

DOI: 10.18372/2310-5461.71.21450

УДК 004.89:004.41

О. О. Дроздюк

Національний університет

«Київський авіаційний інститут»

<https://orcid.org/0009-0005-4785-6548>

e-mail: 9012874@stud.kai.edu.ua

ЕМПІРИЧНИЙ АНАЛІЗ ФАЗОВОЇ ПОВЕДІНКИ ШІ-АГЕНТІВ ПРОГРАМУВАННЯ

Вступ

Агенти штучного інтелекту (ШІ-агенти) дедалі ширше застосовуються для автоматизації практичних завдань програмної інженерії. Отримавши звіт про помилку (bug report), системи на кшталт SWE-agent [1], OpenHands [2] та Agentless [3] без участі людини виконують повний цикл усунення дефекту, а саме знаходять релевантний код у репозиторії, генерують виправлення та перевіряють його тестами. За допомогою таких агентів сьогодні оцінюють здатність великих мовних моделей (Large Language Models, LLM) вирішувати практичні задачі розробки програмного забезпечення. Стандартом такого оцінювання стали спеціалізовані тестові набори (benchmarks), і більшість сучасних порівняльних досліджень спирається на SWE-bench [4].

Попри швидке зростання частки вирішених задач, спосіб їх вирішення досліджено недостатньо. Попередні роботи переважно фіксували лише підсумковий результат, тобто частку задач, які агент зрештою вирішує. Внутрішня динаміка поведінки агентів залишалася поза увагою, а саме: які файли агент читає, як він чергує пошук і редагування коду та чи перевіряє власні зміни.

Актуальність проблеми посилює нещодавно описаний парадокс якості контексту. Додатковий контекст не гарантує кращого результату, а інколи навіть шкодить. Зі збільшенням довжини контексту точність моделей може знижуватися навіть тоді, коли весь доданий матеріал релевантний. Складні схеми керування контекстом і файли інструкцій для агентів подеколи лише підвищують обчислювальну вартість без покращення якості [5–8].

З наведених результатів випливає, що визначальним є не обсяг контексту, а його своєчасність, адже моделі потребують специфічної інформації на кожному етапі вирішення задачі. Обґрунтоване проектування фазово-залежних стратегій керування контекстом потребує точних вимірювань. Необхідно встановити, як сучасні агенти розподіляють зусилля між фазами задачі та як цей розподіл пов'язаний з успіхом, саме цьому й присвячено дослідження.

Аналіз останніх досліджень та публікацій

Сучасні ШІ-агенти програмування суттєво відрізняються за архітектурою – від повністю автономних до жорстко структурованих. На одному полюсі SWE-agent [1] надає LLM доступ до оболонки та спеціальний інтерфейс агент-комп'ютер (Agent-Computer Interface, ACI). Модель сама обирає інструменти та порядок їх застосування. LLM шукає, читає, редагує та запускає тести через єдиний набір команд. OpenHands [2] має подібну подієво-орієнтовану архітектуру, але запускає агента в ізолюваному середовищі виконання з ширшою інтеграцією інструментів.

На іншому полюсі Agentless [3] повністю усуває автономне прийняття рішень. Він використовує фіксований трифазний конвеєр: (1) ієрархічна локалізація звужує пошук від дерева проекту до файлу та функції, (2) генерація патчу створює потенційних кандидатів на виправлення на основі локалізованого контексту, і (3) валідація відбирає серед кандидатів через виконання тестів. Кожна фаза працює з окремим представленням кодової бази. Попри відсутність агентного каркасу, ця архітектура показала конкурентні результати.

Між цими полюсами є проміжні системи: PatchPilot [9], який додає до шаблону Agentless фази відтворення та вдосконалення, а OpenDev [20] – структурує роботу за окремими режимами (Plan Mode і Normal Mode) із власними промптами. Саме ця різноманітність архітектур і породжує питання, чи використовують такі системи контекст по-різному і чи позначається це на успішному вирішенні задач.

Окремим напрямом є контекстна інженерія, а саме практика відбору, упорядкування та форматування інформації для LLM. В роботі [10] розглянуто понад 1 400 таких робіт і запропоновано таксономії пошуку та генерації, обробки та керування контекстом, а генерацію коду з доповненим пошуком (Retrieval-Augmented Code Generation, RACG) окремо розглянуто в [11].

З'явилися й бенчмарки якості пошуку контексту. ContextBench [12], який складається з 1 136 задач і має еталонні контексти, розмічені людьми,

показав, що складні пошукові схеми дають лише незначну перевагу над простими. SWE Context Bench [13] вимірює точність локалізації файлів, а RepoBench [14] – міжфайлове доповнення коду.

В емпіричних роботах поступово складається уявлення про те, які типи контексту корисніші. Blinn та ін. [15] виявили, що визначення типів і сигнатури програмного інтерфейсу (Application Programming Interface, API) є особливо цінним контекстом для генерації коду. Натомість семантично подібні фрагменти коду виявляються найменш корисними, а їх пошук може навіть погіршити результат [6].

Нові дослідження показують, що великий обсяг контексту, всупереч очікуванням, може також погіршити якість роботи агента. Це явище далі позначено як парадокс якості контексту.

Найбільш систематично це розглянуто в роботі [5]. На кількох задачах і моделях показано, що точність LLM зменшується зі зростанням довжини контексту, навіть коли весь доданий матеріал релевантний. Зниження коливається від 13,9 % до 85 % залежно від типу задачі. Отже, вузьке місце криється в обробці контексту, а не в його пошуку.

В [6] показано, що пошук семантично подібного коду через генерацію з доповненим пошуком (Retrieval-Augmented Generation, RAG) знижує точність генерації коду до 15 %, оскільки це частіше додає до контексту шум, ніж корисну інформацію. Цей результат ставить під сумнів базове припущення більшості підходів із доповненим пошуком.

Парадокс стосується й файлів інструкцій для агентів. Gloaguen та ін. [7] дослідили файли AGENTS.md, тобто інструкції на рівні репозиторію, призначені допомогти ШІ-агентам. Виявилося, що ці файли знижують частку вирішених задач і водночас збільшують вартість у токенах на понад 20 %, адже автори таких інструкцій не можуть передбачити, яка інформація знадобиться агенту, і тому часто вписують туди зайві або суперечливі настанови.

Те саме підтвердили й Lindenbauer та ін. в роботі [8], уже для керування контекстом. Вони порівняли два способи дотримання ліміту контексту: сумаризацію історії засобами LLM та просте обрізання старих кроків. Складніший метод не виявився кращим. За часткою вирішених задач сумаризація зрівнялася з обрізанням, але обійшлася вдвічі дорожче. Автори назвали це «пасткою складності».

Отже, на основі цих робіт можна зробити висновок, що неконтрольований обсяг контексту не покращує результат, а часто навіть шкодить. Визначальною є доречність контексту, тобто те, який фрагмент коду потрібен агенту на конкрет-

ному кроці. Саме це підтверджує актуальність фазового підходу.

Розглядати дії агента як цілісний процес почали відносно недавно. Liu та ін. в [16] розклали траєкторію на послідовність позначених дій і ввели таксономію L (Локалізація), P (Генерація патчу), V (Валідація). На основі цих міток вони оцінили, як часто агент повертається назад, наскільки часто пропускає фази та наскільки тримається плану. Усі три ознаки виявилися пов'язаними зі складністю задачі.

Yang та ін. [17] обрали інший підхід. Вони вбудували фази прямо в архітектуру моделі (PA-MoE, Phase-Aware Mixture of Experts) і провели навчання моделі з підкріпленням на середовищах ALFWorld та WebShop (обидва – тестові середовища загального призначення, де є чіткі фази й кроки завдання). Після цього вони дійшли висновку, що різні фази задачі вимагають різних «когнітивних стратегій». Задач програмування ця робота не охоплює, тому напряду на досліджувані агенти її результати не переносяться. Водночас вона є вагомим аргументом на користь фазового підходу, реалізованого навіть на рівні ваг моделі.

