На правах рукописи

Качанов Владимир Владимирович

МЕТОДЫ И ПРОГРАММНЫЕ СРЕДСТВА АВТОМАТИЧЕСКОГО РЕЦЕНЗИРОВАНИЯ ИСХОДНОГО КОДА

2.3.5 — «Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей»

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата технических наук

Москва — 2025

Работа выполнена на кафедре интеллектуальных систем Федерального государственного автономного образовательного учреждения высшего образования «Московский физико-технический институт (национальный исследовательский университет)».

Научный руководитель: **Цурков Владимир Иванович**
доктор физико-математических наук, профессор

Официальные оппоненты: **Кознов Дмитрий Владимирович**
доктор технических наук, Федеральное государственное бюджетное образовательное учреждение высшего образования «Санкт-Петербургский государственный университет», профессор.

Маркин Дмитрий Олегович
кандидат технических наук, Федеральное государственное казённое военное образовательное учреждение высшего образования «Академия Федеральной службы охраны Российской Федерации», сотрудник.

Ведущая организация: Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет» (ФГАОУ ВО СКФУ).

Защита состоится «11» декабря 2025 года в 13 – 00 на заседании диссертационного совета 24.1.120.01 при Федеральном государственном бюджетном учреждении науки Институте системного программирования им. В.П. Иванникова Российской академии наук по адресу: 115035, г. Москва, ул. Садовническая, д. 41, ст. 2.

С диссертацией можно ознакомиться в библиотеке и на сайте Федерального государственного бюджетного учреждения науки Институт системного программирования им. В.П. Иванникова Российской академии наук.

Автореферат разослан « » 2025 года.

Ученый секретарь
диссертационного совета 24.1.120.01,
кандидат физико-математических наук Д. Ю. Турдаков

Общая характеристика работы

Актуальность темы. В условиях современной цифровой трансформации и динамичного развития информационных технологий обеспечение высокого качества программных продуктов становится стратегически важной задачей. Кодовая база, которая является основой функционирования программного обеспечения, требует постоянного мониторинга и совершенствования для минимизации риска возникновения критических ошибок. В этом контексте методы и программные средства автоматического рецензирования исходного кода, основанные на расширяемом наборе детекторов нежелательного или дефектного кода, представляют собой перспективное направление.

Рецензирование кода традиционно рассматривается как неотъемлемая часть жизненного цикла разработки программного обеспечения наряду с тестированием, изменением кода и его рефакторингом. Этап анализа кода, проводимый после его непосредственного написания, позволяет выявить скрытые дефекты и неформальные ошибки, которые могут быть не обнаружены в рамках стандартных систем непрерывной интеграции и автоматизированного тестирования. Такой подход обеспечивает более глубокую и всестороннюю оценку кода, способствуя повышению его качества и уменьшению вероятности возникновения ошибок в будущем. Кроме того, повышается читаемость и структурированность кода: он становится понятен не только одному специалисту, но и всей команде, что значительно упрощает и ускоряет дальнейшую разработку и поддержку проекта. Процесс рецензирования кода оказывает многогранное положительное влияние на качество конечного продукта и продуктивность команды разработчиков.

Однако проведение рецензирования кода вручную сопряжено с рядом существенных проблем. Ручной анализ требует значительных временных затрат и зачастую зависит от субъективной оценки отдельных участников команды, что может приводить к неравномерности качества рецензий. Кроме того, человеческий фактор нередко является источником пропуска небольших, но важных дефектов, особенно в условиях работы с большими объемами кода. В этом контексте внедрение автоматических средств рецензирования способно существенно повысить эффективность данного процесса, обеспечивая систематический и непрерывный контроль качества. Автоматизация помогает стандартизировать процедуру проверки, снизить вероятность возникновения ошибок, а также разгрузить специалистов, позволяя им сосредоточиться на более сложных и креативных аспектах разработки программного обеспечения.

Особое внимание в современных исследованиях уделяется разработке инструментов для детектирования признаков технического долга и антипаттернов исходного кода («запахи кода», англ. code smells). Это понятие, введенное Кентом Бэком и популяризированное Мартином Фаулером, описывает характеристики кода, которые затрудняют его поддержку и развитие. К таким признакам относятся длинные методы, сложные условия, классы с избыточными обязанностями или чрезмерная связанность между модулями. Автоматизированные инструмен-

ты, способные выявлять существующие антипаттерны и прогнозировать потенциальные проблемные места, значительно повышают поддерживаемость системы и улучшают коммуникацию внутри команды.

Уже более десяти лет ведутся активные исследования и разработки в области автоматизированных систем рецензирования исходного кода, в которых приняли участие как академические исследователи, так и промышленные компании. Среди наиболее заметных инициатив следует отметить такие инструменты, как SonarQube, Sourcery и Semgrep, интегрируемые в системы непрерывной интеграции и разработки (CI/CD) и обеспечивающие автоматическое комментирование запросов на включение изменений (pull requests) с предоставлением рекомендаций. В последние годы акцент сместился в сторону облачных решений, таких как GitHub CodeQL, Snyk Code и AWS CodeGuru Reviewer, которые применяют методы анализа данных и машинного обучения для повышения точности анализа и сокращения доли ложных срабатываний. Однако существующие инструменты в значительной степени ориентированы на статический анализ или эвристические правила. Пилотные внедрения систем на основе больших языковых моделей в настоящее время осуществляются в ряде крупных технологических компаний, включая Google, ByteDance и Microsoft.

Согласно исследованиям, внедрение систем автоматизированного рецензирования кода на основе искусственного интеллекта позволяет сократить время проведения проверки с 30 до 10 минут, что приводит к высвобождению до 20% рабочего времени разработчиков. Как отмечают эксперты Института системных наук IBM, стоимость устранения дефекта, обнаруженного на этапе эксплуатации проекта, приблизительно в шесть раз превышает затраты на его исправление в процессе разработки. В данном контексте автоматическое рецензирование кода приобретает особую значимость, поскольку обеспечивает выявление дефектов на ранних стадиях жизненного цикла программного обеспечения. Опыт внедрения подобных систем в крупнейших технологических компаниях подтверждает их эффективность. Например, корпорация Microsoft обрабатывает с их помощью 600 000 запросов на включение изменений ежемесячно, что позволило сократить медианное время внедрения правок на 10–20% в рамках 5000 репозиториев. Компания ByteDance сообщает об обслуживании 12 000 активных пользователей. Точность предлагаемых системой предложений достигает 75%, что является фактором, поддерживающим доверие со стороны разработчиков.

Таким образом, актуальной является задача разработки методов анализа исходного кода и детектирования не только стандартных ошибок и уязвимостей, которые обнаруживаются методами статического и динамического анализа, но и менее формализованных дефектов, а также инструмента, который интегрирует в себе автоматическое рецензирование исходного кода с пояснением и фильтрацией обнаруженных антипаттернов. При этом требуется обеспечить адекватную точность и приемлемую полноту. Разработка таких методов и итогового инструмента не только повысит качество ПО, но и облегчит дальнейшую его поддержку. Также исследования в этом направлении способствуют развитию теоретической

базы, на которой можно строить дальнейшие инновационные решения в области обеспечения качества программного обеспечения.

