УДК 004.4

РЕКОМЕНДАТЕЛЬНАЯ СИСТЕМА ПОИСКА СЕМАНТИЧЕСКИ БЛИЗКИХ ФРАГМЕНТОВ ПРОГРАММНОГО КОДА

В. И. Зорин[1] [0009-0004-0271-1882], Е. К. Липачев[2] [0000-0001-7789-2332]

[1]*Казанский национальный исследовательский технический университет имени А. Н. Туполева — КАИ, г. Казань,* Россия
[2]*Казанский (Приволжский) федеральный университет, г. Казань,* Россия
[2]*Университет Иннополис, г. Иннополис, Россия*

[1]addefan@mail.ru, [2]elipachev@gmail.com

Аннотация

Рекомендательные системы в научном информационном пространстве являются инструментом поиска и навигации при работе с научными документами. Программный код в настоящее время рассматривается как объект научного знания, и, как следствие, важной задачей является создание систем поддержки жизненного цикла программ, в частности поиска близких программных решений, обнаружения заимствований программного кода, анализа и оценки качества кода. В работе предложена рекомендательная система, формирующая для пользователя персонализированный список фрагментов кода, функционально эквивалентных входному коду-запросу, представленному на одном из языков программирования из установленного набора. Базовый алгоритм системы основан на представлении программного кода в виде абстрактного синтаксического дерева с последующим построением векторного пространства программных кодов. Семантическое сходство программных кодов определяется по расстоянию между векторами кодов в многомерном пространстве. Персонализация выдачи достигается за счет модуля фильтрации, который ранжирует найденные фрагменты с учетом профиля пользователя. Рассматриваемыми факторами являются языковые предпочтения пользователя и его области научных интересов, извлекаемые посредством интеграции с ORCID. Для обеспечения работы системы на основе корпуса CodeNet создан специализированный набор фрагментов про-

граммного кода. Решена также задача автоматического определения языка программирования по фрагменту представленного кода на одном из языков, входящих в текущий рейтинговый список языков программирования.

***Ключевые слова**: абстрактное синтаксическое дерево, векторизация кода, контентная фильтрация, кросс-языковой поиск, межъязыковой программный клон, рекомендательная система, сходство программного кода.*

ВВЕДЕНИЕ

В условиях стремительного усложнения программных систем и расширения цифровых научных архивов разработка методов кросс-языкового анализа исходных кодов программ становится приоритетным направлением исследований. Под семантической близостью программных фрагментов в контексте настоящей работы понимается сходство их функционального поведения, которое аппроксимируется совокупностью структурных и синтаксических паттернов, отражающих логику обработки данных. Под межъязыковым (кросс-языковым) клоном будем понимать фрагмент кода, который демонстрирует синтаксическое (текстовое и структурное) или семантическое (функциональное) сходство с фрагментом кода, написанным на другом языке программирования.

Решение задачи поиска близкого программного кода имеет и существенную практическую ценность в оптимизации процессов обучения и промышленной разработке программного обеспечения. Возможность сопоставления алгоритмически эквивалентных решений на различных языках позволяет значительно ускорить освоение новых технологических идиом и упростить поддержку гетерогенных проектов, доля которых, по статистике, в современной индустрии превышает 80% (см., например, [1]). Кроме того, такие инструменты критически важны при глубоком рефакторинге и миграции систем на новые наборы технологий.

Несмотря на интенсивное развитие области кросс-языкового поиска клонов, большинство существующих инструментов, созданных в рамках соответствующих исследований (см., например, [2–6]), характеризуется жесткой привязкой к ограниченному набору синтаксисов или недостаточной глубиной анализа семантических связей.

Задачу поиска близкого по содержанию кода исследуют преимущественно в контексте поиска клонов кода (выявление дубликатов или очень похожих фрагментов) и кросс-языкового поиска кода. Как отмечено в ряде исследований, клоны программного обеспечения наносят ущерб поддержке, развитию и сопровождению программного обеспечения (см., например, [2, 3]).

Рекомендательные системы в научной деятельности

Рекомендательные системы, наряду с системами поиска, являются наиболее распространенными системами в информационном пространстве. Базовой особенностью рекомендательных технологий является использование алгоритмов, способных учитывать предпочтения отдельного пользователя или категории пользователей в процессе создания персонализированных рекомендаций.

Имеется несколько определений рекомендательных систем, в каждом их которых сделан акцент на определенные их особенности (см., например, [7–9]). В работе [10] проведен анализ различных определений рекомендательных систем, даны ссылки на публикации и приведены сравнения определений с обоснованиями.

В основном в информационном пространстве используются рекомендательные системы, основанные на коллаборативной фильтрации. В научной деятельности применяют рекомендательные системы специального типа, в частности основанные на контенте с учетом семантики (Semantics-Aware Content-Based Recommender Systems) (см., например, [8]). К этому типу систем относится и рекомендательная система, представленная в настоящей работе.

Отметим работу [11], в которой предложена рекомендательная система поиска близких документов в физико-математическом контенте. Она основана на использовании онтологии профессиональной математики.

В работах [12, 13] создана рекомендательная система поиска экспертов для рецензирования математических работ в научном журнале. Эту систему авторы отнесли к типу рекомендательных систем конкретного случая (Case-Based Recommender Systems). Этот тип систем детально описан в [14], где представлены основные свойства таких систем и области их применения.

Рекомендательная система тематической классификации научных журналов предложена в [15]. Она основана на использовании рубрикаторов и классификаторов научно-технической информации, а также онтологии семантической библиотеки предметных областей SciLibRu.

Поскольку программный код в настоящее время рассматривается как самостоятельная единица научного знания, необходимы соответствующие программные инструменты, аналогичные созданным для управления научным контентом. Сегодня создаются исследовательские инфраструктуры (см., например, [16, 17]), цель которых – это интеграция и совместное использование научных документов, исследовательских данных и программ. Для исследовательских данных и программ разрабатываются инструменты поддержки их жизненного цикла, в том числе рекомендательные системы поиска семантически близких фрагментов программного кода.

Обзор близких по тематике исследований

Далее дан анализ существующих инструментальных средств и методов, направленных на решение задач анализа программного кода, выявления дубликатов и поиска функционально близких фрагментов в кросс-языковой среде.

Одним из первых значимых решений в области статического анализа межъязыковых клонов является инструмент LICCA, представленный в [2]. В его основе лежит использование обогащенных конкретных синтаксических деревьев (enriched Concrete Syntax Tree, eCST), которые позволяют унифицировать синтаксические конструкции различных языков программирования (C, Java, JavaScript, Modula-2 и Scheme) за счет введения универсальных узлов. Процесс сопоставления реализуется через сериализацию деревьев и применение модифицированного алгоритма поиска наибольшей общей подпоследовательности (Longest Common Subsequence, LCS). Несмотря на высокую точность на уровне синтаксических единиц, LICCA имеет существенные ограничения: система чувствительна к порядку следования инструкций и требует сопоставимой длины фрагментов кода, что затрудняет ее применение для детекции сложных семантических клонов.

Дальнейшее развитие подходов к синтаксическому анализу нашло отражение в модели, описанной в работе [3]. В отличие от инструментов, полагающихся на промежуточные представления, в этой модели производится анализ

сходства на основе 9 специфических синтаксических признаков, значения которых остаются относительно стабильными для функционально эквивалентного кода на различных языках. Архитектура системы включает «фильтр действий», основанный на семантическом сходстве вызовов API, которое вычисляется с помощью модели Word2Vec и анализа документации библиотек. Для классификации использована глубокая нейронная сеть с сиамской архитектурой, обучаемая на размеченных наборах данных для сопоставления признаков в едином векторе.

