Інформаційні технології, комп'ютерні науки

УДК 004.412:004.8

Киселевич Володимир Володимирович
ORCID: 0009-0006-6449-710X
Асистент кафедри комп'ютерних наук та інформаційних технологій
Житомирський державний університет імені Івана Франка

МОДЕРНІЗАЦІЯ УСПАДКОВАНИХ ПРОГРАМНИХ СИСТЕМ ЗАСОБАМИ LLM-АГЕНТІВ: РОЛЬОВА МОДЕЛЬ І УМОВА РОЗДІЛЕННЯ РОЛЕЙ

Розглянуто модернізацію успадкованих програмних систем із застосуванням агентів на основі великих мовних моделей. Обґрунтовано, що основна складність полягає не у трансляції коду, а у відновленні втраченого предметного знання та неявних інваріантів, підстави яких не зафіксовані у вихідному коді. Запропоновано рольову модель, у якій агент послідовно виконує відновлення наміру, трансформацію та доведення еквівалентності поведінки, причому кожна роль спирається на власне джерело істини й на власний шар надійності. Сформульовано умову розділення ролей та визначено вузькі місця для чотирьох типових сценаріїв міграції. Ключові слова: великі мовні моделі, агентні системи, модернізація успадкованих систем, трансляція коду, неявні інваріанти, доведення еквівалентності поведінки, джерело істини.

Постановка проблеми. Успадковані програмні системи, що обслуговують розрахункові й облікові процеси у банківській, страховій та державній сферах, розвивалися десятиліттями, тоді як документація і підстави проєктних рішень втрачалися разом зі зміною персоналу. Задачу модернізації таких систем і в інженерній практиці, і в академічних експериментах переважно ставлять як задачу трансляції коду, тобто як відображення конструкцій мови-джерела у конструкції цільової з умовою збереження зовнішньої поведінки. Емпіричні дані свідчать, що навіть у такому звуженому формулюванні якість автоматичного перекладу залишається невисокою: дослідження 1700 зразків коду з трьох бенчмарків і двох реальних проєктів для пар мов C, C++, Go, Java і Python зафіксувало частку коректних трансляцій від 2,1 до 47,3 відсотка залежно від моделі та виокремило 15 категорій дефектів, внесених під час перекладу [1]. Ще істотнішим є те, що показники, здобуті на задачах рівня окремого методу, не переносяться на рівень класу з контекстними залежностями: за оцінками, які на попередніх наборах перевищували 80 відсотків, перехід до трансляції класів потребував 360 людино-годин лише для побудови еталонного набору, а 1243 випадки помилок найкращої моделі довелося розмічати вручну [2]. Обмеженням, отже, є не тільки точність відображення синтаксису. Транслятор зберігає те, що записано в коді, проте успадкована система містить рішення, підстави яких не записані ніде: чому саме такий порядок операцій, які значення полів неможливі за бізнес-правилами, які збіги поведінки є навмисними, а які випадковими. Саме ці неявні знання визначають, чи є перекладена система коректною, і саме вони не мають носія у вихідному коді.

