Метод поиска семантически близких фрагментов программного кода

В. И. Зорин [0009-0004-0271-1882], **Е. К. Липачёв** [0000-0001-7789-2332]

Казанский федеральный университет,
Институт информационных технологий и интеллектуальных систем

Аннотация. Предложен метод кросс-языкового поиска семантически близких фрагментов программного кода. Метод основан на построении и анализе абстрактных синтаксических деревьев, представляющих код и использовании модели генерации векторных представлений кода для оценки их функционального сходства через косинусную близость полученных векторов.

Ключевые слова: абстрактное синтаксическое дерево, кросс-языковой поиск, межъязыковой клон, сходство программного кода, эмбеддинг кода.

Method for Searching Semantically Similar Fragments of Program Code

V. I. Zorin[0009-0004-0271-1882], **E. K. Lipachev**[0000-0001-7789-2332],
Institute of Information Technologies and Intelligent Systems,
Kazan Federal University

Abstract. We propose a method for cross-language search of semantically similar fragments of program code. This method is based on the construction and analysis of abstract syntax trees using a model for generating vector representations of code and assessing their functional similarity through the cosine proximity of embeddings.

Keywords: abstract syntax tree, code embedding, cross-language clone, cross-language code search, code similarity.

1. Введение

Программный код, в настоящее время, рассматривается как объект научного знания и, как следствие, важной задачей является создание систем оценки его качества, поиска, анализа и обнаружения схожих решений и подходов. Возможность эффективно находить семантически близкий код на различных языках программирования является критически важной задачей для современной компьютерной науки и индустрии (см., например, [1]).

Решение этой задачи имеет и существенную практическую ценность, поскольку позволяет сократить временные затраты при выполнении таких

повседневных задач, как обучение новым языкам программирования – возможность увидеть, как знакомая задача или алгоритм реализован на изучаемом языке, значительно ускоряет процесс обучения и понимания идиом нового языка; работа в мультиязычных проектах, доля которых по статистике превышает 80% (см., например, [2]), а также миграция проектов на другой язык технологический стек – распространенная задача, часто обусловленная устареванием технологий или требованиями производительности.

Несмотря на активное исследование задачи кросс-языкового поиска клонов кода и создание различных решений как результатов этих исследований [3–7], большинство из них обладают рядом ограничений.

Задача поиска схожего кода исследуется, преимущественно в контексте поиска клонов кода (выявление дубликатов или очень похожих фрагментов) и кросс-языкового поиска кода. Как отмечается в ряде исследований, клоны программного обеспечения наносят ущерб поддержке и развитию и сопровождению программного обеспечения (см., например, [Vislavski, Nafi]).

Анализ показывает, что, несмотря на прогресс, существующие инструменты имеют ограничения, препятствующие решению указанных задач в полном объеме.

В статье [3] представлен инструмент для выявления дублирующихся фрагментов кода, представленных на 5 языках программирования (C, Java, JavaScript, Modula-2, Scheme), под названием LICCA. В основе алгоритма этого программного инструмента лежит использование обогащённых конкретных синтаксических деревьев (enriched Concrete Syntax Tree, eCST). Учёт семантической близости достигается за счёт наличия универсальных для различных языков программирования узлов в eCST.

В работе [4] предложен кросс-языковой детектор клонов CLCDSA (Cross Language Code Clone Detection). Предлагаемая модель CLCDSA анализирует различные синтаксические особенности исходного кода на языках программирования Java, C# и Python для обнаружения кросс-языковых клонов.

Кросс-языковой метод, использующий как статический, так и динамический анализ для выявления похожего кода на языках Java и Python предложен в работе [5]. Этот метод назван COSAL (Code-to-Code Search Across Languages) и не требует модели машинного обучения.

В работе [6] предложена модель обнаружения программных клонов под названием C4 – цифра 4 отражает использование четырех букв «C» в полном названии модели: Contrastive Cross-Language Code Clone Detection. C4 использует предобученную модель CodeBERT для преобразования программ на разных языках, включая Java, Python, C++ и C#, в многомерные векторные представления.