Метод COSAL (Code-to-Code Search Across Languages), ориентированный на кросс-языковой поиск, предлагает гибридный подход, объединяющий статический и динамический анализ без использования моделей машинного обучения [4]. Система оценивает релевантность кода по трем независимым критериям: сходство токенов, структурное сходство (на основе деревьев редактирования) и поведенческое сходство (анализ отношений ввода-вывода с использованием инструментария, представленного в [18]). Финальное ранжирование результатов осуществляется с помощью недоминируемой сортировки, что позволяет сбалансировать визуальное сходство кода с его фактическим функциональным поведением без потери нюансов, характерных для агрегированных метрик.

Переход к использованию предобученных моделей представления программного кода сопровождался появлением системы C4 (Contrastive Cross-Language Code Clone Detection) [5]. Этот метод базируется на архитектуре CodeBERT, которая трансформирует фрагменты кода в высокоразмерные векторные эмбеддинги. Ключевой особенностью C4 является применение контрастивного обучения (Contrastive Learning): модель обучается минимизировать расстояние между эмбеддингами функционально эквивалентных программ и максимизировать его для различных задач. Это позволяет эффективно выявлять семантические клоны 4-го типа, которые имеют идентичное поведение при полностью различном синтаксическом исполнении. Декларирована поддержка языков программирования Java, Python, C++ и C#.

Метод, представленный в [6], развивает идеи контрастивного подхода, адаптируя большие языковые модели для поиска кода за счет интеграции статических и динамических признаков на этапе обучения. В отличие от традиционных систем динамического анализа, в этом методе производится кодирование

информации о времени выполнения в виде оценки семантического сходства (Semantic Similarity Score, SSS) только в процессе тренировки модели, что избавляет от необходимости запускать код в момент выполнения поискового запроса. Кроме того, это первая система, использующая как положительные, так и отрицательные эталонные образцы в процессе дообучения, что значительно повышает точность поиска в условиях синтаксических различий между языками запроса и базы данных.

Новым направлением в области кросс-языкового обнаружении программных клонов стала модель, представленная в работе [19]. Эта модель предназначена для поиска клонов в условиях zero-shot-обучения, т. е. без использования параллельных межъязыковых наборов данных. С ее помощью предложено решение задачи выравнивания представлений через три механизма: контрастивное предсказание сниппетов (contrastive snippet prediction, CSP) для построения изоморфного векторного пространства, доменно-ориентированное обучение для устранения языковой специфики и обучение с циклической согласованностью (cycle consistency). Такой подход позволяет системе эффективно сопоставлять функции даже на тех языках программирования, которые не были представлены в обучающей выборке, обеспечивая высокую степень универсальности в открытых научных репозиториях.

Для высокоточного обнаружения межъязыковых клонов на семантическом уровне предложена модель FEGAT (Flow-Enhanced Graph Attention Network) [20]. Используемый в ней метод основан на графовых представлениях, обогащенных информацией о потоках данных и управления. Архитектура решения предполагает построение графа на основе абстрактного синтаксического дерева, дополненного ребрами потоков, которые затем поступают на вход предобученной модели CodeBERT для формирования первичных векторов узлов, насыщенных семантической информацией. Ключевым компонентом системы является нейронная сеть внимания на графах (Graph Attention Network, GAT), которая обучается извлекать компактные представления функциональной логики программ для последующего вычисления оценки их сходства.

Оценка сходства программных кодов Android-приложений представлена в работах [21, 22]. В них задача оценки сходства сведена к оценке сходства мно-

жеств графов потока управления. Значение сходства вычисляется на основе матрицы сходства. Графы потока управления сравниваются при помощи алгоритмов вычисления расстояний редактирования графов и расстояния Левенштейна [23]. Хоть семантика программного кода учитывается во всех приведенных исследованиях, в них поддерживаются до 4 из 20 языков из списка языков программирования, которые наиболее активно используются в настоящее время (см., например, [24, 25]).

Настоящая работа является продолжением исследований, представленных в [26]. Предложен метод кросс-языкового поиска семантически близких фрагментов кода, представленных на 19 языках программирования, входящих в текущий список наиболее активно используемых языков программирования: Bash, C, C#, C++, Go, Haskell, Java, JavaScript, Kotlin, Lua, PHP, Python, R, Ruby, Rust, Scala, Solidity, а также языков разметки CSS и HTML.

1. ПОСТАНОВКА ЗАДАЧИ

В работе используются термины *сходство*, *несходство* и *расстояние*. Эти термины понимаются в классическом смысле (см., например, [27]), а именно, под *сходством* объектов из множества O понимается функция $\sigma: O \times O \to R$, для которой выполнены условия *положительности*: $\forall x, y \in O, \sigma(x, y) \geq 0$, *максимальности*: $\forall x \in O, \forall y, z \in O, \sigma(x, x) \geq \sigma(y, z)$ и *симметричности*: $\forall x, y \in O, \sigma(x, y) = \sigma(y, x)$. Соответственно, двойственное понятие *несходства* объектов определяется как функция $\delta: O \times O \to R$, симметричная, положительная и удовлетворяющая условию *минимальности*: $\forall x \in O, \delta(x, x) = 0$. *Расстояние* на множестве объектов O определяется как функция несходства $\delta: O \times O \to R$, такая что выполнены условие *определенности*: $\forall x, y \in O, \delta(x, y) = 0$ тогда и только тогда, когда $x = y$, и *неравенство треугольника*: $\forall x, y, z \in O, \delta(x, y) + \delta(y, z) \geq \delta(x, z)$.

Обозначим через P множество всех возможных фрагментов программного кода; $DB \subset P$ – база данных программных фрагментов, доступная системе; $c \in P$ – входной программный фрагмент от пользователя; U – множество пользователей системы, где $u \in U$ – текущий пользователь, инициировавший запрос; $\theta \in [0,1]$ – предустановленное пороговое значение семантического сходства;

$\mathrm{sim}: P \times P \to [0,1]$ – функция, вычисляющая семантическое сходство двух программных фрагментов, значение 1 которой означает функциональную идентичность фрагментов, а 0 – полное отсутствие сходства.

Задача заключается в реализации рекомендательной системы, способной в соответствии с профилем пользователя и входным фрагментом кода сформировать набор персональных рекомендаций, состоящий из пар $R = \{(r_1, w_1), (r_2, w_2), \ldots, (r_k, w_k)\}$. Набор R упорядочен по убыванию значений второй компоненты, где $r_i \in DB$ – фрагмент кода из базы данных, $s_i = \mathrm{sim}(c, r_i) \geq \theta$ – сходство фрагмента кода из запроса и фрагмента из базы данных соответственно, значение которого не ниже установленного порога.

Ключевым требованием к системе является вычисление итогового релевантного веса w_i, который должен учитывать не только семантическое сходство s_i, но и контекст пользователя u. Таким образом, вес w_i является функцией трех параметров, а именно: базовой семантической близости s_i, $L_u(\mathrm{lang}(r_i))$ – значения коэффициента предпочтения языка программирования фрагмента r_i (учитывается в профиле пользователя по частоте взаимодействий); $I_u(r_i)$ – значения коэффициента соответствия предметной области фрагмента r_i научным предпочтениям текущего пользователя, извлеченным из его профиля ORCID.

Разрабатываемая система должна быть кросс-языковой, т. е. корректно оценивать функциональную эквивалентность, даже если фрагменты кода c и r_i представлены на различных языках программирования, а также обеспечивать поддержку фрагментов кода, написанных на 19 востребованных языках программирования и разметки из множества

L = {*Bash, C, C#, C++, CSS, Go, Haskell, HTML, Java, JavaScript, Kotlin, Lua, PHP, Python, R, Ruby, Rust, Scala, Solidity*}.