Liu та ін. [18] запропонували підхід «Context as a Tool», який базується на тому, що агент постійно навчається, коли і яким способом стискати свою історію взаємодії у межах обмеженого бюджету на контекст. SWE-Pruner [19] реалізує більш цілеспрямований підхід. Він використовує адаптивне обрізання контексту на основі поточної фази задачі, що загалом зменшує споживання токенів на 23–38 %.

Опубліковані командою Nebius масштабні набори траєкторій для SWE-agent [21] та Open Hands [22] (понад 80 000 та 67 000 траєкторій відповідно) стали джерелом даних для проведеного міжагентного аналізу. Ця робота базується на системі L (Локалізація), P (Генерація патчу), V (Валідація) від Liu та ін. [16] та доповнює її аналізом переходів, декомпозицією за типами файлів і міжагентним порівнянням.

Постановка задачі дослідження

Аналіз літератури показує, що архітектури агентів програмування вже враховують різницю між фазами задачі. Agentless [3] використовує жорсткий трифазний послідовний конвеєр (pipeline) із різними представленнями коду на кожній стадії. PatchPilot [9] структурує роботу у п'ять фаз із різними фазово-залежними джерелами контексту. OpenDev [20] прямо зазначає, що різні фази потребують різних промптів. Про важливість фазової залежності контексту свідчать також емпіричні дослідження парадоксу якості контексту [5–8],

які показують, що збільшення обсягу контексту не гарантує кращої роботи агента.

Мета дослідження полягає у встановленні емпіричних закономірностей використання контексту ШІ-агентами в межах різних фаз задачі та у визначенні статистичного зв'язку між характеристиками процесу і успішністю вирішення задач.

Для досягнення поставленої мети необхідно:

- розробити та валідувати систему класифікації фаз, яка позначає кожен дію агента як Локалізацію (*L*), Генерацію патчу (*P*) або Валідацію (*V*) на основі сигнатур викликів інструментів, та перевірити її на агенті з явною фазовою структурою (Agentless);

- кількісно охарактеризувати розподіл фаз і закономірності переходів між ними для трьох архітектурно відмінних агентів (OpenHands, SWE-agent, Agentless) на вибірці 4 300 траєкторій;

- побудувати статистичну модель (узагальнена лінійна модель), що визначає, які характеристики процесу (розподіл фаз, кількість дій, частота переходів) є значущими предикторами успіху, та оцінити агент-специфічні ефекти взаємодії;

- виконати декомпозицію використання контексту за типами файлів у межах кожної фази та

порівняти розподіли для вирішених і невирішених траєкторій;

- виявити та описати якісно різні стратегії вирішення задач, що відображаються у закономірностях фазових переходів, і сформулювати практичні рекомендації щодо проектування агентів.

Виклад основного матеріалу

Для аналізу свідомо обрано три агенти з відкритим кодом, повні траєкторії яких викладені у вільний доступ – OpenHands, SWE-agent та Agentless. Запропонований метод спирається на покроковий журнал дій агента, без якого класифікувати дії за фазами неможливо. Комерційні інструменти на кшталт Claude Code, Cursor Agent/Composer чи OpenAI Codex таких журналів у відкритому доступі не надають, тому розкласти їхню роботу на фази запропонованим методом неможливо. Крім доступності даних, обрані агенти суттєво різняться архітектурою, від повністю автономних (OpenHands, SWE-agent) до жорстко конвеєрного (Agentless), що дає змогу порівняти різні стратегії вирішення, а не варіації одного підходу. Порівняння інструментів за критеріями відбору наведено в табл. 1.

Таблиця 1

Порівняння ШІ-агентів програмування за критеріями відбору

Інструмент	Відкритий код	Безкоштовний доступ	Публічні траєкторії	Увійшов у вибірку
OpenHands	так	так	так (набір Nebius)	так
SWE-agent	так	так	так (набір Nebius)	так
Agentless	так	так	так	так
Claude Code	ні	ні (підписка або API)	ні	ні
Cursor Agent/Composer	ні	частково (обмежений план)	ні	ні
OpenAI Codex	частково (CLI)	ні (моделі платні)	ні	ні

Вирішальним критерієм була наявність публічних покрокових траєкторій, без яких запропонований фазовий метод не може бути застосований.

Дані та класифікація фаз

У роботі досліджено:

- характер розподілу операцій між фазами локалізації дефектів, генерації патчів та їх валідації в ШІ-агентах;

- типи файлів, які агенти використовують у кожній фазі та їх вплив на успішність вирішення задачі;

- характеристики процесу (розподіл фаз, кількість дій, частота переходів), які впливають на вирішення задачі та вагові коефіцієнти для них;

- вплив архітектурної специфіки досліджуваної системи на детермінованість фазових профілів.

Проаналізовано траєкторії трьох агентів, кожен з яких відповідає окремому архітектурному підходу.

OpenHands ($n = 2\,000$ траєкторій) – подієво-орієнтований автономний агент з ізольованим виконанням коду. Його траєкторії взято з набору даних Nebius OpenHands на SWE-bench Verified. Кожна траєкторія фіксує повну послідовність викликів інструментів (читання файлів, редагування, команди `bash`) та їхні результати. З 2000 траєкторій 989 (49,5 %) завершилися вирішеною задачею.

SWE-agent ($n = 2\,000$ траєкторій) надає LLM спеціальний інтерфейс агент-комп'ютер для навігації репозиторієм та редагування. Його траєкторії взято з набору даних Nebius SWE-agent на SWE-bench Verified. SWE-agent подав патч у 93,7 % спроб. Суттєве застереження стосується визна-

чення результату, оскільки для SWE-agent статус «вирішено» в цьому аналізі визначається за фактом подання патчу (exit status «submitted»), а не за результатом виконання тестів бенчмарка. Отже, зміна результату для SWE-agent відображає подання патчу, а не підтверджене вирішення; докладніше це обмеження розглянуто в розділі «Обмеження дослідження».

Agentless ($n = 300$ траєкторій) використовує детермінований трифазний конвеєр без автономного прийняття рішень. Оскільки фазові межі жорстко закодовані в його архітектурі, то він слугує еталоном для валідації запропонованого класифікатора фаз на автономних агентах. 300 траєкторій відібрано з публічного випуску Agentless.

Усі три агенти оцінювалися на задачах з SWE-bench Verified – ретельно відібраної підмножини SWE-bench, що містить реальні GitHub-задачі з верифікованими тестовими патчами з 12 популярних Python-репозиторіїв.

Кожну дію в траєкторії агента класифіковано за однією з трьох фаз на основі сигнатур викликів інструментів. Оскільки OpenHands і SWE-agent мають різні формати траєкторій, для кожного агента реалізовано окремий конвеєр класифікації.

Для OpenHands класифікатор працює зі структурованими записами викликів інструментів (tool calls). Інструмент `str_replace_editor` класифікується за підкомандою:

$view \rightarrow L, create/str_replace/insert \rightarrow P.$

Інструмент `execute_bash` класифікується за вмістом команди через набір регулярних виразів. Інструмент `think` класифікується як L (етап обмірковування), `finish` – як V (подання результату).

Для SWE-agent класифікатор працює з текстовими повідомленнями. Команди зчитуються з блоків коду в повідомленнях агента за допомогою регулярних виразів. Перше слово команди зіставляється з набором SWE-agent-специфічних команд:

$find_file, search_file, search_dir, open, scroll_up, scroll_down, goto \rightarrow L; edit, create \rightarrow P; submit \rightarrow V.$

Нерозпізнані команди класифікуються через ті самі регулярні вирази, що й `bash`-команди OpenHands.

Класифікація `bash`-команд для обох агентів використовує три набори регулярних виразів: валідація (найвищий пріоритет) – `pytest, unittest, tox, flake8, mypy, ruff, pylint` тощо; генерація патчу – `sed -i, git apply, patch, cp, mv, rm`; локалізація – `find, grep, cat, head, ls, git log/diff/blame` тощо.