Целью данной работы является развитие методов обнаружения признаков технического долга, заданных на основе примеров исходного кода, и построение системы автоматического рецензирования кода, которая интегрирует в себя реализованные методы.

Для достижения поставленной цели необходимо было решить следующие **задачи**:

1. Разработать и реализовать детекторы нежелательного и потенциально дефектного кода. Собрать наборы размеченных примеров антипаттернов кода по исправлениям непосредственных разработчиков проектов.
2. Проанализировать существующие подходы к автоматическому рецензированию исходного кода. Провести обзор тематик рецензий различными авторами. Разработать методы определения полезных и применимых замечаний.
3. Разработать и реализовать инструмент автоматического рецензирования на основе наиболее перспективных подходов.
4. Интегрировать разработанные и реализованные детекторы в единую систему рецензирования.

Основные положения, выносимые на защиту:

1. Метод сбора размеченных примеров и набор данных различных типов дефектов кода, извлеченный из истории разработки проектов;
2. Методы детектирования антипаттернов исходного кода, основанный на анализе исходного кода и методах машинного обучения;
3. Методы автоматического рецензирования исходного кода, основанные на применении языковых моделей;
4. Программный комплекс, интегрирующий работу детекторов антипаттернов исходного кода в систему автоматического рецензирования.

Научная новизна:

1. Сформирован репрезентативный набор данных, охватывающий различные типы ошибок, выявленные в процессе эволюции реальных программных проектов, что создаёт основу для обучения и валидации интеллектуальных моделей анализа кода;
2. Разработаны методы интеллектуального детектирования неформальных дефектов программного кода, основанные на синтезе статического анализа исходного текста и алгоритмов машинного обучения;
3. Проведено сравнительное исследование эффективности методов повышения качества генерации больших языковых моделей в задаче рецензирования исходного кода;
4. Выполнено оригинальное исследование, результатом которого стало проектирование и реализация программного комплекса, интегрирующего разработанные детекторы антипаттернов в единую систему автоматического рецензирования кода. Архитектура комплекса обеспечивает модульность,

расширяемость и совместимость с существующими инструментами разработки.

Теоретическая значимость. Исследование вносит вклад в развитие научной базы автоматизированного анализа исходного кода и рецензирования изменений, вносимых разработчиками. В работе предложены новые методы интеллектуального анализа и сформирован репрезентативный набор данных, что расширяет спектр задач, решаемых традиционным статическим и динамическим анализом и дает основу для обучения и тестирования моделей, ориентированных на практику индустриальной разработки. Проведённое исследование помогает формализовать подходы к фильтрации, объяснению и структурированию замечаний в задаче рецензирования кода, что открывает перспективы для последующих работ в области использования ИИ для обеспечения качества ПО. Таким образом, работа способствует развитию теоретических представлений о сочетании статического анализа и методов машинного обучения.

Практическая значимость. Разработанный программный комплекс позволяет повысить эффективность процесса рецензирования кода, уменьшить зависимость от субъективных факторов и снизить временные затраты на ручной анализ. Автоматизация проверки способствует выявлению как формальных, так и неформальных дефектов, улучшает читаемость и структурированность кода, облегчает его поддержку и ускоряет интеграцию новых участников команды. Применение системы в промышленной среде даёт возможность стандартизировать практики контроля качества и интегрировать их в существующие системы непрерывной разработки и внедрения. Тестирование показало, что более половины предлагаемых исправлений и замечаний к изменениям в исходном коде отмечаются разработчиками как приемлемые или применимые. Реализованная система рецензирования исходного кода внедрена в конвеер непрерывной интеграции и поставки компанией ООО «СОТЕК».

Методы исследования. Для решения задач, поставленных в диссертационной работе, использовались методы машинного обучения, глубокого обучения и программной инженерии.

Достоверность полученных результатов обеспечивается:

- корректностью выбора и применения используемых алгоритмов, методов и моделей для сбора, анализа и обработки данных;
- результатами апробации реализованных методов на внутреннем тестировании программного комплекса;
- согласованностью полученных результатов с результатами, полученными другими авторами.

Апробация работы. Основные результаты работы докладывались на следующих конференциях:

- Создание набора данных для комбинированной классификации рецензий исходного кода, Интеллектуализация обработки информации, 2024.
- Извлечение именованных сущностей из рецензий к исходному коду, Открытая конференция ИСП РАН, 2023.

- Технический долг в жизненном цикле разработки ПО: запахи кода, Открытая конференция ИСП РАН, 2021.

Личный вклад. Все выносимые на защиту результаты, кроме отдельно оговоренных случаев, получены лично автором.

Публикации. Основные результаты по теме диссертации изложены в 6 печатных изданиях, 5 из которых изданы в журналах, рекомендованных ВАК, 1 — в тезисах докладов. Получено 1 свидетельство о регистрации программы для ЭВМ.

В работах [1, 7] все результаты получены лично автором.

В работах [2–4, 6] автором были выполнены разработка и модификации моделей глубокого обучения, сбор и разметка наборов данных, проведение вычислительных экспериментов и анализ результатов.

В работе [5] автором были предложены оптимизации алгоритмов трансформации исходного кода, проведен анализ результатов.

Объем и структура работы. Диссертация состоит из оглавления, введения, 4 глав, заключения, списка иллюстраций, списка таблиц и списка литературы из 129 наименований. Полный текст занимает 134 страницы.

Содержание работы

Во **введении** изложена актуальность исследований, проводимых в рамках данной диссертационной работы, сформулирована цель и задачи. Обоснована научная новизна, практическая и теоретическая значимость представляемой работы.

Первая глава посвящена обзору области автоматического рецензирования исходного кода и обнаружения антипаттернов в исходном коде программ. Вводятся основные понятия из области технического долга и методов поиска его признаков. Особое внимание уделяется систематизации существующих методов детектирования неформальных дефектов исходного кода. В рамках главы рассматриваются ключевые проблемы использования инструментов обнаружения антипаттернов в исходном коде, а также представлено подробное исследование факторов, влияющих на точность и полноту существующих подходов.

В разделе 1.1 формулируются базовые определения, затрагиваемые в работе, такие как, «Рецензирование исходного кода», «Автоматическое рецензирование исходного кода». Также выделены особенности исследуемой предметной области.

В разделе 1.2 проводится анализ существующих исследований по теме автоматического рецензирования. Описаны решения от четырех различных групп исследователей. Существующие инструменты автоматического рецензирования подразделяются на две основные категории: решения, предполагающие повторное использование ранее написанных рецензий в схожих контекстах, и системы, нацеленные на генерацию новых рецензий с применением языковых моделей.

Первый подход менее требователен к вычислительным ресурсам и основывается на предположении, что значительная часть рецензий может быть переиспользована в новых контекстах без изменений. К преимуществам данного подхода относится гарантированная корректность предлагаемых рецензий в их исходных

контекстах. Однако, как показала проведенная в ходе работы оценка, подобные решения демонстрируют относительно низкие показатели применимости на практике; кроме того, пользователям зачастую бывает затруднительно интерпретировать предложения системы.