2. РЕКОМЕНДАТЕЛЬНЫЙ АЛГОРИТМ ПОИСКА БЛИЗКИХ ФРАГМЕНТОВ КОДА

Опишем алгоритм формирования рекомендаций по поиску близких фрагментов программного кода. На вход алгоритма подается запрос, содержащий фрагмент кода на любом из языков программирования из списка L. На выходе формируется набор рекомендаций, содержащий близкие фрагменты кода.

2.1. Общая схема алгоритма

Работа предложенного алгоритма начинается с этапа инициализации, на котором происходит подключение к базе данных DB программных фрагментов и задается пороговое значение θ сходства кодов, ограничивающее выдачу только наиболее релевантными решениями. При получении входного кода-запроса с система автоматически идентифицирует его язык программирования и выполняет процедуру векторизации для формирования эмбеддинга.

Дальнейший процесс организован в виде цикла, в ходе которого осуществляется последовательное сопоставление запроса с каждым фрагментом r из базы данных. Для каждого такого фрагмента также определяется язык программирования, после чего фрагмент преобразуется в векторное представление. На основе полученных векторов вычисляется значение косинусного сходства $\mathrm{sim}(c, r)$. Если полученное значение удовлетворяет установленному порогу θ, пара, состоящая из программного кода и его оценки сходства, включается в итоговое множество рекомендаций R.

Завершающим этапом являются сортировка сформированного набора в порядке убывания значения сходства и вывод ранжированного списка пользователю. Описанная последовательность действий описана в псевдокоде, представленном в Листинге 1.

Листинг 1. Псевдокод алгоритма поиска фрагментов программного кода, близких коду-запросу.

```
GET_SIMILAR_PROGRAMS(DB, c, threshold)
1   c_language ← detect_language(c)
2   c_embedding ← vectorize(c, c_language)
3   ▷ Инициализация пустого массива кортежей R
4   foreach r in DB
5       do r_language ← detect_language(r)
6          r_embedding ← vectorize(r, r_language)
7          s ← sim(c_embedding, r_embedding)
8          if s ≥ threshold
```

```
9                    then R ← R ∪ {(r, s)}
10  ▷ Сортировка R по убыванию второй компоненты (s)
11  return R
```

2.2. Создание набора фрагментов программного кода

Для реализации функционала потребовалось создать набор данных (датасет) фрагментов программного кода. Формирование экспериментального набора данных осуществлялось на базе крупномасштабного корпуса CodeNet, включающего более 13.9 млн программных образцов (порядка 500 млн строк кода) на 55 различных языках программирования [28].

Некоторые из рассматриваемых моделей векторизации (например, InferCode, предобученная на наибольшем количестве языков) для генерации эмбеддингов требуют обязательного предварительного указания языка программирования, на котором написан исходный код. В связи с этим решена задача предварительного определения языка программирования.

В рамках проведенного исследования из исходного корпуса было отобрано по 5000 репрезентативных фрагментов для каждого из целевых языков, поддерживаемых моделью векторизации InferCode. Из процесса выборки были исключены языки Solidity, R, HTML и CSS из-за отсутствия соответствующих данных в репозитории CodeNet. Сформированный набор данных опубликован на Zenodo и доступен для использования [29].

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 языков программирования, используемых в алгоритме. Названия файлов соответствуют идентификаторам решений из датасета CodeNet.

2.3. Метод определения языка программирования

Одной из задач при разработке системы является автоматическое определение языка программирования. Это связано с тем, что фрагменты кода, участвующие в сравнении, могут быть представлены на различных языках программирования. Кроме того, синтаксические конструкции фрагмента кода не всегда позволяют пользователю однозначно определить язык программирования.

Разработанный метод основан на анализе зарезервированных слов в языках программирования. Алгоритм определения языка программирования по содержимому кода состоит из следующих шагов. Работа алгоритма начинается с подключения к системе датасета зарезервированных слов для языков программирования из списка L. Для загруженного в систему фрагмента программного кода проводится предобработка, в частности удаление строковых литералов и комментариев, затем выполняется его токенизация. Далее запускается цикл по списку языков программирования, в ходе которого для каждого языка вычисляется количество вхождений его зарезервированных слов в списке токенов и рассчитывается отношение использованных зарезервированных слов к их общему количеству в загруженном фрагменте кода. В завершение производится сортировка списка языков по убыванию полученных значений, и наиболее вероятным считается тот язык программирования, который находится на первом месте в отсортированном списке. Описанный алгоритм и логика вычислений детально представлены в псевдокоде, приведенном в Листинге 2.

Листинг 2. Псевдокод алгоритма определения языка программного кода.

```
DETECT_LANGUAGE(RESERVED_WORDS, code)
1   tokens ← preprocessing(code)
2   ▷ Инициализация пустого ассоциативного массива scores с нулевыми
    значениями по умолчанию
3   foreach language in RESERVED_WORDS
4         do language_words ← RESERVED_WORDS[language]
5            counter ← 0
6            foreach keyword in language_words
7                  do if keyword in tokens
8                        then counter ← counter + 1
9            scores[language] ← counter / length[language_words]
10  ▷ Сортировка scores по убыванию
11  return scores[0]
```

Для обеспечения функционирования модуля идентификации языков из множества L был сформирован специализированный реестр зарезервированных слов, собранных на основе анализа официальной технической документации соответствующих языков программирования. Для задачи предобработки программного кода были разработаны и систематизированы формализованные шаблоны строковых литералов и комментариев, учитывающие синтаксические спецификации всех языков, заявленных в постановке задачи.

2.4. Векторное представление программного кода

Определение [30]. *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.*

На Рис. 1 в качестве примера приведен фрагмент кода алгоритма Евклида, а на Рис. 2 – представление этого кода в виде абстрактного синтаксического дерева.

```
1  def gcd(a: int, b: int):
2      while a != 0 and b != 0:
3          if a > b:
4              a = a % b
5          else:
6              b = b % a
7      return a + b
```

Рис. 1. Алгоритм Евклида поиска наибольшего общего делителя, представленный в виде кода на языке Python.

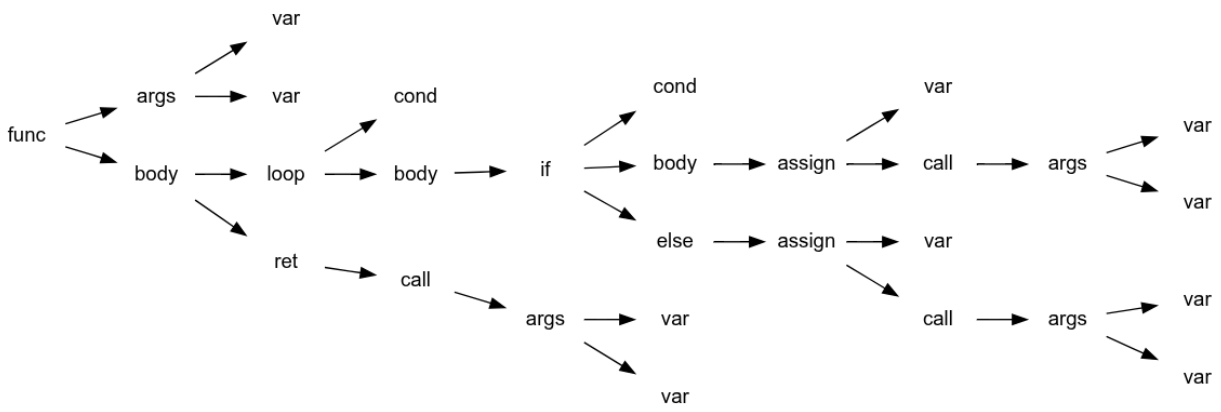

Рис. 2. Абстрактное синтаксическое дерево кода, представленного на Рис. 1.