Локалізація (L) – дії, спрямовані насамперед на пошук релевантного коду, перегляд файлів, по-

шук за ключовими словами, навігація структурою репозиторію.

Генерація патчу (P) – дії, спрямовані на модифікацію коду, створення файлів, застосування патчів.

Валідація (V) – дії, спрямовані на перевірку коректності, запуск тестів, лінтерів, виконання скриптів відтворення помилки.

Класифікатор валідовано на Agentless, де фази задано явно на рівні конвеєра. Окремі модулі відповідають за локалізацію, генерацію патчу та валідацію. Саме з цими модулями звірено отримані мітки.

Метрики та статистичний аналіз

Для кожної траєкторії обчислено такі метрики.

Розподіл фаз. Для кожної фази ϕ з множини $\{L, P, V\}$ визначено частку дій, що належать до цієї фази:

$$D(\phi) = \text{count}(\phi) / N \cdot 100 \quad (1)$$

де N – загальна кількість дій у траєкторії.

Сукупність часток $D(L)$, $D(P)$ та $D(V)$ утворює фазовий профіль траєкторії. Фазовий профіль агента визначено як середні значення цих часток за всіма його траєкторіями. Саме цей показник використано для порівняння агентів.

Побудовано T – матрицю переходів 3×3 , де $T[i][j]$ позначає ймовірність переходу від фази i до фази j . Частота переходів визначається як частка послідовних пар дій, де фаза змінюється:

$$\text{Частота переходів} = (\text{кількість фазових переходів}) / (N - 1). \quad (2)$$

Файли, до яких агент звертався у кожній фазі, поділено на шість категорій: файли вихідного коду, тестові файли, конфігураційні файли, файли документації, `init`-файли та інше. Частку звернень у кожній категорії обчислено окремо для вирішених і невирішених траєкторій.

Для кожної фази та кожного типу файлів також обчислено різницю часток між вирішеними та невирішеними траєкторіями:

$$\Delta(c, \phi) = \text{частка}(c, \phi, \text{вирішено}) - \text{частка}(c, \phi, \text{невирішено}), \quad (3)$$

де $\text{частка}(c, \phi, \text{група})$ – відсоток доступів до файлів типу c серед усіх доступів у фазі ϕ для даної групи траєкторій. Додатні значення означають, що у вирішених траєкторіях звернення до файлів типу c у фазі ϕ трапляються пропорційно частіше, ніж у невирішених.

Для статистичної перевірки гіпотез застосовано два класи методів. Перший – узагальнена лінійна модель (GLM). Побудовано логістичну регресію, що передбачає вирішення задачі (бінар-

ний результат – вирішено чи ні) за характеристиками процесу. Залежна змінна – логарифм відношення шансів вирішення задачі (*logit*). Модель основних ефектів має вигляд [23]:

$$\begin{aligned} \text{logit}(P(\text{вирішено})) = & \beta_0 + \beta_1 \cdot L(\%) + \beta_2 \cdot V(\%) + \\ & + \beta_3 \cdot \text{частота переходів} + \\ & + \beta_4 \cdot \text{агент} + \beta_5 \cdot N \end{aligned} \quad (4)$$

Частку $P(\%)$ виключено, щоб уникнути повної мультиколінеарності ($L + P + V$ приблизно дорівнює 100). Модель взаємодії сумує доданки взаємодії *агент* $\times$ *предиктор*, щоб перевірити, чи відрізняється ефект характеристик процесу між агентами. У моделі взаємодії загальну кількість дій виключено з предикторів, щоб зменшити кількість параметрів і зосередитися на фазових характеристиках.

Другий клас методів – непараметричний критерій Манна–Уїтні. Ним порівняно вирішені та невирішені траєкторії за кожною метрикою процесу, оскільки розподіли кількості дій та відсотків фаз мають правосторонню асиметрію. Загалом перевірено $k = 6$ метрик (*загальна кількість дій*, $L\%$, $P\%$, $V\%$, *частота переходів*, *кількість фазових переходів*). Застосовано

поправку Бонферроні для множинних порівнянь ($\alpha' = 0,05 / 6 \approx 0,0083$). Для групових порівнянь за критерієм Манна–Уїтні наведено рангово-бісеріальну кореляцію (r) як міру розміру ефекту, для коефіцієнтів GLM – 95 % довірчі інтервали.

Тип кожного рисунка обрано за структурою даних. Композицію фаз, знакові різниці часток і коефіцієнти моделі з довірчими інтервалами показано спорідненими горизонтальними діаграмами (відповідно стовпчиковою, діаграмою різниць та *forest plot*); теплові карти застосовано лише для матриць переходів, а діаграми розмаху (*boxplot*) – лише для повних розподілів метрик, де інші типи втратили б частину інформації. Усі рисунки виконано в єдиній кольоровій схемі: синій позначає локалізацію (L), помаранчевий – генерацію патчу (P), зелений – валідацію (V).

Результати дослідження

Розподіл фаз між агентами

Фазові профілі трьох агентів помітно відрізняються (рис. 1). Найбільша частка дій у кожного припадає на локалізацію – 54,4 % в OpenHands, 50,9 % у SWE-agent і 59,1 % в Agentless.

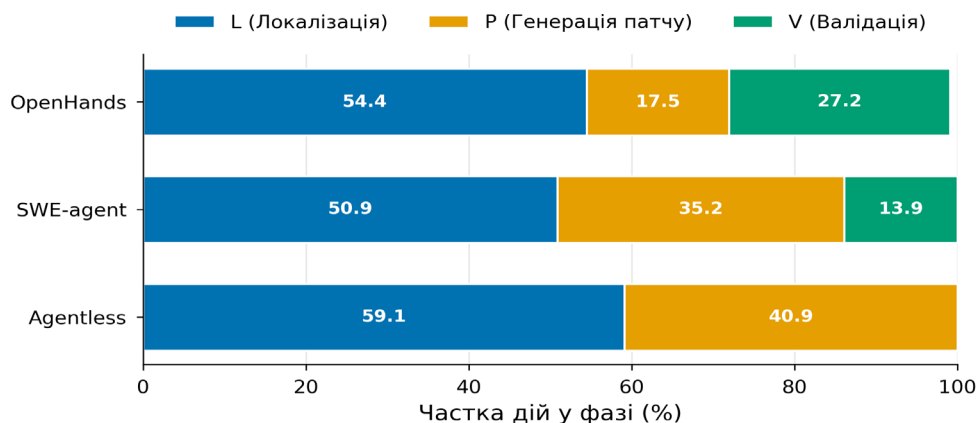


Рис. 1. Розподіл фаз за агентами: частка дій локалізації (L), генерації патчу (P) та валідації (V)

Домінування локалізації очікуване, адже найскладнішим у виправленні помилки є пошук потрібного місця в коді, а не сама зміна. Натомість баланс між генерацією патчу та валідацією у трьох агентів різний, тому саме він становить найбільший інтерес.

OpenHands відводить на валідацію 27,2 % дій, майже вдвічі більше, ніж SWE-agent (13,9 %). Ця різниця, найімовірніше, зумовлена архітектурою, оскільки OpenHands працює в ізольованому середовищі виконання, де запускати тести зручно. Командний інтерфейс SWE-agent такого зручного зворотного зв'язку від тестів не дає. OpenHands також робить помітно більше дій у середньому (64 проти 28). Він досліджує широко, тоді як

SWE-agent діє точково. Чи є це перевагою, розглянуто далі, оскільки довші траєкторії частіше пов'язані з невдачею, ніж із ретельністю.

Agentless має особливий профіль. Його жорсткий конвеєр не відводить автономних дій на валідацію, і валідація виконується зовні, через запуск тестового набору на згенерованих кандидатних патчах. Частка локалізації в нього найвища серед трьох агентів (59,1 %), а решта 40,9 % дій припадає на генерацію патчу. Такий розподіл відповідає стратегії ієрархічного звуження від дерева проєкту до файлу та функції.

