ISSN 3083-7502

УДК 004.42:004.8

Недьошев Максим Владиславович
аспірант,
Національний університет біоресурсів і природокористування України
ORCID: https://orcid.org/0009-0000-9820-0649
E-mail: nedoshev@pm.me

Кириченко Віктор Вікторович
кандидат технічних наук, доцент, доцент кафедри комп'ютерних наук,
Національний університет біоресурсів і природокористування України
ORCID: https://orcid.org/0009-0001-0575-8684
E-mail: v.kyrychenko@nubip.edu.ua

ДОСЛІДЖЕННЯ ВПЛИВУ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ НА РОЗРОБКУ ВЕБСАЙТІВ З ВИКОРИСТАННЯМ ФРЕЙМВОРКУ VUE

Анотація. *У цій статті досліджується трансформаційний вплив великих мовних моделей (LLM) на сучасну компонентно-орієнтовану веброзробку, використовуючи фреймворк Vue.js як конкретний приклад. Синтезуючи результати широкого емпіричного дослідження розробки програмного забезпечення за допомогою LLM [1], ми аналізуємо парадигматичний зсув від нативних робочих процесів фреймворку до процесів, доповнених LLM. Аналіз охоплює весь життєвий цикл проєкту, виявляючи значне підвищення продуктивності на етапах реалізації та розгортання, зокрема у створенні компонентів, автоматизації тестування та налаштуванні інфраструктури. Однак ці переваги врівноважуються критичними викликами, серед яких занепокоєння щодо надійності коду, увічнення конфліктів версій через застарілі навчальні дані та ризик когнітивного розвантаження серед розробників. Ми стверджуємо, що інтеграція LLM переосмислює роль старшого розробника, перетворюючи його з основного генератора коду на експерта-валідатора та архітектурного наглядача. Стаття завершується окресленням ключових ризиків та пропозицією напрямків для майбутніх досліджень, наголошуючи на необхідності створення специфічних для фреймворку бенчмарків для оцінки якості коду, згенерованого ШІ, та лонгітюдних досліджень щодо супроводжуваності Vue.js-застосунків, розроблених за допомогою LLM.*

Ключові слова: *великі мовні моделі, Vue.js, веброзробка, інтелектуальна автоматизація, UI-компоненти, продуктивність, програмна інженерія.*

Актуальність. Веброзробка відіграє велику роль у поточній економіці світу, де всі сервіси і інструменти доступні з більшості гаджетів за пошуковим запитом. Стрімкий розвиток штучного інтелекту, у вигляді Large Language Models. Протягом останніх років інструменти, які базуються на ШІ, як GPT, Github Copilot, Claude та інші, перейшли з розряду експериментальних у категорію практичних інструментів, які щоденно використовуються розробниками. Це призвело до зсуву парадигми розробки до LLM-орієнтованої розробки, що значно відрізняється від традиційних способів розробки. Сфера застосування LLM є відносно новою і є безліч прогалин у систематизованих знаннях. Має сенс дослідити такі ключові фактори: економічна доцільність, ступінь підвищення продуктивності, трансформацію ринку праць, наявність проблем та ризиків використання LLM у розробці. Враховуючи різноманітну природу розробки, яка різниця підходами до розробки, мовами програмування та фреймворками є необхідність систематизувати знання про вплив інтеграції LLM у розробку специфічних доменів і задач.

Мета дослідження – систематично проаналізувати та оцінити вплив великих мовних моделей (LLM) на процеси розробки вебсайтів з використанням фреймворку Vue.js. Оскільки LLM стають все більш інтегрованими в інструменти розробки, виникає нагальна потреба зрозуміти, як ця нова парадигма відрізняється від традиційних підходів до розробки на прикладі веброзробки з використанням Vue.js.

