УДК 004.4:004.8

ВЕЛИКА МОВНА МОДЕЛЬ У ВІДНОВЛЕННІ МІЖСЕРВІСНИХ ЗВ'ЯЗКІВ: ВИБІР СЕРЕД КАНДИДАТІВ І САМОСТІЙНИЙ ПОШУК

Поштаренко Д.К.[1]; Півень О.Б., к.ф.-м.н., доц.[2]
[1,2] Черкаський державний технологічний університет
[1,2] Україна, Черкаси
[1] dmytro.poshtarenko@gmail.com; [2] ol.piven@chdtu.edu.ua

Анотація. *Метод за кодом шукає, куди звертаються частини системи: здебільшого за заготовленими правилами, а решту — мовною моделлю. Модель обирає адресата з підказок правил або називає його сама. На месенджері Rocket.Chat хмарна Opus 5.5 однаково точна в обох, а локальна Qwen3 30B без підказок часто вигадує адресатів. Локальну модель, що не передає код назовні, варто питати лише з підказками.*
Ключові слова: *велика мовна модель; відновлення архітектури; міжсервісні зв'язки; мультирепозиторна система; статичний аналіз*

Вступна частина. Коли програмну систему розбито на окремі сервіси, а їхній код лежить у кількох сховищах коду (репозиторіях), зв'язок між двома частинами видно лише з коду обох сторін і описів інфраструктури, тобто файлів, що описують розгортання системи. Документацію архітектури, створену вручну, часто не встигають оновлювати разом із кодом [1], тож карту таких міжсервісних зв'язків доводиться відновлювати з самого коду.

Великі мовні моделі, тобто нейронні мережі, які за текстовим запитом пишуть текст і код, уже генерують за вихідним кодом архітектурні подання: схеми частин системи і зв'язків між ними [1]. Проте ці схеми часто лишаються на рівні елементів коду, а не частин системи [1]. Засіб ArchAgent поєднує статичний аналіз, тобто аналіз коду без запуску програми, із синтезом таких схем мовною моделлю [2]. Хмарна модель працює на серверах постачальника, тож код доводиться передавати назовні, а командам, які цього не можуть, лишається локальна модель на комп'ютері розробника. У переглянутих джерелах не знайдено порівняння різних способів застосування мовної моделі на рівні окремих міжсервісних зв'язків.

Мета роботи — визначити, як застосовувати мовну модель у методі відновлення міжсервісних зв'язків. Для цього порівняно два способи визначити ціль виклику, тобто його адресата в системі: модель обирає її з кількох можливих, запропонованих правилами, заздалегідь написаними для бібліотек системи, або шукає самостійно. Також з'ясовано, що змінюється, коли хмарну модель замінює локальна, і чи пропускає модель у довгому файлі більше маршрутів, тобто точок входу, через які сервіс приймає запити.

Основна частина. Метод реалізовано в розробленому автором відкритому засобі archmap [3]. Засіб знаходить у коді місця виклику, тобто рядки, з яких одна частина системи звертається до іншої, і визначає ціль кожного виклику: частину системи й маршрут у ній. Основну роботу виконують правила: вони розпізнають виклики та оголошення маршрутів у бібліотеках і каркасах, якими користується система, зокрема в її власних. Описи інфраструктури дають адреси частин і показують, куди шлюз, сервер-посередник для вхідних запитів, спрямовує кожен із них. Мовна модель має два завдання: знаходить оголошення маршрутів там, де правил немає, і для викликів, яких правила не розв'язали, обирає ціль серед щонайбільше 5 кандидатів, запропонованих правилами.

Метод перевірено на Rocket.Chat, відкритому месенджері для команд. Його основний репозиторій містить сервер, вебклієнт і мікросервіси, тобто невеликі окремі програми, кожна

з яких виконує одну функцію, а десктопний і мобільний клієнти лежать в окремих репозиторіях. Клієнти звертаються до сервера звичайними HTTP-запитами (REST) і через постійне з'єднання, яким сервер сам надсилає оновлення (протокол DDP). З'єднання DDP шлюз передає окремому сервісу, а сервер і мікросервіси обмінюються повідомленнями через брокер повідомлень, програму, що доставляє їх від одного сервісу до іншого.

Щоб оцінити метод, автор вручну визначив правильні відповіді, еталон, для випадкової вибірки рядків коду, схожих на виклики: чи є рядок місцем виклику і яка в нього ціль. Файли основного репозиторію та десктопного клієнта розділили навмання: приблизно на третині з них метод налаштовували, потім зафіксували його й перевірили на решті файлів, тестовій частині. Повнота показує, яку частку справжніх зв'язків знайшов метод, а точність — яку частку знайдених зв'язків він визначив правильно. Обидві міри зважено за типом виклику, бо рідкісні типи у вибірці представлено ширше, ніж у коді. Порівнювали дві моделі: хмарну Claude Opus 5.5, до якої зверталися через асистента програміста Claude Code, і локальну Qwen3 30B, запущену програмою Ollama. Файлів редакції Enterprise, що має окрему ліцензію, хмарній моделі не передавали, тож її питали про менше місць.

