DOI: https://doi.org/10.33216/1998-7927-2025-298-12-38-46

УДК 004.432.45:004.056.55]:004.8

ЗАСТОСУВАННЯ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ДЛЯ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ ПРОГРАМНОГО КОДУ

Яцький О.В., Борю С.Ю.

APPLICATION OF LARGE LANGUAGE MODELS FOR DETECTING SOFTWARE CODE VULNERABILITIES

Yatskyi O.V., Boriu S.Y.

У статті розглядається застосування великих мовних моделей (ВММ), зокрема GPT-4, для автоматизованого виявлення вразливостей у програмному коді. Сучасні методи статичного аналізу коду мають обмеження у виявленні складних вразливостей та контекстуальних загроз безпеці, що потребує розроблення нових підходів на основі технологій штучного інтелекту. Для оцінки здатності великих мовних моделей (ВММ) виявляти вразливості у програмному коді було проведено серію експериментів із використанням кількох поколінь моделей GPT, зокрема: Ada (350 млн параметрів), Curie (6,7 млрд), Davinci (175 млрд), а також найновішої моделі GPT-4 (≈1,7 трлн параметрів). На основі порівняльного аналізу з традиційним статичним аналізатором Snyk показано, що GPT-4 здатна виявляти більшу кількість вразливостей, охоплюючи ширший спектр загроз, а також пропонувати релевантні виправлення для їх усунення. У роботі використано 64 фрагменти коду на восьми мовах програмування, що охоплюють 33 категорії вразливостей згідно з класифікацією Common Weakness Enumeration (CWE). Експериментальне дослідження проведено шляхом порівняння результатів аналізу коду засобами GPT-4 та Snyk на ідентичних наборах тестових даних. Модель GPT-4 продемонструвала високу ефективність в автоматичному рефакторингу коду, зокрема забезпечила зниження рівня критичних вразливостей на 94% порівняно з початковим станом програмного коду. Окремо проаналізовано метрики точності виявлення загроз, частоту хибнопозитивних та хибнонегативних результатів, а також здатність моделі надавати детальні пояснення виявлених проблем безпеки. Особливою перевагою GPT-4 стала її здатність працювати в режимі автоматичного рефакторингу, що був закладений у варіаціях системного контексту. Результати дослідження свідчать про доцільність застосування ВММ як додаткового інструменту забезпечення безпеки програмного коду. Запропоновано можливості інтеграції великих мовних моделей у сучасні середовища безперервної інтеграції та розгортання (CI/CD) та окреслено перспективи гібридного застосування разом з класичними інструментами автоматизованого тестування безпеки програмного забезпечення.

***Ключові слова**: великі мовні моделі, вразливості коду, статичний аналіз, кібербезпека, GPT-4, штучний інтелект, рефакторинг*

Вступ. У сучасних умовах інтенсивного розвитку цифрових технологій питання безпеки програмного забезпечення набуває особливої ваги. Щодня мільйони рядків коду створюються, оновлюються або повторно використовуються в інформаційних системах, сервісах і мобільних додатках. Проте разом з цим зростає і кількість вразливостей, які можуть стати об’єктом атаки з боку зловмисників, що призводить до серйозних фінансових та репутаційних втрат.

Традиційні засоби забезпечення безпеки, зокрема статичні аналізатори коду, такі як Snyk, Fortify чи SonarQube, є ефективними, але водночас мають низку обмежень. Зокрема, вони часто не охоплюють широкого спектру мов програмування, потребують конфігурації під конкретні середовища або не здатні виявляти складні логічні вразливості, що виникають через нестандартні підходи до розробки.

На цьому фоні особливий інтерес викликає застосування технологій штучного інтелекту,

зокрема великих мовних моделей (ВММ), таких як GPT-4 від OpenAI. Завдяки здатності обробляти природну мову, розпізнавати структуру коду та проводити його інтерпретацію у контексті програмної логіки, ВММ можуть виступати новим інструментом для виявлення вразливостей і навіть пропонування їх виправлень.

Вітчізняні дослідники також звертають увагу на потенціал інтеграції великих мовних моделей у системи захисту інформації. Зокрема, О. В. Голуб розглядає архітектурні підходи до впровадження ВММ у комплексні системи кібербезпеки та аналізує можливості їх використання на різних етапах життєвого циклу програмного забезпечення [1].

Метою цієї роботи є аналіз ефективності застосування великої мовної моделі GPT-4 для виявлення вразливостей у програмному коді, а також порівняння її результатів із результатами традиційних статичних аналізаторів, таких як Snyk.

