей на міжнародних наукових конференціях: із публікацією результатів у вигляді тез у збірниках матеріалів міжнародної конференції та студентської науково-практичної конференції за результатами виробничої практики.

Публікації. За результатами роботи опубліковано тези доповідей на конференціях: «Можливості оптимізації та рефакторингу коду вебдодатків на основі великих мовних моделей штучного інтелекту», матеріали Міжнародної мультидисциплінарної наукової інтернет-конференції «Світ наукових досліджень. Випуск 45», 21-22 жовтня 2025 року, м. Ополе, Польща; та «Особливості реалізації проєктної діяльності в ТОВ «Інфосвіт ІТ Сервіс» при впровадженні ПК «Універсал», Матеріали науково-практичної конференції за підсумками проходження виробничих практик здобувачів вищої освіти спеціальності 126 Інформаційні системи та технології, кафедра інформаційних систем та технологій ПДАУ, 22 жовтня 2025 р. Вип. XI, м. Полтава.

Структура та обсяг кваліфікаційної роботи. Пояснювальна записка кваліфікаційної роботи складається зі вступу, 3 розділів, висновків, списку використаних джерел (71 найменування), додатків. Кваліфікаційна робота містить 8 таблиць, 18 рисунків, викладена на 65 сторінках.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ПРИНЦИПІВ ФУНКЦІОНУВАННЯ І ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ В РОЗВ'ЯЗАННІ ІНТЕЛЕКТУАЛЬНИХ ЗАВДАНЬ ДІЯЛЬНОСТІ ЛЮДИНИ

1.1 Штучний інтелект, машинне навчання, глибоке навчання та когнітивні обчислення: пояснення термінів

Термін «штучний інтелект» був уперше введений у 1950-х роках і спочатку означав просту ідею прояву машинами інтелекту, подібного до людського [1]. На сучасному етапі штучний інтелект (ШІ) є широкою галуззю в дисципліні комп'ютерних наук, що зосереджується на створенні та розвитку комп'ютерних систем і пов'язаних технологій, які мають здатність виконувати завдання, що традиційно потребують людського інтелекту [2]. Такі завдання охоплюють різноманітні когнітивні здібності, зокрема логічне мислення, навчання з досвіду, розв'язання проблем, розуміння природної мови, сприйняття та здатність взаємодіяти з навколишнім середовищем і змінювати його [3].

Як академічна дисципліна ШІ також виник у 1950-х роках, коли дослідники прагнули створити роботів, здатних мислити, розв'язувати проблеми та навчатися самостійно [4]. Галузь залишалася малопопулярною до 2000-х років, переживаючи повторювані цикли оптимізму, за якими слідували періоди розчарування та скорочення фінансування, відомі як «зима ШІ» [5].

Запровадження алгоритмів, заснованих на даних, та методів машинного навчання (machine learning, ML) докорінно змінили цю сферу, надавши їй унікальну можливість навчатися на історичній інформації для покращення розв'язання проблем. Фінансування та інтерес різко зросли після 2012 р., коли методи глибокого навчання продемонстрували значні можливості навчання й прогнозування, перевершивши майже всі попередні підходи ШІ, а також після 2017 р. з появою архітектури трансформерів [6]. На початок 2020-х років у

США відбувся стрімкий розвиток ШІ, завдяки великим технологічним компаніям, університетам і лабораторіям, що сприяло значним досягненням у цій сфері.

У сучасну епоху стрімкого технологічного розвитку та експоненційного зростання надзвичайно великих наборів даних («великих даних») ШІ перейшов від простої теорії до реального застосування в безпрецедентних масштабах [8]. Від аналізу величезних масивів даних у майже реальному часі, автономних автомобілів та рекомендацій відео, сформованих на основі історії переглядів (Netflix, Лос-Ґатос, Каліфорнія, США), до онлайн-рекомендацій покупок, реклами та виявлення шахрайства (Amazon, Сіетл, Вашингтон, США), ШІ став фундаментально інтегрованим у багато сфер суспільства та часто працює непомітно у фоновому режимі наших персональних електронних пристроїв.

ШІ можна поділити на два типи: вузький або слабкий ШІ та загальний або сильний ШІ [9]. Окрім того, розглядають моделі ML як підгалузь ШІ, глибокі нейронні мережі, глибоке навчання, також інші моделі ШІ. Дамо коротку характеристику кожному з них для розуміння сутності і відмінностей.

Слабкий ШІ створюється спеціально для виконання конкретної діяльності в заданих межах (тобто він спеціалізований і має обмежений спектр застосування). Він не має широких когнітивних можливостей, притаманних людському інтелекту. Переваги слабкого ШІ полягають у його практичності в різних галузях: він забезпечує автоматизацію, оптимізацію та вдосконалення певних процесів. Такі системи широко застосовують у сфері охорони здоров’я, фінансах, виробництві й обслуговуванні клієнтів для оптимізації процесів, підвищення якості прийняття рішень і покращення взаємодії з користувачами. Водночас, слабкий ШІ має значні обмеження. Такі системи мають слабке розуміння за межами свого заданого спектра, що ускладнює їхню роботу з непередбачуваними даними або ситуаціями. Існують занепокоєння щодо конфіденційності даних, упередженості та етичних наслідків, коли такі системи працюють із чутливою інформацією та впливають на процеси

прийняття рішень. Сильний ШІ прагне відтворювати людський інтелект у широкому спектрі та володіти здатністю розуміти, набувати знання і застосовувати їх для виконання різноманітних завдань, як і людина [10-11].

Машинне навчання, яке вважають підгалуззю ШІ, демонструє властивість «навчання» з досвіду, притаманну людському інтелекту, а також здатність навчатися й удосконалювати свої аналітичні можливості завдяки обчислювальним алгоритмам [12]. Ці алгоритми використовують великі набори вхідних і вихідних даних, щоб розпізнавати закономірності та фактично «вчитися», тренуючи машину робити автономні рекомендації або приймати рішення. Після достатньої кількості повторень і змін алгоритму машина здатна отримувати вхідні дані та прогнозувати вихідний результат [12]. Потім ці результати порівнюють із відомими правильними виходами, щоб оцінити точність алгоритму, після чого він багаторазово вдосконалюється з метою покращення здатності передбачати подальші результати [13].

Штучний інтелект та машинне навчання – це дві пов’язані, але окремі теми в інформатиці. Штучний інтелект прагне відтворити людський інтелект у машинах, тоді як машинне навчання зосереджується на створенні алгоритмів, які дозволяють комп’ютерам навчатися на основі даних (рис. 1.1).

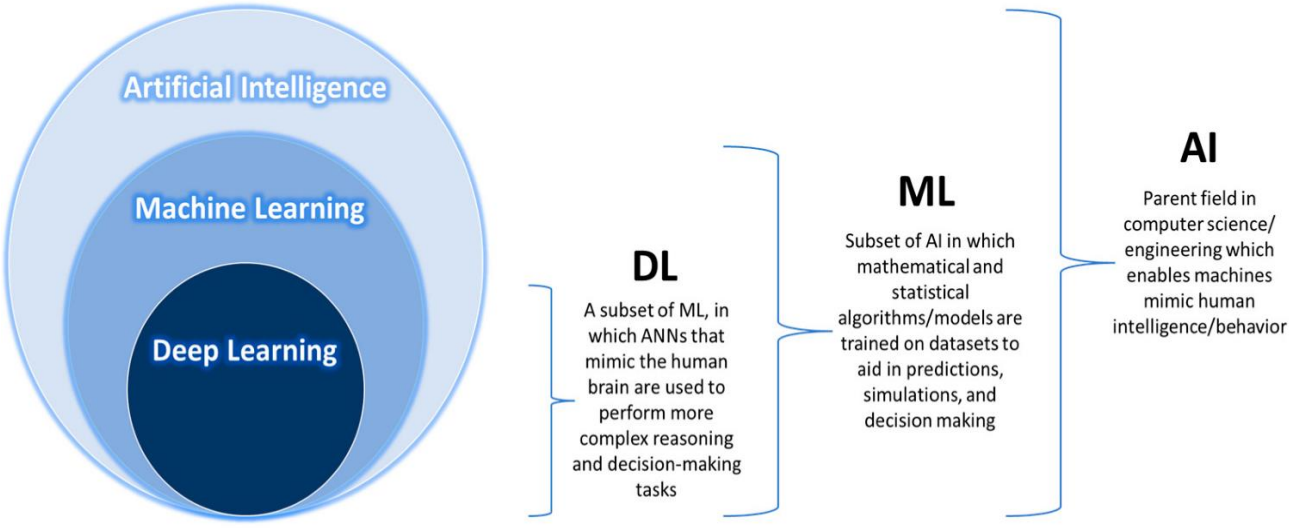

Рисунок 1.1 – Схема ієрархій: штучний інтелект (AI) проти машинного навчання (ML) та глибокого навчання (DL) [14]

Як видно зі схеми (див. рис. 1.1), ШІ є батьківською сферою в комп'ютерних науках та інженерії, у той час, як машинне навчання є нащадком ШІ на основі математичних та статистичних алгоритмів, але батьківською технологією для моделей глибокого навчання (Deep Learning, DL).

Глибокі нейронні мережі (DNN) є складнішими версіями моделей ML, використовують ієрархічні рівні для поділу та обробки результату (рис. 1.2).

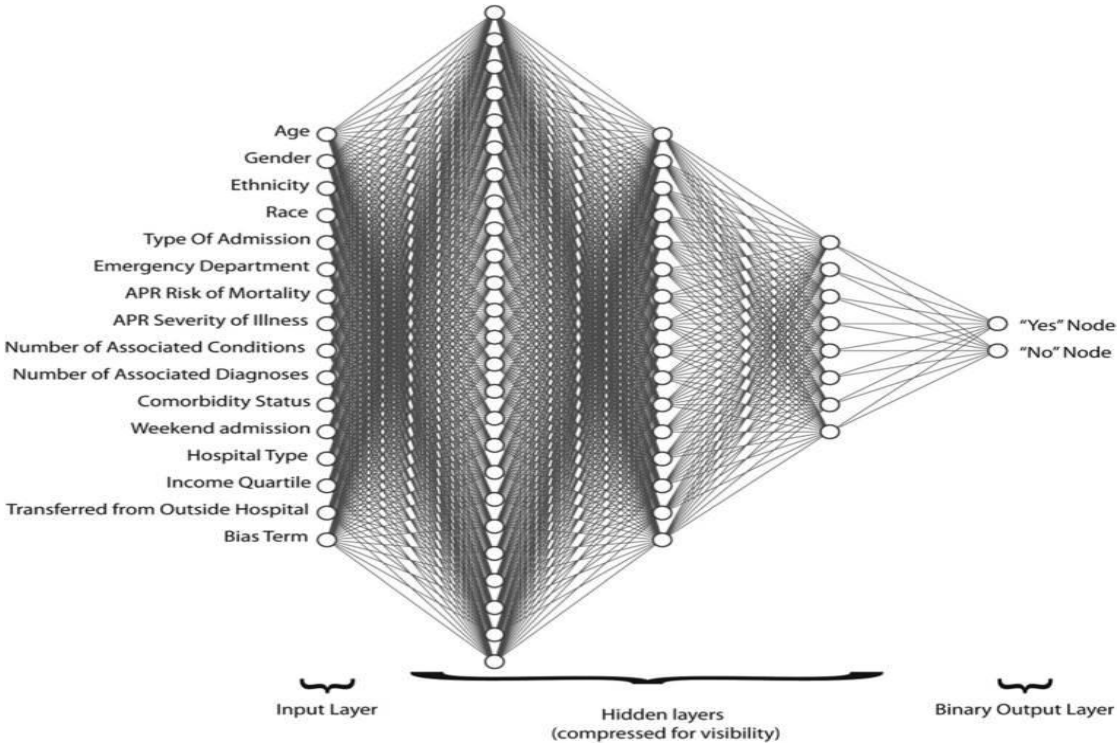

Рисунок 1.2 – Приклад вхідного, прихованого та вихідного шарів DNN [15]

Мережа починається з вхідного рівня, який переходить до певної кількості «прихованих рівнів», де кожний реагує на різні характеристики вхідних даних (див. рис.1.2) [8]. Ці рівні забезпечують підвищення розуміння, коли дані проходять «глибше», що дозволяє створювати моделі без явно запрограмованих інструкцій. Коли машина вивчає певну концепцію на багатьох рівнях, алгоритм може поступово вдосконалюватися в міру надходження нових даних. Подібно до того, як працює людський мозок, машина здатна формувати «нейронні» зв'язки з «дендритичних» зв'язків на різних ієрархічних рівнях даних. Таким чином ці мережі дали поштовх новому напрямку ШІ - «глибокому навчанню» [13].

1.2 Методологічні засади та технічні основи великих мовних моделей ШІ

Традиційні підходи ШІ, хоча й ефективні в багатьох ситуаціях, часто покладаються на структуровані дані та потребують значної роботи з побудови ознак і спеціальних галузевих знань для вирішення конкретних проблем. Хоча такі методи й потужні, вони обмежені потребою у заздалегідь визначених алгоритмах і структурованих наборах даних, які не завжди можуть повністю відобразити складність і тонкощі реальних умов.

Саме тут великі мовні моделі (Large Language Models, LLM) пропонують революційний прорив. На відміну від традиційного ШІ, LLM створені для обробки та розуміння неструктурованих даних, таких як природна мова, зображення та навіть складні сенсорні дані, способом, більш подібним до людського сприйняття [16]. Ця можливість дає змогу LLM працювати з величезною та різноманітною інформацією, характерною для сучасних транспортних систем, роблячи їх більш адаптивними та чутливими до динамічного характеру транспортних викликів.

Для розуміння технологічних аспектів застосування для прикладних завдань необхідно зробити короткий огляд методологічних засад великих мовних моделей (LLM), який базується на огляді наукових публікацій, наприклад, [17-19].

Традиційні рекурентні нейронні мережі (RNN) мали два основні недоліки:

– нездатність ефективно працювати з довгостроковими залежностями;

– неможливість паралельних обчислень, що призводило до низької обчислювальної ефективності.

Потреба у створенні сучасних масштабних моделей вимагала розвитку нових методів, і трансформери виконали цю роль, здійснивши революцію в обробці природної мови завдяки механізму самоуваги (self-attention). Цей механізм дає змогу моделям значно ефективніше вловлювати взаємозв’язки

між окремими словами, визначаючи вагомість кожного слова в реченні під час генерування чи розуміння тексту.

Сучасні LLM, включно з моделями GPT, значною мірою базуються на архітектурі трансформера, запропонованій у [20]. Трансформер, насамперед, вирішує задачу Seq2Seq, де вхідні та вихідні послідовності часто мають різну довжину. Він складається з двох основних компонентів: енкодера та декодера.

Енкодер отримує вхідну послідовність токенів і обробляє їх паралельно. Він складається з багатьох шарів самоуваги та повнозв'язних нейронних мереж. Кожен шар має кілька голів уваги (attention heads), що дають змогу моделі вловлювати різні аспекти вхідної послідовності.

Декодер генерує вихідну послідовність автогоресивно – токен за токеном. Він також містить кілька шарів самоуваги та повнозв'язних мереж, але додатково має механізм уваги до виходів енкодера.

Під час навчання моделі подається вхідна послідовність, і декодер навчається передбачати наступний токен на основі попередніх. На рис. 1.3 показано приклад архітектури «енкодер–декодер» [21].

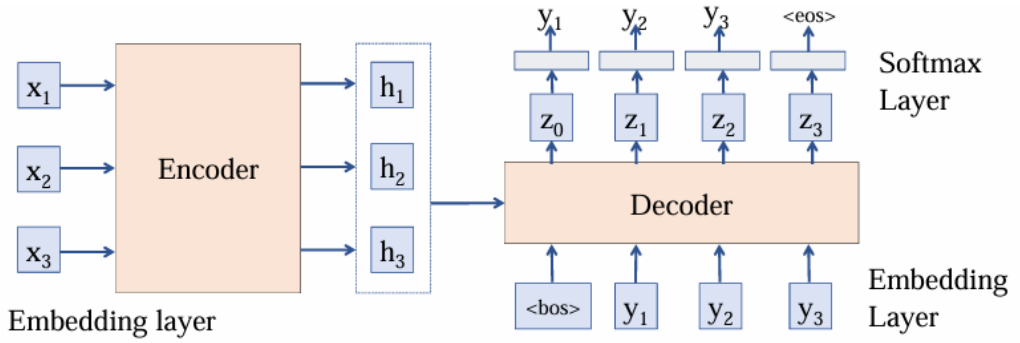

Рисунок 1.3 – Архітектура енкодера-декодера, що використовується в LLM

Вхідна послідовність x_1, x_2, x_3 після проходження енкодера перетворюється на приховані стани h_1, h_2, h_3. Завдання декодера – генерувати слова послідовно. Спершу подається спеціальний токен <bos> (beginning of sentence), після чого декодер генерує y_1. На основі y_1 декодер генерує y_2 і так далі, доки не буде згенеровано токен <eos> (end of sentence), що означає завершення речення. Далі розглянемо реалізацію механізму уваги.