Аналіз матеріалів досліджень за напрямком роботи. Тема інтеграції великих мовних моделей у процеси розробки програмного забезпечення є предметом активних досліджень в академічній та індустріальній спільнотах. Аналіз наукових публікацій дозволяє виділити

кілька ключових напрямків. Перший напрямок стосується емпіричного порівняння продуктивності розробників при використанні LLM-інструментів. Дослідження [1] виявили значні відмінності між традиційними Low-Code платформами та LLM-орієнтованим підходом. Якщо LCP ефективні у вузькоспеціалізованих завданнях (наприклад, створення форм чи простих бізнес-додатків), то LLM демонструють значно вищу гнучкість, охоплюючи ширший спектр програмних завдань, включаючи веб-розробку, аналіз даних та створення алгоритмів. Другий напрямок фокусується на якості та надійності коду, згенерованого LLM. Хоча такі інструменти, як GitHub Copilot, можуть генерувати синтаксично правильний код, роботи [2, 3] вказують на потенційні проблеми, такі як "галюцинації" (генерація неіснуючих функцій або API), вразливості безпеки та неоптимальні алгоритмічні рішення. Це підкреслює критичну роль людини-експерта в процесі перевірки, рефакторингу та валідації коду. Третій напрямок досліджень вивчає вплив LLM на освіту та підготовку майбутніх програмістів. Роботи, такі як [3], аналізують, як використання LLM змінює навчальний процес. З одного боку, вони можуть слугувати потужними інструментами для навчання, надаючи миттєві пояснення та приклади коду. З іншого боку, існує ризик, що студенти стануть надто залежними від цих інструментів, що може негативно вплинути на розвиток фундаментальних навичок розв'язання проблем та алгоритмічного мислення. Четвертий напрямок стосується майбутнього програмної інженерії. У звітах та прогнозах від провідних аналітичних компаній, таких як McKinsey & Company [4], підкреслюється, що ШІ не замінить розробників, а трансформує їхню роль. Очікується, що до 80% рутинних завдань кодування будуть автоматизовані, що дозволить інженерам зосередитися на більш творчих та стратегічних аспектах: архітектурі системи, проєктуванні користувацького досвіду та інноваціях. Нарешті, активно обговорюються етичні аспекти та виклики, пов'язані з використанням LLM у розробці. Питання конфіденційності (оскільки фрагменти коду можуть надсилатися на сторонні сервери для аналізу), авторського права (через навчання моделей на мільярдах рядків відкритого коду) та упередженості (моделі можуть відтворювати помилки та погані практики, присутні в навчальних даних) описані у працях [5, 6]. Таким чином, існуючий масив досліджень підтверджує значний та багатогранний вплив LLM на розробку програмного забезпечення і роботу людини загалом. Однак, багато питань залишаються відкритими, зокрема щодо довгострокових наслідків цієї технологічної революції та розробки найкращих практик для ефективної та безпечної інтеграції LLM у робочі процеси.

Матеріали і методи дослідження. Метою даного дослідження є комплексний аналіз та систематизація знань про вплив великих мовних моделей (LLM) на процеси розробки вебсайтів. Для досягнення поставленої мети було обрано методологію систематичного огляду літератури (Systematic Literature Review, SLR). Інформаційною базою для дослідження слугували наукові та технічні публікації, індексовані у провідних наукометричних базах даних та цифрових бібліотеках, таких як: архів препринтів arXiv.org, Google Scholar. Також використовувались інструменти для пошуку актуальних досліджень з 2020 по 2025 роки Google NotebookLM, Alphaxiv, Research Rabbit, x.com. Пошук відбувався англійською мовою, а пошукові запити включали: Large Lanugage Model, LLM, software engineering, web development, code generation, challenges, opportunities, Github Copilot, AI, Low-code. До аналізу брались джерела з наукових журналів, конференцій, технічні та економічні звіти, публікації присвячені використанню LLM у програмній інженерії та веброзробці. З аналізу були виключені оглядові статті, маркетингові матеріали та блоги без технічного обґрунтування. До методів аналізу можна віднести тематичний аналіз, порівняльний аналіз та синтез даних.

Об’єктом дослідження обрано розробку вебсайтів з використанням штучного інтелекту конкретно з використанням фреймворку Vue.js. Оскільки LLM навчаються на великих об’ємах даних, для розробників є сенс використовувати найбільш популярні інструменти, про які багато інформації у інтернеті. Однак, існує безліч продуктів, які використовують новіші та менш поширені інструменти. Наприклад, згідно аналізу використаних інструментів для створення вебсайтів [7] Vue.js займає далеко не перше місце по використанню, а отже можна дійти висновку що прикладів коду для навчання також невелике, що має створювати

складності для LLM. Такий підхід до аналізу використання LLM у розробці дає більш точніші висновки щодо використання ШІ технологій для роботи у спеціалізованому домені.

