Математическое, алгоритмическое и программное обеспечение

DOI 10.66032/2221-2574-2026-1-2-58-66 УДК 621.372

ГЕНЕРАЦИЯ КОНВЕЙЕРОВ JAVA STREAM API: СРАВНЕНИЕ СПЕЦИАЛИЗИРОВАННОГО АЛГОРИТМА И ТРЁХ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ

Бородин Даниил Вадимович
аспирант, ФГБОУ ВО «Липецкий государственный педагогический университет имени П.П. Семёнова-Тян-Шанского»[1].

Пруцков Александр Викторович
доктор технических наук, профессор кафедры вычислительной и прикладной математики, ФГБОУ ВО «Рязанский государственный радиотехнический университет имени В.Ф. Уткина»[2]; профессор кафедры информатики, информационных технологий и защиты информации, ФГБОУ ВО «Липецкий государственный педагогический университет имени П.П. Семёнова-Тян-Шанского»[1].

[1]*Адрес:* 398020, Российская Федерация, г. Липецк, ул. Ленина, д. 42.
[2]*Адрес:* 390005, Российская Федерация, г. Рязань, ул. Гагарина, д. 59/1.

E-mail для связи: dvoranchik@yandex.ru

Аннотация: Автоматизация проектирования, тестирования и документирования программных конвейеров актуальна для задач программной инженерии и ускорения разработки программного обеспечения. Разработан метод генерации конвейеров. Метод состоит из алгоритмов поиска конвейера, генерации программы, теста и документации. Алгоритм поиска конвейера разработан на основе модифицированного алгоритма поиска в глубину и находит конвейеры минимальной длины. Модификация состояла в итеративном ограничении глубины поиска. Алгоритм поиска конвейера функций преобразований целочисленных массивов *F* Java Stream API сравнивался с большими языковыми моделями (БЯМ) Qwen2.5-Max, DeepSeek и OpenAI ChatGPT4o. План исследования включал формирование множества функций *F*, генерацию тестовых наборов, применение разработанного алгоритма поиска конвейеров и больших языковых моделей к тестовым наборам. Большие языковые модели находили конвейер без указания и с указанием функций множества *F*. Разработанный алгоритм, несмотря на большее время работы при длине конвейера $k = 5$, имеет более высокую точность, чем большие языковые модели без и с указанием функций множества *F*. Средняя длина конвейера, найденного алгоритмом короче, чем у больших языковых моделей без указания функций множества *F* на не менее 12% и чем у больших языковых моделей с указанием функций множества *F* на не менее 7%. Эти результаты подтверждают работоспособность разработанного метода для автоматизации программирования конвейеров. Дальнейшие исследования метода генерации конвейеров состоят в его ускорении за счёт разработки его многопоточной версии и использования алгоритмов поиска ассоциативных правил.

Ключевые слова: автоматизация программирования, генерация программ, генерация конвейеров, Java Stream API, большие языковые модели, поиск в глубину, тестирование, генерация документации.

1. Введение

1.1. Генерация программ и её типы

Автоматическая генерация программ является одной из задач современной программной инженерии и активно развивается благодаря применению различных подходов, которые включают как алгоритмические методы, так и использование больших языковых моделей (БЯМ, англ. LLM — Large Language Models), генерирующих программу по текстовому запросу.

Методы генерации программ подразделяются на пять типов [1].

I. Шаблонные методы — это подход к генерации программного кода, основанный на использовании заранее заданных шаблонов.

II. Генерация кода на основе моделей — метод, заключающийся в преобразовании формальных спецификаций в исполняемый код.

III. Методы поиска и эволюционные алгоритмы — подходы к автоматической разработке, рассматриваемые как задачи поиска оптимального решения в пространстве возможных программ.

IV. Композиционное программирование — основано на создании новых программ путём компоновки уже существующих компонентов или модулей.

V. Методы на основе искусственного интеллекта и машинного обучения — современные подходы к генерации программного кода, основанные на применении БЯМ, нейронных сетей.

В дальнейшем под генерацией программ понимается процедура создания программы, выдающей по одному из наборов входных данных соответствующий набор выходных данных, то есть обладающей свойством массовости.

1.2. Большие языковые модели и их использование для генерации программ

В последние годы БЯМ активно применяются для автоматизации различных задач программной инже-

нерии [2], включая генерацию кода [3], автоматическое тестирование [4] и создание тестовых данных [5].

Современные модели OpenAI Codex, Qwen2.5-Max, DeepSeek и ChatGPT4o генерируют по текстовому описанию задачи или по примеру кода тесты, выявляют ошибки и предлагают исправления [6].