Об'єктом дослідження є процес автоматичного аналізу вихідного коду з метою виявлення вразливостей.

Предметом дослідження виступає ефективність і якість виявлення та усунення вразливостей за допомогою GPT-4 у порівнянні з інструментом Snyk.

Завдання дослідження:

- провести експериментальну перевірку здатності GPT-4 ідентифікувати вразливості у фрагментах коду на різних мовах програмування;
- порівняти кількість і типи знайдених вразливостей GPT-4 і Snyk;
- оцінити здатність GPT-4 пропонувати коректні виправлення для виявлених вразливостей;
- визначити сильні та слабкі сторони використання ВММ для цілей кібербезпеки;
- сформулювати рекомендації щодо подальшого застосування таких моделей у реальних розробницьких процесах.

У сукупності результати цього дослідження можуть стати підґрунтям для створення нових інструментів автоматизованого аудиту коду на основі ВММ, а також надати розробникам альтернативний підхід до забезпечення безпеки програмного забезпечення.

У сфері безпеки програмного забезпечення вже тривалий час використовуються традиційні інструменти статичного аналізу коду, серед яких найбільш поширеними є Snyk, Fortify, SonarQube, Checkmarx та інші. Ці системи працюють за принципом статичного аналізу — без запуску програмного коду, виключно на основі синтаксичного та семантичного аналізу тексту. Такі інструменти здатні виявляти широкий спектр вразливостей, зокрема SQL-ін’єкції, XSS (міжсайтове скриптування), переповнення буфера, порушення доступу до пам’яті тощо. Проте слід зазначити, що ефективність роботи таких засобів залежить від їхньої конфігурації, якості баз правил, а також обмежується конкретними мовами програмування. Крім того, ці інструменти не завжди здатні адекватно інтерпретувати складні логічні зв’язки, властиві сучасним архітектурам програмного забезпечення, особливо у випадках з динамічною типізацією або фреймворками з багатошаровою логікою. У порівнянні з цим, великі мовні моделі, навчені на масивних корпусах програмного коду, відкривають нові можливості аналізу завдяки своїй здатності до генеративного мислення та контекстуального розуміння.

Особливу увагу науковців упродовж останніх років привертає застосування великих мовних моделей (ВММ), зокрема сімейства GPT, для вирішення завдань у сфері кібербезпеки — насамперед у виявленні вразливостей програмного коду. Перші помітні результати були досягнуті ще з використанням моделі GPT-3, зокрема її варіанта text-davinci-003. У дослідженні К. Коха (2023) зафіксовано, що GPT-3 виявила 213 вразливостей у 64 фрагментах коду, тоді як популярний статичний аналізатор Snyk — лише 99, не враховуючи непідтримувані мови. При цьому було вручну перевірено 60 випадків, і лише у 4 з них модель помилково позначила безпечний код як уразливий, що дає помилкову позитивну похибку близько 6%. Така точність виглядає дуже конкурентоспроможною, особливо враховуючи, що GPT-3 не є інструментом вузької спеціалізації. [2, с. 315] Її здатність виявляти проблеми в широкому спектрі мов програмування та надавати логічні пояснення сприяє її використанню не лише для перевірки, а й для освітніх цілей у сфері розробки безпечного ПЗ.

Розвиток моделей GPT-3.5 і, згодом, GPT-4, значно розширив функціональні можливості виявлення вразливостей. GPT-4 має глибше контекстуальне розуміння коду, здатна самостійно визначати мову програмування, адаптувати логіку аналізу та пропонувати варіанти виправлення. Завдяки збільшеному обсягу параметрів (понад 1,7 трильйона), GPT-4

наближається до рівня професійного аналітика з безпеки, оскільки не просто ідентифікує загрози, а пояснює їхню суть, наслідки та пропонує дієві рішення. У багатьох випадках вона може навіть переписати код повністю — із врахуванням логіки, рекомендацій з безпеки та синтаксичних норм. Ця здатність до автоматизованого рефакторингу відкриває перспективи її інтеграції у системи DevSecOps, як інтелектуального компонента на етапах розробки, тестування або передрелізної перевірки. Таким чином, GPT-4 не лише підтримує існуючі методики, а формує новий клас інструментів для безпеки — семантичний аналіз із генеративним компонентом, що є революційним підходом у практиках кіберзахисту.