Отметим, что АСД семантически однозначно представляют фрагменты кода на языках с гомоиконным синтаксисом. Анализ таких представлений представлен в [31].

Представление фрагментов кода в виде плотных векторов фиксированной размерности позволяет перевести задачу анализа программ в область вычислений расстояний в многомерных пространствах, где близость векторов определяет степень функционального или синтаксического сходства объектов. В современных исследованиях выделяется несколько ключевых подходов к векторизации, различающихся архитектурой нейронных сетей и типами используемых данных.

CodeBERT представляет собой бимодальную предобученную модель, предназначенную для захвата семантических связей между естественным языком (natural language, NL) и языками программирования (programming language, PL) [32]. Архитектура модели базируется на многослойном двунаправленном трансформере (Transformer). Обучение CodeBERT осуществляется с использованием гибридной функции потерь, включающей задачу маскированного языкового моделирования (masked language modeling, MLM) и оригинальную задачу детекции замененных токенов (replaced token detection, RTD). Использование RTD позволяет модели эффективно использовать как бимодальные данные (пары «код – документация»), так и большие массивы унимодального кода, что обеспечивает получение универсальных представлений, пригодных для задач поиска кода и генерации документации.

code2vec – это нейронная модель, ориентированная на обучение распределенным представлениям фрагментов кода на основе их синтаксической структуры [33]. Основная идея этого метода заключается в декомпозиции фрагмента кода на набор путей в его абстрактном синтаксическом дереве, соединяющих терминальные узлы. Каждый такой «путь» отображается в вектор, после чего сеть внимания (attention mechanism) вычисляет взвешенное среднее этих векторов для формирования итогового эмбеддинга кода фиксированной длины. Такой подход позволяет модели выделять синтаксические конструкции, наиболее значимые для семантики программы, что было успешно продемонстрировано на задаче предсказания имен методов.

InferCode реализует парадигму самообучения (self-supervised learning) для представления программного кода путем решения задачи предсказания поддеревьев в АСД [34]. В качестве кодировщика в системе используется древовидная сверточная нейронная сеть (tree-based convolutional neural network, TBCNN), которая напрямую обрабатывает иерархическую структуру дерева. Модель обучается предсказывать вероятность появления конкретных поддеревьев в заданном контексте АСД аналогично тому, как модель doc2vec предсказывает слова в документе. Ключевыми преимуществами InferCode являются его кросс-языковая универсальность (polyglot nature) и способность генерировать семантически богатые эмбеддинги, не привязанные к конкретной прикладной задаче.

GraphCodeBERT развивает идеи трансформерных архитектур, интегрируя в процесс обучения информацию о внутренней структуре программ [35]. В отличие от моделей, полагающихся только на токены или АСД, GraphCodeBERT использует графы потока данных, которые отражают семантические отношения зависимости между переменными. В предобучение включены две структурно-ориентированные задачи: предсказание наличия ребер графа потока данных и выравнивание переменных между текстом кода и узлами графа. Для эффективной обработки этих данных применяется специализированная функция маскированного внимания, управляемая графом (graph-guided masked attention).

UniXcoder представляет собой унифицированную кросс-модальную предобученную модель, поддерживающую широкий спектр задач: от классификации до авторегрессионной генерации и дополнения кода [36]. Эта модель использует матрицы маскированного внимания с префиксными адаптерами для гибкого переключения между режимами кодировщика и декодера. UniXcoder эффективно объединяет информацию из исходного кода, комментариев и АСД. Для параллельной обработки древовидных структур предложен метод взаимно однозначного отображения АСД в последовательность токенов, сохраняющий всю структурную информацию. Дополнительно применяются методы контрастивного обучения и кросс-модальной генерации для выравнивания представлений кода на различных языках программирования.

В настоящей работе для формирования векторного пространства программных фрагментов и оценки их семантического сходства в рамках рекомен-

дательной системы была выбрана модель InferCode. Выбор этой модели обусловлен ее способностью работать с широким набором языков программирования, приведенных в постановке задачи, и ориентацией на структурную семантику кода через анализ поддеревьев АСД.

3. РЕКОМЕНДАТЕЛЬНАЯ СИСТЕМА ПОИСКА ПРОГРАММНОГО КОДА

В основе предлагаемого решения лежит парадигма рекомендательных систем, основанных на контенте (Content-Based Recommender Systems, CBRS), которые формируют персональные рекомендации, опираясь на описательные характеристики объектов и профили предпочтений пользователей. Основное допущение таких систем заключается в том, что интересы пользователя остаются стабильными во времени, поэтому ему предлагаются объекты, максимально схожие с теми, которые он положительно оценивал в прошлом. В контексте поиска программного кода это позволяет находить фрагменты, которые по своим функциональным и семантическим свойствам наиболее близки к запросу пользователя (см., например, [37]).

Пользователями настоящей рекомендательной системы являются исследователи и разработчики, работающие в мультидисциплинарных и мультиязычных проектах. Единицей рекомендации выступает фрагмент исходного кода из проиндексированной базы данных, функционально решающий ту же задачу, что и код-запрос, но потенциально написанный на другом языке программирования. Архитектура разработанной контентной рекомендательной системы (см. Рис. 3) включает три основных компонента: анализатор контента (`Content Analyzer`), модуль обучения профиля (`Profile Learner`) и компонент фильтрации и ранжирования (`Filtering Component`).