1.3. Математическая модель конвейера функций как объекта для генерации

Пусть $F = \{f_1, f_2, ..., f_{NF}\}$ – множество функций. Функции множества F обладают следующими свойствами:

1. являются одноместными;

2. $\forall f\left(\left(f \in F\right) \wedge \left(y = f\left(x\right)\right) \wedge \left(x \in T\right) \wedge \left(y \in T\right)\right)$,

где T — некоторое множество объектов.

Даны x и y. Необходимо найти последовательность функций такую, что: $f^{(1)}\left(f^{(2)}\left(...f^{(n)}\left(x\right)\right)\right) = y$, где $f^{(i)} \in F$, $i \in 1,2,...,n$.

Назовём последовательность функций $f^{(1)}\left(f^{(2)}\left(...f^{(n)}\left(x\right)\right)\right) = y$ конвейером (pipeline). Длина конвейера равна количеству функций в нем. Конвейером относительно F назовём конвейер, в котором использованы только функции множества F.

1.4. Конвейеры в программировании, Java Stream API

Конвейеры являются реализацией потоков обработки данных (streams) функционального программирования. Эти потоки широко используются в современном объектно-ориентированном программировании, например в языках программирования Scala, JavaScript, C#, F#, Java [7].

Конкретизируем модель конвейера функций на примере Java Stream API. В конвейерах Java Stream API используются функции фильтрации (filter), отображения (map), сортировки (sorted), агрегации (reduce) и сборки результатов (collect) [8]. Например, для нахождения первичного терма в сегментах кластера информационно-поисковой системы Elasticsearch в классе org.elasticsearch.cluster.metadata.MetadataCreateIndexService с помощью конвейера Java Stream API (листинг 1) генерируется последовательность чисел, равная количеству сегментов (range), извлекаются первичные термы каждого сегмента (mapToLong), отыскивается максимальное значение (max) и преобразуется к типу Long (getAsLong).

Листинг 1. Пример использования Java Stream API Stream API

```
long primaryTerm =
  IntStream.range(0,
      sourceMetadata.getNumberOfShards())
    .mapToLong(
      sourceMetadata::primaryTerm)
    .max()
                .getAsLong();
```

В [9] предложена модель Java Stream API. В отличие от модели конвейера функций эта модель является узкоспециализированной и предназначена для преобразования операторов цикла в конвейеры. Конвейеры используются в не менее, чем 9 % проектов на языке Java [10], а по оценке в [11] — в 21 % проектов.

Конвейеры возможно генерировать. Генерация конвейеров (как и других элементов программы) имеет следующие преимущества:

- сокращает себестоимость и время разработки за счёт автоматизации;
- снижает количество ошибок в программе за счёт устранения человеческого фактора;
- может стать частью платформы разработки с минимальным программированием (low-code development) [12].

Генерация конвейеров Java Stream API исследовалась в [13]. Система S2S (SQL to Stream) автоматически трансформирует запросы на языке SQL в конвейеры Java Stream API для создания тестовых примеров. Особенностью этой системы является то, что генерация тестов возможна только для операторов языка SQL, но не любых преобразований данных. Кроме того, система не поддерживает создание тестов для пользовательских преобразований структуры или содержимого массива по некоторым правилам.

2. Цель работы

Целью работы является автоматизация разработки генерации конвейеров в программах.

Для достижения цели необходимо решить следующие задачи:

1. разработать метод генерации конвейеров;
2. разработать алгоритм поиска конвейера как части этого метода;
3. экспериментально измерить его скорость работы и длину генерируемых конвейеров;
4. выполнить задачу 3 для БЯМ и для поиска конвейера полным перебором;
5. проанализировать полученные измерения для разработанного алгоритма, поиска полным перебором и БЯМ.

3. Метод генерации конвейеров

3.1. Состав метода генерации конвейеров

Предлагается метод генерации конвейеров, включающий следующие алгоритмы:

- поиска конвейера;
- генерации программы;
- генерации теста;
- генерации документации.

Входными данными метода являются исходное и результирующее значения. Результатом метода являются программа, модульный тест и документация к программе.

Задача поиска конвейера — это подзадача генерации программы. Для автоматизации программирования необходимо также преобразовать конвейер в

текст программы, сгенерировать для программы модульный тест и документацию. В связи с этим разработаны алгоритмы генерации программы, теста и документации.

3.2. Алгоритм поиска конвейера

Входными данными алгоритма поиска конвейера являются исходное и результирующее значения. Значением на выходе является конвейер.

Формализуем предметную область с помощью теории графов. Пусть дан орграф: $G = G(V, E)$, в котором каждой дуге $e \in E$ приписана функция $f \in F$, а вершиной $v \in V$ является значение этой функции.

При переходе по дуге к значению родительской вершины v_p применяется функция f, приписанная к дуге. Результат применения функции является значением дочерней вершины $v_s : v_s = f(v_p)$.