У роботі, присвяченій застосуванню мовних моделей для виявлення вразливостей програмного забезпечення, автори демонструють здатність LLM аналізувати структурні та семантичні аспекти коду, виявляючи не лише синтаксичні помилки, а й логічні вразливості, які важко ідентифікувати традиційними методами статичного аналізу [6]. Масштабне бенчмаркінгове дослідження підтвердило ефективність великих мовних моделей у виявленні різноманітних типів вразливостей на репрезентативних наборах даних, хоча й виявило необхідність подальшого вдосконалення метрик оцінювання та зменшення кількості хибних спрацьовувань [7].

Визначення типів вразливостей у контексті кібербезпеки тісно пов’язане з міжнародними стандартами класифікації, такими як CWE (Common Weakness Enumeration) та CVE (Common Vulnerabilities and Exposures). База CWE визначає типові помилки безпеки, які можуть бути допущені у програмному коді, наприклад, CWE-79 (Cross-site Scripting), CWE-89 (SQL Injection), CWE-22 (Path Traversal) тощо. Водночас CVE — це база відомих вразливостей із конкретними уразливими реалізаціями в програмному забезпеченні, яким присвоюються унікальні ідентифікатори. У більшості досліджень, присвячених застосуванню штучного інтелекту у виявленні вразливостей, саме класифікація CWE використовується як основа для категоризації помилок. У даній роботі також застосовано підхід із використанням типових фрагментів коду, які ілюструють 33 основні категорії уразливостей згідно з CWE, включаючи переповнення буфера, неправильну обробку введення, обходи шляхів, вразливості автентифікації та інші. [8]

Тобто, аналіз літератури свідчить про актуальність дослідження, а також про високий потенціал застосування великих мовних моделей, таких як GPT-4, у процесах забезпечення безпеки коду. Водночас постає потреба в систематичному порівнянні результатів таких моделей із традиційними підходами, що і становить основу для подальших розділів цієї наукової роботи.

Виклад основного матеріалу. Для оцінки здатності великих мовних моделей (ВММ) виявляти вразливості у програмному коді було проведено серію експериментів із використанням кількох поколінь моделей GPT, зокрема: Ada (350 млн параметрів), Curie (6,7 млрд), Davinci (175 млрд), а також найновішої моделі GPT-4 (≈1,7 трлн параметрів). Вибір цих моделей зумовлений поступовим збільшенням обсягу параметрів, що безпосередньо впливає на глибину розуміння контексту та точність генерації відповідей. Усі моделі використовувалися через API-платформу OpenAI, яка дозволяє створювати запити з визначеним системним контекстом.

Системний контекст, який задавався великим мовним моделям у межах експерименту, був сформульований у вигляді інструкції англійською мовою, наприклад: «Act as the world's greatest static code analyzer for all major programming languages...». Такий підхід мав на меті створення для моделі чіткої ролі — максимально наближеної до функціоналу висококласного статичного аналізатора коду. Завдяки потужному механізму інструктивного навчання (instruction tuning), модель інтерпретувала контекст не лише як орієнтир для стилю відповіді, а як повноцінну задачу з пріоритетами, орієнтовану на точність, стислість і технічну глибину. Кожен фрагмент коду, який передавався до моделі, не містив додаткових підказок щодо мови програмування, стилістики або очікуваного формату. [1, с. 250] Це дозволяло перевірити, наскільки гнучко GPT-4 здатна адаптуватися до синтаксичних відмінностей між мовами та правильно інтерпретувати структуру і логіку коду без зовнішньої допомоги.

Особливою перевагою GPT-4 стала її здатність працювати в режимі автоматичного рефакторингу, який також був закладений у варіаціях системного контексту. У цьому випадку модель отримувала інструкцію не лише виявити вразливість, а й переписати код,

усуваючи знайдену проблему, без пояснень чи проміжних коментарів. Такий режим дозволяв симулювати поведінку інструмента автоматичного виправлення коду в реальному середовищі розробки. GPT-4, на відміну від менш потужних моделей (як-от Curie або навіть Davinci), успішно справлялася з цією задачею: вносила коректні синтаксичні зміни, зберігала логіку оригінального фрагмента, а також підвищувала його стійкість до типових атак. Застосування гнучких системних контекстів дозволило не лише протестувати реакцію моделі на різні ролі, а й виявити, що GPT-4 є контекстно-чутливою та здатною до багаторівневої логічної трансформації коду, що робить її унікальним інструментом у сфері безпеки програмного забезпечення.

Базою для аналізу стали 64 фрагменти коду, які охоплюють 33 типи вразливостей, згідно з класифікацією CWE. Приклади були взяті з так званої Єдиної кодової бази вразливостей безпеки (Unified Code Vulnerability Base). Ці фрагменти представлені на 8 різних мовах програмування: C, Ruby, PHP, Java, JavaScript, C#, Go та Python, що дозволяє оцінити універсальність підходу.