Аналіз останніх досліджень. Праці останніх років доцільно розподілити за тим, який шар модернізації вони підсилюють. Частина використаних джерел має статус препринту, а саме [3; 4; 5; 6; 7; 8], тому наведені з них числові оцінки слід трактувати як попередні, до завершення рецензування. Перший напрям стосується відновлення розуміння коду. Для мов MUMPS та ALC встановлено, що згенеровані моделлю коментарі здебільшого не містять вигаданих тверджень, є повними та читабельними порівняно з еталонними, проте жодна автоматизована метрика, ані оцінки складності коду, ані метрики на основі еталона не корелюють з якістю коментарів достатньо сильно, щоб прогнозувати ефективність моделі [3]. Доменна адаптація дає приріст саме на задачах розуміння: спеціалізована модель для середовищ великих ЕОМ перевищує базову на 30 відсотків у тестах з множинним вибором, удвічі за показником BLEU у задачах питань і відповідей та у шість разів у підсумовуванні коду COBOL [4]; інша адаптована модель досягає до 73,95 відсотка успішної компіляції та 49,33 за метрикою pass@1 на наборі COBOLEval проти 41,8 відсотка та 16,4 у моделі загального призначення, а на перекладі з Java у COBOL отримує 34,93 за pass@1, тоді як моделі загального призначення дають результати, близькі до нуля [5]. Другий напрям поєднує модель із символьним аналізом програм. Композиційний нейро-символічний підхід застосували до десяти реальних проєктів, розбитих на 17874 фрагменти, і здобули 96,40 відсотка синтаксично коректних перекладених фрагментів, тоді як коректність виконання та функціональну правильність підтверджено лише для 27,03 та 25,14 відсотка фрагментів методів застосунку відповідно, причому інтегрований цикл перекладу й валідації тривав у середньому 34 години на проєкт (від 3 до 121), а виправлення помилок розробниками забирало ще 20,1 години [9]. Третій напрям будує зовнішній оракул коректності. Транслятор у Rust із перевіркою

еквівалентності отримує еталонну програму через компіляцію у WebAssembly, генерує ідіоматичного кандидата моделлю та перевіряє еквівалентність методом перевірки моделей [10]. Агентний метод детерміністичної валідації міграції на трьох дослідженнях випадку COBOL у Java обсягом 430-4114 рядків коду підвищував покриття понад плато випадкового пошуку вхідних даних і досяг 91,90 відсотка покриття гілок на програмі промислового рівня, а згенерована версія збігалася з еталоном за детерміністичними перевірками паритету на всіх прийнятих тестових випадках [6]. Окремим засобом є відновлення формальних умов з природномовного опису: постумови, згенеровані моделлю, виявилися здебільшого коректними та здатними відрізняти дефектний код, зокрема вони дали змогу виявити 64 реальні історичні помилки з набору Defects4J [11]. Четвертий напрям формує еталонні набори для оцінювання. Для міграції репозиторіїв з Java 8 на Java 17 і 21 укладено набір з 5102 репозиторіїв, з яких 300 відібрано для оцінювання, а агентний фреймворк досяг 71,67 та 53,33 відсотка успішності за pass@1 для мінімальної та максимальної міграції [12]. Для переходу з C у безпечний Rust укладено набір зі 100 репозиторіїв з написаними вручну інтерфейсами й тестами, на якому найкраща модель розв’язала лише 15 завдань за один прохід [7], а дослідження реального коду з диференційним фазз-тестуванням показало межу близько 47 відсотків перекладених елементів для найуспішнішої моделі [8]. Спільною рисою цих праць є те, що кожна з них вимірює один шар процесу, а питання про спосіб поєднання шарів і про місце, у якому процес втрачає знання, залишається поза розглядом.

Мета. Мета роботи полягає в тому, щоб обґрунтувати перенесення акценту з трансляції коду на відновлення втраченого предметного знання та неявних інваріантів, побудувати рольову модель агента для процесу модернізації, у якій кожна роль спирається на власне джерело істини й на власний шар надійності, та сформулювати умову розділення ролей як вимогу до архітектури агентної системи модернізації.

Виклад основного матеріалу. Успадкована система є не лише текстом програми, а й нагромадженням рішень, підстави яких залишилися поза кодом. До неявного знання належать: порядок операцій, зумовлений вимогою зовнішнього регулятора, а не логікою обчислення; діапазони значень полів, які не можуть виникнути за бізнес-правилами, а тому ніколи не перевіряються; узгодженість між модулями, що спирається на спільне припущення про формат ідентифікатора; збіги поведінки, серед яких частина є навмисною і використовується суміжними системами, а частина є випадковим наслідком реалізації. Транслятор, що зберігає семантику мови-джерела, відтворює всі ці властивості без розрізнення їхнього статусу, тому перекладена система успадковує і потрібні, і випадкові особливості, а критерію для їх розділення в самому коді немає. Саме тому висока частка синтаксично коректних фрагментів співіснує з набагато нижчою часткою фрагментів, для яких підтверджено функціональну правильність [9], а погіршення якості під час переходу від методу до класу з контекстними залежностями [2] є не аномалією, а очікуваним наслідком зростання обсягу незаписаних припущень. Неявний інваріант відрізняється від документованої вимоги тим, що його порушення стає помітним лише через поведінку суміжних систем, а не через контроль у самому коді.