У каждой вершины, кроме терминальной, $|F|$ дочерних вершин. Циклы в орграфе G отсутствуют. Орграф G является деревом.

Поиск конвейера сводится к поиску в дереве G пути, такого, что $f^{(1)}\left(f^{(2)}\left(\ldots f^{(n)}(v_0)\right)\right) = v_t$, где $v_0 = x$ — корневая вершина G, а $v_t = y$ – некорневая вершина G.

На этом этапе исследования были выбраны простейшие алгоритмы поиска: в глубину и в ширину. Алгоритм поиска в ширину был отвергнут из-за невозможности его практической реализации. При $|F| = 20$ и глубине поиска, равной 3, данные о посещённых вершинах переполняли доступный объем оперативного запоминающего устройства компьютера, используемого в исследовании. Конвейер отыскивается модифицированным алгоритмом поиска в глубину с пометкой посещённых вершин дерева G следующим образом:

0. Начало алгоритма поиска конвейера.

Текущая вершина $v = v_0 = x$, путь (конвейер) $p = <>$, где $<>$ – вектор без элементов, P – множество результатов.

Для каждой глубины d от 1 до D выполнить шаги 3–9.

Множество посещённых вершин $V_p = \varnothing$.

Если $v = y$, то $P = P \cup \{p\}$.

Пометить v как посещённую.

Если у v есть дочерние непосещённые вершины, то взять $v' = f'(v)$, где v' — первая непосещённая вершина; $p = p + f'$, где «+» — операция добавления элемента в конец вектора, f' — функция, приписанная к дуге, ведущей в v'; $v = v'$; перейти к шагу 4.

Если у v есть родительская вершина v_m, то $p = p -$, где «–» — унарная операция исключения последнего элемента из вектора, $v = v_m$, перейти к шагу 6.

Значит $v = v_0$.

Если $P = \varnothing$, то найти в множестве P конвейеры минимальной длины и вернуть их как результат; перейти к шагу 11, иначе перейти к шагу 2.

Вернуть результат — конвейер не найден.

Конец алгоритма поиска конвейера.

Вершины и дуги дерева G генерируются по мере необходимости. В отличие от классического поиска в глубину в предложенном алгоритме используется итеративное ограничение глубины (Iterative Deepening Depth-First Search, IDDFS) [14], при котором поиск выполняется при возрастающем значении глубины d, что позволяет находить путь минимальной длины.

3.3. Модель шаблона генерации программы, теста или документации

Алгоритмы генерации программы, теста и документации к программе являются шаблонными методами в классификации методов генерации программ (раздел 1.1). В основе этих алгоритмов лежит функция: $y = h(x_1, x_2, \ldots, x_{NH})$.

Значением функции h является программа, тест или документация соответственно.

Аргументами функции h являются исходное и результирующее значение, найденный конвейер, а также производные данные:

- представление конвейера на языке программирования для программы и теста;
- словесные описания функций для документации.

Функция h представляет собой выражение (шаблон) и формирует значение подстановкой в шаблон аргументов.

3.4. Алгоритмы генерации программы, теста или документации

Входными данными алгоритмов являются исходное и результирующее значение, найденный конвейер.

Данные алгоритмы имеют одинаковую структуру:

1. включить входные данные в список аргументов функции *h*;
2. преобразовать входные данные в производные данные и включить их в список аргументов функции *h*;
3. подставить аргументы в функцию *h*;
4. считать значение функции *h* результатом работы алгоритма.

Алгоритмы отличаются преобразованиями входных данных и видом функции *h*.

3.5. Конкретизация метода генерации конвейеров

Предлагаемый метод является общим. Для исследо-

вания метода необходимо его конкретизировать:

тип исходного и результирующего значений — массив с целочисленными элементами;

язык программирования, для которого генерируются программа, модульный тест и документация — язык Java;

функции множества F – функции Java Stream API, применяемые поэлементно.

Для подтверждения результативности и оценки возможностей разработанного метода необходимо провести исследование и экспериментальную проверку на тестовых наборах. Чтобы показать применимость алгоритма, сравним его с быстро развивающимися и показывающими хорошие результаты в области генерации программ БЯМ.

3.6. Практический пример работы метода генерации конвейеров

Пусть даны исходный массив $x = \{1,2,3\}$ и результирующий массив $y = \{3,5,7\}$. Алгоритм поиска конвейера, применяя итеративный поиск в глубину, последовательно проверяет функции множества F.

На глубине $d = 1$ алгоритм проверяет все функции, но ни одна из них не дает результирующий массив y. На глубине $d = 2$ он обнаруживает композицию двух функций: сначала выполняется умножение на 2 (doubleItems), затем прибавление 1 (addOne). Массив $\{1,2,3\}$ преобразуется в $\{3,5,7\}$. Найденный алгоритмом конвейер хранится как последовательность индексов функций из F. В нашем случае это массив $\{0,2\}$, где 2 соответствует функции doubleItems, а 0 – функции addOne.