SWE-agent виділяє на генерацію патчу 35,2 % дій, удвічі більше за OpenHands (17,5 %). Це стратегія концентрованого редагування. Знайшовши

релевантний код, SWE-agent витрачає більшу частку зусиль на створення та вдосконалення патчу і відносно менше на його перевірку.

Типи контексту за фазами

У табл. 2 наведено розподіл типів файлів за фазами для вирішених і невирішених траєкторій

OpenHands ($n = 2\,000$; 989 вирішених, 1 011 невирішених). На рис. 2 показано різницю часток між вирішеними та невирішеними траєкторіями для кожного типу файлів у кожній фазі. Додатні значення означають, що у вирішених траєкторіях звернення до відповідного типу файлів трапляються пропорційно частіше.

Таблиця 2

Розподіл типів файлів (%) за фазами та результатом (OpenHands)

Тип файлу	L-Виріш.	L-Невиріш.	P-Виріш.	P-Невиріш.	V-Виріш.	V-Невиріш.
Файл вихідного коду	38,4	40,7	62,8	61,8	7,5	6,4
Тестовий файл	11,4	10,4	27,6	28,9	24,3	24,8
Конфігураційний файл	1,3	1,1	0,1	0,2	0,1	0,2
Файл документації	3,5	3,0	0,3	0,4	0,04	0,04
Init-файл	1,6	1,3	0,8	1,1	0,01	0,01
Інше	43,8	43,5	8,4	7,6	68,0	68,6

Різниця часток типів файлів: вирішені – невирішені (OpenHands)

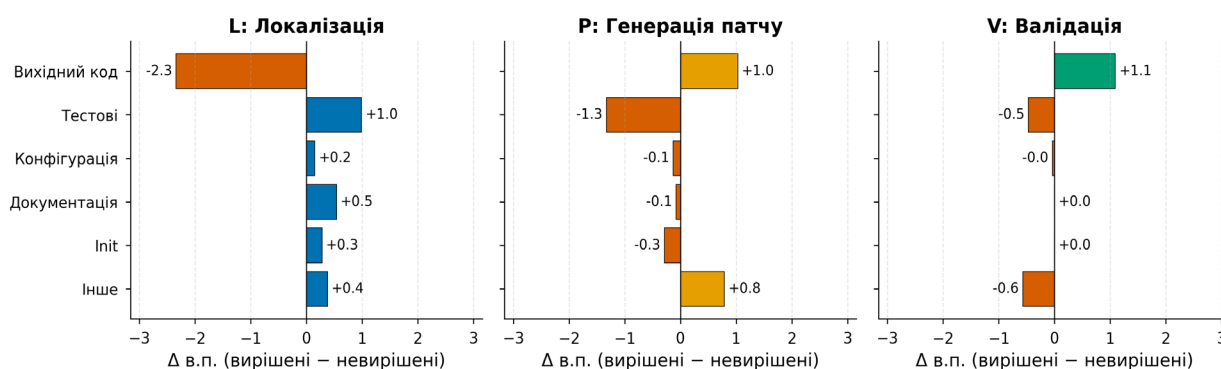


Рис. 2. Фазовий розподіл різниць часток типів файлів (Δ %) між вирішеними та невирішеними траєкторіями OpenHands

З порівняння вирішених і невирішених траєкторій за фазами можна зробити такі висновки. Під час локалізації частка звернень до тестових файлів у вирішених траєкторіях вища, ніж у невирішених (11,4 % проти 10,4 %, різниця 1,0 відсоткового пункту). Можна припустити, що успішні агенти ще до спроби виправлення читають тестові випадки, щоб зрозуміти очікувану поведінку коду. В успішних спробах агент також читає під час локалізації менше файлів вихідного коду (38,4 % проти 40,7 %). Пошук тут цілеспрямований, без зайвого перегляду коду.

Під час генерації патчу закономірність для тестових файлів змінюється на протилежну, в успішних спробах агент рідше редагує тестові файли (27,6 % проти 28,9 %, різниця 1,3 відсоткового пункту). Редагувати тестові файли під час генерації патчу в задачах SWE-bench загалом небажано, адже еталонний тестовий набір фіксо-

ваний. Агент, який модифікує тести, може «обходити» проблему замість вирішувати її.

Під час валідації розподіли типів файлів для вирішених і невирішених траєкторій подібні. Найбільша категорія – «інше» (68,0 % проти 68,6 %). Це переважно результати bash-команд, логи програми запуску тестів та повідомлення про помилки. Це очікувано, адже дії валідації здебільшого визначаються тестовим каркасом, а не стратегією агента.

Файли документації та конфігурації становлять малу частку доступів у всіх фазах (менше 5 % сумарно). Агенти рідко користуються цими типами контексту незалежно від результату.

Статистичний аналіз чинників успішності вирішення задач

Порівняння метрик процесу між вирішеними ($n = 989$) і невирішеними ($n = 1\,011$) траєкторіями OpenHands наведено в табл. 3.

Таблиця 3

Метрики процесу для вирішених та невирішених траєкторій (OpenHands, критерій Манна-Уїтні)

Метрика	Вирішені (медіана)	Невирішені (медіана)	p-значення
Загальна кількість дій	56	66	< 0,0001
V (%)	28,0	26,0	< 0,0001
P (%)	16,4	17,7	0,0001
Фазові переходи	25	29	< 0,0001

Усі чотири наведені порівняння статистично значущі після поправки Бонферроні ($\alpha' = 0,05/6 \approx 0,0083$). Два додаткових порівняння (L % і частота переходів) мають p -значення 0,016 та 0,032 відповідно, що перевищує скоригований поріг, тому в таблиці їх не наведено.

Вирішені траєкторії коротші (медіана 56 проти 66 дій, $p < 0,0001$). Зв'язок між довжиною траєкторії та успіхом стабільний для обох агентів і узгоджується з висновками про «пастку складності» [8]. Отже, велика кількість дій найчастіше свідчить не про ретельність, а про блукання та непродуктивний пошук.

У вирішених траєкторіях частка валідації вища (28,0 % проти 26,0 %, $P < 0,0001$), а частка генерації патчу нижча (16,4 % проти 17,7 %, $p = 0,0001$). Нижча частка генерації патчу свідчить, що успішні агенти редагують ефективніше і їм потрібно менше правок, щоб дійти до коректного виправлення.

Абсолютна кількість фазових переходів у вирішених траєкторіях нижча (медіана 25 проти 29, $p < 0,0001$). Однак вирішені траєкторії ще й коротші, тому частота переходів (переходи на дію)

у них фактично вища. Відносно загальної кількості дій успішні агенти змінюють фази частіше, а пошук, редагування та перевірка чергуються в них щільніше.

У табл. 4 наведено GLM (узагальнену лінійну модель) основних ефектів, що передбачає успіх вирішення за характеристиками процесу для OpenHands і SWE-agent ($n = 4\ 000$; Agentless до моделі не включено, оскільки його валідація зовнішня, а мітки «вирішено» на рівні окремої траєкторії немає). Перш ніж інтерпретувати коефіцієнти, потрібне важливе застереження, оскільки залежну змінну для двох агентів визначено порізно. Для OpenHands «вирішено» означає підтверджене проходження тестів бенчмарка (49,5 % траєкторій), тоді як для SWE-agent лише факт подання патчу зі статусом «submitted» (93,7 % траєкторій). Це змішування пояснює великий коефіцієнт для агента ($OR = 23,244$) і потенційно впливає на оцінку інших коефіцієнтів, тому результати таблиці слід інтерпретувати з урахуванням цього обмеження.

Якість моделі: $AIC = 3474,0$, точність = 76,7 %, $F1 = 0,842$.