Запропоновано розглядати участь агента в модернізації як послідовне виконання трьох різних ролей, кожна з яких має власне джерело істини та власний шар надійності (табл. 1). Роль P1 полягає у відновленні наміру. Джерелом істини тут є артефакти поза кодом: збережена документація, журнали змін і повідомлення систем контролю версій, дані промислової експлуатації, свідчення фахівців предметної області. Шаром надійності є перехресна перевірка кожного відновленого твердження з незалежним артефактом, оскільки модель схильна доповнювати опис правдоподібним змістом там, де свідчень немає. Обмеженість автоматизованого оцінювання цієї ролі підтверджує відсутність метрики, яка корелювала б із якістю згенерованих пояснень [3], тому визнання артефактів P1 придатними залишається людською процедурою. Практичним виявом ролі P1 є перелік тверджень про предметну область із позначенням джерела кожного з них, причому твердження без незалежного підтвердження фіксується як гіпотеза і не переходить далі зі статусом факту.

Роль P2 полягає у трансформації. Джерелом істини є вихідний код, а шаром надійності є машинні перевірки, які не потребують знання наміру: компільованість, перевірка типів, статичний аналіз, дотримання правил цільової мови. Ця роль найкраще піддається автоматизації, і саме на ній зосереджено більшість опублікованих результатів [1; 5; 9]. Роль P3 полягає у доведенні еквівалентності поведінки. Джерелом істини є спостережувана поведінка наявної системи, а не її опис, тому шаром надійності є порівняльне тестування на однакових входах, зокрема на записаному промисловому навантаженні, диференційне порівняння виходів [8] та перевірка інваріантів, сформульованих як виконувані умови [11]. Найсильніші відомі результати для цієї ролі здобуто тоді, коли оракул є зовнішнім щодо моделі, а саме через перевірку еквівалентності відносно програми, породженої компілятором [10], або через детерміністичні перевірки паритету з керованим зростанням покриття гілок [6]. Жодна з перевірок цього шару не здатна виявити втрату наміру, оскільки всі вони посилаються лише на текст програми.

Таблиця 1

Рольова модель агента у процесі модернізації успадкованої системи

Роль	Джерело істини	Шар надійності	Типова відмова
P1: відновлення наміру	Документація, журнали змін, дані експлуатації, свідчення фахівців	Перехресна перевірка з незалежними артефактами	Правдоподібне доповнення відсутніх підстав рішення
P2: трансформація	Вихідний код наявної системи	Компільованість, перевірка типів, статичний аналіз	Втрата випадкової, але використовуваної поведінки
P3: доведення еквівалентності	Спостережувана поведінка наявної системи	Порівняльне тестування на промисловому навантаженні, перевірка інваріантів	Прийняття за критерієм, породженим самим агентом

Джерело: розробка автора.