Функции doubleItems и addOne преобразуется в конвейер Java Stream API (листинг 2).

Листинг 2. Пример конвейера Java Stream API

```
int[] result = Arrays.stream(x)
     .map(n -> n * 2)     // doubleItems
     .map(n -> n + 1)     // addOne
     .toArray();
```

Для генерации программы найденный конвейер подставляется в шаблон метода. Полученный метод имеет следующий вид (листинг 3).

Листинг 3. Пример сгенерированного метода с конвейером Java Stream API

```
public static int[] executePipeline(
            int[] x) {
     return Arrays.stream(x)
            .map(n -> n * 2)
            .map(n -> n + 1)
            .toArray();
}
```

Подобным образом генерируется модульный тест (листинг 4) и документация с использованием HTML (листинг 5).

Листинг 4. Пример сгенерированного модульного теста для конвейера Java Stream API

```
public class Pipeline140Test {
       @Test
       public void testPipeline() {
              int[] input = new int[]{1, 2,
3};
              int[] expected = new int[]{3, 5,
7};
              int[]          result          =
                             Pipe-
line140.executePipeline(input);
              assertArrayEquals(expected,  re-
sult);
       }
}
```

Листинг 5. Пример сгенерированной документации к методу с конвейером Java Stream API

```
  * Executes a pipeline with 2 operations:
   * <ol>
   *   <li>Double = Multiply by 2</li>
   *   <li>Add one</li>
  * </ol>
```

Программа, тест и документация генерируются соответствующими алгоритмами. Шаблоны могут изменены в соответствии с соглашением об оформлении кода и других элементов программы.

4. Исследование алгоритма поиска конвейеров

4.1. План исследования

Основным алгоритмом метода генерации конвейеров является алгоритм поиска конвейера. Этот алгоритм будет исследоваться по следующему плану:

1. сформировать множество функций F, используемых в разработанном алгоритме поиска конвейеров и БЯМ;
2. сгенерировать тестовые наборы;
3. применить разработанный алгоритм поиска конвейеров к тестовым наборам;
4. применить алгоритм полного перебора к тестовым наборам;
5. применить БЯМ к тестовым наборам.

Алгоритм поиска конвейера исследовался на компьютере с процессором Intel Core I9-12900K, оперативной памятью объёмом 60 GB, накопителем SSD Western Digital 512 GB, операционной системой Windows 11 Pro.

4.2. Формирование множества функций *F*

Функции были получены с известных ресурсов по программированию и обмену кодом:

- GitHub (https://github.com);
- Stack Overflow (https://stackoverflow.com);
- Medium (https://medium.com);
- Habr (https://habr.com/ru/articles/).

Функции были отобраны по двум критериям:

1. частота упоминания и использования в коде и обсуждениях на указанных платформах;
2. возможность композиции функций в функции новых видов.

Были сформированы множества из 28, 50 и 100 функций: $|F| = 28;50;100$. Множества меньшей

мощности являются подмножествами множеств большей мощности.

4.3. Генерация тестовых наборов

Был разработан генератор тестовых наборов на языке программирования Java. Тестовые наборы для $k = 1,2,3,4,5$ генерируются следующим образом:

- формируется исходный массив из 20 элементов со значениями: $[1,2,3,...,10,1,2,3,...,10]$;
- формируются все возможные конвейеры относительно F длиной k ($|F|^k$);
- сформированные конвейеры применяются к исходному массиву x: $f^{(1)}\left(f^{(2)}\left(...f^{(n)}\left(x\right)\right)\right) = y$ с получением результирующего массива y; полный перебор всех возможных сочетаний с повторениями функций из множества F используется, поскольку не все функции коммутативны;
- данные о конвейере, исходном и результирующем массиве (листинг 6) записываются в файл.

Листинг 6. Пример исходного тестового набора при $k = 3$

```
Functions: [12, 14, 1]
Original array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Transformed array: [0, 2, 4, 6, 8, 10, 12, 14,
16, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 18]
```

4.4. Применение разработанного алгоритма поиска конвейера к тестовым наборам

При применении разработанного алгоритма и БЯМ к тестовым наборами замерялись следующие показатели:

- точность А, Б, В;
- средняя длина найденных конвейеров за исключением конвейеров с поэлементной заменой;
- среднее время генерации одного конвейера – рассчитывалось время для каждой БЯМ.