Механізм уваги є ключовим компонентом великих мовних моделей і відіграє важливу роль у вловлюванні залежностей та взаємозв'язків між словами у реченні. Він дає змогу моделі фокусуватися на різних частинах вхідної послідовності під час генерування чи інтерпретації кожного токена.

Для заданої послідовності увага обчислює зважену суму векторів значень (value, *v*), де ваги визначаються релевантністю кожного токена. Релевантність визначається порівнянням вектора запиту (query, *q*), який відповідає поточній позиції, з векторами ключів (key, *k*), отриманих з усіх позицій послідовності.

Нехай кожне слово представлено як ($x_1, x_2, \ldots, x_n$). Вектори запитів, ключів і значень обчислюються за допомогою лінійних перетворень:

$$\begin{aligned} q_i &= W\_q\, x_i \\ k_i &= W\_k\, x_i \\ v_i &= W\_v\, x_i \end{aligned} \tag{1.1}$$

Далі вектори *q*, *k*, *v* діляться на кілька голів уваги:

$$\begin{aligned} q_i^h &= W\_q^h\, x_i \\ k_i^h &= W\_k^h\, x_i \\ v_i^h &= W\_v^h\, x_i \end{aligned} \tag{1.2}$$

Основна формула механізму уваги трансформера:

$$Attention(Q, K, V) = softmax(QK^T / \sqrt{d_k})\, V, \tag{1.3}$$

де Q, K, V – матриці запитів, ключів і значень відповідно,

d_k – розмірність ключів.

Для *i*-го слова увага обчислюється як:

$$e_{ij}^h = (q_i^h \cdot (k_j^h)^T) / d_k, \tag{1.4}$$

$$\alpha_{ij}^h = exp(e_{ij}^h) / \sum_j' exp(e_{ij}'^h). \tag{1.5}$$

Остаточне значення для голови:

$$z_i^h = \sum_j \alpha_{ij}^h\, v_j^h. \tag{1.6}$$

Потім результати всіх голів об'єднуються:

$$z_i = Concat(z_{i1}, z_{i2}, \ldots, z_{ih}). \tag{1.7}$$

Далі застосовуються залишкове з'єднання та нормалізація шару:

$$o_i = LayerNorm(z_i + x_i). \quad (1.8)$$

Після цього вихід проходить через feed-forward нейронну мережу:

$$f_i = FFN(o_i) \quad (1.9)$$

У результаті енкодер формує остаточні представлення f_i кожного слова, які потім передаються до декодера для генерації тексту.

Хоча більшість закритих LLM-моделей, таких як GPT-3.5/GPT-4, привертають увагу своєю високою продуктивністю (переважно завдяки ChatGPT), відсутність відкритості щодо даних і коду породжує низку проблем [22].

По-перше, використання закритих LLM може створювати певні етичні ризики для суспільства через невідому архітектуру, непрозорість даних попереднього навчання та способу, у який модель була донавчена.

По-друге, оскільки продуктивність моделей змінюється з часом (згадаймо колосальну еволюцію моделей у 2023-2024 рр.), і деталі залишаються «чорним ящиком», дослідникам дуже складно отримувати відтворювані результати, як показано в недавніх роботах [23-25].

Нарешті, застосування цих закритих LLM у різних предметних галузях, зазвичай, пов’язане зі значними витратами.

Відкриті LLM, натомість, мають потенціал для пом’якшення та розв’язання цих проблем. Відповідно, дослідникам рекомендовано використовувати відкриті моделі, як вхідні, та публікувати власні донавчені моделі для конкретних завдань у відкритому доступі.

Наприклад, після того як LLaMA стала доступною за неліцензованою комерційною угодою, автори [26] обрали попередньо натреновану базову модель LLaMA-7B і донавчили її на доменних знаннях з транспортної безпеки, запропонувавши нову доменну специфічну модель – TrafficSafetyGPT. Крім того, TrafficSafetyGPT перевершила найсучаснішу неспеціалізовану модель LLaMA у різних завданнях, пов’язаних із безпекою.

Отже, використання наявних відкритих LLM у предметно орієнтованих застосуваннях є перспективним напрямом для подальшого розвитку LLM у

сфері транспорту. Відкриті LLM мають очевидні переваги у прозорості та відтворюваності [27].

Однак варто зазначити, що багато відкритих моделей публікуються лише у вигляді архітектури моделі; пов’язані джерела навчальних даних часто не доступні публічно. Відповідно, дослідники повинні забезпечити можливість повторного отримання всіх результатів іншими групами, як, наприклад, у [28], де, використовуючи відкриту платформу з кількома типами симуляцій, вдалося не лише створити реалістичний сценарій для контролю транспорту, а й отримати відтворювані та репліковані результати.

Разом зі зростанням популярності LLM багато дослідників запропонували прототипи, які незабаром почнуть упроваджуватися на практиці. До цього моменту важливо розробити формальні протоколи верифікації та валідації таких моделей, коли вони взаємодіють із симульованими та реальними середовищами. Необхідно поширювати великі еталонні набори даних, які містять описи сценаріїв і формальні визначення коректних відповідей, а також інструменти для виявлення потенційно небезпечних відповідей моделі.

Важливо, що такі зусилля мають здійснюватися не лише для базових моделей, а й для донавчення спеціалізованих моделей (downstream models). Моделі, які проходять донавчення, мають перевагу в тому, що їм не потрібно навчатися з нуля, що дозволяє дослідникам отримувати результати з істотно меншими обчислювальними ресурсами та витратами на навчання. Це особливо корисно для доменно-специфічних даних, які можуть підсилити здатність LLM розуміти особливості певної галузі – наприклад, точне прогнозування дорожньо-транспортних пригод, визначення чинників чи оцінювання відповідальності [27].

У зв’язку зі складністю середовищ оцінювання LLM, потрібно проводити більше експериментів, особливо на великих наборах даних, щоб зменшити випадкові помилки моделей [29]. Водночас, донавчання не лише додає нову інформацію (або змінює наявну) всередині моделі, але й

опосередковано може спричинити побічні ефекти, наприклад, коли зміна ваг під час донавчення змінює поведінку базової моделі поза межами очікуваного. Це становить суттєву загрозу для критично важливих з точки зору безпеки застосувань і потребує протидії з боку розробників LLM.

Не можна обійти й етичні міркування. LLM-моделі навчаються на основі даних з інтернету, які містять значний потенціал упереджень, зокрема, історичних, репрезентативних і семантичних [30]. Тому перед використанням таких моделей на практиці необхідно ретельно оцінити їхні етичні наслідки. Крім того, користувачі LLM дедалі частіше опинятимуться в ситуації дилеми між приватністю та ефективністю. З одного боку, щоб забезпечити ефективність і якість роботи, користувачам доводиться завантажувати персональні дані до хмарних LLM-сервісів, що створює ризик небажаного збору даних провайдерами. З іншого боку, підходи, засновані на локальних (edge) пристроях, зберігають дані локально, але часто не здатні забезпечити належний рівень продуктивності.

Відповідно, мають бути обговорені способи збирання, зберігання, обробки та поширення таких приватних даних, що слугуватиме орієнтиром для розробників LLM і організацій. Нарешті, принцип підзвітності є критично важливим у розробці LLM: мають існувати стандарти та правові норми [31], особливо у контексті етики інтелектуальних транспортних систем.

Такі регулювання, як Загальний регламент про захист даних (GDPR), суттєво вплинуть на розробку та впровадження LLM у різних сферах. Оскільки ці норми висувають дедалі жорсткіші вимоги щодо конфіденційності даних, безпеки та згоди користувача, застосування систем із ШІ має гарантувати, що дані є анонімізованими або зашифрованими для відповідності GDPR. Крім того, розробники LLM повинні впроваджувати механізми, які дозволяють користувачам контролювати власні дані, включно з правом на доступ, зміну або видалення інформації та повну відмову від збирання даних.

Виконання таких нормативів створюватиме не лише технічні виклики, пов’язані з оновленнями та приховуванням часової інформації, а й може

уповільнити впровадження LLM, оскільки компанії будуть змушені інвестувати у юридичну та технічну інфраструктуру для забезпечення відповідності. Крім того, вимоги до прозорості можуть підсилити акцент на пояснюваності, щоб LLM могли надавати чіткі підстави та джерела використаних даних у процесі прийняття рішень, наприклад, у транспортних системах щодо маршрутних рекомендацій або прогнозів трафіку.

Спеціалізовані регулювання у сфері ШІ, як запропонований AI Act ЄС, також впливатимуть на розробку й розгортання LLM. У цьому контексті застосування різних прикладних сфер, ймовірно, буде класифіковано як високоризикове, що вимагатиме суворих стандартів безпеки, надійності та точності, а також масштабних тестувань і сертифікаційних процедур. Заходи підзвітності стимулюватимуть організації створювати структури управління, проводити регулярні аудити та забезпечувати етичне використання, а акцент на упередженнях і справедливості збільшить потребу у ретельних оцінках, щоб запобігти дискримінаційним результатам.

2.3 Дослідження можливостей використання штучного інтелекту в області програмної інженерії

За останні кілька років ШІ став свідком експоненціального зростання в розвитку, збільшення використання та підвищення суспільного інтересу на різних рівнях, від індивідуального до організаційного [32-34]. Сучасний штучний інтелект використовує машинне навчання та інші передові методи для створення нових знань, контенту, гіпотез і навіть інноваційних ідей шляхом визначення шаблонів та інформації, яка зазвичай міститься у великих базах даних. Це стало каталізатором використання ШІ в широкому спектрі програм, включаючи проєктування та розробку програмного забезпечення (ПЗ) [35]. Останнім часом з’являються публікації, в яких наведено приклади використання ШІ на основі LLM для виконання різноманітних завдань: від

навчання на великих даних з подальшою обробкою в прикладних інформаційних системах [36], аналізу функціоналу прикладних систем на основі сформульованих вимог до [37], до виправлення помилок і генерації нового коду із заданими функціями.

Автори масштабного дослідження публікацій за 2018-2025 рр. [38], пов'язаних із використанням ШІ в різних аспектах програмної інженерії, відзначають, що науковці почали широко впроваджувати нові підходи на основі ШІ для вирішення традиційних проблем програмної інженерії, зокрема, на основі мовних моделей [39-40]. Джерелом аналізу стали повторювані тематичні ключові слова авторів у публікаціях, контенті, категоріях, темах. В ролі вихідної інформаційної бібліографічної бази було взято Scopus (Elsevier, Амстердам, Нідерланди), оскільки вона вважається найбільшою базою рефератів і цитат рецензованої наукової літератури. На додаток до розширених аналітичних служб, він дозволяє експортувати 20 000 записів одночасно. Пошуковий запит включав всі можливі комбінації ключових слів, які можуть бути пов'язані з розробкою програмних кодів та ШІ. Приклади формулювання і конструкції запитів наведені нижче.

TITLE-ABS-KEY((«штучний інтелект» АБО «машинне навчання» АБО «глибоке навчання» АБО «інтелектуальна система» АБО «машина опорних векторів» АБО («дерево рішень» І («індукційний» АБО «евристичний»)) АБО «Марковський процес прийняття рішень» АБО «прихована модель Маркова» АБО «нечітка логіка» АБО «k-найближчий сусід» АБО «наївний Байєс» АБО «Байєсівське навчання» АБО «нейронна мережа» АБО «згорточна нейронна мережа» АБО «рекурентна нейронна мережа» АБО «генеративна змагальна мережа» АБО «персептрон» АБО {обробка природної мови} АБО {розуміння природної мови} АБО {загальна модель мови}) і ({розробка програмного забезпечення} АБО {дизайн програмного забезпечення} або { розробка програмного забезпечення})) І PUBYEAR > 2018 І PUBYEAR < 2025. Діаграма популярності запитів показана в додатку А. Розподіл за типами статей показує, що більшість публікацій були доповідями, пов'язаними з конференціями, що

свідчить про те, що дослідження все ще перебуває на стадії дозрівання, і основні знання ще формуються. Цей висновок також підтверджується тим фактом, що трьома найбільш плідними назвами є матеріали конференцій, що вказує на те, що список основних журналів ще потрібно створити.

Тенденція продуктивності досліджень, яка представлена на рис. 1.4, свідчить, що пік продуктивності загальної кількості публікацій був досягнутий у 2020 р. Однак кількість публікацій стабілізувалася у 2022 р. Зменшення кількості публікацій відповідає загальній тенденції продуктивності програмного забезпечення інженерних публікацій, індексованих у Scopus, які також почали демонструвати спад у 2020 р., головним чином через зменшення кількості доповідей на конференціях (сплеск пандемії), тоді як кількість статей почала збільшуватися у 2022 р. Наведені вище дані можуть свідчити про початок позитивної тенденції до досягнення дослідницької зрілості.

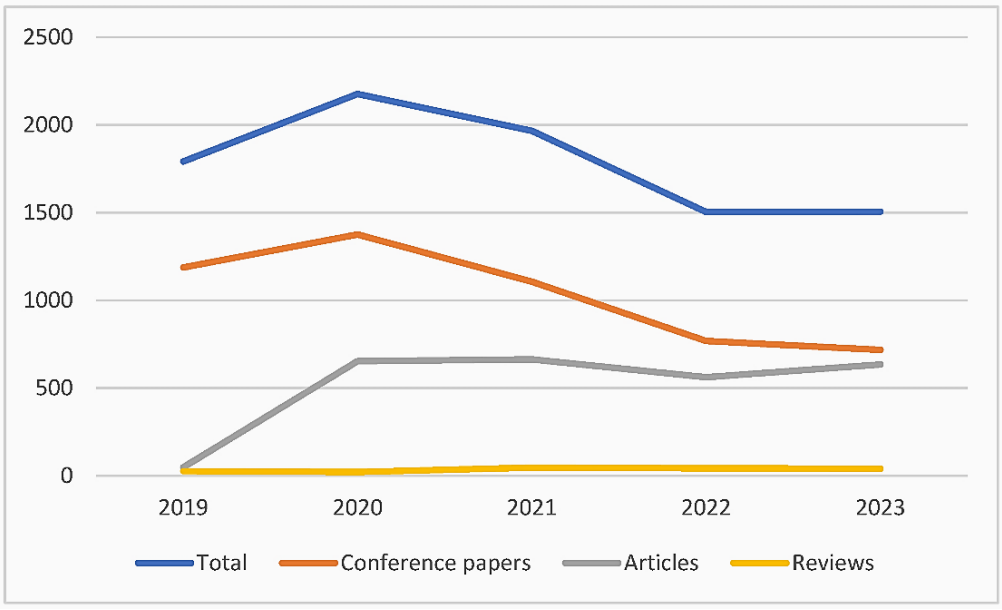

Рисунок 1.4 – Динаміка продуктивності наукової літератури в області ШІ [38]

Щодо географії досліджень, то, якщо позначити *n* як кількість публікацій, пов'язаних зі ШІ, то найпродуктивнішими країнами був Китай (*n* = 2042), Сполучені Штати (*n* = 1193), Індія (*n* = 934), Німеччина (*n* = 445) і Канада (*n* = 381). Це відповідає рейтингу Scimago Country Rankings (Elsevier),

де США посідають перше місце за ПЗ і друге за ШІ, Китай посідає перше місце за ШІ і друге за ПЗ, а інші провідні країни займають місця серед 10 найрезультативніших в обох категоріях. Усі найбільш продуктивні країни також входять до G20 [41]. Найбільш популярними за кількістю публікацій виявилися словосполучення і терміни, а також пов'язані з ними області досліджень, які наведені в табл. 1.1.