Метод поиска кода по коду, представленный в [7] под названием Reinforest, улучшает производительность больших языковых моделей (Large Language Models) за счёт включения в процессе обучения не только статических, но и

динамических признаков, а также использования как положительных, так и отрицательных референсных образцов.

Оценка сходства программных кодов Android-приложений представлена в работе [8]. Задача оценки сходства сведена к задаче оценки сходства множеств графов потока управления. Значение сходства вычисляется на основе матрицы сходства. Графы потока управления сравниваются при помощи алгоритмов расстояния редактирования графов и расстояния Левенштейна.

Хоть семантика программных кодов учитывается во всех приведённых исследованиях, в них поддерживаются максимум 4 из 20 языков из списка востребованных программирования [2].

В настоящей работе предложен метод кросс-языкового поиска семантически близких фрагментов программного кода, представленных на 19 языках программирования, входящих в текущий список наиболее востребованных языков: Bash, C, C#, C++, CSS, Go, Haskell, HTML, Java, JavaScript, Kotlin, Lua, PHP, Python, R, Ruby, Rust, Scala, Solidity.

2. Постановка задачи

Обозначим через P – множество всех возможных фрагментов программного кода; $DB \subset P$ – база данных программных фрагментов, доступная системе; $\theta \in [0,1]$ – предустановленное пороговое значение сходства. sim: $P \times P \to [0,1]$ – функция, вычисляющая сходство двух программных фрагментов,

Задача заключается в реализации системы, способной для входного фрагмента кода $c \in P$ сформировать множество пар $R = \{(r_1, s_1), (r_2, s_2,), \ldots, (r_k, s_k,)\ \}$, упорядоченное по убыванию значений второй компоненты, где $r_i \in DB$ (фрагмент кода из базы данных), $s_i = sim(c, r_i)$ (сходство фрагмента кода c из запроса и фрагмента r_i из базы данных). Пары $(r,\ s)$ со значениями $s < \theta$ не включаются в множество R. Система должна быть кросс-языковой и фрагменты кода c и r_i могут быть представлены на различных языках программирования.

Система обязана обеспечивать корректную работу для фрагментов кода, написанных на языках программирования из множества L = {Bash, C, C#, C++, CSS, Go, Haskell, HTML, Java, JavaScript, Kotlin, Lua, PHP, Python, R, Ruby, Rust, Scala, Solidity}.

3. Метод поиска близких фрагментов кода

Поиск близких фрагментов кода состоит из следующих шагов.

Шаг 1. Загрузка в систему базы программных кодов *DB*.

Шаг 2. Выбор порогового значения сходства θ.

Шаг 3. Загрузка запроса c, содержащего программный код на одном из языков программирования из множества L.

Шаг 4. Определение языка программирования, на котором представлен запрос c.

Шаг 5. Векторизация кода c.

Шаг 6. Инициализация множества пар R=$\{(r, sim(c,r))\}$, $r \in DB, sim()$ – функция сходства.

Шаг 7. Цикл по $r \in DB$.

Шаг 7.1. Определение языка программирования, на котором представлен код r.

Шаг 7.2. Векторизация кода r.

Шаг 7.3. Вычисление значения сходства $sim(c,r)$ запроса c и кода r.

Шаг 7.4. Если $sim(c,r) \geq \theta$, добавляем пару $(r, sim(c,r))$ в множество R.

Завершение цикла.

Шаг 8. Сортировка множества R в порядке уменьшения второй компоненты (значения сходства).

Шаг 9. Вывод результатов поиска.

Дальнейшие разделы содержат детализацию шагов приведённого метода поиска близких фрагментов программного кода.

3.1. Представление программного кода

Определение [9]. *Абстрактное синтаксическое дерево (АСД) – это иерархическая синтаксическая структура в виде конечного помеченного ориентированного дерева, представляющего программный код. Вершины этого дерева сопоставлены с операторами языка программирования, а листья – с передаваемыми в них операндами.*

Представление фрагментов кода в виде векторов фиксированной размерности (эмбеддинга) позволяет вычислять расстояние между векторами, которое определяет значение сходства фрагментов программного кода.

