Ілля Копина, Ярослав Соколовський*
[1] НЛТУ України, Львів, Україна, ORCID: 0009-0007-4241-8738,
ignars3@gmail.com
[2] НУЛП, Львів, Україна, ORCID: 0000-0003-4866-2575,
yaroslav.i.sokolovskyi@lpnu.ua

Алгоритм представлення програмного коду для аналізу та оброблення штучним інтелектом

Анотація. У роботі запропонований і описаний алгоритм представлення програмного коду, який може оброблятися штучним інтелектом для оцінки його якості, виявлення потенційних помилок та подальшої оптимізації. Основна увага приділена процесу побудови абстрактного синтаксичного дерева та його перетворення в універсальне числове представлення, що забезпечує збереження структури та семантичної інформації. Запропонований алгоритм може бути розглянутий як один з підходів для автоматизації аналізу коду у великих програмних системах.

Ключові слова – абстрактне синтаксичне дерево; аналіз коду; векторизація; оптимізація коду; токенізація.

З розвитком інформаційних технологій та зростанням складності програмного забезпечення, виникає потреба у методах автоматизованого аналізу та оптимізації коду. Штучний інтелект відкриває нові можливості для оброблення великих обсягів коду, аналізу його якості, виявлення потенційних помилок та варіантів оптимізації. Проте для того, щоб штучний інтелект зміг ефективно працювати з програмним кодом, необхідно коректно представити його структуру та семантику в числовому форматі, оскільки саме в такому форматі вона буде зрозумілою для моделей машинного навчання.

Одним з перших етапів такої оброблення є синтаксичний аналіз коду, а саме - токенізація. Вона дає змогу розділити код на базові одиниці – токени. Токенами можуть бути ключові слова, оператори, ідентифікатори, що відображають структуру коду та полегшують його аналіз. Наступним етапом розглядається побудова абстрактного синтаксичного дерева, яке відображає синтаксичну ієрархію коду. Подальші етапи передбачають перетворення синтаксичного

* науковий керівник

дерева у числове представлення та збереження не лише синтаксичної, але й семантичної інформації про код.

У цьому дослідженні розглядаються існуючі підходи до побудови абстрактного синтаксичного дерева та представлені нові методи перетворення його у числові значення, які забезпечують ефективну та повну обробку коду. Пропонований підхід має потенціал для використання у великих програмних проектах, де автоматизація аналізу коду може значно покращити продуктивність розробників та якість кінцевого продукту.

Матеріали та методи дослідження. Процес оброблення програмного коду штучним інтелектом починається з токенізації, яка дає змогу розділити код на базові елементи - токени. Токенізація перетворює кожен рядок коду на набір атомарних частин, таких як ключові слова, оператори, ідентифікатори, знаки пунктуації та літерали [1]. Токенізація дає змогу виділити основні компоненти коду, наприклад у випадку функцій, буде відображено її назву, параметри та оператори. Це розбиття необхідне для того, щоб в подальшому побудувати абстрактне синтаксичне дерево.

Побудова абстрактного синтаксичного дерева є дуже важливим етапом аналізу коду, адже саме з його допомогою зберігається структура коду. Абстрактне синтаксичне дерево представляє собою ієрархічну структуру, яка відображає синтаксичну організацію програми. Основна відмінність синтаксичного дерева від простої токенізації заключається в тому, що дерево забезпечує структуроване представлення логіки та взаємозв’язків між елементами коду та програми в цілому [2].

Останнім етапом є перетворення синтаксичного дерева в числовий формат для того, щоб штучний інтелект міг ефективно працювати з кодом. Кожен елемент дерева має бути представлений числом, щоб модель штучного інтелекту могла використовувати ці дані. Це перетворення в числовий формат важливе, адже більшість алгоритмів машинного навчання працюють виключно з числовими даними, тому текстові елементи мають бути закодовані в числа. Також числове представлення дає змогу зберегти не лише синтаксичну структуру ко-

* науковий керівник

ду, але й певну семантику, якщо використовувати методи, які враховують зв’язки між елементами дерева. Зараз найпоширеніший метод перетворення дерева у числа - це використання векторизації або ембедингів. Наприклад, кожен елемент дерева можна закодувати за допомогою моделей, таких як Word2Vec або CodeBERT, які перетворюють дерево в багатовимірні вектори і при цьому зберігають інформацію про контекст і значення. Це дає змогу моделі розрізняти значення різних токенів та їхній зв’язок між собою.

Незважаючи на успішність векторизації, при обробці програмного коду виникають певні труднощі. Наприклад програмний код може мати складну структуру з ієрархічними зв’язками. Прості методи векторизації можуть не зберігати всі ці зв’язки, що призводить до втрати важливої інформації. Також деякі токени можуть мати різні значення залежно від контексту. Наприклад, ключове слово «ref» може мати різне значення, залежно від місця його використання. Це ускладнює створення уніфікованого числового представлення, яке б точно відображало семантику коду. Натомість, при використанні ембедингів кожен токен буде представлений багатовимірним вектором, що збільшує обсяг обчислень і потребує значних ресурсів для оброблення коду великих розмірів. Для подолання таких труднощів, у цій роботі пропонується алгоритм, який перетворює синтаксичне дерево у числове значення, зберігаючи при цьому інформацію про структуру та зв’язки коду. Такий підхід може забезпечити ефективнішу обробку коду, що в кінцевому результаті покращить оптимізацію та якість програмного забезпечення.