Таблиця 1.1 – Кластери та пов'язані з ними ключові слова і категорії

Репрезентативні ключові слова, к-ть	Категорії (області дослідження)
Штучний інтелект (560), розробка програмного забезпечення (173), тестування програмного забезпечення (123), нечітка логіка (98), програмне забезпечення (73), великі дані (65), навчання з підкріпленням (64)	Етичне використання розробки програмного забезпечення на основі штучного інтелекту, використання нечіткої логіки в розробці та тестуванні програмного забезпечення, автоматизація тестування програмного забезпечення в гнучкому середовищі, управління проектом життєвого циклу програмного забезпечення за допомогою нечіткої логіки, наука про дані та великі дані в розробці програмного забезпечення
Розробка програмного забезпечення (673), обробка природної мови (362), розробка вимог (108), гнучка розробка програмного забезпечення (61)	Обробка природної мови при розробці програмного забезпечення, обробка природної мови при розробці вимог до програмного забезпечення, розуміння історій користувачів за допомогою обробки природної мови
Машинне навчання (1504), оцінка зусиль розробки програмного забезпечення (156), класифікація (142), передбачення дефектів програмного забезпечення (205), аналіз даних (102), штучна нейронна мережа (184), показники програмного забезпечення (84), вибір функцій (82)	Оцінка зусиль щодо розробки програмного забезпечення, інтелектуальний аналіз даних у прогнозуванні помилок/дефектів програмного забезпечення, машинне навчання та метрика програмного забезпечення
Якість програмного забезпечення (86), обслуговування програмного забезпечення (62), сховища програмного забезпечення для майнінгу (43)	Репозиторії програмного забезпечення для майнінгу для покращення якості програмного забезпечення та обслуговування програмного забезпечення, Crowdsourcing, GitHub та програмне забезпечення з відкритим вихідним кодом як джерела для даних розробки програмного забезпечення для майнінгу

Як видно з табл. 1.1, найбільш активно за останні 5 років розвивалися напрямки як розробки самих штучних нейронних мереж, ML, так і використання систем ШІ в областях обробки природної мови при розробці ПЗ,

при обробці великих даних. Показово, що на порядки менше було застосовано тих самих тем в областях обслуговування репозиторіїв коду, використання ПЗ з відкритим кодом для розробки ПЗ, прогнозування помилок, підвищення якості програмного забезпечення.

На думку багатьох дослідників, обробка природної мови (Natural language processing, NLP) при розробці ПЗ може значно покращити досягнення результату завдання розробки [42]. NLP може підтримувати категоризацію помилок [43], розробку більш безпечного програмного забезпечення, декомпозицію програми [44], класифікацію зобов'язань [45], програмування та кодування [46], написання узгоджених і фактично правильних читань [47], розробку на основі моделі [48], розгортання шаблонів проектування [49] та управління відстежуваністю [50].

Загалом, вивчення попередніх досліджень штучного інтелекту показали, що вони значно вплинули на розробку ПЗ за останні роки. Від обробки природної мови в розробці ПЗ, використання ШІ в управлінні життєвим циклом розробки ПЗ, використання машинного навчання для прогнозування помилок/дефектів і оцінки зусиль, використання глибокого навчання в інтелектуальній розробці ПЗ та управлінні кодом, до репозиторіїв ПЗ для майнінгу для покращення якості програмного забезпечення, ШІ змінив спосіб створення ПЗ розробниками та інженерами. Конвергенція ШІ та розробки ПЗ має потенціал для значного скорочення необхідних ресурсів, покращення якості та покращення взаємодії з користувачем за допомогою більш інтелектуальних програм, орієнтованих на користувача. У розглянутих роботах відзначено, що на сьогодні все ще потрібне загальне розуміння поточного стану, можливих цільових застосувань, практичних сценаріїв використання штучного інтелекту в програмній інженерії, неминучих обмежень, та інших проблем.

В дослідженні, проведеному дослідницьким інститутом платформи Upwork [51], розглянуто причинно-наслідковий вплив генеративного ШІ на можливості працевлаштування та заробіток фрілансерів. Аналіз даних

продемонстрував, як генеративний ШІ трансформує роботу. Огляд попередніх технологічних інновацій, починаючи від робототехніки до мобільних пристроїв, свідчить про те, що ШІ матиме сильний вплив на роботу.

По-перше, це широко обговорюваний ефект заміщення, коли частина роботи зазнає впливу та зрештою порушується, оскільки деякі завдання можна автоматизувати. Але не менш важливим є менш обговорюваний ефект відновлення, коли новостворені робочі можливості з часом збільшують заробіток, оскільки нові технології створюють нові завдання, в яких робоча сила має порівняльну перевагу. Зазвичай, ефект заміщення більш помітний у короткостроковій перспективі, але оскільки нові робочі місця та завдання створюються в середньостроковій та довгостроковій перспективі, ефект відновлення на роботі переважатиме. Вплив нових технологій часто з часом врівноважує один одного, поступаючись місцем багатьом новим навичкам та можливостям, які раніше були неможливі або просто не існували.

Генеративний штучний інтелект збільшив загальну кількість вакансій і заробіток фрілансерів на кожен новий контракт. Це свідчить про загальний більший попит на роботу, виконану незалежними талантами, з боку клієнтів.

Історично склалося так, що технології створюють більше можливостей працевлаштування, ніж замінюють, оскільки суспільство розвивається, впроваджує інновації та відкриває нові способи роботи.

Співзасновник Microsoft, Білл Гейтс, поділився своїми думками щодо професій, які, на його думку, залишаться за межами можливостей ШІ [52]. Він вважає, що хоча ШІ здатний автоматизувати багато завдань у виробництві, логістиці та сільському господарстві, існують професії, які вимагають людської інтуїції, творчості та критичного мислення, що поки що недоступні для машин. До таких професій був віднесені, зокрема, програмісти, вебдизайнери.

Висновки до розділу 1

Штучний інтелект застсовується у все більшій кількості сфер діяльності людини. Для більш рутинних операцій застосовують моделі машинного навчання, у той час, як для інтелектуальних задач кращим рішенням визнані великі мовні моделі ШІ, або LLM. Значна кількість публікацій у наукових базах даних підтверджує зростання інтересу до тематики ШІ, але перспективи їх використання в реальних умовах залишаються неоднозначними. При цьому з’ясовано, що ці моделі все ще не можуть бути рекомендованими в області охорони та безпеки, оскільки вони не здатні забезпечувати прозоре, детерміноване та відтворюване прийняття рішень у широкому спектрі сценаріїв з високою невизначеністю. LLM не мають жодної з цих трьох властивостей, оскільки процес прийняття рішень все ще значною мірою залишається «чорною скринькою», а дослідники лише поступово поглиблюють розуміння принципів роботи різних шарів і механізмів уваги.

Відтворюваність результатів і далі залишається серйозним викликом у контексті LLM з огляду на стрімкий технологічний прогрес і прагнення дослідників бути першими на ринку. У галузі взаємодії людина–машина необхідно досліджувати методи збереження приватності, оскільки LLM працюють із чутливими даними як користувачів, так і операторів. Це включає розробку методів, які дають змогу зберігати ефективність моделей при дотриманні суворих стандартів конфіденційності.

Майбутні дослідження повинні зосередитися на подальшому розвитку можливостей обробки в реальному часі; особливо важливим є ефективне та масштабоване впровадження технологій генерації з підсиленим пошуком (retrieval-augmented generation), де знання LLM розширюється зовнішньою базою знань. Не менш значущим є удосконалення можливостей мультимодальної інтеграції.

РОЗДІЛ 2
ПІДГОТОВКА ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ ІЗ ЗАЛУЧЕННЯМ МОВНИХ МОДЕЛЕЙ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ОПТИМІЗАЦІЇ ПРОГРАМНОГО КОДУ ВЕБДОДАТКІВ

2.1 Аналіз традиційних підходів до розробки й оптимізації вебдодатків

Класичні методи розроблення вебдодатків включають використання базових технологій HTML&CSS для формування структури вебдодатку і UI дизайну, а також мови програмування JavaScript для забезпечення динамічності та інтерактивності вебдодатку. Добре продуманий дизайн, контент, функції кожного вебдодатку чи вебсайту забезпечують його комерційний успіх і реалізацію очікуваних задач. Навіть для досягнення початкового рівня майстерності веброзробнику доводиться опановувати базові технології, усувати помилки і постійно вдосконалювати свою майстерність. Для професійних розробників необхідно залучати бібліотеки та фреймворки JavaScript, асортимент і призначення яких також мають особливості. Окремим напрямком є розроблення бекенд частини. Часто спеціалізація вебпрограміста може фокусуватися або на фроненді, або бекенді, або графічному дизайні. При цьому важливим аспектом є проводити технічну оптимізацію сайту перед публікацією.

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 [53]. Різноманітність завдань, еволюція технологій веброзробки та зростаючі вимоги до продуктивності, безпеки, масштабованості вебдодатків призвели до значної диверсифікації інструментарію розробника, а також появи засобів спеціалізації та спрощення

при створенні окремих проєктів. Наприклад, основною мовою для розробки динамічного інтерфейсу користувача є мова JavaScript, яка розроблялася та розвивалася від початку запуску перших вебсайтів [54].

JavaScript – мова програмування, яка використовується для створення динамічних і інтерактивних елементів сайту. Існування JavaScript зробило можливим існування вебдодатків – застосунків, в яких оновлення інформації відбувається без оновлення всієї сторінки [55]. У JavaScript фактично немає конкурентів: 98% вебсайтів використовують саме цю мову. Але альтернативи все ж є: Dart, CoffeeScript, Haxe, 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 представлено на рис. 2.1 [53].

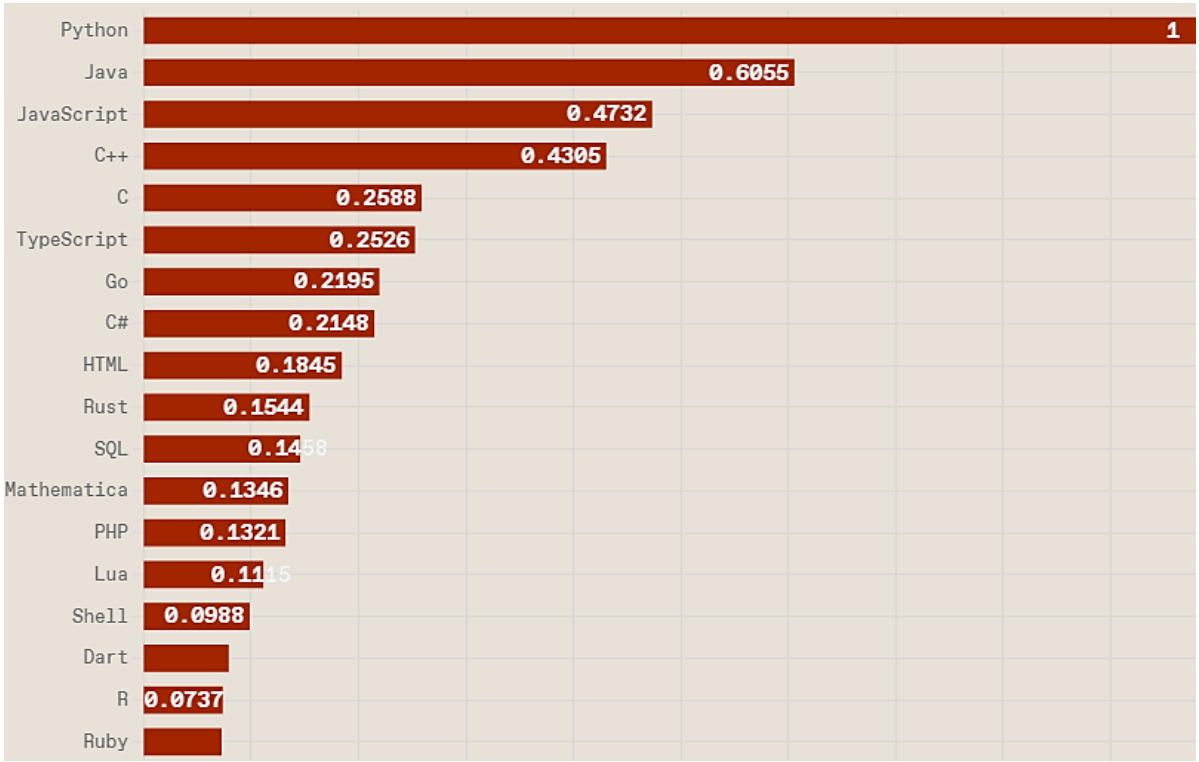

Рисунок 2.1 – Позиція JavaScript у рейтингу мов програмування за даними [53]

Окремо варто розглядати TypeScript – мову програмування, що розширює JavaScript шляхом надання можливості статичної типізації (знаходиться в топі рейтингу мов вебпрограмування, див. рис. 2.1). TypeScript

виник як надмножина JavaScript, яка додає статичну типізацію, що підвищує стабільність і зручність розробки великих проєктів [56]. 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.

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, програмісти експериментують з новими підходами до розробки вебдодатків. Це породжує конкуренцію та прагнення створити кращу мову програмування або фреймворк, що відповідає конкретним вимогам. 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. 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 веброзробникам до цього часу доводиться розбирати й переписувати на сучасний лад. Іншою проблемою є велика кількість коду, який безпосередньо треба писати. Частковим рішенням цих проблем стали бібліотеки та фреймворки.

Серед інших очікувань від якості вебдодатків варто назвати ті, які потребують оптимізації (технічної, користувацької). Через постійний розвиток та ускладнення архітектури вебсайтів, вимоги до швидкості, функціоналу та дизайну зростають з кожним роком. Автори публікації [57] зазначають, що

невідповідність сучасних тенденціям може серйозно вплинути на взаємодію з користувачем та продуктивність вебсайту.

Очікується, що вебдодатки будуть доступні цілодобово з будь-якої точки світу та будуть адаптовані для використання з будь-якого пристрою з будь-яким розміром екрану. Окрім цього, важливими складовими сучасного вебдодатку є гнучка і масштабована система безпеки та багатий досвід користувача, побудований на клієнті, вважають автори [58].

Помітно втрачають актуальність багатосторінкові сайти, особливо в бізнес-сфері, адже швидкість та продуктивність таких сайтів є вкрай низькою. Це пов’язано з тим, що на кожну дію користувача запитується нова сторінка з серверу, та відбувається повне перезавантаження на клієнтській частині. На зміну їм прийшли односторінкові «сайти» Single Page Applications (SPA), які отримують сторінку з сервера лише один раз, а всі інші операції виконуються завдяки асинхронним AJAX та JavaScript скриптам [59]. Подібні зміни вимагали нових ідей та підходів до розробки. Наприклад, поява концепції «компонентів» у веброзробці дала змогу створювати елементи, які можна повторно використовувати в різних частинах коду. Окрім цього, стали з’являтися різні бібліотеки для управління станом додатку (Redux, MobX). Обов’язковою стала можливість інтеграції вебзастосунків з різними платформами і сервісами (соціальні мережі, платіжні системи, API сторонніх сервісів). Для створення real-time застосунків, таких як чати, було створено технологію websockets, яка дозволяє встановлювати постійне двостороннє з’єднання між користувачами, без необхідності постійно створювати нові «коннекти», як при використані традиційних методів [60].

Саме через необхідність постійного створення нового інноваційного інструментарію для реалізації подібних ідей, сформувалась та невпинно розвивається величезна екосистема вебтехнологій: спеціалізовані мови програмування, фреймворки, бібліотеки, бази даних та системи керування, інтегровані середовища розробки тощо. Причому, межі чіткого застосування й приналежності часто не зовсім чітко окреслені. Наприклад, у середовищі

розробників помітні суперечки щодо приналежності React до фреймворків або бібліотек. На вебсайті відомого блогера й програміста Еріка Елліота [60] знаходимо визначення (мовою оригіналу): «React is the Top UI Framework in the World». Однак, інформація на офіційному вебсайті чітко говорить про те, що це бібліотека: «JavaScript-бібліотека для створення користувацьких інтерфейсів». Рекомендації для встановлення програми додаються, тому розпочати роботу в системі можна одразу і безпечно в професійному середовищі [61]. На офіційному сайті викладено необхідні матеріали для вивчення можливостей бібліотеки (рис 2.2), як і посилання на інші бібліотеки.