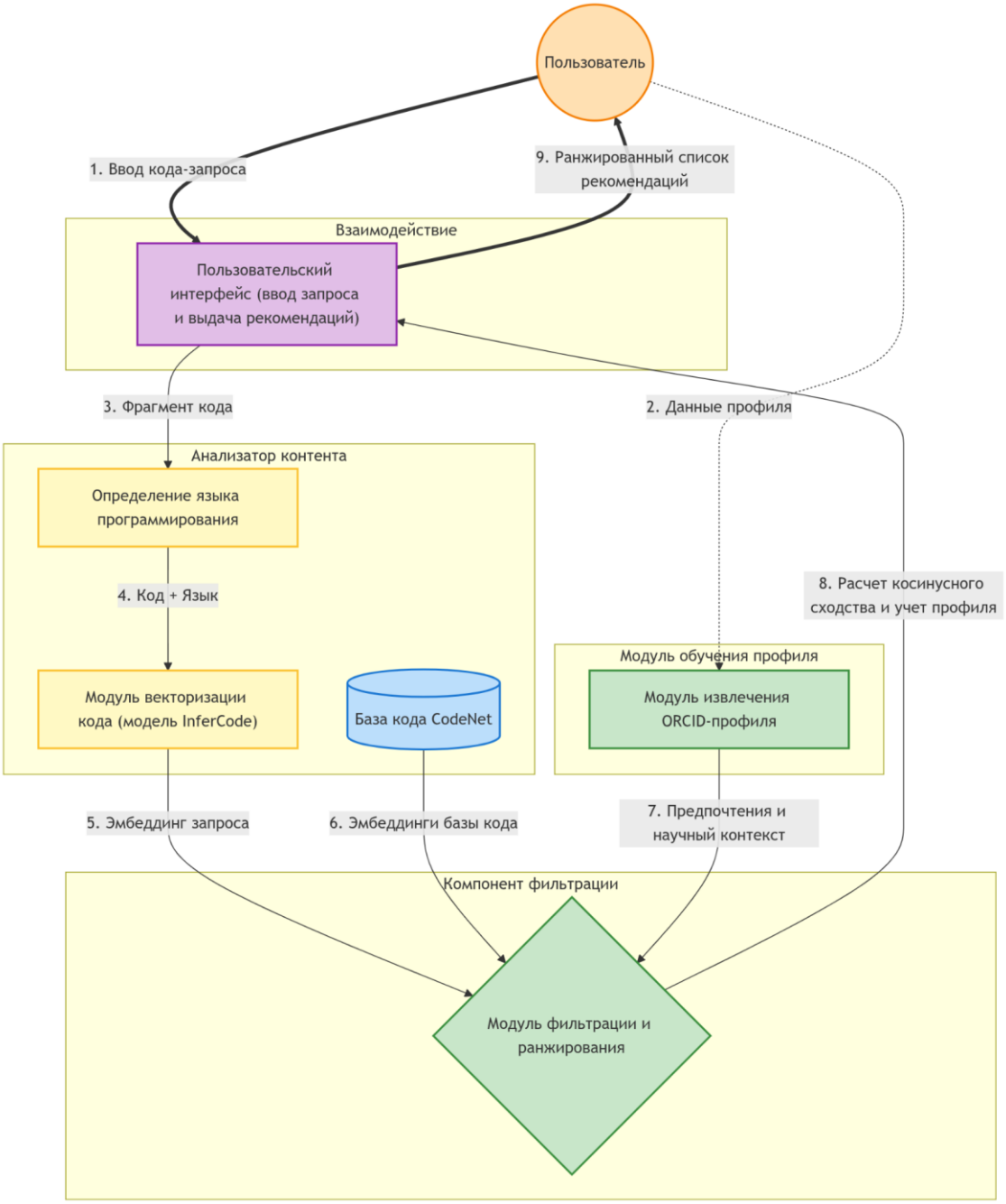

Рис. 3. Архитектура рекомендательной системы поиска близких фрагментов программного кода.

3.1. Анализатор контента

Анализатор контента отвечает за извлечение признаков из описаний объектов и создание структурированного представления, пригодного для машинной обработки. В рамках текущей версии системы этот процесс реализуется через построение абстрактных синтаксических деревьев и их последующую векто-

ризацию моделью InferCode. Для глубокого понимания семантики объектов используются методы дистрибутивной семантики (endogenous semantics), основанные на гипотезе о том, что слова (или элементы кода), встречающиеся в схожих контекстах, имеют схожие значения. Результатом работы таких моделей являются эмбеддинги – плотные векторы фиксированной размерности в многомерном пространстве, где расстояние между векторами служит мерой смысловой близости объектов. Базовой метрикой семантической релевантности между кодом-запросом c и фрагментом из базы данных r_i выступает косинусное сходство их эмбеддингов $s_i = \mathrm{sim}(c, r_i)$.

3.2. Модуль обучения профиля пользователя

Модуль обучения профиля собирает данные о предпочтениях пользователя, формируя модель его интересов на основе истории взаимодействий. В рамках этого модуля было принято допущение, по которому интересы пользователя имеют определенную стабильность, то есть ему предлагаются объекты, соответствующие технологическому стеку и предметной области его исследований. Профиль пользователя формируется на основе следующих двух факторов, относящихся к неявной и явной обратной связях соответственно.

1. Языковые предпочтения, т. е. технологический стек. Система автоматически фиксирует неявную обратную связь – факты взаимодействия с программным кодом на определенных языках программирования. Для каждого пользователя u и языка lang вычисляется нормализованная частота предпочтений $L_u(\mathrm{lang})$, показывающая долю взаимодействия с этим языком в общей истории пользователя.

2. Научный контекст, т. е. предметная область. Явная интеграция реализована через авторизацию по протоколу OAuth с использованием системы ORCID. При входе система извлекает из профиля исследователя набор ключевых слов, описывающих область его научных интересов. Это позволяет сформировать вектор интересов пользователя I_u для дополнительной контекстной фильтрации.

3.3. Компонент фильтрации и формирование рекомендаций

Главная задача компонента фильтрации состоит в преобразовании результатов базового алгоритма кросс-языкового поиска в персонализированную выдачу. Процесс формирования рекомендаций состоит из двух этапов.

На первом этапе производится фильтрация кандидатов. Из базы данных DB извлекается подмножество фрагментов кода $\{r_i\}$, для которых базовое семантическое сходство с запросом превышает установленный порог, т. е. $s_i \geq \theta$. Это гарантирует, что в рекомендации попадет только функционально релевантный код.

На втором этапе производится персонализированное ранжирование. Для каждого фрагмента-кандидата r_i вычисляется итоговый релевантный вес w_i, который определяет его позицию в итоговом списке рекомендаций. Вес рассчитывается как линейная комбинация базового контентного сходства и метрик профиля пользователя:

$$w_i = \alpha s_i + \beta L_u(\text{lang}(r_i)) + \gamma \text{Match}(I_u, \text{Metadata}(r_i)),$$

где s_i – семантическое сходство векторов кода-запроса и кандидата; $L_u(\text{lang}(r_i))$ – вес языкового предпочтения пользователя для языка, на котором написан кандидат r_i; $\text{Match}(I_u, \text{Metadata}(r_i))$ – функция оценки пересечения научных интересов пользователя (из ORCID) с метаданными или тегами контекста, привязанными к фрагменту кода в репозитории (например, принадлежность к математическим библиотекам, веб-разработке и т. д.); α, β, γ – настраиваемые гиперпараметры системы ($\alpha + \beta + \gamma = 1$), балансирующие вклады семантики и персонализации. Значения задаются эмпирически в конфигурации системы (по умолчанию наибольший вес α отдается семантическому сходству).

3.4. Пользовательский сценарий

Взаимодействие пользователя с системой происходит через веб-интерфейс. Пользователь вводит в систему фрагмент кода на любом из 19 поддерживаемых языков. Алгоритм автоматически определяет язык запроса, векторизует код и передает его в рекомендательное ядро.

В результате система возвращает пользователю список топ-N фрагментов кода, упорядоченный по убыванию веса w_i. Таким образом, на верхних позициях списка пользователь видит те фрагменты, которые не только максимально

точно реализуют исходный алгоритм, но и написаны на предпочтительных для него языках программирования и соответствуют профилю его исследований.

Применение контентного подхода обеспечивает системе ряд преимуществ: это независимость от данных других пользователей (рекомендации строятся только на основе интересов конкретного лица) и прозрачность, так как система может обосновать выбор результата наличием конкретных характеристик в коде [37].

4. ОЦЕНКА АЛГОРИТМА ФОРМИРОВАНИЯ РЕКОМЕНДАЦИЙ

Эффективность разработанного решения поиска сходных фрагментов программного кода и рекомендательной системы в целом оценивались с помощью серии вычислительных тестов. Осуществлялась проверка моделей векторизации на эталонном наборе данных, а также анализировалась точность разработанного вспомогательного алгоритма определения языка программирования.

4.1. Валидация моделей векторизации

Для вычисления показателей качества исследуемых моделей векторизации применялись набор данных и программная среда BigCloneEval [38], специально созданные для задачи оценки систем обнаружения программных клонов. В рамках валидации моделей метрика полноты (Recall) оценивалась для 8 различных типов клонов:

- тип-1 (Type-1) – точные копии (за исключением пробельных символов, форматирования и комментариев);
- тип-2 (Type-2) – объединение всех клонов 2-го типа;
- тип-2 несогласованный (Type-2 blind) – переименование переменных без сохранения соответствия;
- тип-2 согласованный (Type-2 consistent) – последовательное переименование переменных (один к одному);
- очень сильный тип-3 (Very-Strongly Type-3) – клоны с синтаксической схожестью в диапазоне [90, 100);
- сильный тип-3 (Strongly Type-3) – клоны с синтаксической схожестью в диапазоне [70, 90);
- умеренный тип-3 (Moderately Type-3) – клоны с синтаксической схожестью в диапазоне [50, 70);

- слабые тип-3/тип-4 (Weakly Type-3/Type-4) – клоны с синтаксической схожестью в диапазоне [0, 50).