Отметим несколько способов векторизации применительно к рассматриваемой задаче.

Модели на основе токенов, например, CodeBERT [10], аналогично обработке естественного языка, работают с последовательностями лексем.

Модели, основанные на деревьях, преобразуют код в древовидные структуры, отражающие синтаксические и семантические правила языка программирования. К таким моделям относятся code2vec и InferCode, использующие абстрактные синтаксические деревья.

Нейронная модель code2vec представляет фрагменты кода в виде непрерывных распределённых векторов фиксированной длины (кодовые векторы), которые используются для прогнозирования семантических свойств кода [11].

Модель InferCode использует сверточную нейронную сеть на основе дерева в качестве кодировщика большого набора кода на языке Java. Эта модель применена к задачам кластеризация кода, обнаружения клонов кода, кросс-языкового поиска кода [12].

Модели на основе графов представляют код в виде различных графов, таких как графы потока управления или графы потока данных, позволяющих учитывать динамические зависимости и поведение кода (см., например, [8, 13].

Модель UniXcoder [14] использует трансформерную архитектуру, включающую формирование последовательности токенов и абстрактных синтаксических деревьях, с последующим их преобразованием в линейные последовательности, которые обрабатываются трансформером.

В настоящей работе предложен метод, основанный на создании собственного набора фрагментов программного кода (см. раздел 3.2) и векторизации на основе модели InferCode (см. раздел 3.4).

3.2. Создание датасета фрагментов программного кода

В этом разделе представлен набор данных (датасет) фрагментов программного кода, сформированный для решения поставленной задачи. Этот набор создан на основе известного набора CodeNet, состоящего из более чем 14 миллионов фрагментов кода и около 500 миллионов строк кода на 55 различных языках программирования [15].

Из датасета CodeNet отобраны по 5000 фрагментов кода для каждого из 19 языков программирования, поддерживаемых моделью InferCode. Исключения составили языки Solidity, R, HTML и CSS, коды которых не представлены в CodeNet.

Датасет организован в виде системы папок, по одной на каждый язык программирования. Каждая папка содержит файлы с исходным кодом для соответствующего языка из множества L (см. раздел 2). Названия файлов соответствуют идентификаторам решений из датасета CodeNet.

Для размещения датасета и получения постоянной ссылки на доступ к нему выбран открытый репозиторий Zenodo, обеспечивающий публикацию научных данных, их хранение и обмен (https://zenodo.org/). Выбор обусловлен возможностью бесплатного использования репозитория и высоким лимитом бесплатного хранилища – до 50 ГБ на все файлы датасета.

Созданный набор данных опубликован на Zenodo и доступен для использования [16]. Описание и README, содержащие дополнительную информацию о способе создания и структуре датасета, представлены на русском и английском языках.

3.3. Определение языка программирования

В рамках задачи определения языка программирования по коду рассмотрены и протестированы два решения.

Нейросетевая модель Guesslang (https://github.com/yoeo/guesslang) обучена примерно на 2 миллионах экземпляров программного кода и способна определять 54 языка программирования. Существенным недостатком этой

модели является скорость работы: полный цикл запуска-завершения определения языка для фрагмента кода в среднем занимает около 5 секунд.

Второе решение, под названием Pygments (https://pygments.org/), является универсальным «подстветчиком» синтаксиса исходного кода и в качестве подзадачи решает задачу определения языка. Решение основано на использовании лексического механизма на основе регулярных выражений, что положительно сказывается на скорости работы. Заявлена поддержка 597 языков программирования и других текстовых форматов. Однако, при тестировании обнаружились проблемы определения языка даже на достаточно простых экземплярах кода с уникальными среди языков программирования зарезервированными словами (например, `fn` в языке Rust).

В связи с недостатками для решения поставленной задачи, обнаруженными в указанных решениях, разработан новый метод, основанный на анализе ключевых (зарезервированных) слов в языках. Важной частью алгоритма является предобработка кода, включающая удаление строковых литералов и комментариев.