Таблиця 4

Логістична GLM для передбачення вирішення задачі за характеристиками процесу

Предиктор	Коеф.	OR	95% ДІ	z	p
Вільний член	-1,573	0,207	[0,103, 0,419]	-4,38	< 0,001
L (%)	0,012	1,012	[1,005, 1,018]	3,59	< 0,001
V (%)	0,037	1,037	[1,026, 1,049]	6,67	< 0,001
Частота переходів	2,060	7,843	[3,934, 15,637]	5,85	< 0,001
Агент (SWE-agent)	3,146	23,244	[16,793, 32,173]	18,97	< 0,001
Загальна кількість дій	-0,016	0,984	[0,981, 0,987]	-9,78	< 0,001

Усі предиктори статистично значущі ($p < 0,001$). Їхній практичний ефект на шанси вирішення показано на рис. 3. Зелений колір на рисунку відповідає предикторам, які підвищують імовірність вирішення, помаранчевий тим, які її знижують, а вертикальна пунктирна лінія позначає відсутність ефекту. Коефіцієнт для агента є технічним контролем, тому на рисунку його не наведено. Найбільший ефект серед характеристик

процесу має частота переходів. Її $OR = 7,843$ відповідає одиничній зміні величини (від 0 до 1), тобто теоретичному максимуму. У практичному діапазоні збільшення частоти переходів на 0,1 (тобто на 10 % більше зміни фаз на дію) збільшує шанси на вирішення приблизно на 23 % ($\exp(2,060 \times 0,1) = 1,23$). Отже, швидке чергування пошуку, редагування й тестування дає кращі результати, ніж тривала робота в одній фазі.

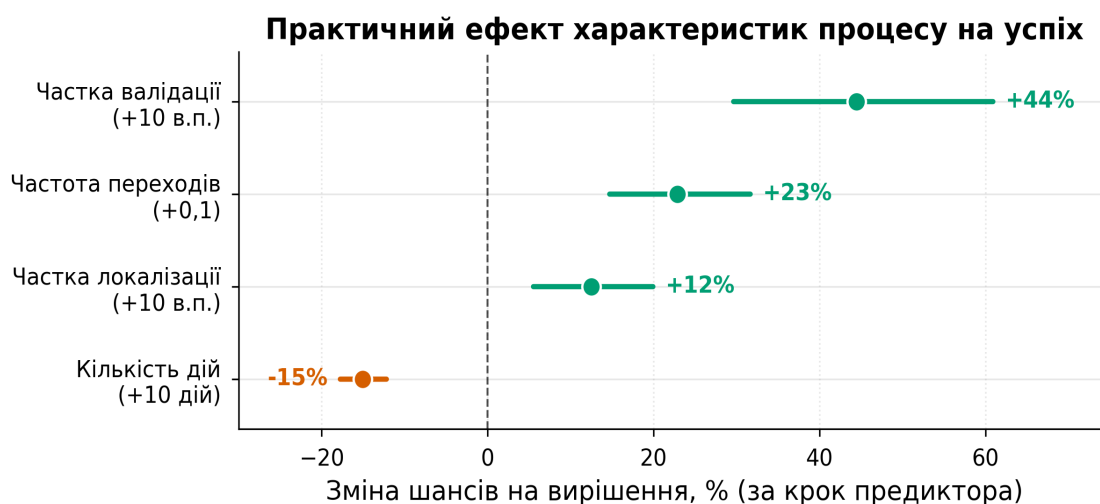


Рис. 3. Зміна шансів на вирішення задачі за реалістичний крок кожного предиктора з 95 % довірчими інтервалами

Серед фазових часток найсильніший предиктор – частка валідації. Кожен додатковий відсотковий пункт V збільшує шанси на вирішення на 3,7 % ($OR = 1,037$). На практиці агент, який відводить 30 % дій на валідацію, має приблизно на 44 % вищі шанси на вирішення, ніж той, який відводить 20 %, за інших рівних умов ($1,037^{10} \approx 1,44$).

Загальна кількість дій діє у протилежний бік ($OR = 0,984$), оскільки кожна додаткова дія зменшує шанси на вирішення приблизно на 1,6 %, тобто довші траєкторії асоціюються з невдачею.

Як ці ефекти виглядають у термінах самої ймовірності успіху, показано на рис. 4. Інші предиктори зафіксовано на типових значеннях, і зі зростанням кожної з двох характеристик ймовірність успіху помітно підвищується.

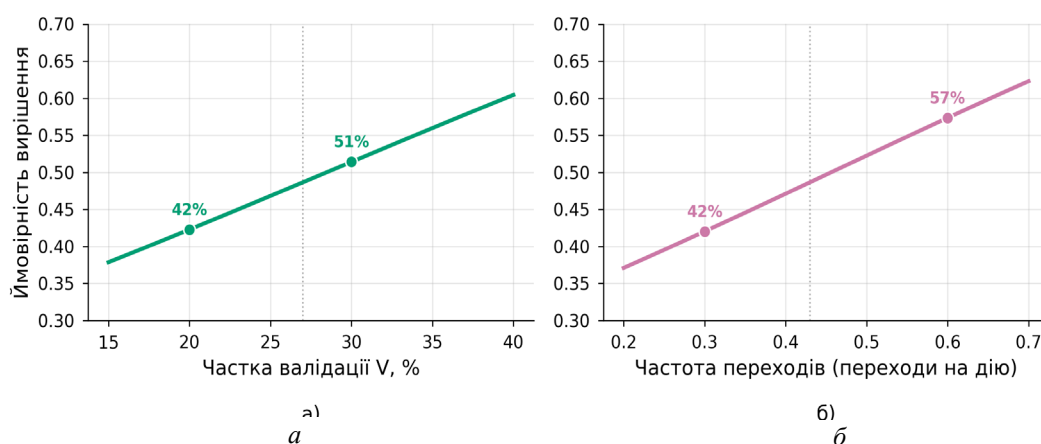


Рис. 4. Ймовірність успішного вирішення задач залежно від параметрів процесу: а – вплив частки валідації; б – вплив частоти фазових переходів

Великий коефіцієнт для агента ($OR = 23,244$) – це згаданий вище контроль базової різниці між агентами, а не характеристика процесу.

Модель взаємодії (табл. 5) дозволяє перевірити, чи однаково характеристики процесу діють для різних агентів.

Таблиця 5

GLM з агент-специфічними ефектами взаємодії

Предиктор	Коеф.	OR	95 % ДІ	z	p
L (%)	0,058	1,060	[1,037, 1,083]	5,26	< 0,001
V (%)	0,080	1,084	[1,062, 1,106]	7,72	< 0,001
Частота переходів	3,270	26,323	[6,416, 107,992]	4,54	< 0,001
L(%) × Агент	-0,056	0,946	[0,924, 0,967]	-4,83	< 0,001
V(%) × Агент	-0,021	0,979	[0,944, 1,015]	-1,17	0,243
Частота переходів × Агент	0,006	1,006	[0,166, 6,080]	0,01	0,995

Взаємодія $L(\%) \times$ Агент статистично значуща ($OR = 0,946, P < 0,001$). Додатний основний ефект $L(\%)$ ($OR = 1,060$) діє для OpenHands, але для SWE-agent комбінований ефект становить $OR = 1,060 \times 0,946 = 1,003$, тобто фактично нульовий. Більші зусилля на локалізацію допомагають Open Hands, але не SWE-agent. Ймовірно, це пояснюється архітектурою: в OpenHands складніша інтеграція інструментів, тому ретельна локалізація окупається. Спрощений ACI SWE-agent, навпаки, зменшує ціну неточної локалізації: агентові легше виправитися вже на етапі генерації патчу.

Взаємодії $V(\%) \times$ Агент та Частота переходів $\times$ Агент статистично не значущі ($p = 0,243$ та $P = 0,995$ відповідно). Перевага валідації та щільного чергування фаз стабільна для обох агентів.

Стабільний додатний зв'язок між часткою валідації та успіхом вирішення ($OR = 1,037, P < 0,001$) є, ймовірно, найважливішим практичним результатом роботи. Він зберігається для обох агентів у моделі взаємодії ($V(\%) \times$ Агент статистично не значуща), тобто не є артефактом одного агента. Найімовірніше, причина полягає у своєчасності перевірки, оскільки часті тести виявляють помилку, поки її легко виправити, а відкла-

дена валідація показує її, коли патч уже фактично готовий.