Определим точность А, Б, В. Пусть n — общее число найденных конвейеров; v — количество найденных конвейеров, прошедших первый этап валидации (описан далее); e — количество найденных конвейеров с поэлементной заменой; t — количество найденных конвейеров с функциями, не принадлежащими множеству F. Тогда точность А вычисляется как $p_A = \frac{v}{n}$, точность вычисляется как $p_Б = \frac{v+e}{n}$, а точность В – как $p_B = \frac{v-t}{n}$.

Была разработана программа на языке Java для применения разработанного алгоритма. Программа выполняет следующее:

1. для каждого тестового набора применяет алгоритм поиска конвейера, записывает в протокол работы данные тестового набора, а также промежуточные массивы;
2. вычисляет среднее время поиска, количество найденных конвейеров, а также среднюю длину конвейера.

Кроме того, была разработана программа на языке Java, отыскивающая конвейер полным перебором конвейеров заданной длины.

4.5. Применение больших языковых моделей к тестовым наборам

Для сравнения с разработанным алгоритмом использовались следующие БЯМ:

ChatGPT-4o: версия gpt-4o (URL: https://chatgpt.com/);

Qwen2.5-Max: версия qwen2.5-max-latest (URL https://chat.qwen.ai/);

DeepSeek: версия deepseek-chat (URL: https://chat.deepseek.com/).

Критериями выбора моделей были: популярность использования, доступность и разнообразие провайдеров. В связи с большим промежутком временем обработки БЯМ одного запроса (от 3 до 12 секунд) были отобраны с помощью генератора случайных чисел [15] 50 тестовых наборов для каждой длины k. Запросы были сформулированы в двух вариантах:

1. без указания функций множества F — БЯМ может использовать любые известные ей преобразования;
2. с указанием функций множества F — БЯМ должна будет строить решение, используя только их.

Запросы формировались одношагово — без уточняющих сообщений, для равных условий сравнения. Каждый текстовый запрос отправлен ко всем трём БЯМ. Для всех моделей использовался следующий системный промпт:

«Ты эксперт Java-программист. Генерируй только корректный компилируемый код Java без комментариев и дополнительных объяснений.»

Эксперименты исследования отличались пользовательским промптом.

Вариант 1 — без указания множества функций F:

«Напиши Java метод, который преобразует входной массив {массив_входа} в выходной массив {массив_выхода}, используя Stream API. Верни только код метода без объяснений. Пример: входной массив [1, 2, 3, 4, 5], выходной массив [4, 16].»

Вариант 2 — с указанием множества функций F:

«Напиши Java метод, который преобразует входной массив {массив_входа} в выходной массив {массив_выхода}, используя ТОЛЬКО следующие функции Stream API: <Список функций>»

Валидация результатов БЯМ выполнялась в два этапа:

1. синтаксическая проверка — компиляция сгенерированного кода с помощью javac. Если код не компилируется, результат отмечается как синтаксически некорректный;
2. проверка соответствия множеству F — извлечение из кода всех вызовов методов Stream API с помощью регулярных выражений и сопоставление с перечнем функций из множества F. Если обнару-

Таблица 1. Результаты применения разработанного алгоритма и поиска полным перебором

Длина конвейера k	Точность А	Средняя длина конвейера для 28/50/100 функций
1	100%	1
2	100%	1,55/1,77/1,68
3	100%	2,01/2,54/2,39
4	100%	2,32/3,11/2,93
5	100%	2,7/3,58/3,49

Таблица 2. Среднее время поиска одного конвейера разработанным алгоритмом и полным перебором конвейеров

Длина конвейера k	Среднее время поиска одного конвейера полным перебором для 28/50/100 функций, с	Среднее время поиска одного конвейера разработанным алгоритмом для 28/50/100 функций, с
1	0,0005/0,0004/0,0003	0,001/0,0004/0,0003
2	0,0005/0,00052/0,002	0,002/0,00048/0,001
3	0,005/0,007/0,08	0,005/0,005/0,03
4	0,063/0,27/7,2	0,062/0,27/3
5	1,33/14,5/621,1	0,093/11,8/283,7

жены функции, не входящие в F, результат отмечается как нарушающий ограничения.

Итоговым результатом БЯМ считался конвейер минимальной длины.

5. Полученные результаты

5.1. Результаты применения алгоритма поиска конвейера

После применения алгоритма поиска конвейера к тестовым наборам были получены следующие результаты (таблицы 1, 2). Для $|F| = 28$ приведено время поиска всех конвейеров минимальной длины, для $|F| = 50;100$ — первого конвейера. Разработанный алгоритм успешно находит конвейер. Результаты разработанного алгоритма сравниваются с результатами поиска полным перебором конвейеров.

5.2. Результаты применения больших языковых моделей

Были получены следующие результаты БЯМ, которые использовали для поиска конвейера все известные им функции (таблица 3).

