№ 5(59) 2026
НАУКА і ТЕХНІКА СЬОГОДНІ
ISSN 2786-6025 Online
УДК 004.423
https://doi.org/10.52058/2786-6025-2026-5(59)-4618-4629

Киселевич Володимир Володимирович асистент кафедри комп'ютерних наук та інформаційних технологій, Житомирський державний університет імені Івана Франка, м. Житомир, https://orcid.org/0009-0006-6449-710X

ГІБРИДНИЙ МЕТОД СТАТИЧНОГО АНАЛІЗУ АСИМПТОТИЧНОЇ СКЛАДНОСТІ ПОТОКІВ ВИКОНАННЯ ПРОГРАМНОГО КОДУ

Анотація. У статті розглядається проблема статичного аналізу асимптотичної часової складності потоків виконання програмного коду TypeScript. Існуючі формальні аналізатори ресурсних меж орієнтовані на байткод JVM та функціональні мови сімейства OCaml, тому екосистему TypeScript із її структурним типуванням, асинхронною семантикою та інтенсивними інтеграціями із зовнішніми системами вони не охоплюють. Системи оцінювання складності виключно на основі великих мовних моделей працюють із довільним кодом, але не дають формальних гарантій і не виражають кількісної міри впевненості, придатної до агрегації. Поза охопленням існуючих підходів залишається задача інтерпроцедурного аналізу асимптотики для сервісів на TypeScript, у якій структурна основа поєднана з резервним оцінюванням семантичних викликів великими мовними моделями, а довіра виражена як ортогональна характеристика оцінки. Запропоновано метод, що поєднує детермінований шаблонний аналіз структурних конструкцій з консенсусним оцінюванням семантичних викликів за допомогою великих мовних моделей за принципом самоузгодженості. Об'єктом аналізу є дерево потоку виконання з підтримкою вбудовування викликів через межі модулів, що дозволяє виявляти приховані квадратичні складності у композиції сервісів. Особливістю методу є кількісно виражена довіра, яка поширюється від листя дерева потоку до кореня за визначеною алгеброю агрегації. Експериментальна валідація виконана на трьох тестових наборах: еталонному корпусі з 30 канонічних алгоритмів, контрольній перевірці чутливості до шаблонів через виключення детектора рекурсивних шаблонів та контрольному корпусі з 30 фрагментів TypeScript із зовнішніми викликами. У детермінованому режимі точність на еталонному наборі становить 80 %, після контрольної перевірки знижується до 76,7 %, а на контрольному корпусі прикладів виробничого коду дорівнює 20 %. У гібридному режимі з консенсусом великих мовних моделей точність на контрольному корпусі досягає 93 %, що становить приріст у 73 відсоткові пункти і обґрунтовує гібридну архітектуру. Запропонований метод

ISSN 2786-6025 Online
заповнює нішу інтерпроцедурного аналізу асимптотичної складності для екосистеми TypeScript, не охоплену формальними аналізаторами ресурсних меж сімейства JVM та OCaml і системами оцінювання складності виключно на основі великих мовних моделей.

Ключові слова: асимптотична складність; статичний аналіз коду; абстрактне синтаксичне дерево; великі мовні моделі; гібридний аналіз; потоки виконання; TypeScript; кількісно виражена довіра.

Kyselevych Volodymyr Volodymyrovych Assistant, Department of Computer Sciences and Information Technologies, Zhytomyr Ivan Franko State University, Zhytomyr, https://orcid.org/0009-0006-6449-710X