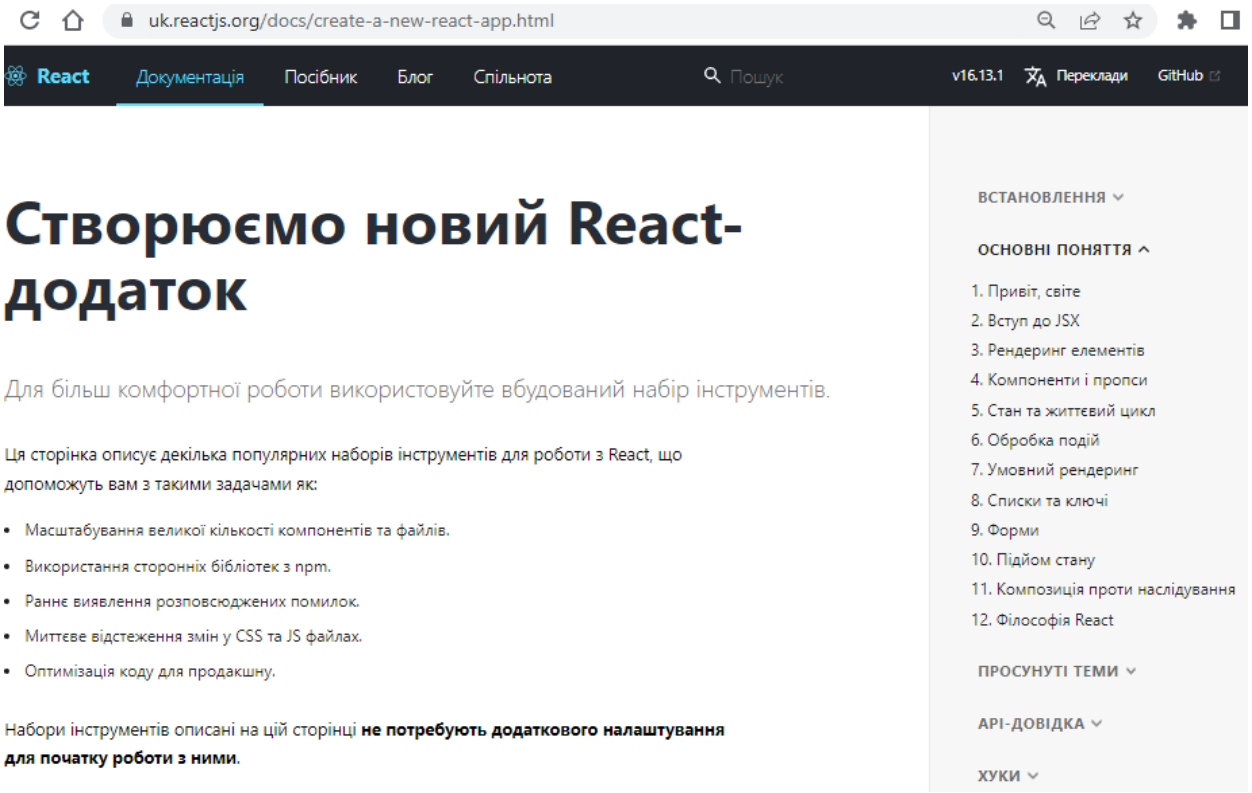

Рисунок 2.2 – Зміст розділу «Документація» на вебсайті React [61]

React-додаток будується з компонентів, саме в них і відбувається оновлення і декларація елементів сторінки. Тобто, компонент це головне поняття в React, єдина сутність, яку він містить. Компоненти можуть допомогти розбити інтерфейс на незалежні частини і використовувати їх повторно за потреби та комбінувати будь-яким чином. Це по суті своїй клас, що унаслідується з кореня бібліотеки. React-компонент може повертати

HTML-код за допомогою JSX. У звичайному JavaScript таке неможливо, це також зручна надбудова над мовою. Компоненти реалізують метод render(), який приймає вхідні дані і повертає те, що буде показано користувачу. Приклад застосування методу наведено в коді (рис. 2.3).