Решения, использующие генерацию рецензий с помощью языковых моделей, напротив, требуют наличия высокопроизводительных вычислительных ресурсов, в частности графических ускорителей, как для обучения, так и для эксплуатации. Такие решения формируют значительно более понятные и релевантные рецензии с точки зрения языка. Однако их существенным недостатком является отсутствие гарантий корректности генерируемых предложений, поскольку в процессе генерации языковая модель может продуцировать недостоверную информацию (феномен «галлюцинации») или внутренне противоречивые утверждения. Рассмотренные решения основаны на различных архитектурах моделей и схемах обучения и дообучения.

В разделе 1.3 рассматриваются основные понятия области детектирования антипаттернов исходного кода, классификация и примеры такого кода.

Антипаттерны разделяют на три категории: Implementation smells (дефекты реализации в пределах метода), Design smells (проблемы проектирования на уровне классов) и Architecture smells (архитектурные недостатки системы).

В статьях зачастую исследуют такие виды антипаттернов, как: «Божественный класс» (God Class), «Завистливый метод» (Feature Envy), «Стрельба дробью» (Shotgun Surgery), «Сложный метод» (Complex Method), «Длинный метод» (Long Method), «Длинный список параметров» (Long Parameter List).

Основной проблемой детектирования таких дефектов кода является отсутствие их формального описания и значительная зависимость граничных значений для конкретной команды разработки и проекта. Подход основанный на метриках является базовым в различных инструментах анализа антипаттернов исходного кода (PMD, SonarQube, JDeodorant, SpotBugs). Однако заранее определенные правила часто не покрывают весь спектр объектов с дефектами. Также созданные эвристики могут быть слишком жесткими или наоборот слишком общими, что приводит к ложно-положительным или ложно-отрицательным срабатываниям.

В разделе 1.4 описан сравнительный анализ существующих решений. Для проведения анализа были отобраны девять проектов с открытым исходным кодом, на которых выполнялась оценка работы трёх инструментов детектирования. На основании сформированных отчётов был проведен анализ корреляции результатов работы указанных инструментов.

Было показано, что для детекторов сложных методов только совпадает треть от общего числа предупреждений. Для длинных методов — около половины. Для оценки точности обнаружения антипаттернов исходного кода часть предупреждений была исследована на основании ручной разметки 100 случайных примеров, отмеченных всеми инструментами.

Антипаттерн типа «длинный метод», являясь наиболее однозначно идентифицируемым, обнаружение которого зачастую связано с количеством строк, имеет

наивысший уровень истинных срабатываний, близкий к 90%. В то же время, в процессе выборочной проверки файлов проектов было выявлено значительное количество методов, которые эксперты отнесли к категории длинных, но которые не были обнаружены инструментами. Дефекты «божественный класс» (god class) и «сложный метод» (complex method) характеризуются крайне низким уровнем истинных срабатываний (10% и 20% соответственно). Учитывая высокую трудоемкость процесса верификации предупреждений, такие показатели точности можно считать неприемлемыми для практического применения.

В первой главе проведено комплексное исследование современного состояния области автоматического рецензирования исходного кода и детектирования антипаттернов. Проведенный анализ позволил выделить два принципиально различных подхода к автоматическому рецензированию: системы, основанные на повторном использовании существующих рецензий, и решения, использующие генерацию новых рецензий с помощью языковых моделей. Систематизированы основные понятия и терминология, связанные с техническим долгом и методами идентификации его признаков.

Полученные результаты подтверждают актуальность разработки новых методов детектирования антипаттернов с повышенной точностью и обосновывают необходимость исследований, направленных на создание гибридных подходов, сочетающих формальные метрики с методами машинного обучения. Выявленные проблемы и ограничения современных решений определяют направления дальнейших исследований, представленных в последующих главах работы.

Вторая глава посвящена основным результатам исследования и разработки методов детектирования антипаттернов кода с применением методов машинного обучения и инструментов статического анализа исходного кода.

Использование методов машинного обучения позволяет автоматизировать выбор наиболее важных метрик (признаков), а также может находить сложные нелинейные взаимосвязи между метриками и наличием в коде того или иного потенциального дефекта. В ряде исследований применялись такие методы машинного обучения, как логистическая регрессия, решающие деревья и метод опорных векторов (SVM), для выявления антипаттернов на основе широкого набора метрик. Позднее было предложено применение глубоких нейронных сетей и автокодировщиков для преобразования метрик, уменьшения размерности и последующей классификации. Также существуют подходы, объединяющие данные от разных детекторов, подавая результат их работы в нейронные сети позволяет продемонстрировать повышение точности. В отдельную категорию можно выделить работы, исследующие применение токенизированного представления кода и нейросетевых архитектур для его обработки.

В разделе 2.1 описываются существующие подходы к подготовке наборов данных. Для построения качественной модели машинного обучения требуется большой набор обучающих данных. Первые открытые наборы данных с признаками технического долга включали около 240 вручную размеченных примеров, но со временем доступ к ним был утрачен. В ряде последующих работ сообщалось о со-

здании набора из десятка тысяч размеченных случаев технического долга, однако сами данные остались закрытыми. Открытый набор, состоящий из 1986 примеров был основан на устаревшем Qualitas Corpus (последнее обновление — 2012г.), что снижает его актуальность. Более современный и ценный ресурс предложен в исследовании MLCQ, где 20 специалистов оценили фрагменты кода, итогом чего стал набор с 14739 пометками и открытым доступом. При этом количество действительно подтверждённых уникальных примеров в нём оказалось около 1500.

Общая особенность ручной разметки заключается в её высоком качестве при ограниченной масштабируемость. Альтернативным путём является автоматическая аннотация с помощью инструментов статического анализа, таких как SonarQube и Designite, применённых к сотням проектов. Хотя такая разметка менее точна, она позволяет работать с крупными выборками и повторяемыми алгоритмами. Ещё одним способом сбора набора данных является анализ истории исправлений реальных проектов: например, в работе ManySStuBs4J рассматриваются однотипные мелкие изменения, внесённые самими разработчиками. Такой принцип применялся а текущей работе к поиску антипаттернов исходного кода.

В разделе 2.2 приводится описание разработанного алгоритма для автоматического сбора примеров дефектов кода с исправлениями от непосредственных разработчиков из проектов с открытым исходным кодом. Для сбора нового набора данных из истории разработки были проанализировано топ-100 проектов с Github по количеству положительных отметок от пользователей, основным языком которых является Java. Разработанный инструмент анализирует каждый проект по очереди, проходя всю историю его изменений. На каждом шаге с помощью реализованных детекторов антипаттернов кода выполняется анализ файлов, измененных в коммите. В случае если анализ выявляет исправление определенного типа дефекта, соответствующий пример вносится в итоговый список для данного детектора.