HYBRID METHOD FOR STATIC ANALYSIS OF ASYMPTOTIC COMPLEXITY OF EXECUTION FLOWS IN PROGRAM CODE

Abstract. The paper addresses the problem of static analysis of asymptotic time complexity of execution flows in TypeScript program code. Existing formal resource bound analyzers target JVM bytecode and functional languages of the OCaml family, and therefore do not cover the TypeScript ecosystem with its structural typing, asynchronous semantics, and intensive integrations with external systems. Complexity prediction systems based exclusively on large language models work with arbitrary code but provide no formal guarantees and no quantitative measure of confidence amenable to aggregation. None of the existing approaches addresses the task of interprocedural analysis of asymptotics for TypeScript services in which a structural backbone is combined with fallback evaluation of semantic calls by large language models and confidence is expressed as an axis orthogonal to the complexity class. The proposed method combines deterministic pattern-based analysis of structural constructs with consensus-based evaluation of semantic calls by large language models according to the self-consistency principle. The object of analysis is the execution flow tree with support for cross-module call inlining, which makes it possible to detect hidden quadratic complexities in service compositions. A distinguishing feature of the method is quantitatively expressed confidence propagated from tree leaves to the root through a defined aggregation algebra. Empirical validation is performed on three test sets: a reference set of 30 canonical algorithms, a control check of sensitivity to patterns through ablation of the recursive shrink-pattern detector, and a corpus of 30 TypeScript fragments with external calls. In the deterministic mode, accuracy on the reference set reaches 80 percent, drops to 76.7 percent after the ablation check, and falls to 20 percent on the corpus of production-pattern fragments. In the hybrid mode with large language model consensus, accuracy on this corpus reaches 93 percent, a 73 percentage point

ISSN 2786-6025 Online
improvement that justifies the hybrid architecture. The proposed method fills the niche of interprocedural complexity analysis for the TypeScript ecosystem that is not covered by formal resource bound analyzers of the JVM and OCaml families or by complexity prediction systems based exclusively on large language models.

Keywords: asymptotic complexity; static code analysis; abstract syntax tree; large language models; hybrid analysis; execution flows; TypeScript; quantified confidence

Постановка проблеми. Продуктивність модульних систем визначається не складністю окремих процедур, а їх композицією. Типовий приклад має таку структуру: ділова операція реалізована як ланцюг сервісних викликів, кожен сервіс ізольовано має прийнятну асимптотику, проте композиція в циклі утворює квадратичну поведінку. Профілювання виявляє такі дефекти лише за достатнього обсягу даних у виробничому середовищі, тобто вже після випуску системи. Класичні статичні аналізатори, орієнтовані на окремі функції, зазначеної проблеми не покривають, оскільки втрачають контекст міжмодульних викликів.

Сучасні інструменти, що працюють до запуску коду, не закривають один спеціальний клас задач, а саме сервіси на TypeScript з активною інтеграцією із зовнішніми системами. Формальні аналізатори ресурсних меж COSTA [4] і RAML [3] орієнтовані на байткод JVM та OCaml відповідно, тому структурне типування TypeScript та його асинхронну семантику не підтримують.

Інфраструктурні фреймворки інтерпроцедурного аналізу Soot, SootUp [5] і WALA, а також композиційний аналізатор Infer [6], будують графи викликів і дозволяють пошук дефектів, проте асимптотичної складності не оцінюють. Інструменти оцінювання складності на основі великих мовних моделей [8, 9] працюють з довільним кодом, але не дають формальних гарантій і не виражають кількісної міри впевненості, придатної до агрегації.

Жоден з перелічених підходів не охоплює задачі інтерпроцедурного аналізу асимптотичної складності для сервісів на TypeScript, у якому структурна основа для типових конструкцій поєднана з резервним оцінюванням семантичних викликів великими мовними моделями, а довіра виражена як ортогональна характеристика оцінки. У статті розв’язано саме цю задачу.

Аналіз останніх досліджень і публікацій. Класичні підходи до аналізу обчислювальної складності беруть початок з робіт Кнута [1] і формалізовані у Кормена та співавторів [2], де викладено алгебру асимптотичних класів. Окремі аспекти аналітичного визначення мажорант для асимптотичних оцінок обчислювальних алгоритмів розглянуто у вітчизняній роботі Потапенка [12]. На цій основі побудовано RAML Хоффмана [3] для OCaml із застосуванням automatic amortized resource analysis і COSTA Альберта та співавторів [4] для

ISSN 2786-6025 Online

байткоду Java через рекурентні рівняння оцінки вартості. Спільне обмеження цих систем полягає у зосередженості на конкретній цільовій мові з суттєво іншою семантикою, ніж екосистема TypeScript.

Інфраструктурні фреймворки інтерпроцедурного аналізу формалізують побудову графа викликів алгоритмами CHA, RTA і k-CFA. Soot та його наступник SootUp [5] орієнтовані на байткод JVM, тоді як WALA охоплює JVM, JavaScript і Python. Композиційний аналізатор Infer [6] застосовує методологію обчислення функціональних підсумків на основі сепараційної логіки у промисловому масштабі. Жоден з цих фреймворків асимптотичну складність не оцінює; інтеграція шару оцінки вартості поверх них становила б окрему задачу. Обмеження глибини рекурсивного вбудовування у запропонованому методі реалізує обмежений варіант методу k-CFA Шиверса [7].