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Привіт, {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Віталій" />,
  document.getElementById('hello-example'),
);
```

Рисунок 2.3 – Приклад застосування елементів та методу render() на React

У цьому прикладі (див. рис. 2.3) використовується XML-подібний синтаксис під назвою JSX. Доступ до вхідних даних, які передаються в компонент, можна отримати за допомогою render() та this.props. Результатом виконання прикладу буде виведення привітання в інтерактивному режимі. Зазвичай, така модель обробки даних поширюється на більш складні мовні інтерактивні взаємодії, і потужність бібліотеки виявляється на повну сутність.

JSX – це препроцесор (розширення JavaScript), що спрощує створення елементів і компонентів React, дозволяє декларативно створювати компоненти користувальницького інтерфейсу. Розширення JSX володіє наступними вагомими можливостями:

– застосування простої декларативною розмітки;

– код розмітки розташований там само, де і код компонента;

– реалізація принципу поділу відповідальностей (наприклад, відділення опису інтерфейсу від логіки стану і від побічних ефектів). При цьому реалізація базується не на використанні різних технологій (наприклад, HTML, CSS, JavaScript);

– абстрагування при управлінні змінами DOM;

– абстрагування від особливостей різних платформ, для яких створюють React-застосунки.

З цим розширенням під час написання коду потрібно набагато менше зусиль, порівняно з класичним JavaScript. JSX трансформується в JavaScript перед запуском у браузері. Він не є обов'язковим під час використання React. Таким чином, фреймворк React є надзвичайно зручним інструментом у веброзробці, який дозволяє будувати сучасні та швидкі вебдодатки.

Файли CSS можна завантажити в заголовок структури вебсторінки для загальних глобальних макетів. Модулі CSS – це локальні файли CSS, які можна імпортувати безпосередньо у файли JavaScript. Для цього знадобиться правильно налаштований завантажувач. Next.js увімкне це за замовчуванням. Styled-jsx дозволяє оголошувати вбудовані стилі у компонентах React, подібно до того, як теги <style> працюють у HTML. Сфера застосування цих стилів є гіперлокальною, що означає, що стилі впливатимуть лише на рідні теги та їхні дочірні теги. Next.js також включає styled-jsx за замовчуванням.

Virtual DOM у React представляє полегшену копію звичайного DOM. І відмінною особливістю React є те, що ця бібліотека працює саме з віртуальним DOM, а не звичайним. Якщо програмі необхідно уточнити інформацію про стан елементів, то вона звертається до віртуального DOM. За рахунок Virtual DOM має збільшуватися продуктивність, що є вимого оптимальності.

React надає декларативний API, і це створює умови, за яких не потрібно турбуватися про зміни, які відбуваються, що спрощує написання додатків. React надає повний контроль над розміром додатка, дає більше гнучкості під час переходу від односторонніх додатків (SPA) до мікросервісів, використовуючи частини колишнього додатка.

React чудово підійде для проєктів, де важлива гнучкість при створенні великих екосистем, що мають тенденцію розростатися. Дуже добре підходить для командної розробки. UI-код читабельний і простий у супроводі. Здатний повністю взяти на себе відповідальність за рівень подання в архітектурі MVC

(модель, подання, контролер). Розробка користувацького інтерфейсу за цим принципом – це сучасний підхід у сфері надання даних.

Важливим моментом оптимізації розробки є адаптивність. Зараз популярним є підхід «Mobile First». Автори роботи [62] закликають завжди дотримуватися цього принципу. Цей принцип означає, що застосунки повинні розроблятися таким чином, щоб в першу чергу гарно виглядати на мобільних пристроях, а вже після цього на широких екранах.

До появи таких інструментів як CSS та JavaScript такий підхід, та й взагалі адаптивність вебсторінки була неможливою. Це важливо, оскільки якщо подивитись на статистику інтернет–трафіку за 2024 р., то можна побачити, що трохи більше ніж 60% трафіку складають користувачі мобільних пристроїв. Крім того, розвиток мобільних технологій також призводить до необхідності впровадження в розробку специфічних фреймворків та інструментів, таких як React Native, Flutter або Xamarin. За допомогою цих інструментів розробники можуть створювати кросплатформенні додатки, які працюють як в браузерах, так і на iOS та Android. Такий підхід дозволяє писати один код, який буде виконуватись на різних видах девайсів, що суттєво економить фінансові ресурси замовника та час розробника [63].

Враховуючи стрімкий розвиток галузі та постійну появу нових інноваційних інструментів, а також велику конкуренцію між вебстудіями, одним із головних завдань сучасного веброзробника є аналіз та підбір найкращого рішення саме для потреб його проєкту.

Високий попит на вебдодатки змушує розробників писати економічно ефективні, безпечні та оптимізовані вебдодатки. Говорячи про шляхи отримання раціонального, оптимального коду, не можна не висвітлити застосування штучного інтелекту. Останнім часом з’являються публікації, в яких наведено приклади використання ШІ на основі великих мовних моделей (LLM) для виконання різноманітних завдань: від навчання на великих даних з подальшою обробкою в прикладних інформаційних системах, аналізу функціоналу прикладних систем на основі сформульованих вимог, до

виправлення помилок і генерації нового коду із заданими функціями [64]. В роботі [65] показано метод навчання моделі кодувальника-декодера на основі мільйонів виправлень помилок в історії змін проектів, розміщених на GitHub. Показано, що така модель здатна перекладати код з помилками у виправлену версію до 50% випадків. Автори [66] представили Codex – модель мови GPT, налаштовану на загальнодоступному коді з GitHub, і вивчали її можливості написання коду Python. Дослідження виявило низку обмежень, які включали труднощі з документаційними рядками, що описують довгі ланцюжки операцій, і з прив'язкою операцій до змінних. Однак, ідея генерації складних кодів показала безумовну перспективність.

2.2 Результати експериментальної перевірки ефективності окремих моделей ШІ для генерації й оптимізації коду

Як було відмічено, LLM є значним досягненням у галузі ШІ та глибокого навчання, особливо у сфері розуміння й генерації природної мови. LLM тренуються на величезних наборах даних, які включають різноманітну текстову інформацію, що надає їм можливість розуміти мову, розпізнавати контекст і формувати зв'язні відповіді.

Найбільш популярною на сьогодні в різних областях застосування є модель ШІ GPT-4, яка стала наступнице попередньої версії GPT-3.5. Розробкою займається компанія OpenAI [67]. Можливості моделі варто розглянути через порівняння версій. GPT-4 – це велика мультимодальна модель (приймає вхідні зображення та текст, видає текстові виходи), яка, хоча й менш здатна, ніж люди в багатьох сценаріях реального світу, демонструє продуктивність на рівні людини на різних професійних і академічних тестах. Наприклад, він здає симуляцію адвокатського іспиту, набравши приблизно 10% найкращих учасників іспиту; навпаки, оцінка GPT-3.5 була близько нижніх 10%. Розробники витратили 6 місяців на ітераційне узгодження GPT-4,

використовуючи уроки змагальної програми тестування, а також ChatGPT, що призвело до найкращих результатів (хоча й далеко не ідеальних) щодо фактичності, керованості та відмови виходити за поручні.

Протягом останніх двох років перебудували весь стек глибокого навчання та разом із Azure спільно з нуля розробили суперкомп'ютер для підтримки робочого навантаження. У результаті тренувальний запуск GPT-4 був, за оцінко розробників, безпрецедентно стабільним, ставши першою великою моделлю, продуктивність навчання якої вдалося точно передбачити заздалегідь.

Як і попередні моделі GPT, базову модель GPT-4 було навчено передбачати наступне слово в документі з використанням загальнодоступних даних (наприклад, даних інтернету), а також даних, які ми отримали ліцензію. Дані – це сукупність даних у вебмасштабі, включаючи правильні та неправильні розв'язки математичних задач, слабкі та сильні міркування, суперечливі та послідовні твердження, які представляють велику різноманітність ідеологій та ідей.

Тож на запит із запитанням базова модель може відповідати різними способами, які можуть бути далекими від намірів користувача. Щоб узгодити її з наміром користувача в межах огороджень, ми точно налаштовуємо поведінку моделі за допомогою навчання з підкріпленням із зворотним зв'язком людини (RLHF).

Щоб зрозуміти різницю між цими двома моделями, були проведені дослідження за допомогою різноманітних тестів, у тому числі симуляції іспитів, які спочатку були розроблені для людей. Були використані найновіші загальнодоступні тести (у випадку олімпіад і питань з безкоштовними відповідями або придбавши видання практичних іспитів 2022–2023 рр. Результати наведені у вигляді графіку в додатку Б.

В табл. 2.1 представлено результати оцінки GPT-4 (за [67]) з найбільш відомими моделями (state-of-the-art, SOTA) та мовними моделями (LM SOTA) за традиційними тестами, розробленими для моделей машинного навчання.

Таблиця 2.1 – Порівняльні результати тестування моделі GPT-4

Тип тесту	GPT-4	GPT-3.5	LM SOTA	SOTA
Завдання з вибором відповідей із 57 предметів (професійні та академічні)	86,4%	70,0%	70,7%	75,2%
Розумні міркування навколо повсякденних подій	95,3%	85,5%	84,2%	85,6%
Питання з природничих наук для початкової школи. Виклик-набір	96,3%	85,2%	82,4%	85,6%
Розсудливі міркування щодо розділення займенників	87,5%	81,6%	84,2%	85,6%
Завдання з програмування на мові Python	67,0%	48,1%	26,2%	65,8%
Розуміння прочитаного та арифметика	80,9%	64,1%	70,8%	88,4%

Як видно з табл. 2.1, модель GPT-4 показала найкращі результати у всіх типах тесту порівняно з іншими, у т.ч. мовними, моделями. Найнижчий результат отримано при вирішенні завдання на програмування мовою Python, однак, він також найкращий серед інших моделей. Можна припустити, що можливості моделі випливають здебільшого з процесу попереднього навчання. Але керування моделлю відбувається після процесу навчання – базова модель потребує швидкого проектування, щоб навіть знати, що вона має відповідати на запитання.

Одним із найбільш трансформаційних застосувань LLM у програмній інженерії є здатність перетворювати описи природною мовою в виконуваний код. Такий процес дозволяє розробникам описувати функціональність звичайною мовою, після чого модель автоматично трансформує її у структурований програмний код.

Першим етапом є аналіз вхідних даних, під час якого LLM отримує текстовий опис і розбиває його на токени для подальшої обробки. Далі відбувається етап розуміння наміру, коли модель інтерпретує мету користувача, визначаючи ключові слова, контекстні ознаки та залежності. На основі цього розуміння розпочинається етап генерації коду, у межах якого модель створює функціональний фрагмент коду, що відповідає вимогам. Згенерований код зазвичай відповідає найкращим практикам і стандартам

цільової мови програмування, але може потребувати додаткового вдосконалення. Схема процесу взаємодії розробника з моделлю ШІ в різних процесах представлена на рис. 2.4.

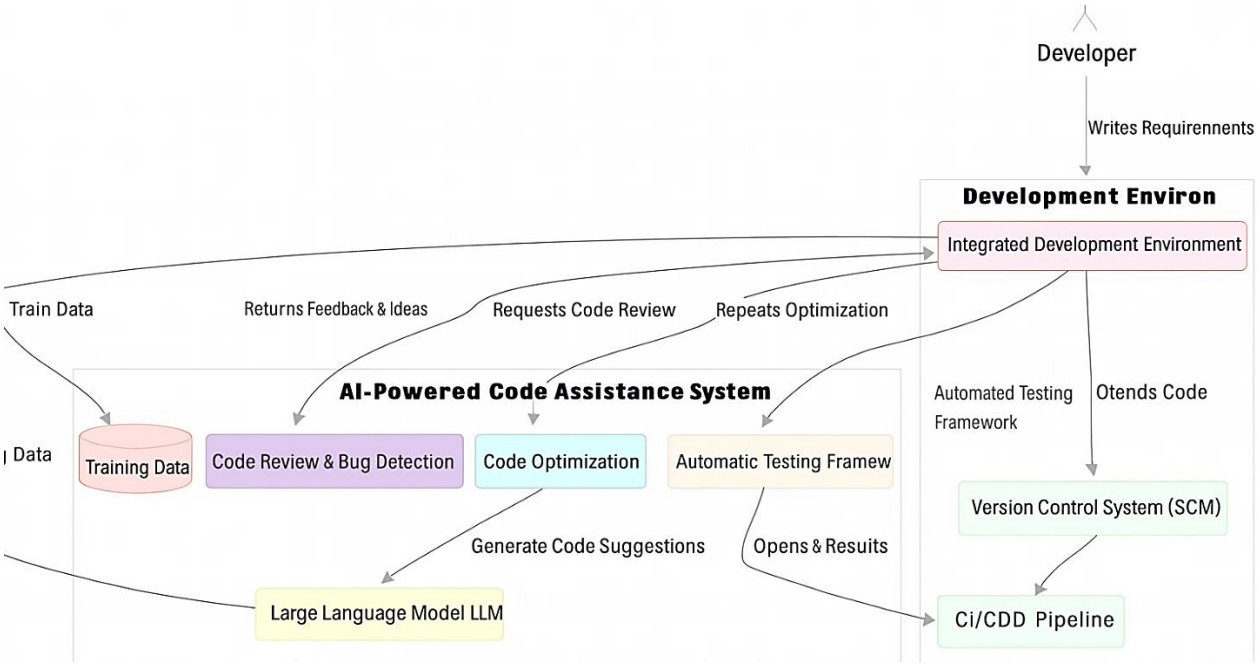

Рисунок 2.4 – Система генерації та оптимізації коду на базі штучного інтелекту

На завершальному етапі – удосконалення результату, модель оптимізує код, усуваючи недоліки, зменшуючи надлишковість і підвищуючи ефективність. Цей процес значно підвищує продуктивність розробки та робить програмування доступнішим для новачків [68].

LLM можуть також виконувати зворотну задачу – перетворювати код у текст, зрозумілий людині, що особливо корисно для документації, ревю коду та навчання. Процес починається з аналізу коду, де модель структурує та токенізує дані. Далі йде етап розуміння коду, під час якого модель визначає змінні, функції, цикли та інші конструкції, щоб зрозуміти логіку програми. Потім модель переходить до генерації тексту – створює пояснення, яке може бути як коротким, так і детальним. Завершальний етап – удосконалення результату для покращення точності, ясності та лаконічності.

Автоматизація пояснення коду сприяє кращій підтримуваності, документуванню та ефективному навчанню нових розробників.

Оптимізацію коду розглянемо в кількох аспектах: статичний і динамічний аналіз коду, автоматизоване ревю, виявлення помилок. Суттєві характеристики застосування цих методів наведені в табл. 2.2 на основі [68]. При цьому будуть описані результати кейсів, проведених за допомогою різних доступних інструментів та узагальнені відгуки, отримані від користувачів.

Таблиця 2.2 – Характеристики методів LLM для оптимізації коду

Метод	Сутність методу	Можливості удосконалення від ШІ
Статичний аналіз коду	Дозволяє знаходити помилки без виконання програми: процес розпочинається зі синтаксичного аналізу - перевірки правильності синтаксису. Далі семантичний аналіз, що досліджує логічну структуру коду	Модель може виявляти неефективні цикли, зайві обчислення, вразливості безпеки, після чого пропонує оптимізаційні рішення: - рефакторинг, - усунення непотрібних операцій, - перебудова алгоритмів, - дотримання найкращих практик.
динамічний аналіз коду	Дозволяє оцінити роботу коду під час виконання. LLM моделюють виконання коду, виявляючи потенційні збої, помилки та вузькі місця. Потім відбувається аналіз продуктивності — оцінюється час виконання, споживання пам’яті та використання ресурсів.	Модель пропонує оптимізаційні рішення, такі як: - удосконалення управління пам’яттю, - розгортання циклів, - поліпшення алгоритмів, - паралелізація.
Автоматизоване ревю коду	Модель визначає анти-патерни, дублікати, помилки дизайну	Покращення - від перейменування змінних до вибору ефективніших алгоритмів. Інтеграція таких інструментів в IDE та системи контролю версій значно спрощує робочий процес і зменшує кількість людських помилок
Виявлення помилок	Моделі аналізують код, розпізнаючи патерни, притаманні помилковим конструкціям, виявляють аномалії - нетипову поведінку, логічні помилки, необроблені крайні випадки та потенційні вразливості	Як результат підвищує надійність і безпеку програмного забезпечення, а також значно скорочує час налагодження

Надалі розглянуто окремі результати застосування різних інструментів ШІ для окремих операцій з кодами, були виміряні метрики ефективності [68].

CodeGen від GitHub – це інструмент на основі ШІ, створений для допомоги розробникам у генерації фрагментів коду, функцій та навіть повноцінних програм на основі описів природною мовою [46]. Використовуючи велику мовну модель, CodeGen розуміє намір користувача та перетворює його на структурований, виконуваний код. Однією з найбільш значущих переваг цього інструменту є його безшовна інтеграція в редактор коду GitHub, що дозволяє розробникам генерувати та вдосконалювати код без необхідності перемикання контексту.

Ефективність CodeGen відображена в його показниках продуктивності: час генерації коду становить менше однієї секунди, а точність – 90%. Ці результати свідчать про те, що інструмент здатний миттєво створювати якісний, функціональний код, що суттєво підвищує продуктивність розробників. Крім того, з рівнем задоволення користувачів у 85% CodeGen отримав високу оцінку серед інженерів (табл. 2.3), оскільки зменшує час, необхідний для написання шаблонного коду, і прискорює цикли розробки.

Ще однією новаторською системою генерації коду на основі штучного інтелекту є Codex, розроблений компанією Anthropic. Подібно до CodeGen, Codex здатний створювати фрагменти коду, функції та повноцінні програми за описами природною мовою. Однак Codex виходить за межі стандартної генерації коду, забезпечуючи можливість виконання коду в рамках того самого середовища. Це дозволяє розробникам тестувати згенерований код і динамічно вдосконалювати результати.

Таблиця 2.3 – Показники продуктивності і їх значення CodeGen і Codex

Показник	Значення для CodeGen	Значення для Codex
Час генерації коду	< 1 секунди	< 2 секунд
Точність	90%	88%
Задоволеність користувачів	85%	87%

Codex інтегрований у кілька середовищ розробки, що робить його доступним широкому колу програмістів. Його ефективність підтверджується

часом генерації коду менше двох секунд і точністю 88%, що демонструє здатність забезпечувати коректні та корисні фрагменти коду. Крім того, рівень задоволеності користувачів становить 87%, що підтверджує його цінність у підвищенні швидкості розробки, зменшенні кількості помилок і покращенні робочих процесів програмного забезпечення (див. табл. 2.3).

Наступні результати дослідження узагальнюють кейси з оптимізації коду, які були проведені для інших моделей: CodeOpt, DeepCode, CodeOpt, CodeGuru. Ці інструменти використовують методи ШІ і мають особливості.

DeepCode це передовий інструмент оптимізації коду на основі ШІ, який використовує методи машинного навчання для аналізу, удосконалення та покращення коду. Він відіграє важливу роль у підвищенні якості програмного забезпечення шляхом виявлення неефективних ділянок, пропонування оптимізації та перебудови коду для покращення продуктивності.

DeepCode широко використовується в таких галузях, як фінанси, охорона здоров’я та технології, де оптимізований і безпомилковий код має критичне значення. Однією з ключових переваг DeepCode є здатність надавати рекомендації, засновані на даних, забезпечуючи ефективність, безпеку та легку підтримку коду. Метрики продуктивності показують, що DeepCode підвищує продуктивність коду на 25%, зменшує складність коду на 30% і знижує кількість помилок на 40%. Результати наведено в табл. 2.4, та само, як і для інших моделей.

Ці покращення значно підвищують надійність програмного забезпечення, роблячи DeepCode незамінним інструментом для розробників, які прагнуть до високої якості.

Таблиця 2.4 – Метрики оптимізації коду при застосуванні спеціальних моделей ШІ

Метрика	Значення DeepCode	Значення CodeOpt
Покращення продуктивності	25%	30%
Зменшення складності коду	30%	35%
Зменшення кількості помилок	40%	45%

CodeOpt – це інструмент оптимізації коду на базі ШІ, розроблений Microsoft. Він поєднує статичний та динамічний аналіз для виявлення неефективних ділянок коду та покращення продуктивності ПЗ. CodeOpt інтегровано в пакет інструментів Microsoft, включаючи Visual Studio, що дозволяє розробникам оптимізувати код прямо у робочих середовищах.

На відміну від традиційних інструментів оптимізації, CodeOpt використовує дані реального виконання програми, що допомагає розробникам краще розуміти вплив їхнього коду на продуктивність. Ефективність інструмента підтверджується такими показниками: 30% покращення загальної продуктивності коду, 35% зниження складності коду та 45% зменшення кількості помилок. Ці результати підкреслюють внесок CodeOpt у створення ефективнішого, зручнішого для підтримки та безпомилкового програмного забезпечення, що підвищує продуктивність розробників.

Наступні моделі застосовувалися для проведення ревю коду та ефективності (кількості) помилок: CodeQL, CodeGuru.

CodeGuru – інструмент на основі штучного інтелекту, розроблений AWS, призначений для автоматизації ревю коду та виявлення помилок. Він використовує машинне навчання для аналізу коду, виявлення потенційних проблем та надання практичних рекомендацій розробникам.

Сильною стороною CodeGuru є його інтеграція з інструментами AWS, що дозволяє здійснювати ревю в реальному часі прямо в робочому середовищі розробника. CodeGuru спрямований на виявлення помилок, вразливостей та «код-смелів», забезпечуючи надійність і безпеку ПЗ. Показники ефективності: 80% виявлення помилок, 50% зменшення код-смелів, 88% задоволеність користувачів, представлені в табл. 2.5.

CodeQL – це інструмент аналізу коду на основі ШІ, розроблений GitHub для виявлення помилок, вразливостей безпеки та неефективності. На відміну від традиційних засобів статичного аналізу, CodeQL дозволяє виконувати семантичні запити до коду, що дає змогу знаходити патерни, які вказують на ризики у великих репозиторіях.

CodeQL інтегрується в CI/CD-процеси GitHub, регулярно скануючи код на наявність проблем. Інструмент досягає 85% рівня виявлення помилок і знижує кількість вразливостей на 60%, рівень задоволеності – 90% (табл. 2.5).

Таблиця 2.5 – Метрики виявлення помилок при застосуванні спеціальних моделей ШІ

Метрика	Значення CodeGuru	Значення CodeQL
Рівень виявлення помилок	80%	85%
Зменшення код-смелів	50%	60%
Задоволеність користувачів	88%	90%

Поряд зі значними вигодами, які надає використання моделей ШІ, навчених для роботи з кодом, виникає і низка обмежень та перешкод. Найбільше це стосується якості даних та інтерпретованості результатів.

Одним із ключових викликів у використанні LLM для генерації та оптимізації коду є забезпечення якості та неупередженості навчальних даних.

Оскільки моделі навчаються на великих масивах відкритого коду та документації, у даних можуть міститися:

- неповні або застарілі фрагменти;
- упереджені коментарі;
- дисбаланс між різними стилями програмування;
- неправильні патерни поводження з безпекою.

Це може призвести до створення неефективного, помилкового або упередженого коду. Для зменшення ризиків варто:

- формувати різноманітні набори навчальних даних;
- застосовувати методи виявлення bias;
- проводити ручну перевірку (human-in-the-loop).

Без відповідних заходів LLM можуть підсилювати існуючі упередження й створювати етичні та технічні ризики у програмній інженерії.

Ще однією проблемою є низький рівень інтерпретованості LLM, що часто називають «чорним ящиком». На відміну від традиційного

програмування, де логіка чітко визначена, LLM видають результат на основі складних нейронних залежностей. Це ускладнює розуміння причин появи певного фрагмента коду або помилки [69]. Проблема є критичною у сферах із жорсткими вимогами до прозорості, таких як:

- медичне програмне забезпечення;
- автономні системи;
- фінансові застосунки.

Для розв’язання проблеми необхідні інструменти Explainable AI (XAI), зокрема:

- візуалізація attention-механізмів;
- аналіз шляхів прийняття рішень;
- правила та пояснення, що генерує сама модель.

Без цього прийняття LLM у відповідальних сферах може бути обмеженим.

Незважаючи на вражаючі можливості інструментів на основі LLM у автоматизованій генерації коду, виявленні помилок і оптимізації, їхнє безшовне впровадження в існуючі середовища розробки залишається складним завданням. Чимало команд розробників покладаються на застарілі системи, індивідуально налаштовані робочі процеси та специфічні для організації стандарти кодування, що ускладнює впровадження інструментів, керованих штучним інтелектом, без суттєвих змін. Розробники часто чинять опір різким змінам у робочих процесах, особливо якщо нові інструменти потребують додаткового навчання або порушують усталені практики. Проблеми затримок, обмеження сумісності та криві навчання також можуть перешкоджати впровадженню.

Для подолання цих труднощів інструменти на основі LLM повинні бути розроблені таким чином, щоб легко інтегруватися з популярними IDE, CI/CD-процесами та системами контролю версій. Наявність детальної документації, зручних інтерфейсів і можливостей налаштування може допомогти подолати

розрив між досягненнями штучного інтелекту та реальними потребами програмної інженерії.

На основі успіху мовних моделей в інших видах модальності, автори роботи [66] припустили, що спеціалізована модель GPT, яка називається Codex, може досягти успіху в різноманітному кодуванні завдань. Показано метод навчання моделі кодувальника-декодера на основі мільйонів виправлень помилок в історії змін проектів, розміщених на GitHub.

Увага була зосереджена на задачі генерації автономних Python-функцій з docstring та оцінюванні правильності зразків коду автоматично через юніт-тести. Це контрастує з генерацією природної мови, де зразки зазвичай оцінюються за допомогою евристики або людських оцінювачів. Для точного бенчмаркінгу обраної моделі створювався набір даних з 164 оригінальних програмних задач з юніт-тестами.

2.3 Характеристики вебдодатків, обраних для експерименту

Для дослідження можливостей та ефективності штучного інтелекту по оптимізації коду вебсайтів та їх редагування були використані кілька подібних вебсторінок, розроблених студентами в кінці проходження базового курсу вебтехнологій. При постановці завдань були враховані вимоги до функцій та якості дизайну комерційних продуктів, які висуваються реальними IT-компаніями, що спеціалізуються на веброзробці і співпрацюють з університетами. При розробці технічного завдання, плануванні дизайну і формуванні вірогідних вимог до змісту та функціональності вебсайтів застосовувався кейс-метод, який дозволяв зрозуміти реальні умови і цілі використання вебсайтів у реальному бізнесі.

В кінцевому варіанті всі сайти були виконані у формі лендінгу, являли собою шаблони для комерційних промо-сайтів різних сфер бізнесу: кав’ярня, школа іноземних мов, енергозбереження та інші. Кожен сайт складався з 4-6

тематичних підрозділів, містив головний банер, меню для навігації, слайдер, відгуки про компанію, форму для контактів користувачів та інше. Макети були розроблені здебільшого за допомогою Figma. Блочна структура кожної сторінки була побудована на основі HTML, при цьому використовувалися, окрім основних, семантичні теги типу <nav>, <footer> <section>, <main>, <aside> і універсальні , . Розробка проводилася в середовищі VS Code. Фрагмент робочої області показано на рис. 2.5.

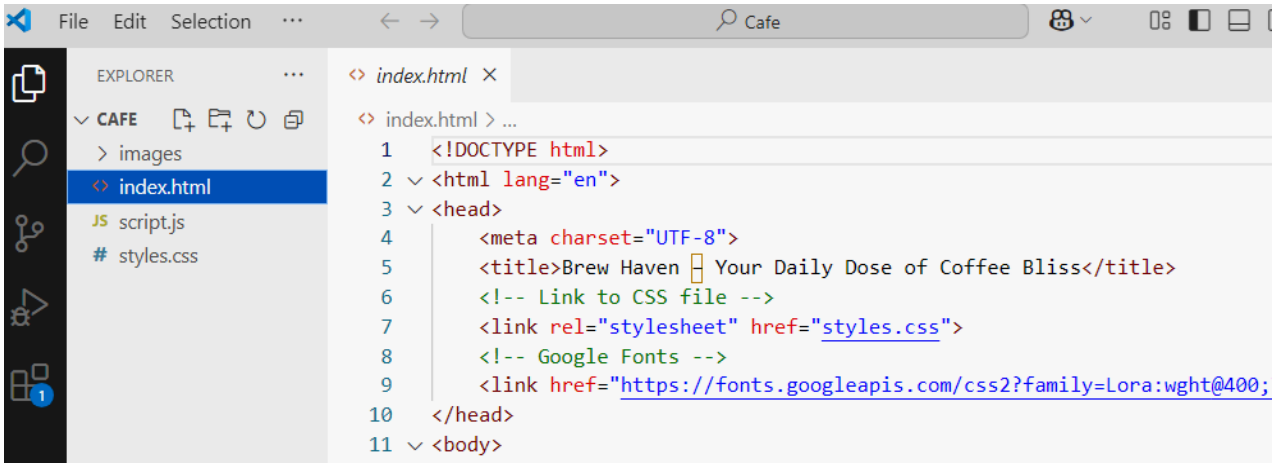

Рисунок 2.5 – Зміст HTML-файлу в середовищі VS Code і структура проєкту для сайту кав’ярні

Довжина HTML-коду вебдодатку складає 160 рядків з тегами. Вигляд дизайну та змісту верхньої частини одного із сайтів показано на рис. 2.6.

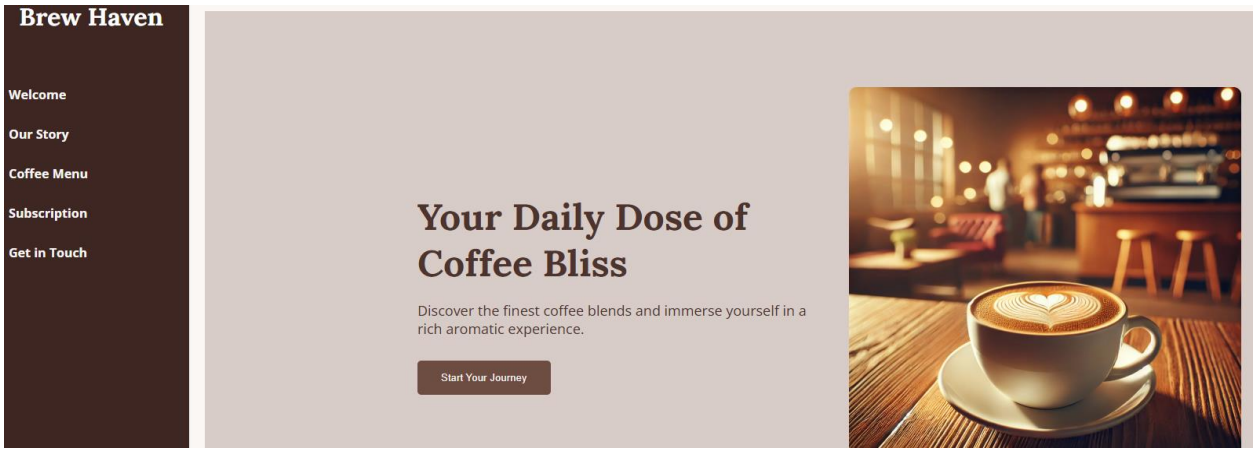

Рисунок 2.6 – Візуальне зображення дизайну верхньої частини і бокового меню для сайту-лендінгу кав’ярні

Одним із оригінальних рішень є блок бокового меню, яке залишається на сторінці постійно. Розмітка меню показана на рис. 2.7.