В качестве критерия отсечения при подсчете метрик использовался статический порог косинусного сходства, равный 0.95. Результаты для базовых архитектур представлены в Табл. 1.

Табл. 1. Результаты валидации моделей для генерации эмбеддингов по программному коду.

Тип клона	CodeBERT	GraphCodeBERT	UniXcoder
Type-1	1.0000	0.9438	1.0000
Type-2	0.9495	0.5889	0.9495
Type-2 (blind)	0.9474	0.6132	0.9474
Type-2 (consistent)	0.9497	0.5867	0.9498
Very-Strongly Type-3	0.9596	0.6482	0.9596
Strongly Type-3	0.9507	0.3670	0.9508
Moderately Type-3	0.9901	0.0919	0.9904
Weakly Type-3 or Type-4	0.9997	0.0360	0.9998

Из таблицы видно, что модели UniXcoder и CodeBERT показывают сопоставимую и наиболее высокую эффективность. При этом UniXcoder демонстрирует незначительное преимущество на согласованном подтипе типа-2 и слабоструктурированных клонах сильного типа-3, умеренного типа-3 и слабых типах-3/4.

4.2. Валидация алгоритма определения языка программирования

Валидация алгоритма определения языка программирования производилась с использованием набора данных, описанного в разделе 2.2. Результаты сравнения предложенного алгоритма по ключевым словам с существующими аналогами представлены в Табл. 2. Разработанный нами алгоритм показал лучшие точность и скорость выполнения среди представленных решений при приемлемом значении полноты.

Табл. 2. Результаты валидации алгоритмов определения языка программирования.

Алгоритм	Accuracy	Precision	Recall	F1 score	Среднее время определения, мс
По ключевым словам	0.71	0.88	0.71	0.71	0.475
Guesslang	0.84	0.86	0.84	0.85	5.576
Pygments	0.03	0.19	0.03	0.03	20.724

ЗАКЛЮЧЕНИЕ

Предложена контентная рекомендательная система, использующая семантический анализ программного кода для кросс-языкового поиска семантически близких фрагментов в пространстве кодов. Разработанное решение акцентирует внимание на функциональной эквивалентности алгоритмов, позволяя находить смысловые аналоги на 19 различных языках программирования.

Основой анализатора контента выступает метод представления исходного кода в виде абстрактных синтаксических деревьев с последующей генерацией эмбеддингов на базе модели InferCode. Для обеспечения корректной работы конвейера обработки данных разработан алгоритм автоматической идентификации языка программирования по входному фрагменту, а также сформирован специализированный мультиязычный датасет на основе корпуса CodeNet.

В рамках рекомендательной системы был реализован модуль персонализации. Итоговое ранжирование рекомендаций осуществляется с учетом индивидуального профиля исследователя: система динамически взвешивает семантическое сходство кода с историей языковых предпочтений пользователя и областью его научных интересов, автоматически извлекаемых через авторизацию по профилю ORCID.

Так, предложенный подход, сочетающий векторизацию на основе синтаксических деревьев и метрическую оценку подобия, позволяет эффективно решать задачи интеллектуальной навигации в современных цифровых научных библиотеках. В перспективе разработанная рекомендательная система будет

интегрирована в исследовательскую инфраструктуру цифровой математической библиотеки Lobachevskii-DML [39].

Благодарности

Выражаем благодарность Наталии Павловне Тучковой за проявленный интерес к исследованию, значимые замечания и советы при оформлении статьи.

СПИСОК ЛИТЕРАТУРЫ

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

2. *Vislavski T., Rakić G., Cardozo N., 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. P. 512–516. https://doi.org/10.1109/SANER.2018.8330250

3. *Nafi K.W., Kar T.S., Roy B., Roy C.K., 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, USA, 2019. P. 1026–1037. https://doi.org/10.1109/ASE.2019.00099

4. *Mathew G., Stolee K.T.* Cross-language code search using static and dynamic analyses // Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, New York, USA, 2021. P. 205–217. https://doi.org/10.1145/3468264.3468538

5. *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. P. 413–424. https://doi.org/10.1145/3524610.3527911

6. *Saieva A., Chakraborty S., Kaiser G.* Reinforest: Reinforcing Semantic Code Similarity for Cross-Lingual Code Search Models // 2024 IEEE International Conference on Source Code Analysis and Manipulation (SCAM). 2023. P. 177–188. https://doi.org/10.1109/SCAM63643.2024.00026

7. *Ricci F., Rokach L., Shapira B.* (Eds.) Recommender Systems Handbook. Springer New York, N.Y., 2022. 1060 p. https://doi.org/10.1007/978-1-0716-2197-4

8. *de Gemmis M., Lops P., Musto C., Narducci F., Semeraro G.* Semantics-

Aware Content-Based Recommender Systems // In: Ricci F., Rokach L., Shapira B. (Eds.) Recommender Systems Handbook. Springer, Boston, MA, 2015. P. 119–159. https://doi.org/10.1007/978-1-4899-7637-6_4

9. *Фальк К.* Рекомендательные системы на практике: практическое руководство. М.: ДМК Пресс, 2020. 256 с.

10. *Manouselis N., Drachsler H., Verbert K., Duval E.* Recommender Systems for Learning. Springer, 2013. https://doi.org/10.1007/978-1-4614-4361-2

11. *Elizarov A.M., Lipachev E.K., Zhizhchenko A.B., Zhil'tsov N.G., Kirillovich A.V.* Mathematical Knowledge Ontologies and Recommender Systems for Collections of Documents in Physics and Mathematics // Doklady Mathematics. 2016. Vol. 93, No. 2. P. 231–233. https://doi.org/10.1134/S1064562416020174

12. *Елизаров А.М., Липачев Е.К., Хайдаров Ш.М.* Метод автоматизированного подбора рецензентов научных статей, реализованный в информационной системе научного журнала // Научный сервис в сети Интернет: труды XXI Всерос. науч. конф. (23–28 сен. 2019, Новороссийск). М.: ИПМ им. М.В. Келдыша, 2019. С. 318–328. https://doi.org/10.20948/abrau-2019-94.
URL: http://keldysh.ru/abrau/2019/theses/94.pdf (дата доступа: 14.03.2026)

13. *Елизаров А.М., Липачев Е.К., Хайдаров Ш.М.* Рекомендательная система поиска экспертов для проведения научного рецензирования в математическом журнале // Электронные библиотеки. 2020. Т. 23, № 4. С. 708–732. https://doi.org/10.26907/1562-5419-2020-23-4-708-732

14. *Smyth B.* Case-based recommendation // In: Brusilovsky A., Kobsa W. (Eds.). The Adaptive Web: Methods and Strategies of Web Personalization, Lecture Notes in Computer Science, Springer, Berlin, 2007. P. 342–376.