Емпіричні дослідження здатності великих мовних моделей оцінювати асимптотичну складність є відносно новою гілкою. Робота Сікка та співавторів [8] застосовує машинне навчання до ознак синтаксичного дерева; за межами канонічних шаблонів моделі узагальнюються погано. Дюб, Парр і Соарес [9] у тестовому наборі BigO(Bench) систематично оцінюють здатність моделей працювати з обмеженнями асимптотичної складності й документують відсутність формальних гарантій. Принцип самоузгодженості Ванга та співавторів [10] застосовується у запропонованому методі не до основної генеративної задачі, як в оригінальній роботі, а до атомарної підзадачі оцінки складності окремого семантичного листка з подальшою інтеграцією результату в загальну агрегацію. У попередній публікації автора [11] обґрунтовано застосування нотації О велике на рівні модулів архітектури; поточна робота розвиває цей напрям через формалізацію потоків виконання та резервне оцінювання за допомогою великих мовних моделей.

На основі огляду виділяємо три невирішені раніше частини проблеми, які перевіряються в експериментальній частині як три гіпотези. Перша гіпотеза стверджує, що детермінований структурний аналіз досягає високої точності на канонічних алгоритмах за наявності відповідних шаблонів, проте втрачає її на викликах до зовнішніх систем і нестандартних структурах коду. Друга гіпотеза стверджує, що консенсусне оцінювання семантичних листків за принципом самоузгодженості великої мовної моделі дає корисну оцінку у випадках, де структурний аналіз повертає невизначеність. Третя гіпотеза стверджує, що кількісно виражена довіра, поширювана від листя до кореня, дає робочий критерій для розрізнення надійних оцінок від тих, що потребують ручної перевірки.

Мета статті – розробка методу статичного аналізу асимптотичної часової складності потоків виконання TypeScript, що поєднує детермінований шаблонний аналіз структурних конструкцій з консенсусним оцінюванням

ISSN 2786-6025 Online

семантичних викликів за допомогою великих мовних моделей і подає рівень довіри в кількісній формі як ортогональну характеристику оцінки. Завдання дослідження передбачають формалізувати дерево потоку виконання з міжмодульним вбудовуванням, описати ієрархію асимптотичних класів і правила агрегації, реалізувати програмний прототип та апробувати його на трьох тестових наборах.

Виклад основного матеріалу. Як мотиваційний приклад розглянемо потік формування фінансового звіту в обліковій системі. За ідентифікатором клієнта система отримує перелік замовлень, для кожного замовлення завантажує товари й обчислює підсумкову суму. Кожна окрема операція має прийнятну складність: завантаження клієнта O(1), завантаження замовлень O(m), завантаження товарів для одного замовлення O(k), обчислення підсумку O(k). Композиція вкладених викликів утворює потік зі складністю O(m·k). Профілювання виявляє цю проблему лише на великих обсягах даних, тоді як статичний аналіз потоку, що враховує характер взаємодії між модулями, здатний вказати на неї до першого запуску. У літературі цей сценарій відомий як шаблон N+1; внесок методу полягає у формалізмі автоматизованого розгортання композиції міжмодульних викликів у TypeScript разом з кількісним вираженням невпевненості.

Дерево потоку виконання визначаємо як орієнтований граф з вершин двох типів. Внутрішні вершини відповідають керуючим конструкціям коду, а саме послідовності, розгалуженню, циклу, асинхронній паралельній композиції та рекурсії. Термінальні вершини відповідають елементарним операціям або зовнішнім викликам. Коренем дерева є функція точки входу потоку. Оцінкою кожної вершини є трійка значень: клас асимптотичної складності з обмеженого набору, змінна, відносно якої вимірюється складність, та рівень довіри в межах [0, 1]. Особливістю формалізму є підтримка вбудовування викликів через межі модулів. При виявленні виклику побудовник дерева через бібліотеку ts-morph знаходить визначення викликаної функції навіть в іншому файлі через імпорт. Якщо поточна глибина не перевищує заданої межі (k=3 у поточній реалізації, що відповідає обмеженому варіанту методу k-CFA), тіло викликаної функції рекурсивно вбудовується у дерево. Виклики глибше k представлено термінальною вершиною типу «зовнішній виклик», яка делегується гібридному механізму оцінювання.