Цей результат узгоджується з дослідженнями про парадокс якості контексту, де Lindenbauer та ін. [8] показали, що прості підходи перевершують складні. Тест відповідає однозначно – пройшов чи ні. Це надійніше за саморефлексію LLM, де модель оцінює сама себе й може видати бажане за дійсне. Водночас зв'язок між $V(\%)$ та успіхом кореляційний і не виключено, що простіші задачі водночас потребують менше дій (коротші траєкторії з пропорційно більшою валідацією) і частіше вирішуються. Модель контролює загальну кількість дій, що частково знімає цей ризик. Але повністю виключити вплив складності задачі без контрольованого експерименту неможливо.

Міжагентне порівняння стратегій

Ймовірності фазових переходів для OpenHands і SWE-agent подано у вигляді теплових карт на рис. 5, де інтенсивність кольору відповідає ймовірності переходу від рядка-фази до стовпця-фази. Діагональні елементи показують частоту збереження поточної фази, позадіагональні відповідають переходам між фазами. Матриці виявляють якісно різні стратегії вирішення задач.

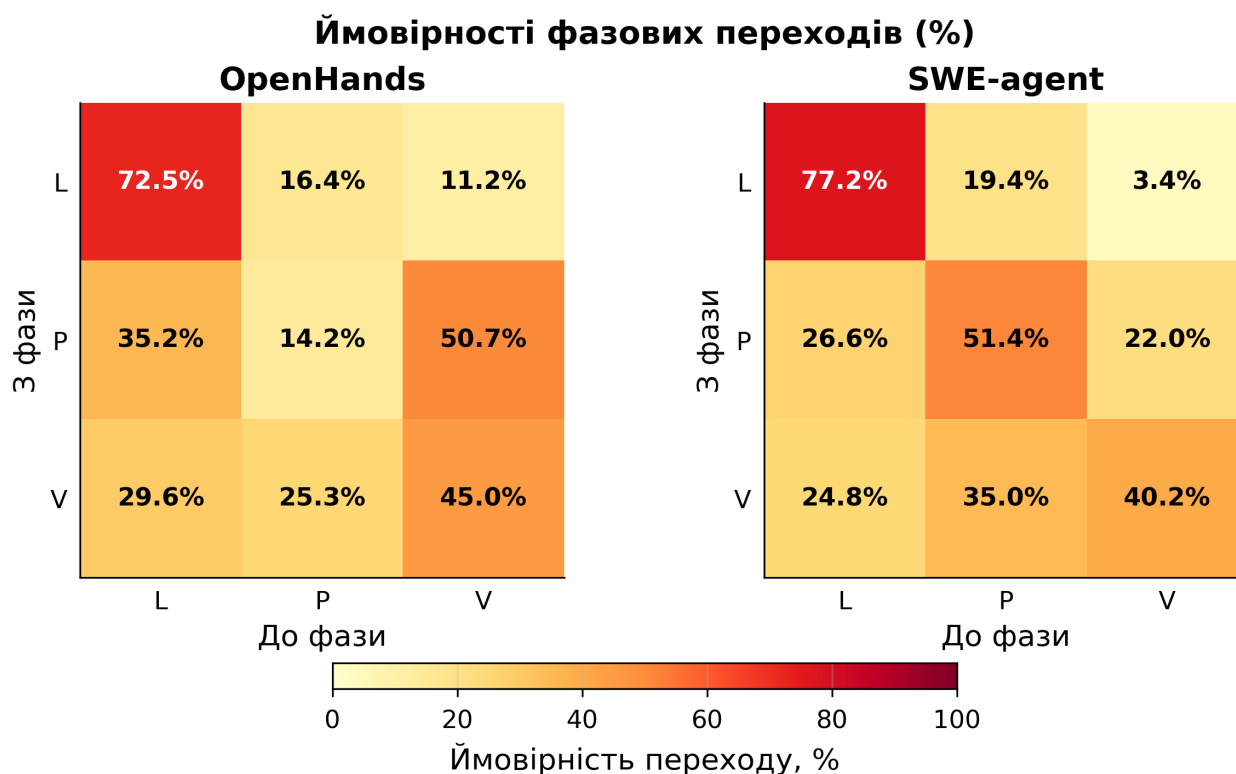


Рис. 5. Теплові карти матриць фазових переходів для OpenHands та SWE-agent

Обидва агенти найчастіше залишаються в межах локалізації (ймовірність $L \rightarrow L$ становить 72,5 % для OpenHands і 77,2 % для SWE-agent), однак поза цією спільною рисою їхні стратегії помітно розходяться.

Для OpenHands характерний ітеративний стиль. Після створення патчу він переходить до валідації у 50,7 % випадків, а повторно редагує без проміжної перевірки лише у 14,2 %. Кілька редагувань поспіль без пошуку чи тестування між ними трапляються рідко. Послідовність «відредагуй, а потім перевір» створює щільні цикли зворотного зв'язку, а після валідації OpenHands найчастіше продовжує перевірку (45,0 %) або повертається до локалізації (29,6 %), щоб уточнити розуміння проблеми.

SWE-agent поводить інакше, після генерації патчу він найчастіше переходить до нової генерації патчу (51,4 % повторних редагувань), а до тестування він переходить лише у 22,0 % випадків. Агент робить кілька послідовних правок, перш ніж перевірити результат. Патч удосконалюється через подальше редагування, а не через зворотний зв'язок від тестів. SWE-agent також частіше переходить одразу від локалізації до редагування (19,4 % проти 16,4 % у Open Hands) і значно рідше виконує проміжну перевірку під час локалізації (3,4 % проти 11,2 %).

Agentless становить окремий крайній випадок, його переходи задані архітектурою. Конвеєр завжди рухається L , потім P , потім V (зовнішня), без циклів зворотного зв'язку. Відсутність ітеративного доопрацювання компенсується тим, що агент генерує кілька патчів-кандидатів і потім обирає найкращий за результатами тестів.

OpenHands і SWE-agent виконують ту саму роботу, але різними шляхами, або ітеративним, або цілеспрямованим. За ітеративної стратегії агент збирає велику кількість інформації, часто її перевіряє та працює через щільні цикли зворотного зв'язку. За цілеспрямованої стратегії він намагається одразу точно локалізувати проблему й редагує точково, з мінімумом додаткових дій. Ітеративна стратегія рано виявляє помилки, оскільки тестування відбувається майже після кожного редагування. Платою за це є більша кількість дій, у середньому 64 на задачу, причому частина перевірок лише підтверджує вже правильний патч. Цілеспрямована стратегія коротша й містить у середньому лише 28 дій, але є менш надійною. Якщо початкова локалізація неточна або перший патч хибний, то через рідкі перевірки помилка встигає поширитися, перш ніж її виявлять.

Різниця зводиться до вибору між тривалим пошуком кращих варіантів і швидкими діями з

тим, що вже знайдено. Найкращий баланс, імовірно, залежить від складності задачі, і що складніша вона, то більше потрібно досліджень. Результати GLM цьому не суперечать. Від'ємний коефіцієнт для кількості дій ($OR = 0,984$) показує, що коротші траєкторії загалом асоціюються з успішним виконанням завдання. Додатний коефіцієнт для частки валідації ($OR = 1,037$) свідчить, що без перевірки не обійтися. Agentless виходить за межі цього компромісу, оскільки замість послідовного прийняття рішень покладається на схему «згенеруй і відфільтруй». Його висока частка локалізації (59,1%) відображає ставку на те, що вирішальною фазою є саме локалізація.

Рекомендації щодо проєктування агентів

Аналіз типів файлів показує, що контекст треба добирати залежно від фази. Під час локалізації корисний сигнал дають тестові файли. В успішних спробах їхня частка у фазі L на 1 в.п. вища. Під час генерації патчу агенту варто зосередитися на файлах вихідного коду й утримуватися від редагування тестів. В успішних спробах частка редагування тестових файлів у фазі P на 1,3 в.п. нижча. Різниця невелика, але стабільна, і це підтверджує, що різним фазам потрібен різний контекст.