Алгоритм определения языка программирования по содержимому кода состоит из следующих шагов.

Шаг 1. Загрузка датасета зарезервированных слов языков программирования, указанных в постановке задачи.

Шаг 2. Предобработка загруженного в систему фрагмента программного кода, в частности, удаление строковых литералов и комментариев.

Шаг 3. Токенизация программного кода.

Шаг 4. Цикл по списку языков программирования.

Шаг 4.1. Подсчёт количества вхождений зарезервированных слов текущего языка программирования в списке токенов.

Шаг 4.2. Вычисление процента использованных зарезервированных слов текущего языка программирования от их общего количества в загруженном фрагменте кода.

Шаг 5.Сортировка списка языков программирования по убыванию значений, полученных на шаге 4.

Язык программирования, находящийся на первом месте в отсортированном списке, считается наиболее вероятным.

Ключевые слова для всех 19 языков программирования и представления, поддерживаемых InferCode, были собраны в датасет зарезервированных слов с их официальных документаций языков программирования. Были составлены и описаны всевозможные шаблоны строковых литералов и комментариев среди языков программирования, указанных в постановке задачи (раздел 2).

3.4. Векторизация программного кода

Преобразование программного кода в эмбеддинги осуществлено на основе модели InferCode [12]. Выбор этой модели обоснован тем, что в ней осуществена

поддержка языков программирования, приведенных в постановке задачи (см. раздел 2). Ещё одно преимущество выбранной модели в том, что она обучена на задаче предсказания поддеревьев АСД, что позволяет ей генерировать эмбеддинги, хорошо отражающие семантику и структуру кода, а не только его лексические особенности. Отметим, что модель обучается самостоятельно, без необходимости ручной разметки данных.

3.5. Оценка сходства программ

Для оценки сходства двух фрагментов программного кода c и r используется косинусное расстояние их векторных представлений

$$sim(c,r) = cos\_sim(vec_c, vec_r) = \frac{vec_c \cdot vec_r}{||vec_c|| \cdot ||vec_r||}.$$

Последовательно для сравниваемых фрагментов программного кода c из запроса и фрагмента r из базы данных на основе модели InferCode выполняется процесс векторизации, в результате которого формируются векторы vec_c и vec_r.

Далее вычисляется косинусное расстояние $sim(c,r)$. Значение этого расстояния, равное 1 означает полное совпадение программных кодов.

Заключение

Предложен метод кросс-языкового поиска семантически близких фрагментов программного кода. Метод основан на построении и анализе абстрактных синтаксических деревьев, представляющих код и использовании модели генерации векторных представлений кода для оценки их функционального сходства через косинусную близость полученных векторов.

В планы развития предложенного в работе метода является создание рекомендательной системы, учитывающей предпочтения пользователей.

Также планируется интеграция сервиса в исследовательскую инфраструктуру цифровой математической библиотеки Lobachevskii-DML [17].

Литература

1. *Thimbleby H.* Improving Science That Uses Code // The Computer Journal. 2023. V. 67, No. 4. P. 1381–1397. https://doi.org/10.1093/comjnl/bxad067. URL: https://academic.oup.com/comjnl/article/67/4/1381/7235536, дата обращения 07.07.2025.

2. *Yang H., Nong Y., Wang S., Cai H.* Multi-Language Software Development: Issues, Challenges, and Solutions // IEEE Transactions on Software Engineering. 2024. V. 50. No 3. P. 512–533. https://doi.org/10.1109/TSE.2024.3358258

3. *Vislavski T., Rakić G., Cardozo N. and Budimac Z.* LICCA: A tool for cross-language clone detection // 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), Campobasso, Italy, 2018, pp. 512–516. https://doi.org/10.1109/SANER.2018.8330250

4. *Nafi K.W., Kar T.S., Roy B., Roy C.K. and Schneider K.A.* CLCDSA: Cross Language Code Clone Detection using Syntactical Features and API Documentation // 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 2019, pp. 1026–1037. https://doi.org/10.1109/ASE.2019.00099