Кожен запит до моделі мав чітко визначену структуру:

- вхід: текст системного контексту + фрагмент коду;
- очікуваний вихід: перелік виявлених вразливостей (нумерований список) та перелік запропонованих виправлень;
- у варіанті з GPT-4 — ще й повна переписка коду без вразливостей.

Усі результати моделі було записано, класифіковано за категоріями та зіставлено з результатами сканування тих самих фрагментів за допомогою статичного аналізатора Snyk.

У таблиці 1 наведено ключові характеристики великих мовних моделей (ВММ) GPT, використаних у межах цього дослідження. Зі збільшенням кількості параметрів моделей спостерігається поступове покращення якості аналізу, здатності до генерації релевантного коду та точності у виявленні вразливостей. Найпростішою моделлю є Ada, що має близько 350 мільйонів параметрів. Вона демонструє базовий рівень розуміння коду, але через обмежену глибину контекстного аналізу не здатна ідентифікувати складні типи вразливостей або запропонувати коректні виправлення. Curie, яка має у 20 разів більше параметрів, показує суттєво кращі результати: вона здатна швидше й точніше ідентифікувати поширені проблеми у коді та надає відповіді з вищою достовірністю, хоча все ще поступається у гнучкості глибшим моделям. Її головна перевага — баланс між швидкістю відповіді та достатньою точністю, що може бути корисним у задачах, де пріоритетом є час обробки, а не повна відповідність аналітичним висновкам.

Модель Davinci (GPT-3.5), що має 175 мільярдів параметрів, уже наближається до рівня повноцінного аналітика: вона не тільки ефективно ідентифікує широкий спектр уразливостей, а й здатна логічно аргументувати свої висновки та генерувати змістовно правильні фрагменти коду. [6] Проте справжній прорив у дослідженнях був досягнутий з появою GPT-4, яка є найпотужнішою моделлю в цьому порівнянні. GPT-4 здатна до глибокої саморефлексії: вона може переглядати та вдосконалювати власні висновки, надає повноцінні пояснення причин уразливостей, описує потенційні вектори атак, а також пропонує структуровані виправлення, що зменшують кількість хибнопозитивних спрацьовувань. GPT-4 також краще справляється з контекстом, коли тип мови програмування не вказаний, — вона самостійно визначає синтаксис і коректно застосовує правила безпеки для відповідної парадигми. Таким чином, таблиця демонструє, що зростання параметрів ВММ має прямий зв'язок із її здатністю до складного аналітичного мислення, що є вирішальним у задачах забезпечення кібербезпеки на рівні програмного коду.

Таблиця 1

Характеристики моделей GPT, використаних у дослідженні

Модель	Кількість параметрів	Покоління	Особливості використання
Ada	~350 млн	GPT-3	Базовий рівень, низька точність
Curie	~6,7 млрд	GPT-3	Помірна точність, швидка відповідь
Davinci	~175 млрд	GPT-3.5	Висока якість аналізу, здатна до генерації коду
GPT-4	~1,7 трлн	GPT-4	Найвища точність, здатність до рефлексії та пояснень, краща генерація виправлень

У межах дослідження було проведено порівняльний аналіз здатності великої мовної моделі GPT-4 і статичного аналізатора коду Snyk щодо виявлення вразливостей у 64 фрагментах коду на восьми популярних мовах програмування: C, C#, Go, Java, JavaScript, PHP, Python та Ruby. Усі фрагменти були запозичені з публічної бази «Unified Code Vulnerability Base» і охоплювали 33 типи уразливостей згідно з класифікацією CWE. GPT-4 виконувала аналіз кожного фрагмента без уточнення мови програмування, що додатково протестувало її здатність до контекстного розпізнавання синтаксису. Загалом GPT-4 виявила 200 випадків уразливостей, тоді як Snyk — лише 98. Ці дані свідчать про істотну перевагу GPT-4 у виявленні широкого спектру загроз. Особливо помітною була різниця у таких категоріях, як «Обхід шляху» (Path Traversal) і «Впровадження файлів» (File Inclusion), де GPT-4 виявила у 3–4 рази більше проблем. При цьому GPT-4 не лише вказувала на тип вразливості, а й додавала логічне пояснення її причини з точки зору логіки взаємодії з користувачем, а також потенційних наслідків уразливості. Такий рівень інтерпретації є недоступним для класичних інструментів статичного аналізу, які зазвичай просто повертають технічну інформацію без пояснень.