Для k = 1 БЯМ успешно нашли конвейеры, схожие с конвейерами из тестовых наборов. При $2 \le k \le 4$ среди найденных конвейеров появляются конвейеры с поэлементной заменой элементов исходного массива элементами результирующего массива (листинг 7). При k = 1 доля конвейеров без поэлементных замен составляет 30%.

Листинг 7. Пример полученного конвейера с поэлементной заменой

```
int[] targetArray = Arrays.stream(startArray)
    .map(x -> {
      if (x == 1 || x == 10) return 125;
      if (x == 2 || x == 3) return 64;
      if (x == 4 || x == 5) return 27;
      if (x == 6 || x == 7) return 8;
      if (x == 8 || x == 9) return 1;
      return 0;})
    .toArray();
```

Чтобы разработанный алгоритм и БЯМ находились в одинаковых условиях, в запросе генерации конвейера были явно указаны функции множества F. Были получены следующие результаты (таблица 4).

Для $|F| = 28;50;100$ значения в таблице отличаются незначительно, поэтому приведены средние для указанных размеров множества $|F|$. С увеличением значения k и, если функции искомого конвейера не очевидны, БЯМ начинают использовать функции, не принадлежащие множеству F. При $k \ge 3$ БЯМ в одном из трёх случаев верно находят конвейер с функциями множества F. В остальных случаях БЯМ либо выдаёт сообщение, что конвейера не существует, либо начинает использовать функции, не принадлежащие множеству F.

6. Анализ полученных результатов исследования

6.1. Точность

Точность разработанного алгоритма равна 100%, как и точность поиска полным перебором, и не зависит от k и $|F|$. Точность до трёх раз лучше, чем точность БЯМ. Точность БЯМ, с указанием функций множества F, уменьшалась при увеличении длины k.

При $k \ge 2$ БЯМ без указания функции множества F находили конвейеры с поэлементной заменой значений исходного и результирующего массивов, в значительной доле случаев.

6.2. Средняя длина найденных конвейеров

Средняя длина найденных конвейеров равна средней длине конвейеров, найденных полным перебором, и была меньше или равна длине k. С увеличением длины k преимущество по этому параметру по сравнению с БЯМ возрастает.

Таблица 3. Результаты применения больших языковых моделей без указания функций множества F

Длина конвейера k	Точность Б	Точность В	Средняя длина конвейера	Среднее время поиска одного конвейера, с
1	95%	90%	1	5,8
2	90%	35%	1,77	5,8
3	80%	30%	2,5	5,8
4	77%	28%	4,83	5,8
5	76%	30%	6,4	5,8

Таблица 4. Результаты применения больших языковых моделей с указанием функций множества F

Длина конвейера k	Точность A	Средняя длина конвейера	Среднее время поиска одного конвейера, с
1	90%	1	8,8
2	50%	1,67	8,8
3	30%	2,71	8,8
4	32%	4,38	8,8
5	31%	4,75	8,8

С увеличением $|F|$ средняя длина конвейера возрастала при $|F| = 50$, а затем снова снижалась при $|F| = 100$. Это объясняется тем, что конвейер $f^{(1)}\left(f^{(2)}\left(...f^{(n)}\left(x\right)\right)\right) = y$ может быть заменён конвейером $f^{(n+1)}\left(x\right) = y$, и с ростом $|F|$ количество таких замен возрастает.

БЯМ, которым не были указаны функции множества F, демонстрировали большие длины найденных конвейеров по сравнению с БЯМ с указанием функций множества F.

6.3. Среднее время поиска одного конвейера

Среднее время ответа БЯМ корректировалось с учётом сетевой задержки (ping) между клиентской машиной и серверами провайдеров API. Время корректировалось по формуле:

$$t_{corr} = t_{obs} - \frac{t_{ping}}{2},$$

где t_{obs} — наблюдаемое среднее время ответа; t_{ping} — измеренная круговая задержка сети.

Среднее время поиска одного конвейера одной БЯМ с учётом коррекции составило 5,8 и 8,8 секунд. Такое значительное время объясняется особенностями работы моделей и необходимостью обработки больших объёмов данных для поиска конвейера.

Среднее время поиска одного конвейера разработанным алгоритмом было значительно быстрее, чем у БЯМ при длине $k \le 4$, и было не медленнее, чем при полном переборе при длине $k \ge 2$. Затем время значительно падает и уступает времени поиска БЯМ при $k = 5$ и $|F| = 100$. Рост количества функций увеличивает время поиска, но оно остаётся приемлемым. Согласно [10], доля конвейеров длиной $k \le 3$ составляет 98,5%.

Компьютер, на котором проводилось исследование, имеет худшие характеристики, чем сервера, используемые БЯМ, что повлияло на среднее время поиска одного конвейера.