Головна теза побудованої моделі полягає в тому, що змішування ролей є типовою причиною відмов. Коли агент в одному кроці відновлює намір і одночасно перекладає код, він добудовує припущення про предметну область, а потім не має з чим порівняти результат, бо джерело істини для перевірки він щойно створив сам. Наслідком є внутрішньо узгоджена, але не обґрунтована система: тести, згенеровані з відновленого опису, проходять саме тому, що і код, і тести походять з однієї гіпотези, а розбіжність з наявною системою залишається невиявленою. Сформульовано вимогу розділення: артефакт, що виник у ролі P1, не може використовуватися як критерій прийняття в ролі P3. Допустимим є використання артефактів P1 як джерела гіпотез для добору вхідних даних, для формулювання кандидатів на інваріанти та для впорядкування ділянок коду за ризиком, проте рішення про прийняття перекладеної ділянки має спиратися виключно на поведінку наявної системи або на оракул, зовнішній щодо агента. Практичним наслідком вимоги є розділення каналів: набір приймальних перевірок формується із записаного промислового навантаження та з інваріантів, підтверджених виконанням на наявній системі, а не з описів, здобутих у P1. Розділення ролей задає також порядок робіт, за яким склад приймальних перевірок визначається до початку трансформації, а не добудовується після неї.

Прикладні сценарії міграції відрізняються тим, яка з трьох ролей є вузьким місцем. Для переходу з COBOL на сучасну платформу вузьким місцем є P1, бо предметні правила розчинені в процедурному коді та у форматах записів, а носіїв знання часто вже немає; це узгоджується з тим, що приріст від доменної адаптації спостерігається саме на задачах розуміння та підсумовування [4; 5], тоді як придатність результату все одно встановлюють зовнішнім детерміністичним оракулом [6]. Для перекладу з C у Rust з вимогою безпечного доступу до пам’яті вузьким місцем є P2, оскільки цільова мова відкидає частину конструкцій джерела і потрібна зміна структури володіння даними, а не відображення операторів; на це вказує розрив між обсягом набору репозиторіїв і кількістю завдань, розв’язаних за один прохід [7], а також межа близько 47 відсотків у дослідженні реального коду [8]. Для підвищення версії платформи в межах однієї мови вузьким місцем є P3, бо намір здебільшого зрозумілий і частково зафіксований у наявних тестах, натомість змінюється поведінка бібліотек і стандартних засобів; різниця між успішністю мінімальної та максимальної міграції репозиторіїв з Java 8 [12] показує, що результат визначає ускладнення саме перевірки, а не розуміння. Для заміни фреймворка навантаження розподіляється між P1 і P3, оскільки намір прикладної логіки переважно зрозумілий, проте частина поведінки була делегована фреймворку і ніде не описана, тому її доводиться відновлювати як неявний інваріант і окремо перевіряти на однакових входах. Спільним правилом є те, що інженерні витрати доцільно зосереджувати на ролі, визнаній вузьким місцем, тоді як для решти ролей достатньо типових машинних перевірок.

Обмеження. Робота не містить власного експерименту і спирається на числові результати, здобуті іншими дослідницькими групами на їхніх наборах даних та в їхніх умовах запуску, тому пряме порівняння наведених показників між собою є некоректним. Частина оцінок походить з препринтів [3; 4; 5; 6; 7; 8] і може змінитися після рецензування. Рольова модель є аналітичною конструкцією: поділ на три ролі та відповідність між роллю, джерелом істини й шаром надійності обґрунтовано якісним розбором опублікованих архітектур, а не вимірюванням частоти відмов у контрольованому дослідженні. Вимогу розділення не перевірено статистично, бо для цього потрібне порівняння двох груп проєктів, які відрізняються лише дотриманням цієї вимоги, а таких даних у відкритих джерелах немає. Вибірка опрацьованих джерел обмежена дванадцятьма працями, дібраними за темою модернізації та трансляції коду, тому вона не є систематичним оглядом і може не охоплювати підходів, описаних у промислових звітах без публікації. Віднесення вузького місця до певної ролі для кожного сценарію міграції є гіпотезою, придатною для перевірки, а не встановленим фактом. Оцінки трудомісткості, наведені за [9], залежать від складу конкретних проєктів і не переносяться на інші кодові бази.