Спершу перевірили, чи потрібен моделі перелік кандидатів. Для цього обидва способи, вибір з переліку й самостійний пошук, застосували до всіх місць виклику, які метод знайшов серед відібраних рядків тестової частини, а не лише до нерозв'язаних, як у повному методі, щоб для порівняння вистачило викликів. Відповідь, яку не вдалося зіставити з жодною відомою методу ціллю, вважали вигаданою: вона не дає зв'язку, тож знижує повноту, а не точність. Перевіряли припущення, що модель допомагає знайти більше зв'язків без втрати точності лише тоді, коли обирає з переліку. Воно не підтвердилося для жодної моделі, але з різних причин (табл. 1).

Таблиця 1

Повнота і точність варіантів методу на тестовій частині Rocket.Chat, %

Варіант	Повнота, Opus	Точність, Opus	Повнота, Qwen	Точність, Qwen
Без моделі	95,8	98,8	95,8	98,8
Повний метод	97,4	98,7	96,0	98,7
Вибір на всіх місцях	97,4	96,1	91,1	87,8
Пошук на всіх місцях	97,4	96,4	42,8	72,9

Хмарній моделі перелік для точності не потрібен: в обох способах вона була майже однаково точною. Шукаючи самостійно, вона вигадала ціль лише раз на 181 місце, тож не втратила й повноти. Локальна модель без переліку вигадала ціль у 101 з 218 відповідей, тобто майже в половині. Але й з переліком, відповідаючи про всі місця, вона замінювала хибною ціллю правильні зв'язки, знайдені правилами. Хмарна модель жодного правильного зв'язку правил не зіпсувала, проте обидві моделі ще й зв'язували виклики, ціль яких працює в тій самій частині системи, що й код виклику, хоча міжсервісного зв'язку там немає. Такі хибні й зіпсовані зв'язки підтверджують рішення, закладене в метод: питати модель лише про місця, яких не розв'язали правила.

З переліком відповіді хмарної моделі ще й стабільніші. За всіма прогонами частину запитів ставили тричі й вважали запит стабільним, якщо всі три рішення збіглися. Обираючи з переліку, хмарна модель була стабільною в усіх таких запитах, а шукаючи самостійно — у 36 з 44. Температуру, тобто випадковість у виборі слів, для неї задати не можна, а локальну модель запускали з нульовою температурою, і вона змінила рішення лише в одному запиті.

Далі моделі порівняли в повному методі, де модель питають лише про нерозв’язані місця. Тут результати двох моделей майже збіглися (табл. 1), а локальна модель додала до методу без моделі лише один правильний зв’язок. Різнилися моделі лише на викликах, які шлюз передає окремому сервісу: це 13 підписок клієнтів на потоки подій і 2 виклики методів. Хмарна модель правильно зв’язала 13 з 15 таких викликів, а локальна лише один. Ця різниця значуща, тобто навряд чи випадкова: точний тест Мак-Немара [4], що порівнює дві моделі на тих самих викликах, дає $p = 0{,}0018$. Це ймовірність отримати таку чи більшу різницю випадково, якби моделі насправді не різнилися. Причина в оголошеннях потоків: їх зроблено через функцію, якої не знають правила, і хмарна модель знайшла всі такі оголошення, а локальна жодного, хоча інші оголошення поза правилами знаходить. Отже, на Rocket.Chat локальна модель не замінила хмарну саме в пошуку нестандартно оголошених потоків, а на викликах поза шлюзом дала той самий результат.

Перевага хмарної моделі в пошуку оголошень могла частково йти з її пам’яті: Rocket.Chat є відкритим проєктом, і його код міг потрапити до даних, на яких модель навчалася. Щоб це перевірити, взяли 12 модулів REST сервера Rocket.Chat із відкритою ліцензією, у яких оголошено 59 маршрутів, і дали моделям лише шляхи цих файлів, без коду. Хмарна модель назвала 50 з них, локальна лише кілька, а коли шляхи перейменували, жодна не назвала жодного. Збіг даних, на яких модель навчалася, з даними перевірки є відомою загрозою достовірності висновків [5], тому для наступної перевірки код перейменували.

Нарешті перевірили, чи пропускає модель більше маршрутів у довгому файлі. Ті самі модулі з перейменованими шляхами й маршрутами доповнювали кодом до різної довжини, і модель мала перелічити всі маршрути файлу. Хмарна модель знайшла всі 59 маршрутів на кожному з трьох рівнів довжини, аж до 221 тис. символів. У довших файлах локальна модель загалом пропускала більше маршрутів: у найкоротших вона знайшла 98,3 % з них, у найдовших лише 86,4 % (рис. 1). Статистична перевірка спаду, яка враховує, що маршрути одного файлу пов’язані між собою [6], дала $p = 0{,}0038$, тож спад навряд чи випадковий. Отже, припущення про шкоду довжини підтвердилося лише для локальної моделі, і в методі довгі файли ділять на частини тільки для неї. Це узгоджується з тим, що мовні моделі ненадійно використовують відомості з довгого тексту [7].