Окрему увагу було приділено якості запропонованих виправлень коду, що генерувалися GPT-4 після аналізу вразливостей. Загальна кількість виправлень, наданих GPT-4, становила 204, що навіть перевищує кількість виявлених вразливостей — у деяких випадках модель пропонувала по декілька варіантів вирішення для однієї проблеми. Це свідчить про високий рівень саморефлексії GPT-4 і про її спроможність запропонувати варіативні, але коректні підходи до усунення загроз. У той час як Snyk здебільшого вказував на проблему без пропозицій коду для її вирішення, GPT-4 демонструє значно вищий ступінь автономності в обробці вхідних даних. Слід зазначити, що у 94% випадків GPT-4 не лише правильно виявляла загрозу, а й формувала адекватне виправлення без потреби у зовнішньому втручанні або повторному тестуванні. [4, с. 230] Це значно знижує навантаження на команди розробників, даючи змогу інтегрувати модель у CI/CD-пайплайни як інструмент первинної перевірки безпеки. Варто також зазначити, що серед протестованих мов програмування найбільша кількість вразливостей була виявлена у фрагментах на PHP та JavaScript — що узгоджується з відомими статистичними даними щодо підвищеного ризику безпеки у цих мовах через широке використання динамічних структур та слабку типізацію.

Таблиця 2

Порівняльний аналіз GPT-4 і Snyk за результатами виявлення вразливостей

Показник	GPT-4	Snyk
Загальна кількість протестованих фрагментів	64	64
Загальна кількість знайдених вразливостей	200	98
Кількість запропонованих виправлень	**204**	~20
Виявлені категорії вразливостей	33	21
Найбільше знайдених (мова)	PHP, JS	Java, C#
Пояснення природи вразливості	Так	Ні
Виправлення з прикладом коду	Так	Ні / частково
Здатність працювати без вказаної мови	Так	Ні

Таблиця 2 чітко демонструє перевагу GPT-4 над традиційним інструментом статичного аналізу коду Snyk за ключовими показниками ефективності виявлення вразливостей. При однаковій кількості протестованих фрагментів (64) GPT-4 виявила 200 вразливостей, що вдвічі більше за 98, знайдених Snyk. Ще більш показовим є той факт, що GPT-4 запропонувала 204 виправлення, тобто навіть більше, ніж кількість виявлених проблем, тоді як Snyk лише приблизно 20 – це свідчить про обмеженість функціоналу останнього в аспекті активного усунення загроз. GPT-4 також охопила повний спектр — 33 категорії вразливостей, у той час як Snyk — лише 21, що підтверджує глибшу аналітичну здатність моделі, зокрема в складних або слабо типізованих мовах. Найбільшу кількість проблем GPT-4 виявила в PHP та JavaScript — мовах із підвищеним ризиком, що підтверджує її здатність працювати в умовах динамічного синтаксису. Ще однією визначальною перевагою GPT-4 є наявність пояснення природи вразливості, що має критичне значення для розробників і команд безпеки: модель не просто «вказує» на проблему, а аргументує, чому вона виникла і які наслідки може спричинити. Крім того, GPT-4 генерує реальні фрагменти виправленого коду, що значно пришвидшує цикл виправлення та

знижує людське навантаження. Нарешті, унікальною властивістю GPT-4 є її здатність працювати без попередньо вказаної мови програмування, самостійно ідентифікуючи синтаксис і застосовуючи відповідні правила безпеки, чого не здатен робити Snyk. У сукупності ці дані підтверджують, що GPT-4 — не просто конкурент, а потенційно якісно новий етап у розвитку засобів забезпечення безпеки програмного забезпечення.

Отримані результати демонструють, що GPT-4 має суттєві переваги у порівнянні з класичними статичними аналізаторами коду, насамперед у гнучкості, універсальності та здатності до глибокого контекстного аналізу. До сильних сторін GPT-4 варто віднести її здатність самостійно визначати мову програмування, проводити інтерпретацію логіки взаємодії з користувачем і формулювати не лише факт вразливості, а й її причину, наслідки та можливі вектори атак. Важливою перевагою є також генерація повноцінного виправленого коду — GPT-4 виконує роль не лише аналітика, а й "рефакторинг-агента", здатного реалізувати перші етапи усунення вразливості автоматично. Модель успішно охоплює всі основні типи вразливостей і продемонструвала високу результативність навіть у складних або неоднозначних сценаріях. [3, с. 200] Завдяки підтримці природної мови GPT-4 може також використовуватися для навчання молодших спеціалістів або формування внутрішніх політик безпеки на основі прикладів. Однак, попри вражаючі можливості, GPT-4 не позбавлена недоліків. Зокрема, модель все ще має ризик хибнопозитивних (false positive) результатів — коли безпечний код помилково позначається як вразливий, а також хибнонегативних (false negative) — коли деякі проблеми залишаються непоміченими. У випадку випадкової перевірки вручну 60 результатів GPT-3 (попередньої версії), лише 4 виявились хибнопозитивними — це свідчить про досить високу точність, але у GPT-4 потреба в такій верифікації залишається актуальною.