Для решения задачи бинарной классификации «Длинных методов» и «Божественных классов» использовались модели машинного обучения «Случайный лес» (Random Forest). Гиперпараметры для моделей подбирались экспериментально. В качестве признакового описания кода для «Длинных методов» применялись метрики LOC (количество строк кода), NCLOC (количество непустых строк кода), MCC (цикломатическая сложность) и TokenCount (количество токенов); для «Божественных классов» — метрики LCOM5 (отсутствие связности методов), NAD (количество атрибутов) и NMD (количество методов). Для работы с «Сложными методами» была реализована сверточная нейронная сеть.

С целью повышения точности отбора примеров были реализованы дополнительные фильтры на основе эвристик, определенных в ходе ручного анализа результатов. Псевдокод разработанного метода сбора данных представлен на алгоритме 1.

Полученный набор данных, состоящий из 5912 примеров может быть использован для обучения новых моделей машинного обучения с целью создания более точных детекторов технического долга. Набор включает 2517 примеров «Длин-

Алгоритм 1: Псевдокод алгоритма автоматического сбора примеров дефектов кода с исправлениями

```
Function collect_dataset(repositories):
    dataset ← [ ];
    foreach repo ∈ repositories do
        commits ← get_commit_history(repo);
        foreach commit ∈ commits do
            changed_files ← get_changed_files(commit);
            foreach file ∈ changed_files do
                old_code ← get_old_version(file, commit);
                new_code ← get_new_version(file, commit);
                foreach detector_type ∈ detectors do
                    check_and_add_example(detector_type, old_code,
                     new_code, commit, dataset);
    return dataset;
Function check_and_add_example(detector_type, old_code, new_code,
 commit, dataset):
    detector ← detectors[detector_type];
    old_defect ← detector.detect_fix(old_code);
    new_defect ← detector.detect_fix(new_code);
    if old_defect and ¬ new_defect then
        example ← create_example(old_code, new_code, commit);
        if detector[heuristics_filter].check(old_code, new_code) then
            dataset.add(example);
```

ных методов», 2967 примеров «Сложных методов» и 428 примеров «Божественного класса». Собранный набор данных опубликован в открытом доступе.

В работе также проводилось сравнение моделей, обученных на новых данных с предыдущими результатами. Поскольку количество примеров «Длинных Методов» значительно превысило объем, доступный в наборе данных MLCQ, появилась возможность обучить более сложную модель, а именно сеть LSTM. Детектор «Божественных классов» представляет собой алгоритм Случайный Лес, а для «Сложных методов» используется сверточная нейронная сеть. В качестве валидационных данных применялись размеченные вручную примеры из следующих проектов: ant, drill, error-prone, giraph, hive, truth.

При разметке валидационного набора данных использовались три категории: истинные срабатывания (TP), ложные срабатывания (FP) и истинные срабатывания, не требующие исправления (Won't Fix, WF). Применение категории «не требующие исправления» обусловлено случаями, когда наличие «дефекта» мо-

жет быть осознанным решением разработчиков либо исправление такого участка кода потребует значительных затрат ресурсов. Оценивались точность (PR), полнота (RE) и F1-мера (F1).

Модель	F1	PR	RE	WF	TP	FP
Длинный Метод						
Базовая	0.277	0.163	0.928	116	39	84
Новая	**0.493**	0.346	0.857	47	36	**21**
Сложный Метод						
Базовая	0.518	0.35	1.0	36	70	94
Новая	**0.757**	0.626	0.957	23	67	**17**
Божественный Класс						
Базовая	0.092	0.05	0.625	68	5	27
Новая	**0.141**	0.079	0.625	37	5	**21**

Таблица 1: Результаты валидации обученных моделей

Как видно из табл. 1, результаты моделей по всем типам дефектов улучшились. По метрике F1 улучшение составило от 46% до 78%, количество ложно положительных срабатываний уменьшилось на 22% для «Божественных классов» и более чем на 75% для других типов дефектов.

Раздел 2.3 содержит описание разработанных методов детектирования антипаттернов исходного кода и методов их дополнительной фильтрации. На примере детектора длинных методов демонстрируются базовый алгоритм классификации, этапы препроцессинга исходного кода, построение признакового пространства и применение дополнительных фильтров. Помимо непосредственного детектирования и классификации проблем, разработанный алгоритм предоставляет варианты исправления участков кода с дефектами.

Возможность генерации исправлений также способствует решению задачи классификации антипаттернов, поскольку существуют случаи, когда код по формальным признакам соответствует характеристикам дефекта, однако его реализация обусловлена объективной необходимостью. К таким случаям относятся, например, объемные конструкции `switch-case` или сложные последовательности математических вычислений. Хотя такие конструкции могут быть разбиты на отдельные методы, это не всегда целесообразно, так как может привести к снижению читаемости и понятности кода.

Таким образом, если алгоритм разделения длинного метода может предложить семантически осмысленное разбиение на два или более метода, каждый из которых не превышает допустимую длину, детектор идентифицирует данный случай и предоставляет вариант исправления. Кроме того, в разделе представлены результаты количественной оценки эффективности разработанных методов дополнительной фильтрации. На примере детектора «длинных методов» показана схема работы разработанных и реализованных детекторов на рисунке 1.

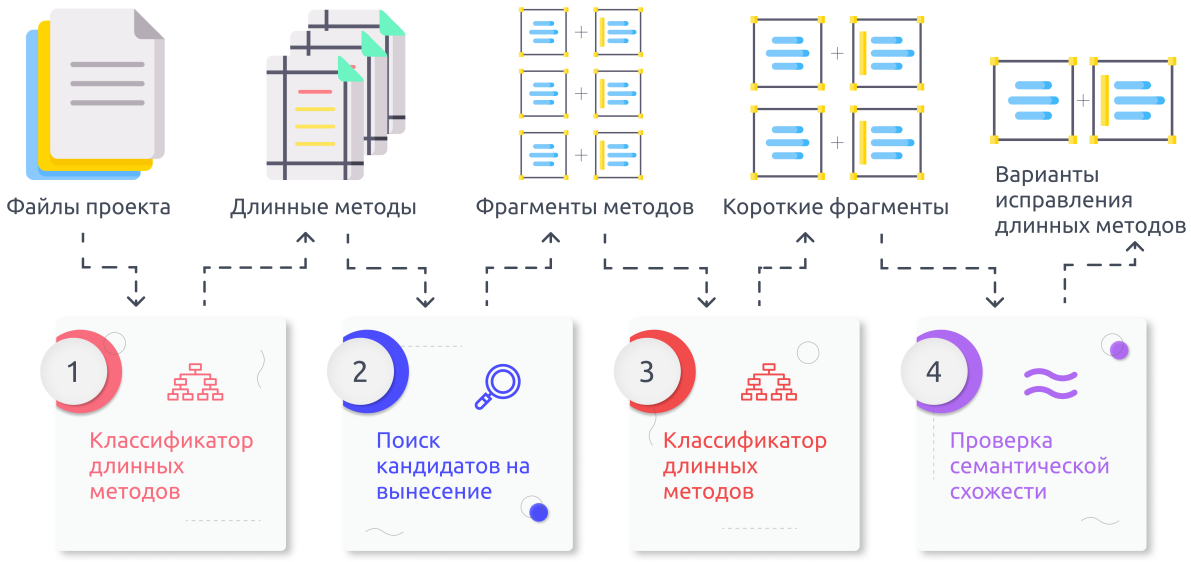

Рисунок 1: Схема работы детектора «длинных методов»

Во второй главе были представлены методы детектирования элементов технического долга. Разработанные методы используют гибрадный подход на основе метрик кода и методов машинного обучения, что позволяет повысить точность и полноту результатов, а также дает возможность подстраиваться под конкретную группу разработчиков или проект. Предложенный подход к дополнительной фильтрации результатов базового детектора значительно (на 75%) сокращает число ложноположительных срабатываний.