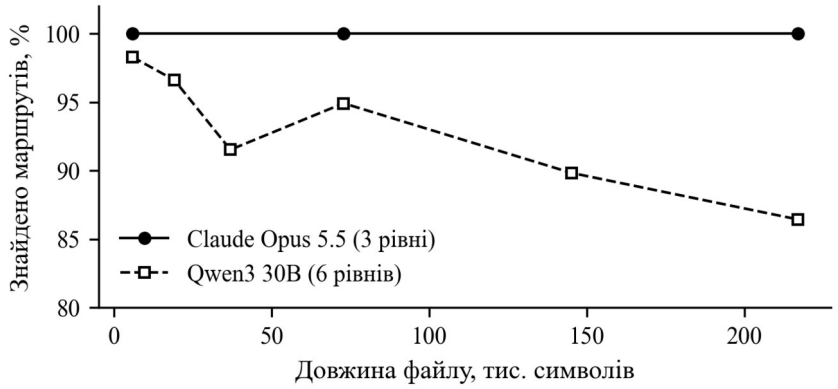


Рис. 1. Частка знайдених маршрутів залежно від довжини файлу

Висновки. Мовну модель у методі відновлення міжсервісних зв’язків варто питати лише про виклики, яких не розв’язали правила, і давати їй перелік кандидатів. Хмарній моделі Claude Opus 5.5 перелік не потрібен для точності, але з ним її відповіді стабільніші. Локальна Qwen3 30B без переліку часто вигадує цілі, а відповідаючи про всі виклики, і з переліком псує правильні зв’язки правил. У повному методі локальна модель дала той самий результат, що й хмарна, крім викликів через шлюз: оголошень потоків, зроблених нестандартно, вона

не знайшла. Довгі файли варто ділити на частини лише для локальної моделі. Висновки про способи застосування й заміну моделі отримано на одній системі з еталоном одного розмітника. Перевагу хмарної моделі в пошуку оголошень могла посилити її пам'ять про код Rocket.Chat, тоді як довгі файли перевіряли на перейменованому коді. Далі метод варто перевірити на інших системах і моделях інших розробників.

Використання інструментів ШІ. Складовими методу були Claude Opus 5.5 через Claude Code 2.1.303 і Qwen3 30B (qwen3:30b-a3b-instruct-2507, квантування Q4_K_M, тобто зі стиснутими вагами) через Ollama. Моделі вилучали оголошення маршрутів, обирали ціль виклику серед кандидатів або шукали її самостійно, а в перевірці запам'ятовування називали маршрути за самим шляхом файлу (основна частина). Код засобу archmap написав Claude Code (Claude Opus 5.5) за вимогами автора, а автор перевірив і виправив його. Текст і рисунок тез підготовлено з тим самим асистентом на основі даних і вимог автора й перевірено автором (усі частини тез).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Miryala, S., Dhar, R., & Vaidhyanathan, K. (2026). LLM-based automated architecture view generation: Where are we now? arXiv:2603.21178. Retrieved from https://arxiv.org/abs/2603.21178.

[2] Pan, R., Mao, B., Ma, T., & Ling, Z. (2026). ArchAgent: Scalable legacy software architecture recovery with LLMs. arXiv:2601.13007. Retrieved from https://arxiv.org/abs/2601.13007.

[3] Poshtarenko, D. (2026). archmap [Computer software]. Retrieved from https://github.com/dmytro-poshtarenko/archmap.

[4] McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2), 153-157. doi:10.1007/BF02295996

[5] Sallou, J., Durieux, T., & Panichella, A. (2024). *Breaking the silence: The threats of using LLMs in software engineering*, Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results (pp. 102-106). New York: ACM. doi:10.1145/3639476.3639764

[6] Liang, K.-Y., & Zeger, S. L. (1986). Longitudinal data analysis using generalized linear models. *Biometrika*, 73(1), 13-22. doi:10.1093/biomet/73.1.13

[7] Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2024). Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12, 157-173. doi:10.1162/tacl_a_00638

Poshtarenko D.K., Piven O.B.

A large language model in recovering inter-service links: selection among candidates and free search

Abstract. From code, a method finds where the parts of a system send their calls: mostly by rules written in advance, and the rest with a language model. The model either picks the receiver from the rules' suggestions or names it itself. On the Rocket.Chat messenger, the cloud Claude Opus 5.5 is equally precise both ways, while the local Qwen3 30B often invents receivers without suggestions. A local model, which keeps the code in-house, should be asked only with suggestions.

Keywords: *large language model; architecture recovery; inter-service links; multi-repository system; static analysis*