Метод використовує ієрархію з восьми впорядкованих класів асимптотичної складності, доповнену спеціальним елементом невизначеності O(?):

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!).$$

Множення асимптотичних класів реалізовано через статичну таблицю всіх можливих добутків. Випадки, що виходять за межі ієрархії, класифікуються як O(?) з нульовим рівнем довіри. Листя поділяється на

ISSN 2786-6025 Online

детерміновані та семантичні. До детермінованих належать виклики з відомою асимптотикою з вбудованої таблиці типових операцій над масивами, асоціативними масивами і множинами. До семантичних належать виклики через межі модуля, що не піддаються вбудовуванню, звернення до зовнішніх систем за лексичною евристикою імен та рекурсивні виклики з нерозпізнаним шаблоном скорочення аргументу.

Детерміновані листя оцінюються зіставленням із шаблонами, що дає максимальний рівень довіри. Семантичні листя оцінюються через консенсусний механізм на основі великих мовних моделей за принципом самоузгодженості. Запит до моделі формується із вихідного коду функції, контексту виклику й первинної вхідної змінної; запит надсилається N=3 рази з температурами 0,3, 0,4 і 0,5 за базової моделі claude-sonnet-4-20250514. За наявності більшості (понад N/2 запусків) результат дорівнює мажоритарному класу з рівнем довіри, що становить частку узгоджених відповідей. За відсутності більшості результатом стає найгірший клас за частковим порядком, тобто верхня межа серед отриманих оцінок, з малим рівнем довіри 1/N. Така консервативна стратегія надає перевагу безпеці перед оптимістичною оцінкою.

Алгебра агрегації формалізує обчислення оцінки кореня з оцінок дочірніх вершин. Для послідовної композиції та розгалуження результуюча складність дорівнює максимуму складностей дочірніх вершин, а рівень довіри відповідає мінімуму.

Для циклу складність становить добуток складності тіла на функцію межі: O(log n) для циклу зі скороченням аргументу навпіл і O(n) для лінійного циклу. Для асинхронної паралельної композиції результат дорівнює максимуму складностей конкурентних дочірніх операцій, оскільки загальна робота над колекцією розміру n становить O(n) щодо обсягу незалежно від паралелізму. Для рекурсії результат залежить від форми скорочення аргументу: скорочення навпіл дає множник O(log n), лінійне скорочення дає множник O(n), а нерозпізнаний шаблон дає O(?) з нульовим рівнем довіри.

Прототип реалізовано мовою TypeScript із використанням бібліотеки ts-morph для аналізу абстрактного синтаксичного дерева та офіційного SDK Anthropic для консенсусу великих мовних моделей. Експериментальна валідація виконана на трьох тестових наборах за фіксованих параметрів консенсусу й глибини вбудовування k=3.

Перший експеримент перевіряє першу гіпотезу на канонічних алгоритмах. Еталонний набір містить 30 функцій, поділених на 15 структурно простих шаблонів (constant, linearSearch, nestedLoop, binarySearchIterative, bubbleSort та інші) і 15 структурно складних шаблонів (sortWithLibrary, dijkstra, mergeSort, quickSort, levenshteinDistance, nQueens та інші). Зведені результати у детермінованому режимі наведено в таблиці 1.

ISSN 2786-6025 Online

Таблиця 1

Точність прототипу на еталонному наборі у детермінованому режимі.

Категорія	Кількість	Точність	Середня довіра
Структурно прості шаблони	15	100 % (15/15)	0,82
Структурно складні шаблони	15	60 % (9/15)	0,55
Загалом по вибірці	30	80 % (24/30)	0,68

Прототип коректно класифікував 100 % структурно простих шаблонів і 60 % структурно складних, що дає загальну точність 80 %. Цей показник збігається з раніше опублікованим у конференційних тезах автора результатом і є коректним показником якості наявної системи прописаних вручну шаблонів. Відмова прототипу проявляється на алгоритмах, чий рекурсивний шаблон не передбачено у вбудованому детекторі (permutations, nQueens, mergeSort, quickSort, topologicalSort, powerSet). Такі випадки доцільно делегувати на оцінювання великій мовній моделі, що підтверджує першу гіпотезу.