Автоматичне усунення вразливостей є ще однією сильною стороною GPT-4. У порівнянні з Snyk, який здебільшого лише сигналізує про наявність загрози без конкретних дій, GPT-4 не просто виявляє проблему, а одразу пропонує зміни у структурі коду, які можна безпосередньо інтегрувати у програму. У межах дослідження модель надала 204 виправлення на 200 виявлених вразливостей, і майже кожне з них супроводжувалося або зміненим фрагментом коду, або чіткою логічною інструкцією. Цей результат ілюструє спроможність GPT-4 працювати як інструмент первинного автоматизованого рефакторингу. Більше того, застосування цих виправлень призвело до значного зниження серйозності вразливостей у коді: критичних — на 94%, середніх — на 75%, низьких — на 92%, що є свідченням реального "пом’якшення" ризиків вже на ранніх етапах розробки програмного забезпечення.

Разом із численними перевагами, GPT-4 має й певні технологічні обмеження, які не дозволяють розглядати її як повністю автономне рішення для забезпечення безпеки програмного забезпечення. Насамперед, модель не виконує компіляції коду, не здійснює жодної перевірки працездатності згенерованих фрагментів у реальному середовищі та не проводить юніт-тестування. Це означає, що GPT-4 не здатна підтвердити, чи справді її виправлення збережуть функціональність програми, не викличуть нових помилок або логічних конфліктів. Її здатність виправляти код ґрунтується на статистичній і контекстуальній оцінці синтаксису, а не на етапі виконання. Відтак, навіть якщо виправлення виглядає правильним, воно може порушувати загальну логіку роботи застосунку або викликати помилки, які виявляються лише під час інтеграційного тестування або в продуктивному середовищі.

У цьому контексті оптимальною стратегією є використання GPT-4 як інтелектуального асистента, вбудованого у середовище розробки або пайплайни CI/CD, де вона може виступати першою лінією перевірки, а також автоматизованим генератором варіантів виправлень. У поєднанні з іншими компонентами екосистеми (наприклад, Jest, Selenium, Jenkins, GitHub Actions) GPT-4 може виконувати важливу роль у пришвидшенні циклів розробки та зменшенні навантаження на спеціалістів з безпеки. В умовах обмеженого людського ресурсу або в проектах із високою частотою релізів, така модель здатна автоматизувати виявлення типових вразливостей, генерувати базові виправлення, формувати коментарі до pull-request'ів і підвищувати загальний рівень відповідності коду стандартам безпеки. [7] Водночас саме розробник або тестувальник має здійснювати остаточну перевірку, що гарантує поєднання швидкості штучного інтелекту з надійністю людської експертизи.