Также представлен метод сбора данных о антипаттернах и примерах их исправлений из непосредственного процесса разработки проектов с открытым исходным кодом. Показано, что собранные таким образом данные позволяют обучать более совершенные детекторы признаков технического долга. Улучшение по мере F1 составило от 46% до 78% для разных типов антипаттернов.

Третья глава посвящена исследованию методов автоматического рецензирования исходного кода. Приводится детальный анализ результатов работы двух воспроизведенных подходов: подбор готовых рецензий из «исторического» набора и генерация рецензий с использованием языковых моделей, реализующих архитектуры, предложенные в работах AUGER и CodeReviewer. Выделены проблемные аспекты апробированных подходов и предложены модификации для их решения. Кроме того в главе описано решение двух подзадач: формирование качественного набора данных для обучения и классификация рецензий.

Общая схема рецензирования представлена на рисунке 2.

Автоматизированная генерация комментариев к изменениям кода на основе исторических данных рецензирования представляет собой задачу повышения эффективности процессов анализа кода. Такой инструмент должен анализировать различия между версиями кода, выявлять значимые изменения и формировать

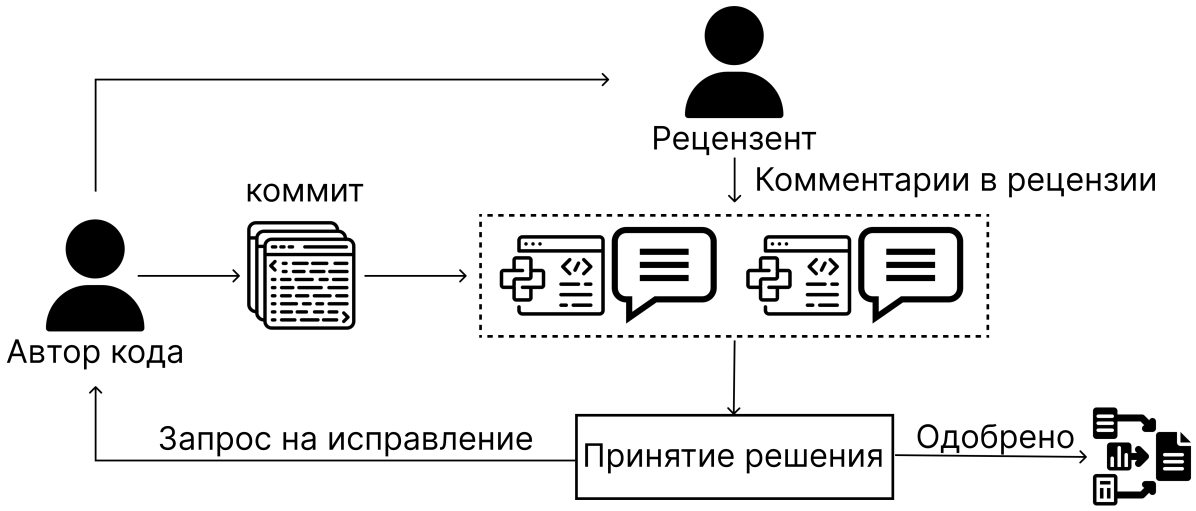

Рисунок 2: Общая схема рецензирования

осмысленные текстовые комментарии, аналогичные тем, которые оставляют эксперты.

Пусть:

- $\mathcal{C}_{\text{prev}}$ — исходное состояние кода до изменения,
- $\mathcal{C}_{\text{new}}$ — состояние кода после изменения,
- $\mathcal{D}$ — разница между $\mathcal{C}_{\text{prev}}$ и $\mathcal{C}_{\text{new}}$, представленная в унифицированном формате,
- $\mathcal{H} = \{(\mathcal{C}_{\text{prev}}^{(i)}, \mathcal{C}_{\text{new}}^{(i)}, \mathcal{D}^{(i)}, \mathcal{R}^{(i)})\}_{i=1}^{N}$ — исторический набор данных, содержащий N примеров изменений кода и соответствующих им рецензий $\mathcal{R}^{(i)}$.

Требуется построить модель $\mathcal{M}$, которая для произвольного изменения кода, заданного кортежем $(\mathcal{C}_{\text{prev}}, \mathcal{C}_{\text{new}}, \mathcal{D})$, генерирует релевантный, точный и применимый текстовый комментарий $\mathcal{R}$:

$$\mathcal{R} = \mathcal{M}(\mathcal{C}_{\text{prev}}, \mathcal{C}_{\text{new}}, \mathcal{D}; \Theta, \mathcal{H})$$

где Θ — параметры модели.

Для успешного внедрения в рабочий процесс автоматизированная система генерации комментариев должна удовлетворять нескольким базовым требованиям. Генерируемые тексты должны быть грамматически правильными, осмысленными и точно отражать суть модификаций кода, без включения лишней информации. Ключевым фактором является их практическая направленность: комментарии должны давать разработчику четкий совет — например, указание на необходимость изменения алгоритма или добавления проверок. Кроме того, время генерации должно оставаться приемлемым для интеграции в процесс рецензирования.

В подразделе 3.1.1 описан принцип работы инструмента CommentFinder, проведен анализ набора данных, собранного авторами оригинального исследования.

В ходе ручного анализа его результатов были выявлен ряд проблем, среди которых: некорректное использование библиотечного метода для расчета метрики BLEU, высокий процент неинформативных тривиальных комментариев, повышенные требования к оперативной памяти при решении задачи поиска схожих участков кода, отсутствие предобработки рецензий из набора данных, что негативно сказывается на качестве поиска релевантных рецензий.

В подразделе 3.1.2 были предложены модификации подхода CommentFinder, а именно: оптимизация работы с алгоритмом SequenceMatcher позволила в 20 раз сократить время его выполнения; использование библиотеки tree-sitter для обработки исходного кода и его токенизации повысило качество входных данных для основного алгоритма; замена косинусного расстояния на гармоническое среднее по токенам значительно повысила точность поиска схожих фрагментов кода; был добавлен механизм взвешенного учёта вклада различных типов токенов (переменные, константы, имена методов) в итоговую оценку схожести между участками кода. Эффективность предложенных улучшений отражена в таблице 2.