Найсильніший предиктор успіху – частота переходів ($OR = 7,843$). За отриманими даними, що частіше агент чергує пошук, редагування й перевірку, то вищі його шанси. Отже, замість поширеної схеми «сплануй, потім виконай» варто використовувати цикл «шукай, редагуй, тестуй, повтори».

Від'ємний зв'язок між загальною кількістю дій та успіхом ($OR = 0,984$) свідчить, що агентам потрібні явні бюджети дій або евристики раннього завершення, адже сучасні агенти зазвичай вичерпують увесь бюджет, перш ніж припинити роботу, і витрачають ресурси на спроби, які вже не мають перспективи.

Інфраструктура валідації теж впливає на результат. Стабільна перевага валідації в обох агентів означає, що варто вкладатися у швидке та надійне виконання тестів. Ізольовані середовища виконання, як в OpenHands, дають змогу працювати ітеративно, з частими перевітками, а саме це побудована модель пов'язує з успішним виконанням завдання. Агенти без зручного доступу до тестової інфраструктури цієї переваги не мають.

Обмеження дослідження

Першою загрозою внутрішній валідності є різне визначення залежної змінної для різних

агентів. Для OpenHands статус «вирішено» визначається проходженням тестів бенчмарка, тоді як для SWE-agent – фактом подання патчу (93,7% траєкторій). Це змішує дві різні залежні змінні в GLM, що ймовірно пояснює великий коефіцієнт для агента ($OR = 23,244$) і може впливати на інші коефіцієнти.

Другою загрозою є вплив побічних змінних, оскільки застосована GLM виявляє лише асоціації, а не каузальні зв'язки. Складність задачі – правдоподібна побічна змінна. Прості задачі можуть давати коротші траєкторії з більшою часткою валідації, і саме це створює видимий зв'язок з успіхом, хоча причинного механізму немає. Включення загальної кількості дій як коваріати частково знімає цей ризик, проте повністю розвести кореляцію й причинність на спостережних даних неможливо, тому що для цього довелось б штучно задавати розподіл фаз в експерименті.

Окремим джерелом похибки є і сам класифікатор фаз. Його валідовано на явних фазах Agentless, але в автономних агентів дії на межах фаз інколи можна віднести до фази лише умовно. Така похибка скоріше послаблює реальні ефекти, ніж створює фальшиві.

Зовнішня валідність обмежена обсягом аналізу. Три агенти й один бенчмарк (SWE-bench Verified), що складається виключно з Python-репозиторіїв, тому на інші мови програмування результати можуть і не перенестися. Додає невизначеності й те, що агенти працюють на різних базових LLM (GPT-4, Claude, відкриті моделі залежно від конфігурації), отже, неможливо впевнено приписати ефекти саме стратегії, а не базовій моделі. Та сама обставина частково працює й на користь дослідженню, оскільки те, що результати тримаються для двох архітектурно різних агентів і на різних моделях, радше підвищує впевненість у їхній узагальненості.

Спрощеність таксономії L/P/V обмежує конструктивну валідність. Хоча вона добре закріплена в попередніх роботах [16], поза нею лишаються дії, що поєднують фази (наприклад, читання файлу перед редагуванням) або не належать до жодної (наприклад, налаштування середовища). Категорія «інше» в аналізі типів файлів частково охоплює такі випадки. Окрім того, фаза валідації для Agentless є зовнішньою стосовно траєкторії агента, що ускладнює пряме порівняння часток V між усіма трьома агентами, тому, де це доречно, Agentless про-аналізовано окремо.

Існує застереження і щодо статистичної валідності, тому що GLM припускає незалежність спостережень лише за умови коваріат.

Задачі у SWE-bench походять із різних репозиторіїв та часових періодів, що зменшує, але не усуває залежність. В ідеалі варто було б використати модель змішаних ефектів із випадковими вільними членами для репозиторію та задачі. Поточна модель є GLM із фіксованими ефектами, і вона може занижувати стандартні похибки за наявності залишкової кластеризації. Для множинних порівнянь у критеріях Манна–Уїтні застосовано поправку Бонферроні ($k = 6$, $\alpha' \approx 0,0083$), і два з шести порівнянь ($L\%$ та частота переходів) не досягли скоригованого порогу значущості.

Висновки

У статті подано міжагентний емпіричний аналіз того, як III-агенти використовують контекст за фазами задачі. Класифікувавши за фазами 4 300 траєкторій архітектурно відмінних агентів, встановлено, що агенти вирішують задачі по-різному і що ці відмінності пов'язані з успішністю.

Ключові кількісні результати показують, що локалізація домінує у всіх архітектурах (50,9–59,1 % дій), а пошук потрібного коду забирає в агентів найбільше зусиль і лишається їхнім вузьким місцем. GLM-модель виявляє три статистично значущі предиктори успіху, а саме частоту фазових переходів ($OR = 7,843$), частку валідації ($OR = 1,037$) та загальну кількість дій ($OR = 0,984$). Аналіз матриць переходів виявляє дві якісно різні стратегії вирішення, ітеративну (OpenHands) та цілеспрямовану (SWE-agent).

Відкритих питань залишається багато. Спостережні дані не дозволяють відокремити кореляцію від причинності, а з'ясувати це міг би контрольований експеримент, що маніпулює розподілом фаз, наприклад із примусовою перевіркою після кожного редагування. Аналіз обмежений трьома агентами на одному бенчмарку (SWE-bench Verified, Python), тож його варто розширити на інші мови та класи задач, такі як рефакторинг, генерація з нуля, оптимізація продуктивності. Розгляд контексту на рівні токенів, а не доступу до файлів, показав би, яка саме інформація визначає успіх у кожній фазі. Відмінність між стратегіями підказує ідею адаптивного підходу, за якого стратегію обирають динамічно, за попередньою оцінкою складності задачі. Також корисною була б валідація класифікатора на людських розмітках, яка дала б змогу оцінити похибку на межах фаз.

Наукова новизна полягає у виконанні міжагентного емпіричного аналізу використання контексту III-агентами на рівні фаз задачі. Запропоновано уніфіковану систему класифі-

кації дій агентів різних архітектур (автономних та конвеєрних) за фазами L/P/V. Узагальненою лінійною моделлю на вибірці 4 300 траєкторій кількісно встановлено зв'язок між характеристиками процесу (частка валідації, частота переходів, кількість дій) та ймовірністю успіху. Виявлено й описано дві якісно різні стратегії вирішення, ітеративну та цілеспрямовану.