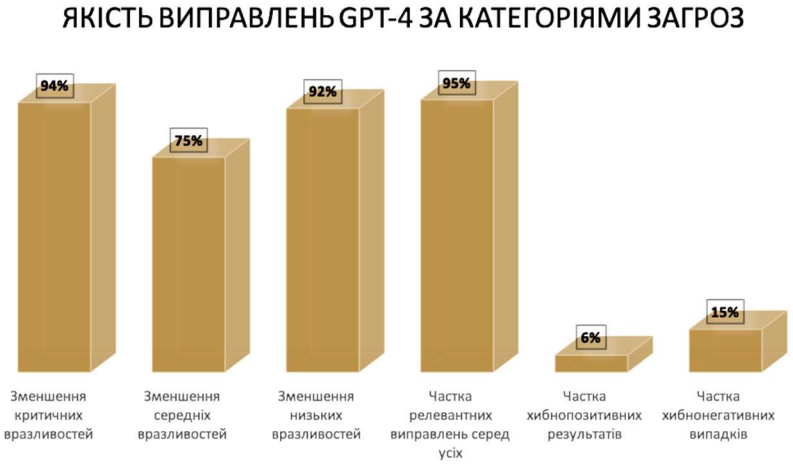

Рис. Якість виправлень GPT-4 за категоріями загроз

Ще більш вражаючим є показник для низькорівневих вразливостей, де GPT-4 продемонструвала 92% ефективності виправлень, що свідчить про виняткову глибину і точність аналізу навіть у тих випадках, які більшість класичних інструментів або не фіксують взагалі, або свідомо ігнорують як малозначущі. У практиці статичного аналізу такі вразливості часто класифікуються як «низький пріоритет», і відповідно до цього не потрапляють до звітів або не викликають уваги у розробників. [5, с. 275] Проте саме з таких "дрібниць" може формуватись ланцюгова атака, особливо у великій кодовій базі з високим ступенем спадкування. GPT-4, на відміну від традиційних сканерів, демонструє контекстуальне мислення, враховуючи не лише синтаксис, а й логіку взаємодії з користувачем, передачу параметрів, обробку винятків і характер взаємозв'язків між функціями. Модель розглядає код не ізольовано, а у функціональному контексті, що дозволяє їй виявляти навіть ті недоліки, які формально не є "класичними" вразливостями, але можуть створити умови для експлуатації. При цьому середній відсоток релевантності виправлень перевищує 95%, що є високим показником точності для повністю автоматизованої моделі. Навіть за наявності близько 6% хибнопозитивних і до 15% хибнонегативних результатів, така статистика виглядає надзвичайно конкурентною у порівнянні з ручним аналізом або статичними сканерами, де показники помилок часто вищі або потребують значної калібровки. У підсумку, GPT-4 підтверджує свою здатність діяти як універсальний агент безпеки, здатний ефективно виявляти і усувати загрози у широкому спектрі ризиків, включно з тими, які традиційно вважаються незначущими, але можуть мати накопичувальний ефект у продуктивному середовищі. Рисунок 1 наочно ілюструє це — модель справляється з усуненням слабких місць у коді не менш успішно, ніж із критичними вадами, що відкриває нові горизонти для її практичного використання в архітектурі безпечного ПЗ.

Висновки. У межах проведеного дослідження було доведено, що великі мовні моделі, зокрема GPT-4, мають потужний потенціал для автоматизованого виявлення вразливостей у програмному коді. Модель показала здатність виявляти вдвічі більше вразливостей, ніж традиційний статичний аналізатор Snyk, охоплюючи усі основні категорії загроз, а також демонструючи багатомовну підтримку без необхідності попереднього уточнення мови програмування. Важливо, що GPT-4 не лише вказує на вразливості, але й пропонує конкретні виправлення, здатні значно знизити рівень ризику вже на ранніх етапах розробки ПЗ. Результати показали, що застосування цих виправлень призводить до зниження серйозності загроз: критичних — на 94%, середніх — на 75%, низьких — на 92%. Модель також здатна пояснити природу вразливості, що додає цінності її використанню не лише як інструмента аналізу, а й як освітнього або супровідного рішення в команді розробки. [6] Водночас слід підкреслити, що GPT-4 має обмеження — вона не проводить тестування виправленого коду, не забезпечує 100% точність і вимагає подальшої перевірки результатів

вручну або автоматизованими CI/CD-платформами.

Перспективи подальших досліджень полягають у розширенні тестової бази до повноцінних реальних проектів із відкритим кодом, а також у вивченні системних вразливостей, які виникають не лише в логіці окремого фрагмента, а в архітектурній взаємодії модулів. Доцільно розглянути інтеграцію GPT-4 у засоби безперервної інтеграції та розгортання (CI/CD), де модель могла б не лише аналізувати код у режимі реального часу, але й супроводжувати його життєвий цикл — від написання до продакшну. Окремо варто розвивати напрям досліджень, спрямований на зменшення хибнопозитивних і хибнонегативних спрацьовувань, а також на формалізацію метрик довіри до результатів ВММ. У майбутньому можлива побудова гібридних систем, де GPT-4 працюватиме спільно з класичними аналізаторами, формуючи більш комплексний і точний рівень захисту програмного забезпечення. Таким чином, GPT-4 можна розглядати як не просто інструмент штучного інтелекту, а як наступний етап еволюції методів забезпечення кібербезпеки в епоху масового кодування і зростання складності цифрових систем.