	Пройдено	Провалено
Исходный	41	42
+ бинаризация	53	30
+ F1-score	70	13
+ веса токенов	76	7

Таблица 2: Результаты тестирования после внесения улучшений

В ходе исследования также была обнаружена существенная проблема, связанная с большим количеством бесполезных и неинформативных рецензий в исторических данных, что потребовало разработки комплексного решения. Был реализован краулер для сбора рецензий с платформ GitHub и Gerrit Review System с целью формирования собственного набора данных. Для фильтрации немотивационных комментариев (т.е. не побуждающих разработчика к изменению кода) были разработаны специальные правила. Например, если после оставления комментария код не был изменен в рамках того же pull request или patchset, такой комментарий не считается мотивационным. Также был предложен набор регулярных выражений для отсеивания неинформативных рецензий, таких как «LGTM but see comment», «Ok, thanks», «Ditto for others», «Ack». В собранном и вручную размеченном наборе данных из 419 примеров успешно прошли фильтрацию 343 рецензии.

В подразделе 3.1.3 проведен сравнительный анализ подходов, представленных в работах AUGER и CodeReviewer. Оба основаны на дообучении языковых моделей архитектуры T5, предварительно обученных на обширных корпусах текстов, связанных с программированием. Ключевые различия заключаются в постановке задач дообучения, составе используемых наборов данных и методах предобработки входных данных.

В ходе ручного анализа нескольких сотен примеров были обнаружены следующие проблемы:

- присутствуют галюцинации модели с генерацией повторяющихся сообщений «this is bad. this is bad. this is bad...»;
- генерируются неинформативные сообщения «Why this change?» или схожие, без предложений к исправления;
- комментарии сохраняют специфичные имена сущностей из проектов обучающей выборки, например «it should be done in the ‘foo‘ method».

Анализ обучающей выборки также выявил признаки аугментации данных методом дублирования примеров с минимальными изменениями, а в некоторых случаях и без изменений кода и рецензии. Кроме того, обнаружено наличие комментариев по типу «Done», «understood», «thanks», «oh my god». Таким образом была идентифицирована системная проблема с качеством обучающих данных и поставлена задача создания классификатора рецензий к исходному коду, который позволит выделять из общего потока собираемых данных только полезные.

В разделе 3.2 описана методология решения задачи фильтрации качественных рецензий и представлены результаты очистки набора данных. Для решения этой задачи применяется многоэтапная фильтрация. На первом этапе используются исключающие эвристические правила для удаления заведомо нерелевантных комментариев (таких как выражения благодарности или краткие подтверждения). Затем предпринимаются попытки создания включающих фильтров для отбора конструктивных рецензий. Однако эвристические подходы и тематическое моделирование (BERTopic) продемонстрировали ограниченную эффективность из-за сильной зависимости от языка программирования и большого количества аномалий в данных. В качестве перспективного направления выделен подход с применением few-shot обучения для интеллектуальной классификации комментариев на основе небольшого размеченного набора данных. В результате комплексной очистки данных из почти 500 тысяч собранных комментариев было отобрано около 100 тысяч качественных примеров для обучения моделей. Анализ результатов финальной классификации показал сохранение значительного количества шумных примеров, включая нерелевантные предложения и избыточно длинные рецензии, охватывающие множество различных тем.

В разделе 3.3 описано решение задачи классификации рецензий исходного кода с целью точного определения тематической направленности комментариев и обеспечения селективной фильтрации релевантной информации. В ходе исследования был создан и представлен в открытом доступе новый набор данных, размером 10045 комментариев, объединяющий четыре открытых датасета и дополненный 3200 комментариями, размеченными вручную. Для разметки была разработана иерархическая классификация, комментарии были разделены на 16 классов, а затем объединены в 5 групп. Итоговая разбиение содержало следующие группы: CodeStyle, Discussion, Development, User, Other.

Комментарии, связанные с удобочитаемостью кода (англ. readability), именованием переменных и визуальным представлением кода, были отнесены к группе

CodeStyle. Рецензии, затрагивающие вопросы разработки, оптимизации, отладки, ошибок, дефектов, функциональности и архитектурных решений, классифицированы как Development. Комментарии, содержащие обсуждения без указания конкретных проблем либо заключающие вердикты, отнесены к категории Discussion. В группу User включены сообщения, касающиеся вопросов пользовательского опыта: поддержки использования кода, вывода ошибок и документации. В результате предложенная классификация охватила большинство комментариев, остальные были отнесены к категории Other.

Проведено сравнительное тестирование различных методов классификации, включая RandomForest, FastText, TfIdf, Word2Vec, BERT и CodeBERT. Модели, использующие классические векторные представления (CountVectorizer и TfidfVectorizer), показали результаты на 10 пунктов ниже по метрике F1-macro по сравнению с моделями на основе Word2Vec и FastText. Наилучший результат (F1-macro = 0.775) продемонстрировали модели на архитектуре трансформеров BERT. Также проведена оценка по вычислительным затратам. Полученные результаты подтверждают применимость автоматической классификации рецензий для выявления обсуждений, требующих повышенного внимания разработчиков.

В разделе 3.4 описано исследование определения наилучшей метрики семантической схожести комментариев. Основной проблемой является оценка смысловой, а не текстуальной близости рецензий. Классические метрики (BLEU, ROUGE) оказались неэффективны, так как не улавливают семантику. Для решения данной проблемы создана тестовая база из 105 троек комментариев с экспертной оценкой семантической близости.

Экспериментальное сравнение метрик показало, что наилучший результат (83/105 корректных определений) демонстрирует метрика BertScore на модели Deberta-XLarge с применением предобработки текста. Однако ввиду высокой вычислительной сложности данного подхода был выбран компромиссный вариант: косинусное расстояние между эмбеддингами модели CodeT5 (110M параметров) с предобработкой текста, показавший результат 82/105 корректных определений. Предобработка текста статистически значимо улучшала результаты всех исследуемых метрик.

В разделе 3.5 описана разработка системы автоматического рецензирования на основе больших языковых моделей. Проведен разбор четырех существующих реализаций аналогичных систем. В рамках работы также освещаются: архитектура предложенной системы, поэтапная эволюция и настройка используемых инструкций (промптов, англ. prompt engineering), механизмы фильтрации выходных данных для повышения их качества и результаты комплексной оценки эффективности генератора на репрезентативных наборах реальных производственных данных. В качестве базового решения использован подход из публикации DeepCRCEval, в котором модель получает на вход унифицированный формат (unidiff) измененного метода в качестве контекста исходного кода.

В ходе применения и анализа результатов базового решения были выделены следующие группы неприменимых рецензий:

1. Избыточные и дублирующие предложения — рекомендации, уже реализованные в коде или добавляющие излишние проверки;
2. Нарушение принципа разумной достаточности — излишнее усложнение кода, например, повсеместное добавление try-except;
3. Игнорирование проектного контекста — предложения, противоречащие установленным практикам проекта;
4. Необоснованные рекомендации — комментарии с неявной мотивацией или ложным определением проблем.

Несмотря на значительный процент таких предложений, анализ показывает, что оставшиеся содержат релевантные замечания. Многие из приемлемых рецензий качественно идентифицируют реальные проблемы кода и предлагают обоснованные пути улучшения. Важно отметить, что большая часть выявленных проблем носит системный характер и может быть устранена через донастройку инструкций и внедрение дополнительного контекста.