5. *Mathew G., Stolee K.T.* Cross-Language Code Search using Static and Dynamic Analyses // arXiv:2106.09173. 2021. https://doi.org/10.48550/arXiv.2106.09173

6. *Tao C., Zhan Q., Hu X., Xia X.* C4: contrastive cross-language code clone detection // ICPC '22: Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, New York, USA, 2022, pp. 413–424. https://doi.org/10.1145/3524610.3527911

7. *Saieva A., Chakraborty S., Kaiser G.* REINFOREST: Reinforcing Semantic Code Similarity for Cross-Lingual Code Search Models // arXiv:2305.03843. 2023. https://doi.org/10.48550/arXiv.2305.03843

8. *Петров В.В.* Система автоматизации численной оценки сходства Android-приложений // Научный сервис в сети Интернет: труды XXV Всероссийской научной конференции (18-21 сентября 2023 г., онлайн). — М.: ИПМ им. М.В.Келдыша, 2023. — С. 283–297. https://doi.org/10.20948/abrau-2023-33. https://keldysh.ru/abrau/2023/theses/33.pdf

9. *Ахо А.В., Сети Р. Ульман Дж.Д.* Компиляторы: Принципы, технологии и инструменты. М.: Вильямс, 2003. 1184 с. URL: https://linux-doc.ru/programming/assembler/book/compilers.pdf.

10. *Feng Z., Guo D., Tang D., Duan N., Feng X., Gong M., Shou L., Qin B., Liu T., Jiang D., Zhou M.* CodeBERT: A Pre-Trained Model for Programming and Natural Languages // arXiv:2002.08155. 2020. https://doi.org/10.48550/arXiv.2002.08155

11. *Alon U., Zilberstein M., Levy O., Yahav E.* code2vec: learning distributed representations of code // Proc. of the ACM on Programming Languages. 2019. V. 3. Iss. POPL. P. 1–29. https://doi.org/10.1145/3291636

12. *Bui N.D.Q., Yu Y., Jiang L.* InferCode: Self-Supervised Learning of Code Representations by Predicting Subtrees // arXiv:2012.07023. 2020. https://doi.org/10.48550/arXiv.2012.07023

13. *Guo D. et al.* GraphCodeBERT: Pre-training Code Representations with Data Flow // arXiv:2009.08366. 2020. https://doi.org/10.48550/arXiv.2009.08366

14. *Guo D., Lu S., Duan N., Wang Y., Zhou M., Yin J.* UniXcoder: Unified Cross-Modal Pre-training for Code Representation // arXiv:2203.03850. 2022. https://doi.org/10.48550/arXiv.2203.03850

15. *Puri R. et al.* CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks // arXiv:2105.12655. 2021. https://doi.org/10.48550/arXiv.2105.12655

16. *Зорин В.И.* Programming Language Detection Dataset (1.0.0) [Набор данных; Электронный ресурс]. https://doi.org/10.5281/zenodo.15661548

17. *Elizarov A., Kirillovich A., Lipachev E., Nevzorova O., Nevzorov V.* Digital OntoMath Ecosystem Tools for Managing and Developing Mathematical Knowledge // In: Aliev R.A., et al. 12th World Conference “Intelligent System for Industrial Automation” (WCIS-2022). WCIS 2022. Lecture Notes in Networks and Systems. Springer, Cham, 2024. V. 912. P. 110–117. https://doi.org/10.1007/978-3-031-53488-1_13

References

1. *Thimbleby H.* Improving Science That Uses Code // The Computer Journal. 2023. V. 67, No. 4. P. 1381–1397. https://doi.org/10.1093/comjnl/bxad067. URL: https://academic.oup.com/comjnl/article/67/4/1381/7235536, last access 07.07.2025

2. *Yang H., Nong Y., Wang S., Cai H.* Multi-Language Software Development: Issues, Challenges, and Solutions // IEEE Transactions on Software Engineering. 2024. V. 50. No 3. P. 512–533. https://doi.org/10.1109/TSE.2024.3358258