15. *Атаева О.М., Тучкова Н.П., Дегтев А.Г.* Рекомендательная система на основе обобщенного указателя журналов // Онтология проектирования. 2025. Т. 15, № 4. С. 598–613. https://doi.org/10.18287/2223-9537-2025-15-4-598-613

16. The MaRDI consortium. MaRDI: Mathematical Research Data Initiative Proposal. 2022. https://doi.org/10.5281/zenodo.6552436

17. *Kalinin N.A., Skvortsov N.A.* Difficulties of FAIR Principles Implementation in Cross-Domain Research Infrastructures // Lobachevskii J. Math. 2023. Vol. 44, No. 1. P. 147–156. https://doi.org/10.1134/S199508022301016X

18. *Mathew. G, Parnin C., Stolee K.T.* SLACC: simion-based language agnostic

code clones // Proc. of the ACM/IEEE 42nd International Conference on Software Engineering. 2020. P. 210–221. https://doi.org/10.1145/3377811.3380407

19. *Li J., Tao C., Jin Z., Liu F., Li J., Li G.* ZC^3: Zero-Shot Cross-Language Code Clone Detection // 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2023. P. 875–887. https://doi.org/10.1109/ASE56229.2023.00210

20. *Hu M., Yang J., Zhou W.* Cross-language code clone detection via flow-enhanced graph attention network // The Computer Journal. 2026. https://doi.org/10.1093/comjnl/bxaf146

21. *Петров В.В.* Система автоматизации численной оценки сходства Android-приложений // Электронные библиотеки. 2024. Т. 27, № 3. С. 336–365. https://doi.org/10.26907/1562-5419-2024-27-3-336-365

22. *Petrov V.V.* Automated system for numerical similarity evaluation of android applications // Automatic documentation and mathematical linguistics. 2024. Vol. 58, No. 3. P. 131–142. https://doi.org/10.3103/S0005105525700207

23. *Riesen K.* Structural Pattern Recognition with Graph Edit Distance. Springer, Cham, 2015. 158 p. https://doi.org/10.1007/978-3-319-27252-8

24. The Top Programming Languages 2025 // IEEE Spectrum. 2025. URL: https://spectrum.ieee.org/top-programming-languages-2025 (дата доступа: 14.03.2026)

25. Топ языков программирования в 2025 году: рейтинг IEEE и влияние на него языковых моделей // Хабр-блог, 2025. URL: https://habr.com/ru/companies/selectel/articles/951348 (дата доступа: 14.03.2026)

26. *Зорин В.И., Липачев Е.К.* Метод вычисления меры сходства фрагментов программного кода // Системы высокой доступности. 2026. Т. 22, № 1. С. 47−50. https://doi.org/ 10.18127/j20729472-202601-09

27. *Euzenat J., Shvaiko P.* Basic similarity measures // In: Ontology Matching. Springer, Berlin, Heidelberg, 2013. P. 85–120. https://doi.org/10.1007/978-3-642-38721-0_5

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

29. *Зорин В.И.* Programming Language Detection Dataset (1.0.0). https://doi.org/10.5281/zenodo.15661548

30. *Ахо А.В., Сети Р. Ульман Дж.Д.* Компиляторы: принципы, технологии и инструменты. М.: Вильямс, 2003. 768 с.

31. *Городняя Л.В.* Формы для показа результатов сравнения языков программирования на примере диалектов языка LISP // Электронные библиотеки. 2026. Т. 29 (1). С. 24–59. https://doi.org/10.26907/1562-5419-2026-29-1-24-59

32. *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 // Empirical Methods in Natural Language Processing. 2020. P. 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

33. *Alon U., Zilberstein M., Levy O., Yahav E.* code2vec: learning distributed representations of code // Proc. of the ACM on Programming Languages. 2019. Vol. 3, No. POPL. P. 1–29. https://doi.org/10.1145/3290353

34. *Bui N.D.Q., Yu Y., Jiang L.* InferCode: Self-Supervised Learning of Code Representations by Predicting Subtrees // 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). 2020. P. 1186–1197. https://doi.org/10.1109/ICSE43902.2021.00109

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

36. *Guo D., Lu S., Duan N., Wang Y., Zhou M., Yin J.* UniXcoder: Unified Cross-Modal Pre-training for Code Representation // Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers), Dublin, 2022. P. 7212–7225. https://doi.org/10.18653/v1/2022.acl-long.499

37. *Musto C., Gemmis M.d.F., Lops P., Narducci F., Semeraro G.* Semantics and Content-Based Recommendations // F. Ricci, L. Rokach, B. Shapira (Eds.) Recommender Systems Handbook. Springer New York, N.Y., 2022. P. 251–298. https://doi.org/10.1007/978-1-0716-2197-4_7

38. *Svajlenko J., Roy C.K.* BigCloneEval: A Clone Detection Tool Evaluation Framework with BigCloneBench // 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2016. P. 596–600. https://doi.org/10.1109/ICSME.2016.62

39. *Елизаров А.М., Кириллович А.В., Липачев Е.К., Невзорова О.А.* Цифровая экосистема OntoMath как подход к построению пространства математических знаний // Электронные библиотеки. 2023. Т. 26. № 2. С. 154–202. https://doi.org/10.26907/1562-5419-2023-26-2-154-202

A RECOMMENDATION SYSTEM FOR FINDING SEMANTICALLY SIMILAR FRAGMENTS OF PROGRAM CODE

V. I. Zorin[1] [0009-0004-0271-1882], E. K. Lipachev[2] [0000-0001-7789-2332]

[1]*Kazan National Research Technical University named after A. N. Tupolev — KAI, Kazan, Russia*

[2]*Kazan Federal University, Kazan, Russia*

[2]*Innopolis University, Innopolis, Russia*

[1]addefan@mail.ru, [2]elipachev@gmail.com

Abstract

Recommendation systems in the scientific information space serve as essential tools for search and navigation when working with scientific documents. Software code is currently considered as an object of scientific knowledge and, as a result, an important task is to create software lifecycle support systems, in particular, to find similar software solutions, detect code borrowings, analyze and evaluate code quality.

This paper proposes a content-based recommender system that provides users with a personalized list of code fragments that are functionally equivalent to the input query code presented in one of the programming languages from the established set.

The basic algorithm of the system is based on the representation of the program code in the form of an abstract syntax tree followed by the construction of a vector space of program codes. The semantic similarity of program codes is determined by the distance between code vectors in a multidimensional space.

The personalization of recommendations is achieved through a filtering module that ranks the retrieved fragments taking into account the user's profile. The factors under consideration are the language preferences of the user and his areas of scientific interests, extracted through integration with ORCID.

To ensure the system's operation, a specialized dataset was created based on

the CodeNet corpus. The problem of automated language detection from a snippet of the presented code in one of the 19 languages included in the current rating list of programming languages has also been solved.

Keywords*: abstract syntax tree, code embedding, content-based filtering, cross-language clone, cross-language code search, code similarity, recommender system.*

REFERENCES

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

2. *Vislavski T., Rakić G., Cardozo N., 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. P. 512–516. https://doi.org/10.1109/SANER.2018.8330250

3. *Nafi K.W., Kar T.S., Roy B., Roy C.K., 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, USA, 2019. P. 1026–1037. https://doi.org/10.1109/ASE.2019.00099

4. *Mathew G., Stolee K.T.* Cross-language code search using static and dynamic analyses // Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, New York, USA, 2021. P. 205–217. https://doi.org/10.1145/3468264.3468538

5. *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. P. 413–424. https://doi.org/10.1145/3524610.3527911

6. *Saieva A., Chakraborty S., Kaiser G.* Reinforest: Reinforcing Semantic Code Similarity for Cross-Lingual Code Search Models // 2024 IEEE International Conference on Source Code Analysis and Manipulation (SCAM). 2023. P. 177–188. https://doi.org/10.1109/SCAM63643.2024.00026

7. *Ricci F., Rokach L., Shapira B.* (Eds.) Recommender Systems Handbook. Springer New York, N.Y., 2022. 1060 p. https://doi.org/10.1007/978-1-0716-2197-4

8. *de Gemmis M., Lops P., Musto C., Narducci F., Semeraro G.* Semantics-Aware Content-Based Recommender Systems // In: Ricci F., Rokach L., Shapira B.

(Eds.) Recommender Systems Handbook. Springer, Boston, MA, 2015. P. 119–159. https://doi.org/10.1007/978-1-4899-7637-6_4

9. *Falk K.* Recommender Systems in Practice: A Practical Guide. Springer, Berlin, 2016. 340 p.