Трансформація життєвого циклу веброзробки. Традиційний процес веб розробки включає створення компонентів, управління станом (state management), маршрутизацію та тестування, зазнає значних змін під впливом LLM:

1. Етап ініціалізації та налаштування проєкту**.** На цьому етапі LLM виступають як інтелектуальні асистенти, що генерують конфігураційні файли для збирачів проєкту (Vite, Webpack), налаштовують інтеграцію з TypeScript, ESLint, Prettier та створюють базову структуру каталогів. Це значно скорочує час, який раніше витрачався на рутинні операції, дозволяючи розробнику швидше перейти до бізнес-логіки. Хоча цей процес вже був автоматизований за допомогою консольних інструментів (CLI), використання LLM замінює використання шаблонів (boilerplate) на інтерактивну генерацію стартових проєктів. Для цього можна використовувати сервіси v0 або bolt.new.
2. Розробка UI-компонентів відчуває найбільший вплив LLM. Замість ручного кодування, розробник може сформувати запити природною мовою. Це значно підвищує продуктивність: час виконання завдань може зменшитись на 0,8 стандартних відхилень, а якість результату зрости на 0,4 стандартних відхилення [6].
3. Тестування та зневадження. Написання юніт-тестів та компонентних тестів (наприклад, з використанням Vitest та Vue Testing Library) є одним з найбільш трудомістких і часозатратних процесів. LLM здатні автоматично генерувати тестові сценарії, мокувати залежності та описувати базові перевірки, що значно підвищує тестове покриття. Проте, згенеровані тести не завжди охоплюють усі граничні випадки, що вимагає ретельного аудиту з боку розробника. Дослідження безпеки коду, згенерованого ШІ, вказують на те, що він може містити до 40% вразливостей, що підкреслює критичну необхідність перевірки [2].

Ключові виклики та ризики у контексті Vue.js. Незважаючи на очевидні переваги, інтеграція LLM у розробку на Vue.js породжує специфічні виклики:

1. Надійність та ідіоматичність коду. LLM, навчені на величезному масиві коду, не завжди розрізняють ідіоматичний (відповідний "духу" фреймворку) та функціональний, але неоптимальний код. Для Vue.js це може проявлятися у неправильному використанні системи реактивності, ігноруванні можливостей Composition API або генерації коду, що призводить до невиправданих повторних рендерів компонентів.
2. Конфлікти версій. Екосистема Vue.js нещодавно зазнала значних змін, а саме: перехід з Vue 2 на Vue 3, еволюція від Vuex до Pinia, Webpackt до Vite. LLM, навчені на застарілих даних, можуть генерувати код, несумісний з останніми версіями фреймворку та його бібліотек. Це створює "прихований технічний борг" і вимагає від розробника постійної пильності та глибоких знань актуального стану екосистеми.
3. Моделі можуть "вигадувати" (галюцинації) неіснуючі функції або властивості API Vue.js, що призводить до помилок під час виконання і вимагає додаткового часу на зневадження. Ця проблема детально описана в систематичних оглядах LLM у програмній інженерії [1].

Глобальне тестування LLM для генерації коду для Vue.js. Для тестування роботи LLM у задачах різного типу у дослідженні [8] було створено LLM-Bench. У даному дослідженні Gemeni 2.5 pro, Claude, GPT4.1 LLM показали найкращі результати у задачах генерації коду. Однак, відсоток вирішних задач менший за 50%. У контексті задач специфічних для Vue.js в дослідженні було описано 20 задач типових для Vue та 20 задач для Nuxt, які можуть зустрічатися під час розробки і розподілені на 3 категорії: “easy”, “moderate” та “challenging”. Так, наприклад, для Gememi 2.5 pro було виконано всього 45% завдань з другої спроби і лише 20% з першої спроби.

Специфічне тестування. Для розширення тестування LLM для специфічної задачі в дослідженні було розроблено два тести: генерація таблиці користувачів без та з використанням бібліотек готових компонентів. Використання бібліотеки компонентів робить задачу для LLM більш специфічною та складною, однак це широка поширена задача на

практиці, оскільки більшість вебзастосунків використовують бібліотеки компонентів. Для тестів було використано бібліотеку компонентів Vuestic UI, як приклад нової та не самої популярної бібліотеки, що має малу кількість прикладів коду, хоча має велику за обсягом документацію.