В качестве первого улучшения был применен структурированный вывод (англ. Structured Output), позволяющий генерировать выходные данные в строго заданном формате (JSON, XML или таблицы). Такой метод обеспечивает предсказуемость, машинную читаемость и упрощает интеграцию с внешними системами (например, базами данных или API).

Следующим шагом для улучшения восприятия моделью исходного кода и внесенных изменений было расширение контекста с уровня конкретного метода до уровня файла. В случае добавления нового файла в коммите, он передается модели целиком, что позволяет генерировать рецензии для всего файла, а не для отдельных методов. Для измененных файлов в модель передается соответствующий метод вместе со всеми изменениями в файле в унифицированном формате. Для модифицированных методов предлагается подавать на вход модели все соответствующие изменения в файле.

В работе BitsAI представлена таксономия правил рецензирования исходного кода, состоящая из четырех категорий. На основе внутренних обсуждений и тематического моделирования данный список был пересмотрен. Итоговые категории для системы автоматического рецензирования: Best Practices, Bug Fixes, Optimization и Modern Python Features. В системный промпт также добавлены требования для предотвращения ложных выводов, когда необходим анализ внешнего контекста, и правила валидации предлагаемых изменений.

Внедрение ограничений значительно сократило объем генерируемых рецензий, сузив область применения системы. Для расширения кругозора модели и контроля стиля генерации применяется few-shot learning с добавлением в промпт примеров. Было опробовано два подхода: добавление трёх фиксированных примеров и автоматический подбор трёх релевантных примеров с помощью семантического поиска описанного в подразделе 3.1.1. Система с автоматическим подбором значительно превосходит по показателям рецензента с фиксированными примерами.

Следующим использованным подходом является «модель как судья» (lm-as-a-judge) — это метод автоматической оценки текстовых ответов, сгенерированных большими языковыми моделями (LLM), с использованием другой LLM в качестве судьи.

Первоначальная инструкция, которая взята с работы DeepCRCEval показала крайне низкую согласованность с человеческой оценкой (коэффициент каппа Коэна $< 0{,}01$), однако чаще отсеивала низкокачественные комментарии по сравнению с качественными. После дальнейшей доработки инструкции с фокусом на пять ключевых категорий (Understanding, Relevance, Actionability, Accuracy, Clarity) и введения трехуровневой шкалой оценок («ошибочный» / «средний» / «применимый») согласие выросло до 0.14. С такими результатами метод нельзя использовать в качестве «судьи», но можно в качестве постобработки: он позволяет фильтровать комментарии, отсеивая только оценки категории «ошибочно» (мягкий режим) или оставляя исключительно оценки «применимый» (строгий режим).

Разработанные в главе 2 детекторы антипаттернов исходного кода органично встраиваются в инструмент рецензирования, поскольку все необходимые для их работы входные данные уже используются алгоритмом рецензирования. При обнаружении антипаттернов выходные данные детекторов поступают в качестве дополнительного контекста для большой языковой модели. Общая архитектура разработанного программного комплекса автоматического рецензирования исходного кода, интегрирующего работу детекторов антипаттернов представлена на рисунке 3.

В разделе 3.6 описаны подходы к дообучению большой языковой модели на основе пользовательских оценок генерируемых рецензий: контролируемое дообучение (SFT), обучение с подкреплением на основе человеческих предпочтений (RLHF), прямая оптимизация предпочтений (DPO) и оптимизация по методу Канемана-Тверского (KTO). В связи с ограниченным объемом собранных данных и сложностью формирования для каждого случая парных примеров (положительных и отрицательных) был выбран подход KTO, допускающий использование несвязанных примеров работы модели с бинарной оценкой релевантности комментария заданному контексту. Экспериментально установлено, что соотношение положительных и отрицательных примеров в обучающей выборке должно быть приблизительно равным, поскольку значительное преобладание отрицательных примеров снижает эффективность дообучения.

В разделе 3.7 рассматриваются практические аспекты применения систем автоматического рецензирования в современные системы непрерывной разработки и интеграции. Исследуются подходы к реализации для платформ GitHub и Gerrit.

В разделе 3.8 приведены выводы по анализу методов автоматического рецензирования кода, выявлены их ограничения, предложены решения, разработана система авторецензирования на основе LLM с интеграцией детекторов антипаттернов для интеграции в процесс разработки.

Глава 4 посвящена экспериментальному исследованию систем автоматиче-

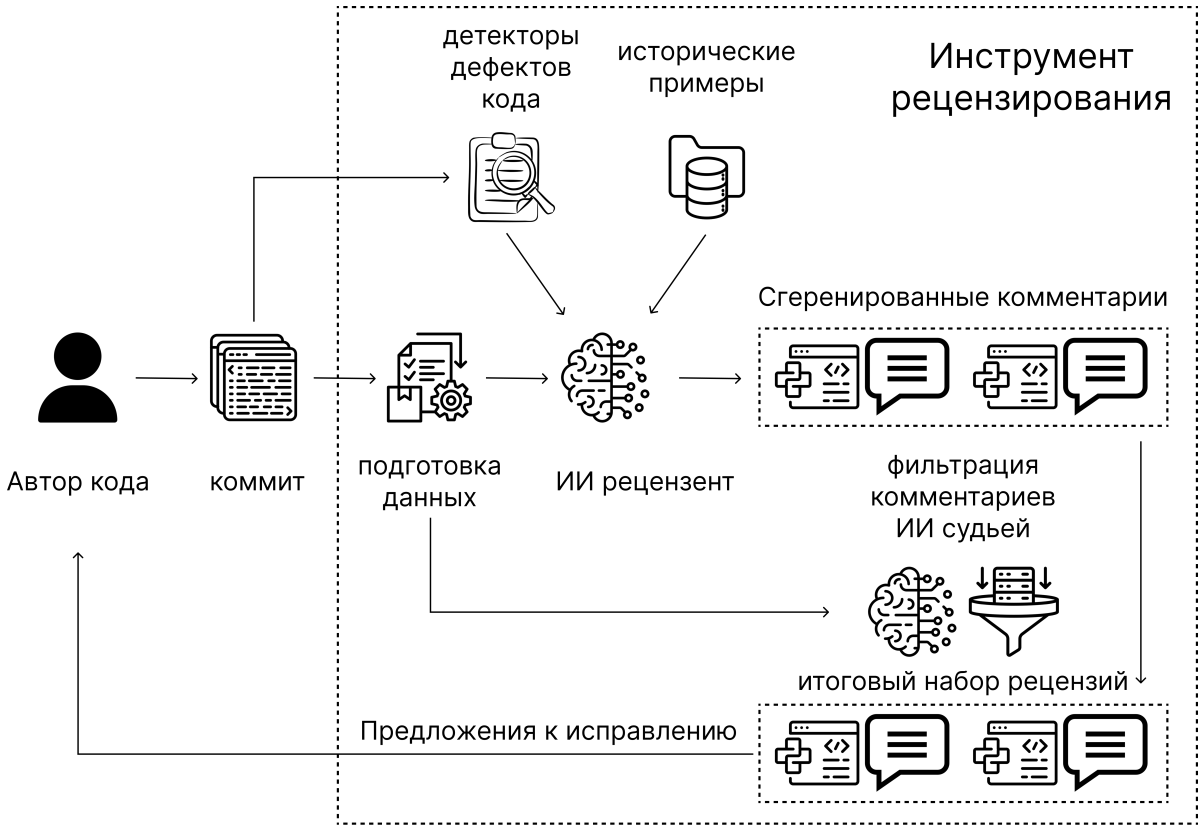

Рисунок 3: Схема работы системы автоматического рецензирования

ского рецензирования исходного кода с использованием больших языковых моделей. Для оценки разработанных инструментов была реализована система тестирования и проведена серия экспериментов.