10. *Manouselis N., Drachsler H., Verbert K., Duval E.* Recommender Systems for Learning. Springer, 2013. https://doi.org/10.1007/978-1-4614-4361-2

11. *Elizarov A.M., Lipachev E.K., Zhizhchenko A.B., Zhil'tsov N.G., Kirillovich A.V.* Mathematical Knowledge Ontologies and Recommender Systems for Collections of Documents in Physics and Mathematics // Doklady Mathematics. 2016. Vol. 93, No. 2. P. 231–233. https://doi.org/10.1134/S1064562416020174

12. *Elizarov A.M., Lipachev E.K., Khaydarov S.M.* Method of automated selection of reviewers of scientific articles, implemented in the scientific journal information system // Nauchny`j servis v seti Internet. M: IPM im. Keldysha, 2019. P. 318–328. https://doi.org/10.20948/abrau-2019-94

13. *Elizarov A.M., Lipachev E.K., Khaydarov S.M.* Recommender system in the process of scientific peer review in mathematical journal // Russian Digital Libraries Journal. 2020. Vol. 23, No. 4. P. 708–732. https://doi.org/10.26907/1562-5419-2020-23-4-708-732

14. *Smyth B.* Case-based recommendation // In: Brusilovsky A., Kobsa W. (Eds). The Adaptive Web: Methods and Strategies of Web Personalization, Lecture Notes in Computer Science, Springer, Berlin, 2007. P. 342–376.

15. *Ataeva O.M., Tuchkova N.P., Degtev A.G.* Recommendation system based on a generalized journal index // Ontology of Designing. 2025. Vol. 15, No. 4. P. 598–613. https://doi.org/10.18287/2223-9537-2025-15-4-598-613

16. The MaRDI consortium. MaRDI: Mathematical Research Data Initiative Proposal. 2022. https://doi.org/10.5281/zenodo.6552436

17. *Kalinin N.A., Skvortsov N.A.* Difficulties of FAIR Principles Implementation in Cross-Domain Research Infrastructures // Lobachevskii J. Math. 2023. Vol. 44, No. 1. P. 147–156. https://doi.org/10.1134/S199508022301016X

18. *Mathew. G, Parnin C., Stolee K.T.* SLACC: simion-based language agnostic code clones // Proc. of the ACM/IEEE 42nd International Conference on Software Engineering. 2020. P. 210–221. https://doi.org/10.1145/3377811.3380407

19. *Li J., Tao C., Jin Z., Liu F., Li J., Li G.* ZC3: Zero-Shot Cross-Language Code

Clone Detection // 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2023. P. 875–887.
https://doi.org/10.1109/ASE56229.2023.00210

20. *Hu M., Yang J., Zhou W.* Cross-language code clone detection via flow-enhanced graph attention network // The Computer Journal. 2026.
https://doi.org/10.1093/comjnl/bxaf146

21. *Petrov V.V.* Automated system for numerical similarity evaluation of android applications // Russian Digital Libraries Journal. 2024. Vol. 27, No. 3. P. 336–365. https://doi.org/10.26907/1562-5419-2024-27-3-336-365

22. *Petrov V.V.* Automated system for numerical similarity evaluation of android applications // Automatic documentation and mathematical linguistics. 2024. Vol. 58, No. 3. P. 131–142. https://doi.org/10.3103/S0005105525700207

23. *Riesen K.* Structural Pattern Recognition with Graph Edit Distance. Springer, Cham, 2015. 158 p. https://doi.org/10.1007/978-3-319-27252-8

24. The Top Programming Languages 2025. 2025.
URL: https://spectrum.ieee.org/top-programming-languages-2025 (Accessed: 22.03.2026).

25. Top yazy`kov programmirovaniya v 2025 godu: rejting IEEE i vliyanie na nego yazy`kovy`x modelej // Habr-blog, 2025. URL: https://habr.com/ru/companies/selectel/articles/951348 (Accessed: 22.03.2026).

26. *Zorin V.I., Lipachev E.K.* A method for calculating the similarity measure of program code fragments // Highly Available Systems. 2026. Vol. 22, No. 1. P. 47−50. https://doi.org/10.18127/j20729472-202601-09

27. *Euzenat J., Shvaiko P.* Basic similarity measures // In: Ontology Matching. Springer, Berlin, Heidelberg, 2013. P. 85–120.
https://doi.org/10.1007/978-3-642-38721-0_5

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

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

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

31. *Gorodnyaya L.V.* Forms for displaying the results of comparison of programming languages using the example of dialects of the LISP language // Russian Digital Libraries Journal. 2026. Vol. 29, No. 1. P. 24–59. https://doi.org/10.26907/1562-5419-2026-29-1-24-59

32. *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 // Empirical Methods in Natural Language Processing. 2020. P. 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

33. *Alon U., Zilberstein M., Levy O., Yahav E.* code2vec: learning distributed representations of code // Proc. of the ACM on Programming Languages. 2019. Vol. 3, No. POPL. P. 1–29. https://doi.org/10.1145/3290353

34. *Bui N.D.Q., Yu Y., Jiang L.* InferCode: Self-Supervised Learning of Code Representations by Predicting Subtrees // 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). 2020. P. 1186–1197. https://doi.org/10.1109/ICSE43902.2021.00109

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

36. *Guo D., Lu S., Duan N., Wang Y., Zhou M., Yin J.* UniXcoder: Unified Cross-Modal Pre-training for Code Representation // Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Dublin, 2022. P. 7212–7225. https://doi.org/10.18653/v1/2022.acl-long.499

37. *Musto C., Gemmis M.d., Lops P., Narducci F., Semeraro G.* Semantics and Content-Based Recommendations // F. Ricci, L. Rokach, B. Shapira (Eds.) Recommender Systems Handbook. Springer New York, N.Y., 2022. P. 251–298. https://doi.org/10.1007/978-1-0716-2197-4_7

38. *Svajlenko J., Roy C.K.* BigCloneEval: A Clone Detection Tool Evaluation Framework with BigCloneBench // ICSME, 2016. P. 596–600. https://doi.org/10.1109/ICSME.2016.62

39. *Elizarov A.M., Kirillovich A.V., Lipachev E.K., Nevzorova O.A.* Digital Ecosystem OntoMath as an Approach to Building the Space of Mathematical Knowledge // Russian Digital Libraries Journal. 2023. Vol. 26, No. 2. P. 154–202. https://doi.org/10.26907/1562-5419-2023-26-2-154-202

СВЕДЕНИЯ ОБ АВТОРАХ

ЗОРИН Виталий Иванович – магистрант Института компьютерных технологий и защиты информации Казанского национального исследовательского технического университета им. А.Н. Туполева – КАИ. Научные интересы: программная инженерия, обработка естественного языка, рекомендательные системы.

Vitaly Ivanovich ZORIN – graduate student at the Institute of Computer Technology and Information Security of Kazan National Research Technical University named after A.N. Tupolev – KAI. Research interests: software engineering, natural language processing, recommendation systems.

email: addefan@mail.ru

ORCID: 0009-0004-0271-1882

ЛИПАЧЕВ Евгений Константинович – кандидат физико-математических наук, доцент кафедры цифровой аналитики и технологий искусственного интеллекта Института информационных технологий и интеллектуальных систем Казанского федерального университета, доцент Университета Иннополис. Научные интересы: цифровые библиотеки, интеллектуальный анализ данных, рекомендательные системы, технологии извлечения знаний.

Evgeny Konstantinovich LIPACHEV – Candidate of Physics and Mathematics, Associate Professor, Kazan Federal University, Innopolis University. Research interests: digital libraries, data mining, recommender systems, knowledge extraction technologies.

email: elipachev@gmail.com

ORCID: 0000-0001-7789-2332

Материал поступил в редакцию 14 марта 2025 года