3. *Vislavski T., Rakić G., Cardozo N. and Budimac Z.* LICCA: A tool for cross-language clone detection // 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), Campobasso, Italy, 2018, pp. 512–516. https://doi.org/10.1109/SANER.2018.8330250

4. *Nafi K.W., Kar T.S., Roy B., Roy C.K. and Schneider K.A.* CLCDSA: Cross Language Code Clone Detection using Syntactical Features and API Documentation // 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 2019, pp. 1026–1037. https://doi.org/10.1109/ASE.2019.00099

5. *Mathew G., Stolee K.T.* Cross-Language Code Search using Static and Dynamic Analyses // arXiv:2106.09173. 2021. https://doi.org/10.48550/arXiv.2106.09173

6. *Tao C., Zhan Q., Hu X., Xia X.* C4: contrastive cross-language code clone detection // ICPC '22: Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, New York, USA, 2022, pp. 413–424. https://doi.org/10.1145/3524610.3527911

7. *Saieva A., Chakraborty S., Kaiser G.* REINFOREST: Reinforcing Semantic Code Similarity for Cross-Lingual Code Search Models // arXiv:2305.03843. 2023. https://doi.org/10.48550/arXiv.2305.03843

8. *Petrov V.V.* Automated system for numerical similarity evaluation of Android applications // Nauchnyi servis v seti Internet: trudy XXV Vserossiiskoi nauchnoi konferentsii (18–21 sentiabria 2023 g., g. Novorossiisk). — M.: IPM im. M.V.Keldysha, 2023. — P. 283–297. https://doi.org/10.20948/abrau-2023-33. https://keldysh.ru/abrau/2023/theses/33.pdf

9. *Aho A.V., Lam M.S., Sethi R., Ullman J.D.* Compilers: Principles, Techniques, and Tools (2 ed.). Boston, Massachusetts, US: Addison-Wesley, 2006. 1006 p.

10. *Feng Z., Guo D., Tang D., Duan N., Feng X., Gong M., Shou L., Qin B., Liu T., Jiang D., Zhou M.* CodeBERT: A Pre-Trained Model for Programming and Natural Languages // arXiv:2002.08155. 2020. https://doi.org/10.48550/arXiv.2002.08155

11. *Alon U., Zilberstein M., Levy O., Yahav E.* code2vec: learning distributed representations of code // Proc. of the ACM on Programming Languages. 2019. V. 3. Iss. POPL. P. 1–29. https://doi.org/10.1145/3291636

12. *Bui N.D.Q., Yu Y., Jiang L.* InferCode: Self-Supervised Learning of Code Representations by Predicting Subtrees // arXiv:2012.07023. 2020. https://doi.org/10.48550/arXiv.2012.07023

13. *Guo D. et al.* GraphCodeBERT: Pre-training Code Representations with Data Flow // arXiv:2009.08366. 2020. https://doi.org/10.48550/arXiv.2009.08366

14. *Guo D., Lu S., Duan N., Wang Y., Zhou M., Yin J.* UniXcoder: Unified Cross-Modal Pre-training for Code Representation // arXiv:2203.03850. 2022. https://doi.org/10.48550/arXiv.2203.03850

15. *Puri R. et al.* CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks // arXiv:2105.12655. 2021. https://doi.org/10.48550/arXiv.2105.12655

16. *Zorin V.I.* Programming Language Detection Dataset (1.0.0). https://doi.org/10.5281/zenodo.15661548

17. *Elizarov A., Kirillovich A., Lipachev E., Nevzorova O., Nevzorov V.* Digital OntoMath Ecosystem Tools for Managing and Developing Mathematical Knowledge // In: Aliev R.A., et al. 12th World Conference “Intelligent System for Industrial Automation” (WCIS-2022). WCIS 2022. Lecture Notes in Networks and Systems. Springer, Cham, 2024. V. 912. P. 110–117. https://doi.org/10.1007/978-3-031-53488-1_13