Так було описано запит для LLM без конкретного запиту для використання бібліотеки:

```
Make Table.vue component.
<columns> Avatar, Name, Phone, Email, Edit Button, Delete Button. </columns> Fill
with mock data. Show modal confirmation on delete.
Freeze Avatar and Name column.
```

Цей запит було дано на обробку LLM у хмарі Google Gemini 2.5 Prop, Google Gemini 2.5 Flash, Sonnet 4 та локальним LLM Qwen3 4b, Google Gemma 12b. Результати тестів представлено у табл. 1. Критеріями тестування є ручний аналіз згенерованого тесту та його запуск. За результатами тестування можна прийти висновку що жодна LLM не впоралась з завданням на 100% і згенерований код має схожі проблеми, однак більшість результатів можна запустити, змінити і мати робочий результат.

Таблиця 1 – Результати роботи LLM для генерації коду

Тест	Gemini 2.5 pro	Gemini 2.5 flash	Sonnet 4	Qwen3	Gemma 3
Код може бути скомпільований та запущений без змін	TRUE	TRUE	TRUE	TRUE	FALSE
Код має коректний Vue синтаксис	TRUE	FALSE	FALSE	TRUE	TRUE
Код має реалізацію відображення аватара у таблиці	TRUE	TRUE	FALSE	TRUE	TRUE
Код має справну реалізацію вікна підтвердження	TRUE	TRUE	FALSE	TRUE	FALSE
Код має справну реалізацію колонок	TRUE	FALSE	FALSE	FALSE	TRUE
Код не використовує додаткові бібліотеки	FALSE	FALSE	FALSE	FALSE	TRUE
Код запускається і немає візуальних проблем	FALSE	FALSE	FALSE	TRUE	FALSE

Далі для тестування LLM наближено до реальних продуктів до запиту було додано “using Vuestic UI”. Жодна LLM не впоралась з завданням. Всі LLM додали зайвого коду, що не відповідає використання бібліотеки людиною. Локальні LLM з малою кількістю параметрів не впорались з завданням взагалі і використовували код з інших бібліотек, що призвело до некоректного синтаксису (табл. 2).

Висновки. Інтеграція великих мовних моделей у процес розробки на фреймворку Vue.js є потужним каталізатором продуктивності, що дозволяє автоматизувати рутинні завдання на всіх етапах життєвого циклу проєкту. Найбільший ефект спостерігається у генерації UI-компонентів, написанні тестів та налаштуванні інфраструктури, що відповідає загальним тенденціям впливу ШІ на програмну інженерію.

Однак, результати специфічного тестування, проведеного в рамках даної роботи, виявляють значні обмеження сучасних LLM, навіть таких просунутих, як Gemini 2.5 Pro. Хоча моделі задовільно справляються із загальними завданнями, їхня ефективність різко падає при роботі зі спеціалізованими інструментами, такими як нішеві бібліотеки компонентів (на

прикладі Vuestic UI). Згенерований код часто виявляється неробочим, містить зайві елементи та не відповідає найкращим практикам використання бібліотеки. Це емпірично доводить, що сліпа довіра до LLM у реальних проєктах є невиправданою і може призвести до накопичення технічного боргу.

Таблиця 2 – Результати роботи LLM для генерації коду з використанням специфічної бібліотеки

Тест	Gemini 2.5 pro	Gemini 2.5 flash	Sonnet 4	Qwen3 (4b)	Gemma 3 12B
Код може бути скомпільований та запущений без змін	False	TRUE	TRUE	FALSE	FALSE
Код має коректний Vue синтаксис	TRUE	FALSE	TRUE	TRUE	TRUE
В коді присутній зайві виклики функцій, стилі та інше.	TRUE	TRUE	TRUE	TRUE	TRUE
Код має справну реалізацію вікна підтвердження	TRUE	TRUE	TRUE	FALSE	FALSE
Код має справну реалізацію колонок	FALSE	FALSE	FALSE	FALSE	FALSE
Код запускається і немає візуальних проблем	FALSE	FALSE	FALSE	FALSE	FALSE

Ключовим висновком дослідження є те, що інтеграція LLM не нівелює потребу у кваліфікованих інженерах, а, навпаки, підвищує вимоги до їхньої експертизи. Роль розробника трансформується з автора коду на архітектора, валідатора та наглядача, відповідального за критичну оцінку та адаптацію рішень, запропонованих штучним інтелектом.