```
<nav id="side-nav">
    <div class="logo">
        <h1>Brew Haven</h1>
    </div>
    <ul>
        <li><a href="#home">Welcome</a></li>
        <li><a href="#story">Our Story</a></li>
        <li><a href="#menu">Coffee Menu</a></li>
        <li><a href="#subscription">Subscription</a></li>
        <li><a href="#contact">Get in Touch</a></li>
    </ul>
    <div class="social-media">
        <a href="#"><img src="images/icon-facebook.svg" alt="Facebook"></a>
        <a href="#"><img src="images/icon-instagram.svg" alt="Instagram"></a>
        <a href="#"><img src="images/icon-twitter.svg" alt="Twitter"></a>
    </div>
</nav>
```

Рисунок 2.7 – Розмітка блоку меню для сайту кав’ярні

Властивості блоків були описані за допомогою стилів CSS, при цьому використані тегові селектори, класи .class, ідентифікатори #id із власними іменами, а також спеціальні функції для трансформації зображень типу a: hover, Scale. Для адаптивності використовували властивість flex і її функції. Наприклад, на рис. 2.8. показано опис властивостей меню при наведенні миші.

```
#side-nav ul li a {
    display: block;
    color: #faf7f5;
    padding: 15px 20px;
    text-decoration: none;
    font-weight: bold;
    transition: background-color 0.3s;
}

#side-nav ul li a:hover {
    background-color: #5d4037;
}
```

Рисунок 2.8 – CSS-властивості для пунктів меню при наведенні та активації

При цьому вирішувалися завдання раціоналізації коду, наприклад, з урахуванням каскадності та успадкування властивостей класу. На рис. 2.9 показаний розділ сайту заголовком Coffee Menu з блоками зображень напоїв.

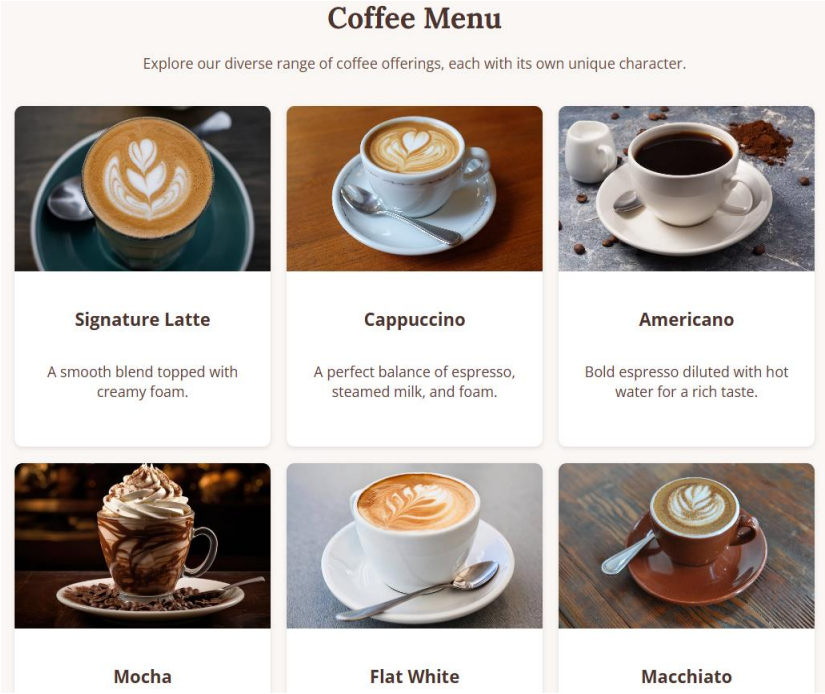

Рисунок 2.9 – Дизайн середньої частини промо-сайту для кав’ярні

Всі блоки описані типово в класі .card. На рис. 2.10 представлено код із описом CSS-властивостей всіх типових елементів (рисунків і тексту), які належать одному класу.

```
.card {
    background-color: #fff;
    width: 300px;
    margin: 20px;
    border-radius: 10px;
    overflow: hidden;
    text-align: left;
    box-shadow: 0 2px 6px rgba(0,0,0,0.1);
}

.card img {
    width: 100%;
    height: 200px;
    object-fit: cover;
}

.card h3 {
    padding: 20px;
    font-size: 24px;
    color: #4e342e;
}

.card p {
    padding: 0 20px 20px;
    color: #5d4037;
```

Рисунок 2.10 – Зміст каскадних таблиць стилів у styles.css для опису елементів, що належать класу .card

Завдяки каскадності стилів, код з описом CSS вийшов компактним і керованим. В основному класі .card для кожного з блоків секції визначено форму, розмір, фон background-color: #fff;, відступи, заокруглену форму (border-radius: 10px;), а також тінь (box-shadow: 0 2px 6px rgba(0,0,0,0.1); та деякі інші елементи.

За допомогою JavaScript були розроблені окремі інтерактивні елементи: спливаючі вікна повідомлень, каруселі та ін. Загалом, частина HTML&CSS була виконана доволі якісно, академічно. Кнопки, меню, форми, таблиці, графіки, карти та інші інтерактивні компоненти можуть бути створені на сторінках за допомогою JavaScript. Застосування JavaScript на початковому рівні викликає труднощі, оскільки потребує вивчення значного обсягу синтаксису, використання змінних, функцій. Тому доводиться використовувати готові шаблони, приклади з довідкових сайтів.

Метою спілкування зі штучним інтелектом було вирішення таких задач, як отримання зовнішнього аналізу якості розроблених вебсайтів, знаходження різних помилок, отримання пропозицій щодо варіантів удосконалення і доповнення. При цьому були зафіксовані всі діалоги для подальшого оцінювання їх якості, точності, достатності, а також відповіді. Спілкування відбувалося через ChatGPT.

Висновки до розділу 2

Інтеграція штучного інтелекту та великих мовних моделей у сферу розробки програмного забезпечення становить трансформаційний зсув у тому, як створюється, підтримується та оптимізується програмний код. Розглянуті моделі ШІ демонструють вражаючі можливості в автоматизації генерації коду, оптимізації продуктивності програм та підвищенні ефективності процесів ревю коду. Використовуючи LLM, розробники можуть суттєво підвищити продуктивність, скоротити час виходу продукту на ринок та покращити якість

коду. Інструменти на основі ШІ також відіграють важливу роль у тому, щоб зробити розробку ПЗ більш доступною, оскільки дозволяють людям із мінімальним досвідом програмування генерувати та вдосконалювати код за допомогою взаємодії природною мовою.

Попри переваги, впровадження LLM у розробку програмного забезпечення не позбавлене викликів. Однією з найкритичніших проблем є якість даних та упередженість. Оскільки LLM значною мірою залежать від даних, на яких їх тренують, будь-які помилки або упередження в цих наборах даних можуть призвести до субоптимальної генерації коду, вразливостей безпеки або до відтворення наявних упереджень у практиках програмування. Тому забезпечення різноманітності, високої якості та ретельної підготовки тренувальних наборів даних є необхідною умовою для надійності та справедливості моделей.

LLM часто працюють як «чорні скриньки», що ускладнює для розробників розуміння того, як і чому були згенеровані певні результати. Така відсутність прозорості може обмежувати використання цих систем у критично важливих застосуваннях, де важливі відповідальність, можливість налагодження та дотримання стандартів.

Загалом, ідея генерації та оптимізації, тестування складних кодів показала безумовну перспективність і потребує подальших досліджень.

РОЗДІЛ 3
РЕЗУЛЬТАТИ ВИПРОБУВАННЯ ВІДОМИХ МОВНИХ МОДЕЛЕЙ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ОПТИМІЗАЦІЇ КОДУ ВЕБДОДАТКУ

3.1 Результати експерименту при використанні ChatGPT в ролі AI-асистента при розпізнанні вебкоду і його оптимізації

У практичній частині представлені результати проведеного експерименту, основною метою якого було дослідити можливості популярних великих мовних моделей LLM ШІ 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.

Спілкування проводилося на основі ChatGPT, який є генеративним чат-ботом зі штучним інтелектом, був розроблений OpenAI та випущений 30 листопада 2022 р. [67]. Він використовує великі мовні моделі, такі як GPT-4o, для генерації людських відповідей у тексті, мові та зображеннях.

Перший запит (для різних вебсайтів виконувався різними користувачами) включав, перш за все, опис програмного продукту і задачу, наприклад: «Я створив сайт-лендінг на тему (названо загальну тему або вид бізнесу). Сайт у формі SPA, використано мову HTML, CSS і JavaScript. Сайт має навігацію, слайдер, блокові текстові елементи (перелік інших елементів). Проаналізуй сайт і запропонуй, що можна тут покращити».

Відповідь 1 ШІ: «Звісно, допоможу! Завантаж необхідний код або всі файли». Були завантажені просто файли сайту безпосередньо в чат.

Результат обробки файлів ШІ складався з трьох блоків повідомлень. Перший блок відповіді включав перелік досягнень, тобто сильні сторони роботи. Для переважної більшості сайтів були відмічені такі позитивні сторони (текст збережено від ChatGPT):

– структура у формі SPA як універсальна та легка для роботи в браузері;

– стильна навігація – фіксована зверху, з плавним скролом;

– витриманий стиль і дизайн (якщо все відповідало тематиці);

– наявність анімації (слайдер, карусель або інше було точно названо);

– адаптивність.

Другий блок містив перелік пропозицій для вдосконалення по критеріям:

1. SEO та доступність:

– в <html lang="ua"> краще замінити "ua" на "uk" (офіційний ISO-код для української);

– додай alt-описи до всіх зображень, зокрема до банерного фону (через aria-label або інші способи), якщо хочеш зробити сайт дружнім до екранних читалок;

– додай <meta name="description">.

2. Адаптивність на мобільних:

– в деяких блоках можна додати snap-x snap-mandatory overflow-x-auto — тоді слайдери будуть працювати свайпом на мобілках.

– у футері трохи забагато колонок — на мобільних це може бути надмірно. Варто зробити їх в 2 ряди або акордеони.

3. Функціональність слайдера (в кількох сайтах):

– зараз він не нескінченний, тобто рухається між 0, 1, 2, але не в обидва боки циклічно. Далі запропоновано фрагмент, приклад, як переробити, щоб він рухався циклічно;

– додати плавний перехід між слайдами: «Ти вже додаєш transition-transform, але для ще кращого UX можна додати easing»:

```
#slider {
  transition: transform 0.5s ease-in-out;
}
```

Третій блок містив різні пропозиції від ШІ: «Якщо цікаво, можу додати (далі перелік пропозицій)».

Далі потрібно було вести діалог щодо конкретних пропозицій і називати їх. Для прикладу, в одному з випадків погодилися із пропозицією зробити

адаптивне меню у вигляді гамбургера. Промпт виглядав так: « так, створи для адаптивності меню гамбургер-меню».

ШІ сформував послідовно фрагменти для змін HTML-коду, властивості CSS (рис. 3.1), а також в програмний код JavaScript (рис. 3.2), які були впроваджені в код сайту для кав'ярні методом копіювання.

Далі слідувала пропозиція додати ефект анімації (перетворення трьох рисочок на хрестик), яку також легко було використати.

```
460  /* Hamburger Styles */
461  #hamburger {
462      display: none;
463      flex-direction: column;
464      justify-content: center;
465      gap: 5px;
466      background: none;
467      border: none;
468      cursor: pointer;
469      margin-bottom: 20px;
470  }
471
472  #hamburger .bar {
473      width: 25px;
474      height: 3px;
475      background-color: #faf7f5;
476      transition: all 0.3s;
477  }
```

Рисунок 3.1 – Додавання властивостей гамбургер-меню в ідентифікатор `#hamburger`, розроблений за допомогою ChatGPT

```
41  /* Hamburger menu toggle */
42  const hamburger = document.getElementById('hamburger');
43  const navLinks = document.querySelector('#side-nav ul');
44
45  hamburger.addEventListener('click', () => {
46      navLinks.classList.toggle('active');
47  });
```

Рисунок 3.2 – Зміст скрипта від ChatGPT для згортання меню в гамбургер

При проведенні попереднього аудиту розробленого промо-сайту отримали вигляд згорнутого меню, як показано на рис. 3.3.

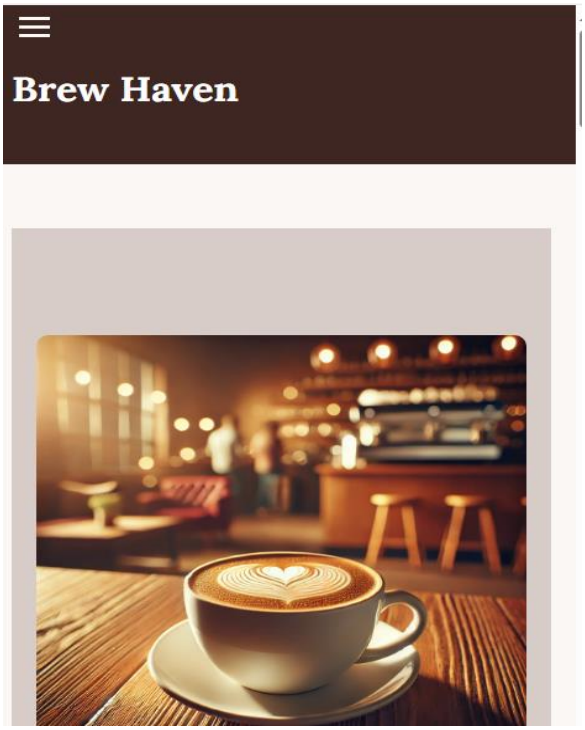

Рисунок 3.3 – Вигляд згорнутого меню у верхній частині сайту

Один із сайтів, використаних для експерименту, відрізнявся від попередніх: в ньому замість окремого файлу таблиць стилів для опису елементів був застосований CSS-фреймворк Tailwind. Його використання саме по собі сприяє оптимізації сайту, якщо це лендінг. Завдяки цьому фреймворку вдається уникнути дублювання класів, він має вбудовану систему breakpoints та дизайн-систему за замовчуванням: кольори, відступи, типографіка, шрифти вже узгоджені. Tailwind дає змогу стилізувати елементи просто в HTML-коді за допомогою класів., хоча це в певній мірі захаращує HTML. Наприклад, опис кнопки виглядає так:

```
<button class="bg-blue-500 text-white px-4 py-2 rounded hover:bg-blue-600">
```

Повний код вебсайту, записаний в єдиному файлі HTML був завантажений в ChatGPT з проханням проаналізувати та розділити висхідний код на три складових і винести всі описи стилів в окремий файл. Особливості цього файлу були розпізнані ШІ, він погодився розділити код сторінки на структуру, стилі та JS. Однак, після кількох спроб ця задача так і не була вирішена коректно. Навіть при створенні зовнішнього файлу .CSS вся

висхідна структура зберігалася із вбудованими стилями. Зі збільшенням кількості версій переробленого сайту ШІ починав свої варіації зі змістом, додавав все нові елементи і, зрештою, довів код до нечитабельного й абстрактного сюжету. При цьому почали накладатися задачі, які були реалізовані раніше в тому ж аккаунті користувача і зберігалися в архіві. Можна припустити, що ШІ підлаштовується під потреби користувача, якщо в нього з'являються подібні завдання.

Коли було вдруге запропоновано виділити в окремий файл лише скрипти, то ШІ оцінив цю задачу як важливий крок до масштабованості та оптимальності, провів коротке обґрунтування правильності такої операції та навів пропозиції рефакторингу.

3.2 Результати експерименту з використання Copilot для аналізу, оптимізації, рефакторингу коду вебсайтів

Аналогічний експеримент для порівняння проведено за допомогою Copilot. Розробником є GitHub (власність Microsoft) спільно з OpenAI. Copilot базується на мовній моделі OpenAI Codex (розширена версія GPT) і є генеративним штучним інтелектом для автодоповнення коду. За призначенням він підтримує десятки мов: Python, JavaScript, TypeScript, Go, C++, Java, HTML, CSS та інші, інтегрований зі спеціалізованими IDE VS Code, Visual Studio, Neovim, JetBrains IDE та інші. Спілкування також відбувалося через Copilot чат. На відміну від ChatGPT, через чат можна було завантажити лише лістинги коду, оскільки формат готових файлів не підтримувався.

Аналіз коду кожної частини кожного вебсайту був досить загальним і описовим, без технічної деталізації досягнень розробника. Однак, пропозиції щодо удосконалення коду були не менш конструктивними, а часто й більш детальними, з коментарями і готовими до впровадження фрагментами.

Пропозиції для SEO-оптимізації так само включали рекомендації:

– додати мета-теги: meta name="description" для опису сайту та meta name="viewport" для адаптивності;

– забезпечити достатній контраст між текстом і фоном (можна перевірити через tools like WebAIM).

Оптимізація в частині Accessibility:

– додати aria-label до посилань соцмереж і форм (наприклад, до aria-label="Your name");

– забезпечити достатній контраст між текстом і фоном (можна перевірити через tools like WebAIM).

Copilot сформував більше пропозицій для покращення за результатами тестування декількох вебсайтів і одразу подавав обґрунтування та оновлений або генерував свій код, що полегшувало прийняття рішень.

Функціональні покращення від Copilot стосувалися коду JavaScript:

– Smooth Scroll з урахуванням висоти навігації: щоб адаптуватись до змінної висоти меню (як от на мобільних), варто динамічно обчислювати її;

– очистка повідомлення після надсилання форми і фрагмент коду;

– додати валідацію (перевірку на порожні поля або email-формат), якщо планується інтеграція з сервером.

На рис. 3.4 показано відповідний код для прокрутки, запропонований ШІ, в оригінальній редакції:

```
const navHeight = document.getElementById('side-nav').offsetHeight;
window.scrollTo({
    top: targetSection.offsetTop - navHeight,
    behavior: 'smooth'
});
```

Рисунок 3.4 – Код для динамічного обчислення висоти меню при навігації

Покращення для UX-дизайну стосувалися кнопок закриття, додавання темної теми та анімації гамбургер-меню (ті самі, що і в ChatGPT).

Загальні рекомендації дозволяли зробити сайти іще більш професійними та технологічно досконалими, наприклад:

– створити модульну структуру JS: винести частини коду в окремі функції або файли (navigation.js, form.js) для зручної підтримки;

– якщо буде багато інтерактивну, можна додати невеличку бібліотеку, наприклад GSAP для анімації або Validate.js для форм;

– додати обробку помилок, якщо scroll target не знайдено, або форма не заповнена.

Загалом, робота з доопрацювання й оптимізації коду, або його рефакторингу була оцінена всіма учасниками експерименту доволі високо. В ході експерименту досягли результату щодо зручності й швидкості перевірки вірності коду та виправлення неточності, внесення цікавих нюансів, які сприяють більш професійному виконанню кінцевого програмного продукту у вигляді вебсайту. Оцінка роботи за 10-бальною шкалою з двома мовними моделями штучного інтелекту на основі GPT-моделі показана в табл. 3.1. Для кожної з моделей показано усереднений бал по кожному з критеріїв за результатами суб’єктивних оцінок 24 учасників експерименту.

Таблиця 3.1 – Порівняльні оцінки зручності та повноти окремих результатів GPT-моделей ШІ при роботі з кодами вебсайтів

Зміст критерію	ChatGPT	GitHub Copilot	Примітки
Розробник та призначення в програмуванні	OpenAI, генерація, пояснення, виправлення коду, а також діалог	GitHub + OpenAI (власність Microsoft), більше застосовується для автодоповнення коду в IDE	Довідкова інформація
Зручність і комфорт ведення бесіди в чаті	10 основна функція ChatGPT	8 Наявність режиму часту як додаткова функція	Враховано меню в чаті, збереження розмов, структурування відповіді і т. ін.
Способи завантаження даних для аналізу	10 Лістинг, частини коду, файли різних типів, архіви	8 Лістинг коду, фрагмент коду на мові програмування	На основі використаних форматів завантаження коду

Продовження таблиці 3.1

Здатність розуміти призначення чужого коду і розбір, пояснення алгоритмів	10 Повний аналіз зі структурою	8 Достатнє розуміння, обмежене пояснення алгоритмів	Завантаження вебсайтів на різну тематику
Генерація коду за результатами розбору коду	10 Швидко і коректно	10 Швидко, детально, з коментарями	Генерація коду відбувалася після погодження з розробником
Проведення заходів оптимізації чужого коду	10 Всі види оптимізації	10 Всі види оптимізації	Оцінювалося розуміння всіх складових оптимізації та зміст пропозицій
Ефективність як наставника	10 Діє як досвідчений вчитель	9 Дає пояснення, розкладає власний код, але обмежено	Для навчання потрібно опанувати основи програмування, макетування сайтів, щоб спілкуватися із ШІ на зрозумілій мові та чітко формулювати завдання

Як бачимо, GPT-моделі показали високе розуміння змісту і призначення коду, здатність проводити синтаксичний аналіз, ефективно справлялися з пошуком недосконалостей в роботах, пропонували шляхи покращення.

3.3 Оцінювання ефективності розробки вебдодатку за участю ШІ

В процесі підготовки кваліфікаційної роботи була проведена серія експериментів з оптимізації програмного коду вебсайтів, 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 ШІ може бути економічно обгрунтованою альтернативою для створення оптимізованого коду на етапі розробки (завершення, тестування) коду вебсайту порівняно з послугами комерційних маркетингових компаній, які проводять оптимізацію вебдодатків.

Для економічного аналізу розробки сайту використаємо витратний (Cost-based) метод.

Для визначення вартості розробки вебсайту спочатку проведемо розрахунок трудомісткості. Враховуємо, що були використані технології HTML&CSS JavaScript, Visual Studio Code, Figma. Роботу виконував один фронтенд розробник, графічні елементи та дизайн підготував графічний дизайнер. В розрахунках враховуємо:

– роботу фронтенд-розробника (погодинні ставки);

– графічного дизайнера;

– використання інструментів (VS Code, Figma – безкоштовні/умовно-безкоштовні);

– час виконання.

Самим важливим пунктом при розробці продукту є розрахунок його собівартості й визначення кінцевої ціни, оскільки саме вона має основну вагу при визначенні переваг над конкурентами та доцільності розробки системи.

Для розрахунків затрат на оплату праці візьмемо за основу, що розробник працює по стандартному графіку: восьмигодинний робочий день, п’ять днів на тиждень. Оплата праці – погодинна. Вартість години роботи може бути різна в залежності від наступних факторів:

– область застосування системи;

– обсяг застосовуваних інструментів та методів;

– термін виконання;

– складність завдання.

Оплату праці розробника приймаємо 200 грн/год (на DOU) [70]. На розробку системи в нашому проєкті виділено 2 тижня (10 робочих днів).

Витрати на оплату праці графічного дизайнера середнього рівня складають 180 грн/год, при цьому він працював 40 год, включно з усіма фото. Додатково в проєкт закладається очікувана норма запасу, рівна 20 %.

Раніше було встановлено, що розробник буде працювати по 8 годинному графіку з оплатою 200 грн/год. Формула розрахунку витрат оплати праці:

$$З_п = (К_{днів} \cdot В_{роб}) \cdot Т_г, \quad (3.1)$$

де $З_п$ – Витрати на оплату праці спеціалістів;

$К_{днів}$ – кількість робочих днів;

$В_{роб}$ – тривалість робочого дня в годинах;

$Т_г$ – погодинна ставка розробника.

Підрахунок оплати праці розробника становить $З_{пр} = (10 \cdot 8) \cdot 200 = 16000$ грн. Оплата праці дизайнера становитиме $З_{пд} = (5 \cdot 8) \cdot 180 = 7200$ грн. Загальний підрахунок витрат представлено в табл. 3.2.

Таблиця 3.2 – Розрахунок прямих витрат на розроблення вебсайту

№ з/п	Види витрат на розробку і впровадження технології	Витрати часу (од.)	Грошовий еквівалент робіт, грн/од.часу	Сумарна вартість, грн
1	Розроблення макету вебсайту в програмі Figma (ПЗ безкоштовно) і фото	40 год.	180	7200
2	Розроблення сайту (програміст)	80 год.	200	16000
3	Оплата домену (хостинг) від компанії TheHost, пакет «Діловий»	1 рік	799	799
4	Вартість домену (Міжнародний інформаційний домен Info) на рік	1 рік	249	249
5	Вартість послуг інтернет за період роботи (виготовлення сайту)	14 днів	20	280
6	Сумарна вартість індивідуального замовлення вебсайту	=(п.1+п.2+ п.3+ п.4+ п.5)		24528

Дані, які використані для підрахунку витрат на розроблення вебсайту, було взято з відкритих джерел. Зокрема, актуальні дані про хостинг взято із вебсайту компанії Thehost [71].

Таким чином, загальна вартість виготовлення вебсайту одноосібним графічним дизайнером (макет сайту) і вебпрограмістом складе близько 24528 грн станом на початок 2025 р. Час та вартість на тестування вебсайту перед передачею на хостинг знаходиться в межах 10-20 % часу, відведеного на програмування.

Використання ШІ для генерації коду вебсторінок стає все більш затребуваним завдяки здатності оптимізувати процеси розробки. Основні

економічні переваги застосування ШІ в розробці вебсторінок можна розглянути в кількох ключових аспектах.

1. Зниження витрат на розробку: ШІ здатний автоматизувати процес написання коду, що зменшує обсяг ручної праці та скорочує час, необхідний для виконання рутинних завдань. Це дозволяє компаніям економити на витратах, пов’язаних з наймом додаткових програмістів. Наприклад, ШІ може генерувати HTML, CSS або JavaScript код для типових компонентів, таких як кнопки, форми чи інтерактивні елементи, що суттєво зменшує робоче навантаження. З огляду на розцінки, економічний ефекти на розробці може скласти від 20% на програмуванні, тобто 4906 грн (або $120 по курсу НБУ станом на 01.11.2025 р.), враховуючи безкоштовний доступ до ШІ.

2. Підвищення продуктивності та швидкості розробки. Інструменти ШІ дозволяють розробникам працювати швидше, оскільки вони можуть генерувати готові фрагменти коду на основі коротких запитів або описів функцій. Це означає, що проекти завершуються швидше, що важливо для компаній, які хочуть швидше виводити свої продукти на ринок. Швидкість розробки особливо критична в галузі, де нові продукти і функції повинні швидко відповідати на зміни в потребах користувачів та ринкові тенденції.

3. Оптимізація якості коду: ШІ може використовуватися для перевірки коду на помилки, дотримання стандартів або оптимізацію продуктивності, що знижує витрати на тестування та покращує якість кінцевого продукту. Наприклад, інструменти ШІ 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, тобто, орієнтовно 2453 грн.

Таким чином, навіть часткове залучення ШІ дозволяє отримати економію коштів SUM = 4 905,6 + 2453 = 7359 грн.

По відношенню до вартості цілого проєкту економія складе:

7359÷24528×100 = 30 %.

Окрім фінансових вигод розглядаються також і організаційні переваги.

4. Інновації та конкурентна перевага: використання ШІ дозволяє компаніям тестувати нові ідеї без значних інвестицій в ресурси, а також бути більш гнучкими до змін. ШІ 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.

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 отримані результати експерименту з використання мовних моделей ШІ для оптимізації і написання коду згідно зразка або визначеного завдання.

Провівши різнопланові дослідження варіантів залучення великих мовних моделей штучного інтелекту на прикладі ChatGPT та Copilot у процесі удосконалення і оптимізації кодів вебсайтів, отримали низку позитивних результатів. Обидві залучені в ході експерименту моделі можуть бути ефективними для програмістів у написанні, завершенні, поясненні та рефакторингу коду.

З технічної точки зору були визначені або підтверджені аналогічні результати інших досліджень, що Copilot більш органічно працює для доповнення та завершення коду, надає підказки в IDE, завдяки інтегрованості

в середовища розробки. Це потужний фактор, який дозволяє уникнути простих помилок на етапах написання коду.

Обидві моделі є ефективними для аналізу готового коду і формування пропозицій щодо SEO-оптимізації та технічної оптимізації. У процесі навчання важливо бачити перспективу покращення власної роботи, зростання рівня майстерності. В цьому велику допомогу може надати ШІ як зовнішній аудитор та помічник. Всі пропозиції, які були надані для модернізації або доповнення властивостей елементів, розширення функціоналу у вигляді фрагментів готового коду, були легко інтегровані та сприяли покращенню оригінальності кінцевого програмного продукту.

Однозначний висновок 95% всіх учасників експерименту, що використання ШІ для генерації повністю готового коду без набуття власних знань і тренування навичок у самостійному програмуванні є тупиковим. Натомість, детальний аналіз пропозицій, розбір коду по частинам сприяє навчанню і розкриває вектори подальшого руху. В цьому сенсі ChatGPT – чудовий як асистент-наставник, аналітик і тлумач. Добре підходить для навчання, обговорення, дебагу, роботи з алгоритмами та великими фрагментами коду. Однак, ШІ також може помилятися або невірно вас розуміти.

Все, сказане в даній роботі, означає формувати відношення до ШІ як до корисного професійного інструменту, який суттєво підвищує продуктивність програмістів**,** зменшує час на рутинні задачі та допомагає навчатися. Однак він не замінює повноцінного розробника, іноді помиляється в контексті і потребує критичного підходу до використання його підказок.

Подальшими напрямками досліджень можуть бути цікавими експерименти з кодами, в яких застосований більш складний набір технологій, інші мови програмування, а також генерація графічних зображень для графічного дизайну вебсайтів.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проаналізовано значну кількість інструментарію веброзробника, який впливає на якість та оптимізацію вебдодатків. Підсумовуючи результати проведеного дослідження, яке стосувалося пошуку можливих шляхів технічної оптимізації програмних кодів, які постають перед програмістами при розробці вебдодатків, можна сформулювати наступні висновки.

1. Значна кількість публікацій у наукових базах даних підтверджує зростання інтересу до тематики ШІ, але перспективи їх використання в реальних умовах залишаються неоднозначними.

2. Конвергенція штучного інтелекту та розробки програмного забезпечення має потенціал для значного скорочення необхідних ресурсів, покращення якості та покращення взаємодії з користувачем за допомогою більш інтелектуальних програм, орієнтованих на користувача.

3. Провівши різнопланові дослідження варіантів залучення великих мовних моделей штучного інтелекту на прикладі ChatGPT та Copilot у процесі удосконалення і оптимізації кодів вебсайтів, отримали низку позитивних результатів. Обидві залучені в ході експерименту моделі можуть бути ефективними для програмістів у написанні, завершенні, поясненні та рефакторингу коду.

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.

5. З технічної точки зору були визначені або підтверджені аналогічні результати інших досліджень, що Copilot більш органічно працює для доповнення та завершення коду, надає підказки в IDE, завдяки інтегрованості в середовища розробки. Це потужний фактор, який дозволяє уникнути простих

помилок на етапах написання коду. Обидві моделі є ефективними для аналізу готового коду і формування пропозицій щодо SEO-оптимізації та технічної оптимізації.

Проведено економічний аналіз, порахована орієнтовна вартість реалізації розробки вебдодатку для середнього бізнесу. На основі оцінок визначено, що застосування штучного інтелекту при оптимізаці й частковій генерації коду для вебсторінок може значно підвищити ефективність розробки, знизити витрати (за оцінками від 7360 грн або 30 %), прискорити вихід на ринок та покращити якість продукту.

Результати дослідження опубліковані в збірниках матеріалів конференцій різного рангу. Копії повного тексту публікацій представлено в додатку В.

Подальшими напрямками досліджень можуть бути цікавими експерименти з кодами, в яких застосований більш складний набір технологій, інші мови програмування, а також генерація графічних зображень для графічного дезайну вебсайтів.

Запропонована методика досліджень може бути корисною для досягнення оптимізації кодів більшої складності з мінімальними витратами часових і людських ресурсів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bini SA. Artificial intelligence, machine learning, deep learning, and cognitive computing: what do these terms mean and how will they impact health care? *The Journal of Arthroplasty*. 2018. Vol. 33. Issue 8. P. 2358–2361. doi: 10.1016/j.arth.2018.02.067.

2. Gbashi, S.; Njobeh, P.B. Enhancing Food Integrity through Artificial Intelligence and Machine Learning: A Comprehensive Review. *Applied Science*. 2024. 14. 3421. https:// doi.org/10.3390/app14083421

3. Shah Romil F., Martinez Alejandro M., Pedoia Valentina, Majumdar Sharmila, Vail Thomas P., Bini Stefano A. Variation in the Thickness of Knee Cartilage. The Use of a Novel Machine Learning Algorithm for Cartilage Segmentation of Magnetic Resonance Images. *The Journal of Arthroplasty.* 2019. Vol. 34(10). P. 2210–2215. doi: 10.1016/j.arth.2019.07.022.

4. Cannataro, M.; Guzzi, P.H.; Agapito, G.; Zucco, C.; Milano, M. Chapter 3—Artificial Intelligence. In *Artificial Intelligence in Bioinformatics*; Cannataro, M., Guzzi, P.H., Agapito, G., Zucco, C., Milano, M., Eds.; Elsevier: Amsterdam, The Netherlands, 2022. P. 29–33. ISBN 978-0-12-822952-1.

5. Galanos, V. Expectations and Expertise in Artificial Intelligence: Specialist Views and Historical Perspectives on Conceptualisation, Promise, and Funding. Ph.D. Thesis, University of Edinburgh, Edinburgh, UK, 2023.

6. Delipetrev, B.; Tsinaraki, C.; Kostic, U. Historical Evolution of Artificial Intelligence. JRC Publications Repository. URL: https://publications.jrc.ec.europa.eu/repository/handle/JRC120469 (дата звернення: 04.12.2025).

7. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention Is All You Need. In Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017.

8. Topol Eric J. High-performance medicine: the convergence of human and artificial intelligence. *Nature Medicine*. 2019. Vol.25. Issue 1. P.44–56. doi: 10.1038/s41591-018-0300-7.

9. Flowers, J.C. Strong and Weak AI: Deweyan Considerations. *In Proceedings of the AAAI Spring Symposium: Towards Conscious AI Systems, Palo Alto, CA, USA, 25–27 March 2019*; 2019. Volume 2287.

10. Fjelland, R. Why General Artificial Intelligence Will Not Be Realized. *Humanit. Soc. Sci. Commun.* 2020. Vol. 7. P.1–9.

11. Kuusi, O.; Heinonen, S. Scenarios from Artificial Narrow Intelligence to Artificial General Intelligence—Reviewing the Results of the International Work/Technology 2050 Study. *World Futures Rev.* 2022. Vol. 14. P. 65–79.

12. Naylor CD. On the Prospects for a (Deep) Learning Health Care System. *JAMA.* 2018;320(11):1099–1100. doi:10.1001/jama.2018.1110

13. Haeberle Heather S., Helm James M., Navarro Sergio M., Karnuta Jaret M., Schaffer Jonathan L., Callaghan John J., Mont Michael A., Kamath Atul F., Krebs Viktor E., Ramkumar Prem N. Artificial Intelligence and Machine Learning in Lower Extremity Arthroplasty: A Review. *The Journal of Arthroplasty*. 2019. Vol. 34(10). P. 2201–2203. doi: 10.1016/j.arth.2019.05.055.

14. Gbashi, S.; Njobeh, P.B. Enhancing Food Integrity through Artificial Intelligence and Machine Learning: A Comprehensive Review. *Appl. Sci.* 2024. Vol.14. P. 3421. https:// doi.org/10.3390/app14083421

15. PMC Copyright Notice. https://pmc.ncbi.nlm.nih.gov/about/copyright/ (дата звернення: 04.12.2025).

16. Deng, X.; Cao, S.; Horn, A.L. Emerging Applications of Machine Learning in Food Safety. *Annual Revu Food Science Technol.* 2021. 12. P. 513–538.

17. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog*. 2019. 1. p. 9.

18. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. Adv. Neural Inf. Process. Syst. 2020. Vol. 33. P. 1877-1901.

19. Min, B.; Ross, H.; Sulem, E.; Veyseh, A.P.B.; Nguyen, T.H.; Sainz, O.; Agirre, E.; Heintz, I.; Roth, D. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Comput. Surv*. 2023.56. P.1–40.

20. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *In Proceedings of the 31st Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017*. 2017. p. 6000–6010.

21. Wandelt, S., Zheng, C., Wang, S., Liu, Y., & Sun, X. Large Language Models for Intelligent Transportation: A Review of the State of the Art and Challenges. *Applied Sciences.* 2024. *14*(17). P. 7455. https://doi.org/10.3390/app14177455

22. Chen, H.; Jiao, F.; Li, X.; Qin, C.; Ravaut, M.; Zhao, R.; Xiong, C.; Joty, S. ChatGPT's One-year Anniversary: Are Open-Source Large Language Models Catching up? *arXiv.* 2023. arXiv:2311.16989 https://doi.org/10.48550/arXiv.2311.16989.

23. Rau, D.; Kamps, J. Query Generation Using Large Language Models: A Reproducibility Study of Unsupervised Passage Reranking. *In Proceedings of the European Conference on Information Retrieval, Glasgow, Scotland, UK, 24–28 March 2024*. 2024. pp. 226–239.

24. Ziems, C.; Held, W.; Shaikh, O.; Chen, J.; Zhang, Z.; Yang, D. Can Large Language Models transform computational social science? *Comput. Linguist.* 2024. 50. P.237–291.

25. Franc, J.M.; Cheng, L.; Hart, A.; Hata, R.; Hertelendy, A. Repeatability, reproducibility, and diagnostic accuracy of a commercial Large Language Model (ChatGPT) to perform emergency department triage using the Canadian triage and acuity scale. *Can. J. Emerg. Med.* 2024. 26. P.40–46.

26. Zheng, O.; Abdel-Aty, M.; Wang, D.; Wang, C.; Ding, S. TrafficSafetyGPT: Tuning a Pre-trained Large Language Model to a Domain-Specific Expert in Transportation Safety. *arXiv* 2023. arXiv:2307.1531.

27. Liu, Z.; Qiao, A.; Neiswanger, W.; Wang, H.; Tan, B.; Tao, T.; Li, J.; Wang, Y.; Sun, S.; Pangarkar, O.; et al. LLM360: Towards Fully Transparent Open-Source LLM. *arXiv* 2023. arXiv:2312.06550.

28. Da, L.; Gao, M.; Mei, H.; Wei, H. LLM powered sim-to-real transfer for traffic signal control. *arXiv*. 2023. arXiv:2308.14284.

29. Mo, B.; Xu, H.; Zhuang, D.; Ma, R.; Guo, X.; Zhao, J. Large Language Models for Travel Behavior Prediction. *arXiv* .2023. arXiv:2312.00819.

30. Head, C.B., Jasper, P., McConnachie, M., Raftree, L., & Higdon, G. Large language model applications for evaluation: Opportunities and ethical implications. *New Directions for Evaluation.* 2023. Vol. *2023*. P.33 - 46.

31. P. Robinson *et al*. Modelling Ethical Algorithms in Autonomous Vehicles Using Crash Data," in *IEEE Transactions on Intelligent Transportation Systems*, July 2022. Vol. 23. No. 7. P. 7775-7784. doi: 10.1109/TITS.2021.3072792.

32. Ooi, K.-B.; Tan, G.W.-H.; Al-Emran, M.; Al-Sharafi, M.A.; Capatina, A.; Chakraborty, A.; Dwivedi, Y.K.; Huang, T.-L.; Kar, A.K.; Lee, V.-H.; et al. The Potential of Generative Artificial Intelligence Across Disciplines: Perspectives and Future Directions. J. Comput. Inf. Syst. 2023. P. 1–32.

33. Završnik, J.; Kokol, P.; Žlahtiˇc, B.; Blažun Vošner, H. Artificial Intelligence and Pediatrics: Synthetic Knowledge Synthesis. *Electronics*. 2024. Vol. 13. p. 512.

34. Lo, D. Trustworthy and Synergistic Artificial Intelligence for Software Engineering: Vision and Roadmaps. *arXiv 2023. arXiv*:2309.04142.

35. Belzner, L.; Gabor, T.; Wirsing, M. Large Language Model Assisted Software Engineering: Prospects, Challenges, and a Case Study. *In Proceedings of the Bridging the Gap Between AI and Reality; Steffen, B., Ed.; Springer Nature: Cham, Switzerland, 2024*. pp. 355–374.

36. Kopishynska O., Utkin Y., Sliusar I., Muravlov V., Makhmudov K., Chip L. Application of Modern Enterprise Resource Planning Systems for Agri-Food Supply Chains as a Strategy for Reaching the Level of Industry 4.0 for Non-

Manufacturing Organizations. *Engineering Proceedings*. 2023. Vol. 40(1). P.15. https://doi.org/10.3390/engproc2023040015

37. Kopishynska, O., Utkin, Y., Makhmudov, K., Kalashnik, O., Moroz, S., Somych, M. Digital Transformation of Resource Management of Territorial Communities Based on the Cloud ERP System in the Concept of Industry 4.0. In N. Callaos, J. Horne, B. Sánchez, M. Savoie (Eds.), *Proceedings of the 17th International Multi-Conference on Society, Cybernetics and Informatics: IMSCI 2023.* International Institute of Informatics and Cybernetics, 2023. P. 13-20. https://doi.org/10.54808/IMSCI2023.01.13

38. Kokol P. The Use of AI in Software Engineering: A Synthetic Knowledge Synthesis of the Recent Research Literature. *Information*. 2024. Vol. 15(6). P. 354. https://doi.org/10.3390/info15060354.

39. Ozkaya, I. The Next Frontier in Software Development: AI-Augmented Software Development Processes. *IEEE Software.* 2023.Volume 40. Issue 3. P.4–8. https://doi.org/10.1109/MS.2023.324840

40. Bano, M.; Hoda, R.; Zowghi, D.; Treude, C. Large language models for qualitative research in software engineering: exploring opportunities and challenges. *Automated Software Engineering*. 2024. Vol. 31. No 1. P.1-12. https://doi.org/10.1007/s10515-023-00407-8.

41. G20. Wikipedia 2024. URL: https://en.wikipedia.org/wiki/G20 (дата звернення: 04.12.2025).

42. Sawant, A.A.; Devanbu, P. Naturally!: How Breakthroughs in Natural Language Processing Can Dramatically Help Developers. *IEEE Softw*. 2021. Vol. 38. P. 118–123.

43. Ahmed, H.A.; Bawany, N.Z.; Shamsi, J.A. Capbug-a Framework for Automatic Bug Categorization and Prioritization Using Nlp and Machine Learning Algorithms. *IEEE Access.* 2021. Vol. 9. P. 50496–50512.

44. Charitsis, C.; Piech, C.; Mitchell, J.C. Using NLP to Quantify Program Decomposition in CS1. *In Proceedings of the Proceedings of the Ninth ACM Conference on Learning, New York, NY, USA, 1–3 June 2022*. 2022. pp. 113–120.

45. dos Santos, G.E.; Figueiredo, E. Commit Classification Using Natural Language Processing: Experiments over Labeled Datasets. 2020. URL: https://cibse2020.ppgia.pucpr.br/images/artigos/4/S04_P1.pdf (дата звернення: 04.12.2025).

46. Wong, M.-F.; Guo, S.; Hang, C.-N.; Ho, S.-W.; Tan, C.-W. Natural Language Generation and Understanding of Big Code for AI-Assisted Programming: A Review. *Entropy*. 2023. Vol.25. P.888.

47. Koreeda, Y.; Morishita, T.; Imaichi, O.; Sogawa, Y. LARCH: Large Language Model-Based Automatic Readme Creation with Heuristics. *In Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, Birmingham, UK, 21–25 October 2023*. 2023. P. 5066–5070.

48. Di Sipio, C.; Di Rocco, J.; Di Ruscio, D.; Nguyen, P.T. MORGAN: A Modeling Recommender System Based on Graph Kernel. *Softw. Syst. Model*. 2023. Vol. 22. Pp. 1427–1449.

49. Jánki, Z.R.; Bilicki, V. The Impact of the Web Data Access Object (WebDAO) Design Pattern on Productivity. *Computers*. 2023. Vol.12. P.149.

50. Pauzi, Z.; Capiluppi, A. Applications of Natural Language Processing in Software Traceability: A Systematic Mapping Study. *J. Syst. Softw*. 2023. Vol. 198. P. 111-116.

51. Monahan, K.: Studying Generative AI's Impact on Work. https://www.upwork.com/. https://www.upwork.com/blog/generative-ai-impact-on-work (дата звернення: 04.12.2025).

52. Younker, S.: Bill Gates just predicted the death of every job thanks to AI - except for these three. Tom's Guide. URL: https://www.tomsguide.com/ai/bill-gates-just-predicted-the-death-of-every-job-thanks-to-ai-except-for-these-three (дата звернення: 04.12.2025).

53. The Top Programming Languages. *IEEE Spectrum.* 2024. URL: https://spectrum.ieee.org/top-programming-languages-2024 (дата звернення: 04.12.2025).

54. JavaScript - Dynamic client-side scripting. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript (дата звернення: 04.12.2025).

55. Marjin Haverbeke. Eloquent JavaScript 3rd edition. No Starch Press, 2018. 472 p.

56. TypeScript vs JavaScript: Know The Difference. URL: https://radixweb.com/blog/typescript-vs-javascript#vs (дата звернення: 04.12.2025).

57. Chuck Musciano, Bill Kennedy, HTML & XHTML: The Defenitive Edition. O'Reilly, 2006. 680 p.

58. Krzysztof Boczkowski, Beata Pańczyk. Comparison of the performance of tools for creating a SPA application interface - React and Vue.js. JCSI 14 (2020). P. 73-77. https://doi.org/10.35784/jcsi.1579

59. Pazriy Ihor. Comparative Analysis of Software Development Systems Based on Frameworks. Herald of Khmelnytskyi national university. *Technical sciences.* 2023. №1(317). P.155–161. https://doi.org/10.31891/2307-5732-2023-317-1-155-161.

60. Eric Elliott. The Missing Introduction to React. 2020. URL: https://medium.com/javascript-scene/the-missing-introduction-to-react-62837cb2fd76 (дата звернення: 04.12.2025).

61. React: офіційний вебсайт. URL: https://uk.reactjs.org/ (дата звернення: 04 грудня 2025).

62. Cross–Platform Mobile App Development with Flutter–Xamarin–React Native. *Medium.* URL: https://cutt.ly/iwT5ttuc (дата звернення: 04 грудня 2025).

63. Majida Laaziria, Khaoula Benmoussa,Samira Khouljic, Mohamed Larbi Kerkebd. A Comparative study of PHP frameworks performance. Procedia Manufacturing 32 (2019) 864-871. URL: https://doi.org/10.1016/j.promfg.2019.02.295

64. Kopishynska O., Utkin Y., Sliusar I., Muravlov V., Makhmudov K., Chip L. Application of Modern Enterprise Resource Planning Systems for Agri-Food Supply Chains as a Strategy for Reaching the Level of Industry 4.0 for Non-Manufacturing Organizations. *Engineering Proceedings*. 2023. Vol. 40(1). P.15. https://doi.org/10.3390/engproc2023040015.

65. Tufano, M., Watson, C., Bavota, G., Poshyvanyk, D., Di Penta, M., Oliveto, R., & De Lucia, A. An empirical study on learning bug-fixing patches in the wild via neural machine translation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*. 2019. 28(4).1-29. https://doi.org/10.48550/arXiv.1812.08693

66. Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J. & Zaremba, W.. Evaluating large language models trained on code. 2021. arXiv preprint arXiv:2107.03374. https://arxiv.org/abs/2107.03374.

67. OpenAI. URL: https://openai.com/index/gpt-4-research/ (дата звернення: 04.12.2025).

68. 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. Ополе, Польща. [електронне видання] URL: https://www.economy-confer.com.ua/full-article/6477/ (дата звернення: 04.12.2025).

69. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*. 2020. Vol. 21(140). 1-67.

70. Nataliia Rekunenko. Ціна роботи. Скільки коштують робочі інструменти та освіта QA. *DOU: вебсайт*. URL: https://dou.ua/lenta/articles/price-of-work-qa/ (дата звернення: 04.12.2025).

71. Ціни на створення сайту 2024. *STUDIO IFISH: вебсайт*. URL: https://ifish.com.ua/ua/tsini-na-sajt/ (дата звернення: 04.12.2025).