7. Заключение

Был разработан метод генерации конвейеров, включающий алгоритмы поиска конвейера, генерации программы, теста и документации. Алгоритм поиска конвейера разработан на основе модифицированного поиска в глубину и находит конвейеры минимальной длины.

Результативность алгоритма поиска конвейера была исследована с функциями преобразований целочисленных массивов Java Stream API и сравнивалась с результатами поиска полным перебором и БЯМ: Qwen2.5-Max, DeepSeek и OpenAI ChatGPT4o, которые находили конвейер без указания и с указанием функций множества F.

Разработанный алгоритм, несмотря на увеличение времени поиска при росте k, имеет более высокую точность чем БЯМ без и с указанием функций множества F и меньшее среднее время по сравнению с поиском полным перебором при $k > 1$. Средняя длина конвейера, найденного алгоритмом, короче, чем у БЯМ без указания функций множества F не менее 12% и чем у найденного БЯМ с указанием функций множества F не менее 7% при $k > 1$, и равна средней длине поиска полным перебором.

Полученные результаты подтверждают работоспособность разработанного метода для автоматизации программирования конвейеров.

Исходный код алгоритма, тестовые наборы данных и результаты запусков размещены в репозитории: https://osf.io/yhc9e/overview?viewonly=1500e82db54a45ed82cd49c127e6a724.

Литература

1. *Borodin D., Prutzkow A.* Program generation methods: types and instances // International Journal of Open Information Technologies. 2025. Vol. 13. No. 7. Pp. 83–91.
2. *Jin H., Huang L., Cai H., Yan J., Li B., Chen H.* From LLMs to LLM-based agents for software engineering: a survey of current, challenges and future // arXiv preprint arXiv:2408.02479. 2024.
3. *Liu F., Liu Y., Shi L., Huang H., Wang R., Yang Z., Zhang L., Li Z., Ma Y.* Exploring and evaluating hallucinations in LLM-powered code generation // arXiv preprint arXiv:2404.00971. 2024.
4. *Capretta V., Uustalu T., Vene V.* Corecursive algebras: A study of general structured corecursion // Brazilian Symposium on Formal Methods. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Pp. 84–100.
5. *Chen Y., Hu Z., Zhi C., Han J., Deng S., Yin J.* Chatunitest: A framework for llm-based test generation // Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering. 2024. Pp. 572–576.
6. *Sapkota R., Raza S., Karkee M.* Comprehensive analysis of transparency and accessibility of ChatGPT, DeepSeek, and other SOTA large language models // arXiv preprint arXiv:2502.18505. 2025.
7. *Khatchadourian R., Tang Y., Bagherzadeh M., Ray B.* An Empirical Study on the Use and Misuse of Java 8 Streams. Fundamental Approaches to Software Engineering. 2020. Pp. 97–118.
8. *Urma R.G., Fusco M., Mycroft A.* Modern Java in Action. Lambdas, Streams, Functional and Reactive Programming. Shelter Island: Manning, 2019. 592 p.
9. *Иванов Р.А., Валеев Т.Ф.* Автоматический рефак-

торинг Java-кода с использованием Stream API // Вестник НГУ. Серия: Информационные технологии. 2019. №2. С. 59–60.

10. *Rosales E., Basso M., Rosà A., Binder W.* Large-scale Characterization of Java Streams // Software: Practice and Experience. 2023. Vol. 53. No. 9. Pp. 1763–1792.

11. *Nostas J., Alcocer J.P.S., Costa D.E., Bergel A.* How Do Developers Use the Java Stream API? // International Conference on Computational Science and Its Applications. 2021. Pp. 323–335.

12. *Schenkenfelder B., Brandstätter U., Kirchtag H., Wimmer M.* Low-Code Development and End-User Development: (How) Are They Different? // 1st International Workshop on Participatory Design & End-User Development. Building Bridges. 2024. Pp. 110–115.

13. *Schiavio F., Rosa A., Binder W.* SQL to Stream with S2S: An Automatic Benchmark Generator for the Java Stream API // Proceedings of the 21st ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences. 2022. Pp. 179–186.

14. *Korf R. E.* Depth-first iterative-deepening: an optimal admissible tree search // Artificial intelligence. 1985. Vol. 27. No. 1. Pp. 97–109.

15. *Marsaglia G.* XorShift RNGs // Journal of Statistical Software. 2003. Vol. 8. No. 13. Pp. 17–31.

Поступила 19 января 2026 г.

English

JAVA STREAM API PIPELINE GENERATION: A COMPARISON OF A SPECIALIZED ALGORITHM AND THREE LARGE LANGUAGE MODELS

Daniil Vadimovich Borodin — Postgraduate Student, Lipetsk State Pedagogical University[1].