ЛІТЕРАТУРА

- [1] Yang J., Jimenez C. E., Wettig A., et al. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. *Advances in Neural Information Processing Systems (NeurIPS 2024)*. 2024. URL: <https://arxiv.org/abs/2405.15793>.
- [2] Wang X., Li B., Song Y., et al. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. *International Conference on Learning Representations (ICLR 2025)*. 2024. URL: <https://arxiv.org/abs/2407.16741>.
- [3] Xia C. S., Deng Y., Dunn S., Zhang L. Agentless: Demystifying LLM-based Software Engineering Agents. *ACM International Conference on the Foundations of Software Engineering (FSE 2025)*. 2024. URL: <https://arxiv.org/abs/2407.01489>.
- [4] Jimenez C. E., Yang J., Wettig A., et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *International Conference on Learning Representations (ICLR 2024)*. 2024. URL: <https://arxiv.org/abs/2310.06770>.
- [5] Du Y., Tian M., Ronanki S., et al. Context Length Alone Hurts LLM Performance Despite Perfect Retrieval. *Findings of the Association for Computational Linguistics: EMNLP 2025*. 2025. URL: <https://arxiv.org/abs/2510.05381>.
- [6] Gu W., Chen J., Wang Y., et al. What to Retrieve for Effective Retrieval-Augmented Code Generation? An Empirical Study and Beyond. *arXiv preprint*. 2025. URL: <https://arxiv.org/abs/2503.20589>.
- [7] Gloaguen T., Mündler N., Müller M., Raychev V., Vechev M. Evaluating AGENTS.md: Are Repository-Level Context Files Helpful for Coding Agents? *ICLR 2026 Workshop on Memory for LLM-Based Agentic Systems*. 2026. URL: <https://arxiv.org/abs/2602.11988>.
- [8] Lindenbauer T., Slinko I., Felder L., et al. The Complexity Trap: Simple Observation Masking Is as Efficient as LLM Summarization for Agent Context Management. *NeurIPS 2025 Workshop on Deep Learning for Code (DL4C)*. 2025. URL: <https://arxiv.org/abs/2508.21433>.
- [9] Li H., Tang Y., Wang S., Guo W. PatchPilot: A Cost-Efficient Software Engineering Agent with Early Attempts on Formal Verification. *International Conference on Machine Learning (ICML 2025)*. 2025. URL: <https://arxiv.org/abs/2502.02747>.
- [10] Mei L., Yao J., Ge Y., et al. A Survey of Context Engineering for Large Language Models. *arXiv preprint*. 2025. URL: <https://arxiv.org/abs/2507.13334>.
- [11] Tao Y., Li Y., Qin Y., Liu Y. Retrieval-Augmented Code Generation: A Survey with Focus on Repository-Level Approaches. *arXiv preprint*. 2025. URL: <https://arxiv.org/abs/2510.04905>.
- [12] Li H., Zhu L., Zhang B., et al. ContextBench: A Benchmark for Context Retrieval in Coding Agents. *arXiv preprint*. 2026. URL: <https://arxiv.org/abs/2602.05892>.
- [13] Zhu J., Wu J., Hu M., et al. SWE Context Bench: A Benchmark for Context Learning in Coding. *arXiv preprint*. 2026. URL: <https://arxiv.org/abs/2602.08316>.
- [14] Liu T., Xu C., McAuley J. RepoBench: Benchmarking Repository-Level Code Auto-Completion Systems. *International Conference on Learning Representations (ICLR 2024)*. 2024. URL: <https://arxiv.org/abs/2306.03091>.
- [15] Blinn A., Li X., Kim J. H., Omar C. Statically Contextualizing Large Language Models with Typed Holes. *Proceedings of the ACM on Programming Languages (OOPSLA 2024)*. 2024. Vol. 8. P. 468–498. URL: <https://doi.org/10.1145/3689728>.
- [16] Liu S., Chen Y., Krishna R., et al. Process-Centric Analysis of Agentic Software Systems. *arXiv preprint*. 2025. URL: <https://arxiv.org/abs/2512.02393>.
- [17] Yang S., Li Y., He S., et al. Phase-Aware Mixture of Experts for Agentic Reinforcement Learning. *arXiv preprint*. 2026. URL: <https://arxiv.org/abs/2602.17038>.
- [18] Liu S., Yang J., Jiang B., et al. Context as a Tool: Context Management for Long-Horizon SWE-Agents. *arXiv preprint*. 2025. URL: <https://arxiv.org/abs/2512.22087>.
- [19] Wang Y., Shi Y., Yang M., et al. SWE-Pruner: Self-Adaptive Context Pruning for Coding Agents. *arXiv preprint*. 2026. URL: <https://arxiv.org/abs/2601.16746>.
- [20] Bui N. D. Q. Building Effective AI Coding Agents for the Terminal: Scaffolding, Harness, Context Engineering, and Lessons Learned. *arXiv preprint*. 2026. URL: <https://arxiv.org/abs/2603.05344>.
- [21] Nebius Team. Nebius SWE-agent Trajectories Dataset. Hugging Face Hub. 2025. URL: <https://huggingface.co/datasets/nebius/SWE-agent-trajectories>.
- [22] Nebius Team. Nebius OpenHands Trajectories Dataset. Hugging Face Hub. 2025. URL: <https://huggingface.co/datasets/nebius/SWE-rebench-openhands-trajectories>.
- [23] McCullagh P., Nelder J. A. Generalized Linear Models. 2nd ed. London: Chapman and Hall, 1989.

Дроздюк О. О.**ЕМПІРИЧНИЙ АНАЛІЗ ФАЗОВОЇ ПОВЕДІНКИ ШІ-АГЕНТІВ ПРОГРАМУВАННЯ**

Для автоматичного виправлення помилок у програмному коді дедалі частіше застосовують ШІ-агентів, зокрема OpenHands, SWE-agent та Agentless. Ці інструменти проходять схожий шлях для вирішення задачі, шукають потрібний фрагмент у репозиторії, вносять зміни і перевіряють їх тестами, однак кожен з них робить це по-своєму. Як саме агенти розподіляють зусилля між цими етапами і чи пов'язаний розподіл з результатом, досліджено недостатньо. Щоб це з'ясувати, проаналізовано 4 300 траєкторій цих трьох архітектурно різних агентів на задачах SWE-bench (Software Engineering Benchmark) Verified. Кожну дію агента за сигнатурами викликів інструментів (tool calls) віднесено до однієї з трьох фаз, а саме Локалізації (пошук релевантного коду), Генерації патчу (редагування коду) або Валідації (запуск тестів). Далі порівняно розподіл фаз, переходи між ними і типи файлів для вирішених і невирішених траєкторій. Узагальнена лінійна модель (Generalized Linear Model, GLM) показала, що шанси на успіх зростають, коли агент більше перевіряє свої зміни ($OR = 1,037$, $p < 0,001$) і частіше чергує фази ($OR = 7,843$, $p < 0,001$), а знижуються, коли дій стає надто багато ($OR = 0,984$, $p < 0,001$). Матриці переходів виявили дві різні стратегії. За ітеративної агент тестує майже після кожного редагування (OpenHands, 50,7% переходів до перевірки після правки), за цілеспрямованої робить кілька правок поспіль і лише потім перевіряє результат (SWE-agent, 51,4% повторних редагувань). Успішні траєкторії загалом коротші, містять більше валідації і частіше змінюють фази. Отримані результати дають емпіричну основу для добору контексту залежно від фази задачі.

Ключові слова: ШІ-агенти програмування; великі мовні моделі; контекстна інженерія; SWE-bench; фазовий аналіз; узагальнена лінійна модель; матриці переходів; OpenHands; SWE-agent; Agentless.

Drozdyuk O.**EMPIRICAL ANALYSIS OF PHASE-BASED BEHAVIOR OF AI CODING AGENTS**

To fix bugs in program code automatically, developers increasingly rely on AI coding agents, including OpenHands, SWE-agent, and Agentless. These tools take a similar route to a solution, searching for the relevant fragment in the repository, making changes, and verifying them with tests, yet each agent does this in its own way. How exactly agents divide effort between these steps, and whether this division relates to the outcome, has not been studied enough. To find out, 4,300 trajectories of these three architecturally distinct agents were analysed on SWE-bench (Software Engineering Benchmark) Verified. Using tool-call signatures, each agent action was assigned to one of three phases, namely Localization (finding relevant code), Patching (editing code), or Validation (running tests). Phase distributions, transitions between phases, and file types were then compared for solved and unsolved trajectories. A Generalized Linear Model (GLM) showed that the odds of success rise when an agent validates its changes more ($OR = 1.037$, $p < 0.001$) and switches phases more often ($OR = 7.843$, $p < 0.001$), and fall when the number of actions grows too large ($OR = 0.984$, $p < 0.001$). The transition matrices revealed two distinct strategies. In the iterative one the agent tests after almost every edit (OpenHands, 50.7% of post-patch transitions go to validation), while in the targeted one it makes several edits in a row before checking the result (SWE-agent, 51.4% repeated edits). Successful trajectories are generally shorter, contain more validation, and switch phases more often. These findings provide an empirical basis for tailoring agent context to the phase of the task.

Keywords: AI coding agents; large language models; context engineering; SWE-bench; phase-based analysis; generalized linear model; transition matrices; OpenHands; SWE-agent; Agentless.

Дата першого надходження: 15.07.2026 р.

Дата прийняття до друку: 21.08.2026 р.

Дата публікації: 10.09.2026 р.