Другий експеримент перевіряє внутрішню валідність першого: чи не є точність 80 % наслідком надмірного налаштування шаблонів під еталонний корпус. Для перевірки функцію detectShrinkPattern замінено на повернення фіксованого значення «unknown», що вимикає всі прописані вручну шаблони розпізнавання форм скорочення аргументу при рекурсії. Прогон на тому ж еталонному наборі дав загальну точність 76,7 %: на структурно простих шаблонах 100 % збереглися, а на структурно складних впала до 53,3 %. Сім алгоритмів (memoizedFibonacci, permutations, powerSet, mergeSort, quickSort, topologicalSort, nQueens) без шаблонів скорочення переходять у клас невизначеності. Цей результат означає, що ефективність детермінованого шару істотно залежить від спеціалізованих шаблонів, тому 80 % коректніше інтерпретувати як верхню межу детермінованого шару на корпусі канонічних алгоритмів, а не як показник узагальнюючої здатності методу.

Третій і четвертий експерименти перевіряють другу гіпотезу на корпусі, що систематично виходить за межі області налаштування шаблонів. Контрольний корпус містить 30 функцій TypeScript, адаптованих з типових шаблонів виробничого коду серверних застосунків: 10 інтеграцій HTTP через axios і fetch, 10 запитів через ORM (Prisma, Mongoose і сирий драйвер pg), 5 операцій з кешем Redis, 5 змішаних потоків. Для кожної функції встановлено еталонну складність на основі ручного аналізу із задокументованим обґрунтуванням. Результати порівняння режимів наведено в таблиці 2.

ISSN 2786-6025 Online

Таблиця 2

Контрольний корпус: детермінований режим
проти гібридного режиму з консенсусом.

Підкатегорія	Розмір	Без моделі	Гібрид	Приріст
HTTP	10	50 % (5/10)	90 % (9/10)	+40 в.п.
ORM	10	10 % (1/10)	100 % (10/10)	+90 в.п.
Cache	5	0 % (0/5)	100 % (5/5)	+100 в.п.
Mixed	5	0 % (0/5)	80 % (4/5)	+80 в.п.
Загалом	30	20 % (6/30)	93 % (28/30)	+73 в.п.

Гібридний режим підняв точність із 20 % до 93 % і повністю усунув клас невизначеності O(?) у вибірці. 21 функція, що в детермінованому режимі повертала O(?), отримала фінальну оцінку від консенсусу. Найбільший приріст спостерігається в категоріях ORM (+90 відсоткових пунктів) і Cache (+100 відсоткових пунктів), де лексична евристика семантичних викликів виявляє листя послідовно й модель має чіткий контекст. Середній рівень довіри у гібридному прогоні становить 1,00, тобто всі три запуски консенсусу зійшлися мажоритарно для кожної функції. Жодний випадок не потребував переходу до консервативної стратегії.

Дві помилки гібридного режиму заслуговують окремого аналізу. Функція fetchOrderItemsParallel (очікуване $O(n^2)$, отримане O(n)) виявляє обмеження на етапі побудови абстрактного синтаксичного дерева: вкладений Promise.all усередині асинхронного зворотного виклику, що повертається з Array. prototype.map, не потрапляє у дерево потоку як паралельна композиція, тому резервна оцінка великою мовною моделлю на цьому фрагменті не запускається. Функція drainQueueWorker (очікуване O(?), отримане O(1)) є контрольним випадком, у якому правильною відповіддю є саме O(?), оскільки завершення циклу залежить від стану зовнішньої черги, а не від вхідного параметра. Консенсус повернув одностайне O(1) у трьох запусках, тобто модель надала перевагу синтаксично видимим шаблонам над семантично складною формою невизначеності. У логах прогону зафіксовано також 40 попереджень про перевищення таймауту на 90 викликів моделі, проте мажоритарність двох з трьох запусків стабільно компенсувала одиничні збої й не вплинула на жодну фінальну оцінку.