Aleksandr Viktorovich Prutzkow — Grand Dr. in Engineering, Professor, Lipetsk State Pedagogical University[1]; Professor, Ryazan State Radio Engineering University named after V.F. Utkin[2].

[1]*Address:* 398020, Russian Federation, Lipetsk, Lenin st., 42.
[2]*Address:* 390005, Russian Federation, Ryazan, Gagarin st., 59/1.

E-mail: dvoranchik@yandex.ru

Abstract: Automation of design, testing and documentation of software pipelines is relevant for software engineering and acceleration of software development. We introduce a method for generating pipelines. The method falls into algorithms of pipeline search, program generation, test, and documentation. The pipeline search algorithm is based on a modified depth-first search algorithm and finds pipelines of minimal length. The modification consisted in iteratively limiting the search depth. The performance of the algorithm that search for a pipeline on the functions of transforming integer arrays F of the Java Stream API compared with large language models Qwen2.5-Max, DeepSeek and OpenAI ChatGPT4o. The experiment plan included forming a function set F, generating test sets, applying the pipeline search algorithm and large language models to test sets. Large language models found a pipeline with and without specifying the functions of the F set. The algorithm has higher accuracy than large language models with and without specifying the functions of the set F in spite of more time for searching pipeline of length $k = 5$. The average length of the pipeline found by the algorithm is shorter than the large language models without specifying the functions of the F set by at least 12% and than the large language models with specifying the functions of the F set by at least 7%. The results prove the efficiency of the method for generating pipelines. Further studies of the pipeline generation method consist of accelerating it by developing its concurrent version and using association rules mining.

Keywords: programming automation; program generation; pipeline generation; Java Stream API; large language models; depth-first search; testing; documentation generation.

References

1. *Borodin D., Prutzkow A.* Program generation methods: types and instances. International Journal of Open Information Technologies. 2025. Vol. 13. No. 7. Pp. 83–91.

2. *Jin H., Huang L., Cai H., Yan J., Li B., Chen H.* From LLMs to LLM-based agents for software engineering: a survey of current, challenges and future. arXiv preprint arXiv:2408.02479. 2024.

3. *Liu F., Liu Y., Shi L., Huang H., Wang R., Yang Z., Zhang L., Li Z., Ma Y.* Exploring and evaluating hallucinations in LLM-powered code generation. arXiv preprint arXiv:2404.00971. 2024.

4. *Capretta V., Uustalu T., Vene V.* Corecursive algebras: A study of general structured corecursion. Brazilian Symposium on Formal Methods. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Pp. 84–100.

5. *Chen Y., Hu Z., Zhi C., Han J., Deng S., Yin J.* Chatunitest: A framework for llm-based test generation. Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering. 2024. Pp. 572–576.

6. *Sapkota R., Raza S., Karkee M.* Comprehensive analysis of transparency and accessibility of ChatGPT, DeepSeek, and other SOTA large language models. arXiv preprint arXiv:2502.18505. 2025.

7. *Khatchadourian R., Tang Y., Bagherzadeh M., Ray B.* An Empirical Study on the Use and Misuse of Java 8 Streams. Fundamental Approaches to Software Engineering. 2020. Pp. 97–118.

8. *Urma R.G., Fusco M., Mycroft A.* Modern Java in Action. Lambdas, Streams, Functional and Reactive Programming. Shelter Island: Manning, 2019. 592 p.

9. *Ivanov R.A., Valeev T.F.* Automatic refactoring of Java code using Stream API. Vestnik NGU. Seriya: Informatsionnye tekhnologii. 2019. No. 2. Pp. 59–60.

10. *Rosales E., Basso M., Rosà A., Binder W.* Large-scale Characterization of Java Streams. Software: Practice and Experience. 2023. Vol. 53. No. 9. Pp. 1763–1792.

11. *Nostas J., Alcocer J.P.S., Costa D.E., Bergel A.* How Do Developers Use the Java Stream API?. International Conference on Computational Science and Its Applications. 2021. Pp. 323–335.

12. *Schenkenfelder B., Brandstätter U., Kirchtag H., Wimmer M.* Low-Code Development and End-User Development: (How) Are They Different?. 1st International Workshop on Participatory Design & End-User Development. Building Bridges. 2024. Pp. 110–115.

13. *Schiavio F., Rosa A., Binder W.* SQL to Stream with S2S: An Automatic Benchmark Generator for the Java Stream API. Proceedings of the 21st ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences. 2022. Pp. 179–186.

14. *Korf R. E.* Depth-first iterative-deepening: an optimal admissible tree search. Artificial intelligence. 1985. Vol. 27. No. 1. Pp. 97–109.

15. *Marsaglia G.* XorShift RNGs. Journal of Statistical Software. 2003. Vol. 8. No. 13. Pp. 17–31.