Висновки. Показано, що подання модернізації успадкованих систем як задачі трансляції коду залишає поза увагою основне джерело складності, а саме відновлення втраченого предметного знання та неявних інваріантів, підстави яких не зафіксовані у вихідному коді. Запропоновано рольову модель, у якій агент послідовно виконує відновлення наміру, трансформацію та доведення еквівалентності поведінки, причому кожна роль спирається на власне джерело істини й на власний шар надійності (табл. 1). Сформульовано умову розділення, за якою артефакт, породжений у ролі відновлення наміру, не може бути критерієм прийняття в ролі доведення еквівалентності. Виокремлено вузькі місця для чотирьох типових сценаріїв міграції, що дає орієнтир для розподілу інженерних витрат між ролями. Подальші дослідження доцільно спрямувати на емпіричну перевірку умови розділення на промислових проєктах міграції та на побудову показників, що оцінюють не частку скомпільованого коду, а повноту відновлених інваріантів.

1. Pan R., Ibrahimzada A. R., Krishna R. [та ін.] Lost in Translation: A Study of Bugs Introduced by Large Language Models while Translating Code. Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. 2024. P. 1-13. DOI: 10.1145/3597503.3639226.
2. Xue P., Wu L., Yang Z. [та ін.] ClassEval-T: Evaluating Large Language Models in Class-Level Code Translation. Proceedings of the ACM on Software Engineering. Vol. 2, No. ISSTA. 2025. P. 1421-1444. DOI: 10.1145/3728940.
3. Diggs C., Doyle M. W., Madan A. [та ін.] Leveraging LLMs for Legacy Code Modernization: Challenges and Opportunities for LLM-Generated Documentation : препринт. arXiv:2411.14971. 2024. URL: https://arxiv.org/abs/2411.14971.
4. Dau A. T. V., Dao H. T., Nguyen A. T. [та ін.] XMainframe: A Large Language Model for Mainframe Modernization: препринт. arXiv:2408.04660. 2024. URL: https://arxiv.org/abs/2408.04660.
5. Dau A. T. V., Tan S. H., Yang J. [та ін.] COBOL-Coder: Domain-Adapted Large Language Models for COBOL Code Generation and Translation: препринт. arXiv:2604.03986. 2026. URL: https://arxiv.org/abs/2604.03986.
6. Ferenczi A., Docherty J., Bessonov M. [та ін.] Agentic Method for Deterministic Validation of Legacy Code Migration: препринт. arXiv:2607.28271. 2026. URL: https://arxiv.org/abs/2607.28271.
7. Khatry A., Zhang R., Pan J. [та ін.] CRUST-Bench: A Comprehensive Benchmark for C-to-safe-Rust Transpilation: препринт. arXiv:2504.15254. 2025. URL: https://arxiv.org/abs/2504.15254.
8. Eniser H. F., Zhang H., David C. [та ін.] Towards Translating Real-World Code with LLMs: A Study of Translating to Rust: препринт. arXiv:2405.11514. 2024. URL: https://arxiv.org/abs/2405.11514.
9. Ibrahimzada A. R., Ke K., Pawagi M. [та ін.] AlphaTrans: A Neuro-Symbolic Compositional Approach for Repository-Level Code Translation and Validation. Proceedings of the ACM on Software Engineering. Vol. 2, No. FSE. 2025. P. 2454-2476. DOI: 10.1145/3729379.
10. Yang A. Z., Takashima Y., Paulsen B. [та ін.] VERT: Polyglot Verified Equivalent Rust Transpilation with Large Language Models. 2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2025. P. 1453-1463. DOI: 10.1109/ase63991.2025.00123.
11. Endres M., Fakhoury S., Chakraborty S. [та ін.] Can Large Language Models Transform Natural Language Intent into Formal Method Postconditions? Proceedings of the ACM on Software Engineering. Vol. 1, No. FSE. 2024. P. 1889-1912. DOI: 10.1145/3660791.
12. Liu L., Liu X. M., Zhou Q. [та ін.] MigrationBench: Repository-Level Code Migration Benchmark from Java 8. Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2. 2026. P. 9427-9438. DOI: 10.1145/3770855.3817526.