Перший експериментальний приклад ілюструє роботу механізму міжмодульного вбудовування на композиції типу N+1. Архітектуру складають чотири анотовані функції в трьох модулях: сервіс клієнтів (O(1) пошук за

ISSN 2786-6025 Online

ідентифікатором), сервіс замовлень (O(1) щодо клієнта), сервіс товарів (O(1) пошук за ідентифікатором замовлення плюс O(n) обчислення підсумкової суми) та контролер з кореневою функцією формування звіту, яка об'єднує ці виклики у циклі за списком замовлень. Прототип у детермінованому режимі коректно вбудував тіла функцій сервісу товарів у дерево потоку контролера, виявив вкладений цикл і обчислив підсумкову складність як $O(n^2)$ з повним рівнем довіри. Жодна з ізольованих функцій сама по собі не сигналізує про проблему. Метод розпізнає квадратичну складність композиції викликів на етапі статичного аналізу через явне розгортання потоку через межі модулів.

Другий експериментальний приклад демонструє повний цикл гібридного режиму на функції fetchAllPayloads з викликом до зовнішнього клієнта HTTP у циклі. Прототип не має шаблону для довільного користувацького клієнта HTTP, тому у детермінованому режимі повертає O(?) з нульовим рівнем довіри для семантичного листка httpClient.fetch(), і ця невизначеність поширюється до кореня. У гібридному режимі всі три запуски консенсусу одностайно класифікували виклик як O(1). Алгебра агрегації перемножила оцінку листка на лінійний цикл за ids довжиною n, давши кінцеву оцінку кореня O(n) з рівнем довіри 1,00. Цей випадок демонструє розподіл відповідальності між компонентами: детермінований шар обробляє все, що піддається структурному зіставленню, шар на основі великої мовної моделі відповідає за випадки, де структурний аналіз принципово не може дати відповіді, а кількісно виражена довіра підсумовує надійність ланцюга.

Висновки. Запропонований метод заповнює нішу інтерпроцедурного аналізу асимптотичної складності для екосистеми TypeScript, не охоплену формальними аналізаторами ресурсних меж сімейства JVM та OCaml і системами оцінювання складності виключно на основі великих мовних моделей, що не дають формальних гарантій.

Наукові результати роботи можна сформулювати таким чином. Уперше запропоновано гібридний метод статичного аналізу асимптотичної складності для коду TypeScript, який поєднує детермінований шаблонний аналіз структурних конструкцій з консенсусним оцінюванням семантичних викликів за принципом самоузгодженості великої мовної моделі та поширює кількісно виражений рівень довіри як ортогональну характеристику оцінки.

Удосконалено формалізм дерева потоку виконання шляхом введення міжмодульного вбудовування викликів через межі модулів з обмеженням глибини k=3, що дозволяє виявляти приховані квадратичні складності у композиції сервісів, не доступні аналізаторам окремих функцій. Удосконалено алгебру агрегації асимптотичних оцінок за рахунок поширення довіри від листя дерева потоку до кореня з консервативною стратегією у випадках відсутності мажоритарності консенсусу. Набуло подальшого розвитку застосування

ISSN 2786-6025 Online

принципу самоузгодженості великих мовних моделей у статичному аналізі коду: на відміну від оригінальної постановки, де самоузгодженість застосовується до основної генеративної задачі, у запропонованому методі вона редукована до атомарної підзадачі оцінки складності окремого семантичного листка з подальшою інтеграцією результату у загальну агрегацію. Практичне значення результатів полягає у можливості виявлення квадратичної складності композиції викликів у сервісах TypeScript до запуску коду, що раніше потребувало профілювання у виробничому середовищі.

Емпірична валідація на трьох наборах функцій підтвердила сформульовані гіпотези. Перший експеримент на еталонному наборі канонічних алгоритмів дав 80 % точності в детермінованому режимі. Другий експеримент через виключення детектора рекурсивних шаблонів знизив точність до 76,7 % і показав чутливість показника до конкретних шаблонів, що підтвердило рамку інтерпретації результату 80 %. Третій експеримент на контрольному корпусі прикладів виробничого коду TypeScript продемонстрував падіння точності детермінованого режиму до 20 %. Четвертий експеримент на тому ж корпусі у гібридному режимі підняв точність до 93 %, що становить приріст у 73 відсоткові пункти й обґрунтовує гібридну архітектуру. Перший експериментальний приклад проілюстрував механізм міжмодульного вбудовування на шаблоні N+1, а другий продемонстрував повний цикл гібридного режиму з одностайним консенсусом 3 з 3.