Реалізація власного алгоритму. Запропонований алгоритм для перетворення абстрактного синтаксичного дерева у числовий формат базується на концепції використання бази даних у форматі ключ-значення, яка забезпечує унікальне числове представлення кожного елемента дерева з урахуванням контексту його використання. Не менш важливим аспектом алгоритму є поділ на окремі таблиці для різних видів дерев. Наприклад, для дерева, що представляє функцію, і для дерева, що представляє клас, використовуються окремі таблиці. Такий підхід дає змогу уникнути неоднозначності при роботі з токенами, які

* науковий керівник

можуть мати різне значення залежно від контексту. Як приклад можна навести токен «ref», який у дереві функції позначає модифікатор параметра, а у дереві структури додає обмеження на саму структуру. Використання різних таблиць для цих дерев дає змогу забезпечити унікальні числові значення для такого токена в кожному з контекстів, що є важливим для коректної інтерпретації коду штучним інтелектом.

Додатково, алгоритм враховує особливості власних найменувань, таких як імена змінних або функцій. Для цього, кожному символу алфавіту і певним спеціальним символам присвоюється унікальне числове значення. Назва представляється у вигляді масиву чисел, де кожне число відповідає конкретному символу. Це дає змогу зберегти довжину та структуру найменувань, що є важливим для перевірки стандартів найменування. Наприклад, так можна перевірити чи починаються приватні поля з символу «_». Такий підхід також забезпечує можливість порівняння схожих найменувань за допомогою моделей машинного навчання.

Окремо потрібно виділити алгоритм, який відповідальний за розмежування логічних частин дерева в числовому масиві. Для цього, пропонується використовувати зарезервовані негативні числові значення, які слугують роздільниками між різними частинами дерева. Наприклад, у дереві функції можна розділити назву функції, її параметри та тіло функції за допомогою спеціальних значень, таких як «-1» або «-2». Це дає змогу штучному інтелекту чітко ідентифікувати, до якої саме частини дерева відносяться ті чи інші числові значення. Також, це забезпечує збереженню ієрархічної структури дерева, необхідної для точного аналізу та оптимізації коду.

Алгоритм має багато переваг порівняно з аналогами. Використання бази даних ключ-значення дає змогу ефективно управляти числовими значеннями для різних токенів, забезпечуючи точну відповідність між токеном і його числовим представленням у певному контексті. Завдяки поділу на таблиці зберігається гнучкість у роботі з різними типами синтаксичних дерев, а також вирішується проблема неоднозначності токенів. Застосування числового представлен-

* науковий керівник

ня найменувань дає можливість моделювати як семантику, так і особливості стилю коду. А механізм розмежування забезпечує логічну структурованість числового масиву, що полегшує аналіз коду для штучного інтелекту.

З іншої сторони, алгоритм має недоліки, наприклад необхідність створення та підтримки багатьох таблиць для різних видів дерев. З цього також випливає, що процес аналізу самих абстрактних синтаксичних дерев буде більш трудомісткий. Окрім цього, при складних деревах масиви можуть ставати досить великими, що вимагатиме додаткових ресурсів для оброблення. Звідси випливає, що проблема збереження ресурсів не повністю вирішена, але кількість чисел в масивах все одно буде значно меншою.

Тому, цей алгоритм є хорошою альтернативою теперішнім аналогам. Більше того, він значно ефективніше підходить для автоматизації оброблення програмного коду.

Висновок. Підсумовуючи, у сучасному аналізі програмного коду для його числового представлення часто використовуються методи векторизації та ембедингу, які дають змогу моделювати семантику коду. Проте, такі підходи мають значні недоліки. Зокрема, векторизація часто не враховує складну ієрархічну структуру коду, а також не вирішує проблему контекстної неоднозначності токенів. Ембединг, в свою чергу, збільшує обчислювальну складність через використання багатовимірних векторів. Запропонований алгоритм перетворення абстрактного синтаксичного дерева у числовий формат вирішує ці проблеми.

Цей алгоритм забезпечує стабільність числових представлень, зменшує загальний обсяг даних для оброблення та дає змогу більш детально провести аналіз коду та його залежностей. Запропонований підхід є ефективною альтернативою традиційним методам і має потенціал для широкого застосування в автоматизації аналізу та оптимізації коду.

У майбутньому його можна покращити завдяки більш деталізованому розмежуванню таблиць для абстрактних синтаксичних дерев та завдяки скоро-

* науковий керівник

ченню числових значень токенів в залежності від обставин. Отже, можна досягнути ще більшої ефективності та точності процесу.

Список використаних літературних джерел

[1] Grune, D., Reeuwijk, K., Bal, H, E., Jacobs, C. J. H., Langendoen, K. (2012). Modern Compiler Design. Springer. doi:10.1007/978-1-4614-4699-6.

[2] Mogensen, T. E. (2011). Introduction to Compiler Design. Springer. doi:10.1007/978-3-319-66966-3.

Algorithm for representing program code for analysis and processing by artificial intelligence

Illia Kopyna

The paper proposes and describes algorithm for representing program code that can be later processed by artificial intelligence to evaluate its quality, identify potential problems, and optimize it. The main focus is on the process of constructing an abstract syntax tree and converting it into a universal numerical representation that preserves the structure and semantic information. The proposed algorithm can be considered as one of the approaches for automating code analysis in large software systems.

* науковий керівник