Таким чином, ефективне використання LLM у розробці на Vue.js вимагає збалансованого підходу: використання ШІ як інструменту для прискорення рутинних операцій, поєднаного з глибокою експертизою розробника для верифікації, рефакторингу та забезпечення якості й довгострокової супроводжуваної кодової бази. Подальші дослідження повинні бути спрямовані на створення спеціалізованих бенчмарків для оцінки LLM у контексті конкретних фреймворків і бібліотек та аналіз життєвого циклу проєктів, розроблених за допомогою ШІ. Можна також припустити що висновки цього дослідження також справедливі для інших фреймворків типу React, Angular та Svelte.

Список використаних джерел

1. Liu, Y., Chen, J., Bi, T., Grundy, J., Wang, Y., Yu, J., Chen, T., Tang, Y., & Zheng, Z. (2024). An empirical study on low code programming using traditional vs large language model support [Preprint]. arXiv. https://arxiv.org/abs/2402.01156.
2. Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2025). Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. Communications of the ACM, 68(2), 96–105. https://doi.org/10.1145/361072.
3. Jošt, G., Taneski, V., & Karakatič, S. (2024). The impact of large language models on programming education and student learning outcomes. Applied Sciences, 14(10), Article 4115. https://doi.org/10.3390/app14104115.
4. Chui, M., Hazan, E., Roberts, R., Singla, A., Smaje, K., Sukharevsky, A., Yee, L., & Zemmel, R. (2023). The economic potential of generative AI: The next productivity frontier. McKinsey Global Institute. https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier.

5. Urlana, A., Kumar, C. V., Singh, A. K., Garlapati, B. M., Chalamala, S. R., & Mishra, R. (2024). LLMs with industrial lens: Deciphering the challenges and prospects - A survey [Preprint]. arXiv. https://arxiv.org/abs/2402.14558. https://doi.org/10.48550/arXiv.2402.14558.
6. Noy, S., & Zhang, W. (2023). Experimental evidence on the productivity effects of generative artificial intelligence. Science, 381(6654), 187–192. https://doi.org/10.1126/science.adh2586
7. W3Techs. Usage statistics and market shares of JavaScript libraries. https://w3techs.com/technologies/overview/javascript_library (25.09.2025).
8. Xu, K., Mao, Y., Guan, X., & Feng, Z. (2025). Web-Bench: A LLM code benchmark based on web standards and frameworks [Preprint]. arXiv. https://arxiv.org/abs/2505.07473. https://doi.org/10.48550/arXiv.2505.07473.

Nedoshev Maksym
PhD student in Computer Science,
National University of Life and Environmental Sciences of Ukraine
ORCID: https://orcid.org/0009-0000-9820-0649
E-mail: nedoshev@pm.me

Kyrychenko Viktor
PhD in Physical and Mathematical Sciences, Associate Professor, Associate Professor of the Department of Computer Science,
National University of Life and Environmental Sciences of Ukraine
ORCID: https://orcid.org/0009-0001-0575-8684
Email: v.kyrychenko@nubip.edu.ua

RESEARCH ON THE IMPACT OF LARGE LANGUAGE MODELS ON WEBSITE DEVELOPMENT USING THE VUE FRAMEWORK

Abstract. *This paper investigates the transformative impact of large language models (LLMs) on modern component-based web development, using the Vue.js framework as a representative case study. By synthesizing the results of a broad empirical study on software development with the assistance of LLMs, we analyze a paradigmatic shift from native framework-driven workflows to workflows augmented by LLMs. The analysis spans the entire project lifecycle, revealing significant productivity gains during the implementation and deployment phases, particularly in component creation, test automation, and infrastructure configuration. However, these advantages are counterbalanced by critical challenges, including concerns about code reliability, the perpetuation of version conflicts due to outdated training data, and the risk of cognitive offloading among developers. We argue that the integration of LLMs redefines the role of the senior developer, transforming it from a primary code generator into an expert validator and architectural overseer. The paper concludes by outlining key risks and proposing directions for future research, emphasizing the need to develop framework-specific benchmarks for evaluating the quality of AI-generated code and to conduct longitudinal studies on the maintainability of Vue.js applications developed with the assistance of LLMs.*

Keywords*: large language models, Vue.js, web development, intelligent automation, UI components, productivity, software engineering.*