Метод має кілька явних обмежень. Залежність від викликів, що розв’язуються статично, виключає підтримку динамічного диспетчеризування та конструкцій eval. Замикання з захопленими змінними, що змінюють складність, аналізуються неповно. Межа глибини вбудовування k=3 виключає глибокі ланцюги викликів. Емпірично виявлено два класи помилок гібридного режиму. Перший клас полягає в тому, що вкладений Promise.all усередині асинхронного зворотного виклику Array.map не розкривається у дерево потоку. Другий клас охоплює функції із завершенням, що залежить від зовнішнього стану, а не від вхідного параметра, які інтерпретуються консенсусом як O(1) замість O(?). Друге обмеження свідчить, що поточна реалізація консенсусу надає перевагу синтаксично видимим шаблонам над семантично складними формами невизначеності й потребує окремого класифікатора, який визначав би, чи залежить асимптотика функції від вхідного параметра, чи від зовнішнього стану. Середній рівень довіри 1,00 на контрольному корпусі є характеристикою цього корпусу, а не універсальною гарантією методу: всі семантичні листки у вибірці належали до шаблонів виробничого коду, на яких модель консенсусу досягає одностайності; на складніших семантичних формах очікується нижча мажоритарність та активація консервативної стратегії, що передбачена алгеброю агрегації. Пряме порівняння на спільному корпусі з COSTA, RAML,

ISSN 2786-6025 Online

Soot та Infer не виконано через різницю цільових мов і вихідних форматів цих систем.

До перспектив подальших досліджень належать розширення прототипу на Java, Kotlin і Python через підключення відповідних аналізаторів синтаксичного дерева; інтеграція у процеси безперервної автоматизованої перевірки коду у формі правил для статичних аналізаторів із порогом на рівень довіри; калібраційне дослідження довіри консенсусу на корпусі з відкритих репозиторіїв з аналізом стабільності між моделями різного покоління; розширена алгебра асимптотичних класів, що враховує не тільки часову, а й просторову складність потоків; створення спільного тестового набору асимптотичних оцінок для прямого порівняння з аналогами. Програмний прототип, тестові набори, експериментальні приклади та скрипти запуску тестування доступні у відкритому репозиторії за ліцензією ISC за адресою https://github.com/kyselevych/flow-complexity.

Література:

1. Knuth D. E. The Art of Computer Programming. Vol. 1: Fundamental Algorithms. 3rd ed. Reading, MA : Addison-Wesley, 1997. 650 p.

2. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 4th ed. Cambridge, MA : MIT Press, 2022. 1312 p.

3. Hoffmann J., Das A., Weng S.-C. Towards Automatic Resource Bound Analysis for OCaml. Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL'17). New York : ACM, 2017. P. 359–373. DOI: 10.1145/3009837.3009842.

4. Albert E., Arenas P., Genaim S., Puebla G., Zanardini D. Cost Analysis of Object-Oriented Bytecode Programs. Theoretical Computer Science. 2012. Vol. 413, no. 1. P. 142–159. DOI: 10.1016/j.tcs.2011.07.009.

5. Karakaya K., Schott S., Klauke J., Bodden E., Schmidt M., Luo L., Tan D. SootUp: A Redesign of the Soot Static Analysis Framework. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2024). LNCS, vol. 14570. Cham : Springer, 2024. P. 229–247.

6. Calcagno C., Distefano D., O'Hearn P. W., Yang H. Compositional Shape Analysis by Means of Bi-Abduction. Journal of the ACM. 2011. Vol. 58, no. 6. P. 1–66. DOI: 10.1145/2049697.2049700.

7. Shivers O. Control-Flow Analysis of Higher-Order Languages. PhD thesis. Carnegie Mellon University, 1991.

8. Sikka J., Satya K., Kumar Y., Uppal D., Shah R. R., Zimmermann R. Learning Based Methods for Code Runtime Complexity Prediction. Advances in Information Retrieval. ECIR 2020. Lecture Notes in Computer Science, vol. 12035. Cham : Springer, 2020. P. 313–325. DOI: 10.1007/978-3-030-45439-5_21.