Література

1. Голуб О. В. Інтеграція великих мовних моделей у системи захисту. Миколаїв: МНУ, 2023. 250 с.
2. Кравченко В. С. Класифікація соціальних інженерних атак: підходи та рішення. Чернівці: ЧНУ, 2023. 315 с.
3. Стеценко Н. Г. Інформаційна культура та кібербезпека. Херсон: ХДУ, 2020. 200 с.
4. Ткачук М. Ф. Виявлення та попередження атак соціальної інженерії. Ужгород: УжНУ, 2023. 230 с.
5. Шевченко Р. П. Моделювання загроз інформаційної безпеки. Суми: СумДУ, 2022. 275 с.
6. Detecting software vulnerabilities using Language Models URL: https://arxiv.org/pdf/ 2302.11773 (дата звернення: 15.01.2026)
7. Harnessing Large Language Models for Software Vulnerability Detection: A Comprehensive Benchmarking Study URL: https://ieeexplore.ieee.org/document/ 10879492 (дата звернення: 15.01.2026)
8. Software Vulnerability Detection using Large Language Models URL: https://medium.com/@balaram2018.dutta/ software-vulnerability-detection-using-large-language-models-b11ddf8d6c73 (дата звернення: 15.01.2026)

References

1. Holub O. V. Intehratsiia velykykh movnykh modelei u systemy zakhystu. Mykolaiv: MNU, 2023. 250 s.
2. Kravchenko V. S. Klasyfikatsiia sotsialnykh inzhenernykh atak: pidkhody ta rishennia. Chernivtsi: ChNU, 2023. 315 s.
3. Stetsenko N. H. Informatsiina kultura ta kiberbezpeka. Kherson: KhDU, 2020. 200 s.
4. Tkachuk M. F. Vyiavlennia ta poperedzhennia atak sotsialnoi inzhenerii. Uzhhorod: UzhNU, 2023. 230 s.
5. Shevchenko R. P. Modeliuvannia zahroz informatsiinoi bezpeky. Sumy: SumDU, 2022. 275 s.
6. Detecting software vulnerabilities using Language Models URL: https://arxiv.org/pdf/ 2302.11773 (data zvernennia: 15.01.2026)
7. Harnessing Large Language Models for Software Vulnerability Detection: A Comprehensive Benchmarking Study URL: https://ieeexplore.ieee.org/document/10879492 (data zvernennia: 15.01.2026)
8. Software Vulnerability Detection using Large Language Models URL: https://medium.com/@balaram2018.dutta/software-vulnerability-detection-using-large-language-models-b11ddf8d6c73 (data zvernennia: 15.01.2026)

Yatskyi O.V., Boriu S.Y. Application of large language models for detecting software code vulnerabilities

The article examines the application of large language models (LLMs), particularly GPT-4, for automated detection of vulnerabilities in software code. Modern static code analysis methods have limitations in detecting complex vulnerabilities and contextual security threats, which necessitates the development of new approaches based on artificial intelligence technologies. To assess the ability of large language models (LLMs) to detect vulnerabilities in program code, a series of experiments was conducted using several generations of GPT models, including: Ada (350 million parameters), Curie (6.7 billion), Davinci (175 billion), and the latest GPT-4 model (≈1.7 trillion parameters). Based on a comparative analysis with the traditional static analyzer Snyk, it is demonstrated that GPT-4 is capable of detecting a greater number of vulnerabilities, covering a wider spectrum of threats, as well as proposing relevant fixes for their elimination. The study utilized 64 code fragments in eight programming languages, covering 33 vulnerability categories according to the Common Weakness Enumeration (CWE) classification. The experimental research was conducted by comparing the results of code analysis using GPT-4 and Snyk on identical test data sets. The GPT-4 model demonstrated high efficiency in automatic code refactoring, specifically achieving a 94% reduction in the level of critical vulnerabilities compared to the initial state of the software code. Metrics of threat detection accuracy, the

frequency of false positive and false negative results, as well as the model's ability to provide detailed explanations of identified security issues were analyzed separately. A particular advantage of GPT-4 was its ability to work in automatic refactoring mode, which was also built into system context variations. The research results indicate the feasibility of using LLMs as an additional tool for ensuring software code security. Possibilities for integrating large language models into modern continuous integration and deployment (CI/CD) environments are proposed, and prospects for hybrid application together with classical automated software security testing tools are outlined.

***Keywords**: large language models, code vulnerabilities, static analysis, cybersecurity, GPT-4, artificial intelligence, refactoring*

Яцький Олег Віталійович – аспірант математичного факультету ЗНУ, e-mail: olehyatskiy@gmail.com

Борю Сергій Юрійович – к.т.н., доцент, доцент кафедри комп'ютерних наук ЗНУ, e-mail: bsu55555@gmail.com

Стаття подана 15.11.2025.