9. Dube E., Parr T., Soares G. BigO(Bench): Can LLMs Generate Code with Controlled Time and Space Complexity. arXiv preprint arXiv:2503.15242. 2025. URL: https://arxiv.org/abs/2503.15242 (дата звернення: 28.04.2026).

10. Wang X., Wei J., Schuurmans D., Le Q. V., Chi E. H., Narang S., Chowdhery A., Zhou D. Self-Consistency Improves Chain of Thought Reasoning in Language Models. International Conference on Learning Representations (ICLR 2023). URL: https://arxiv.org/abs/2203.11171 (дата звернення: 28.04.2026).

ISSN 2786-6025 Online

11. Киселевич В., Іванов Д. Використання нотації О велике у розробці інформаційних систем. Сучасні інформаційні технології в освіті та науці : IX Всеукр. наук.-практ. конф. з міжнар. участю, 21–22 листопада 2024 р. Житомир, 2024. С. 375–379.

12. Потапенко С. Д. Про деякі аспекти асимптотичного оцінювання складності обчислювальних алгоритмів. Моделювання та інформаційні системи в економіці : зб. наук. пр. Київ : КНЕУ, 2023. Вип. 103. С. 177–187.

References:

1. Knuth, D. E. (1997). The Art of Computer Programming. Vol. 1: Fundamental Algorithms (3rd ed.). Reading, MA: Addison-Wesley.

2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to Algorithms (4th ed.). Cambridge, MA: MIT Press.

3. Hoffmann, J., Das, A., & Weng, S.-C. (2017). Towards Automatic Resource Bound Analysis for OCaml. In Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL'17) (pp. 359–373). New York: ACM. https://doi.org/10.1145/3009837.3009842

4. Albert, E., Arenas, P., Genaim, S., Puebla, G., & Zanardini, D. (2012). Cost Analysis of Object-Oriented Bytecode Programs. Theoretical Computer Science, 413(1), 142–159. https://doi.org/10.1016/j.tcs.2011.07.009

5. Karakaya, K., Schott, S., Klauke, J., Bodden, E., Schmidt, M., Luo, L., & Tan, D. (2024). SootUp: A Redesign of the Soot Static Analysis Framework. In Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2024) (LNCS 14570, pp. 229–247). Cham: Springer.

6. Calcagno, C., Distefano, D., O'Hearn, P. W., & Yang, H. (2011). Compositional Shape Analysis by Means of Bi-Abduction. Journal of the ACM, 58(6), 1–66. https://doi.org/ 10.1145/2049697.2049700

7. Shivers, O. (1991). Control-Flow Analysis of Higher-Order Languages (PhD thesis). Carnegie Mellon University.

8. Sikka, J., Satya, K., Kumar, Y., Uppal, D., Shah, R. R., & Zimmermann, R. (2020). Learning Based Methods for Code Runtime Complexity Prediction. In Advances in Information Retrieval. ECIR 2020. Lecture Notes in Computer Science, 12035, 313–325. Cham: Springer. https://doi.org/10.1007/978-3-030-45439-5_21

9. Dube, E., Parr, T., & Soares, G. (2025). BigO(Bench): Can LLMs Generate Code with Controlled Time and Space Complexity. arXiv:2503.15242. https://arxiv.org/abs/2503.15242

10. Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., & Zhou, D. (2023). Self-Consistency Improves Chain of Thought Reasoning in Language Models. In International Conference on Learning Representations (ICLR 2023). https://arxiv.org/abs/2203.11171

11. Kyselevych, V., & Ivanov, D. (2024). Vykorystannia notatsii O velyke u rozrobtsi informatsiinykh system [Application of Big-O notation in information systems development]. In Suchasni informatsiini tekhnolohii v osviti ta nautsi (pp. 375–379). Zhytomyr [in Ukrainian].

12. Potapenko, S. D. (2023). Pro deiaki aspekty asymptotychnoho otsiniuvannia skladnosti obchysliuvalnykh alhorytmiv [On some aspects of asymptotic estimation of computational algorithm complexity]. Modeliuvannia ta informatsiini systemy v ekonomitsi, 103, 177–187. Kyiv: KNEU [in Ukrainian].

Дата першого надходження статті до видання: 13.05.2026
Дата прийняття статті до друку після рецензування: 27.05.2026