В разделе 4.1 представлены результаты оценки качества работы систем автоматического рецензирования с использованием методики, описанной в работах CommentFinder и AUGER. Оценка проводилась автоматически с помощью метрик схожести, представленных в разделе 3.3, на большом наборе примеров. Тестовый набор данных был сформирован на основе рецензий из открытых Gerrit-репозиториев (Opendev, OpenBMC, Chromium, Android). Данные разделили на обучающую и валидационную выборки: по временному признаку (старые/новые) и по проектам на непересекающиеся множества. Такой подход позволил исследовать важность исторических данных целевого проекта в сравнении с данными из внешних источников.

Валидационная выборка содержит 15 000 рецензий (по 5 000 на Python, Java, C++), прошедших классификацию по категориям и получивших метку Development. Обучающая выборка включает 300 000 примеров. Дополнительно созданы уменьшенные подвыборки (150 000, 90 000, 45 000) для анализа зависимости качества от объема данных. Из каждой подвыборки были выделены комментарии классифицированные как Development.

На указанных данных были оценены подходы, описанные в разделах 3.1.1 и 3.1.2. В качестве оценочных метрик использовались обще-

принятая метрика BLEU-4 и выбранная в разделе 3.4 метрика косинусного сходства (cosine_similarity) на векторных представлениях модели CodeT5_110m_embedding.

Из полученных результатов валидации реализованных инструментов на основе CommentFinder, AUGER и CodeReviewer можно сделать вывод о росте результатов при увеличении выборки: с 0,2422 до 0,2786 (на 15%) на cosine_sim метрике для топ-1 и с 0,37 до 0,531 (на 43%) на метрике BLEU4 для топ-1. Для топ-10 BLEU4 вырос с 1.56 до 2.04 (на 30,7%), а cosine_sim – с 0,3898 до 0,4286 (на 10%). Также следует отметить, что практически во всех случаях метрики, полученные при обучении на отфильтрованной по типу сообщений выборке, демонстрируют более высокие результаты по сравнению с общей выборкой, несмотря на меньший объем данных.

Также стоит отметить время работы инструментов — если работа MLCF на валидационной выборке занимает около 3 минут, то для подходов на базе AUGER и CodeReviewer — около 90 минут на то же количество примеров.

В разделе 4.2 приводится этап тестирования и оценки результатов работы разработанных систем при реальном использовании. Оценка качества подобных систем в автоматическом режиме по метрикам схожести сообщений имеет довольно посредственное отношение к итоговой целевой метрике — эффективности/полезности данных комментариев при реальном рецензировании коммитов на проектах. Для этого был разработан небольшой плагин для локального Gerrit сервера, который позволяет ставить отметку для robot_commit, в виде которых публикуются результаты работы системы автоматического рецензирования. Были установлены три типа оценки: применимые — хорошие предложения от системы, к которым автор кода согласен прислушаться и исправить; средние — неплохие, но не обязательные к исправлению в данном контексте; ошибочные. Также при анализе результатов использовалась сумма применимых и средних комментариев как «неплохие».

Результаты разметки результатов вышеупомянутых MLCF, AUGER и CodeReviewer, а также базовой реализации системы с использованием языковой модели из раздела 3.5 таким методом предоставлены в таблице 3. Модель LLM значительно превосходит традиционные подходы (MLCF, AUGER, CodeReviewer) по качеству генерации рецензий. Доля «неплохих» (применимых и средних) комментариев у LLM составляет 23.13%, что более чем в два раза выше показателей других моделей (8.33-9.79%).

При этом все модели демонстрируют высокий процент ошибочных рецензий (76-92%), что указывает на необходимость дальнейшего улучшения систем автоматического рецензирования. LLM генерирует наибольшее количество применимых рецензий (9), что подтверждает ее практическую полезность, несмотря на общую проблему качества.

В том же тестовом окружении было проведено сравнение ряда языковых моделей: Nous-Hermes-2-8x7B, Qwen2.5-Coder-7B, Qwen2.5-Coder-32B, Qwen2.5-72B, DeepSeek-Coder-7B, DeepSeek-Coder-32B, Llama 3.1-70B, Code Llama-70B. По ито-

	Всего	Ошибочные	Средние	Применимые	Неплохие, %
MLCF	120	110	9	1	8.33%
AUGER	296	267	21	8	9.79%
CodeReviewer	100	91	8	1	9%
LLM	160	122	28	9	23.13%

Таблица 3: Результаты рецензирования на реальных проектах

гам разметки результатов было решено взять Qwen2.5-Coder-32B, так как она показала наилучшие результаты для моделей размером до 32 миллиардов параметров.

В разделе 4.3 приведены результаты оценки различных этапов улучшения промптов и фильтраций описанных в разделе 3.5. Для получения более быстрого отклика о качестве внедряемых улучшений был собран набор из 10 патчсетов, извлеченных из локального Gerrit сервера. Данный корпус включает 11 новых файлов и 25 модифицированных файлов, содержащих 69 добавленных и 54 измененных метода, реализованных на языке Python. Аннотирование данных выполнялось двумя авторами исходного кода с целью точной классификации предложений, сгенерированных LLM. Межэкспертная согласованность при ручной разметке, измеренная каппой Коэна, составила 0.72.

В таблице 4 приведены результаты разметки сгенерированных рецензий до применения фильтров. Результаты показывают, что подход с улучшенной инструкцией и автоматическим подбором примеров значительно превосходит базовую версию и метод с фиксированными примерами. Точность полностью применимых рецензий увеличилась в 2 раза, а доля неплохих — в ~ 1.4 раза.

В таблице 5 приведены результаты применения мягкой и жесткой пост-фильтрации сгенерированных рецензий, соответственно. Жесткая пост-фильтрация позволяет достичь высокой точности (до 37% применимых рецензий), но сильно сокращает общее количество комментариев. Сочетание улучшенной инструкции с фильтрацией дает наибольший эффект, повышая точность в 2.5 раза по сравнению с базовым подходом.

Таблица 4: Результаты оценки качества рецензий (без фильтров)

Метод	Плохие	Средние (шт, %)	Применимые (шт, %)	Общее
Базовая инструкция	95	33, 22	22, 14.7	150
Расширенный контекст	74	27, 21.9	22, 17.89	123
Улучшение требований	**48**	16, 17.77	**26**, **28.89**	90
Фиксированные примеры	92	37, 24.6	21, 14.00	150
Автоподбор примеров	60	**38**, 31.14	24, 19.67	122

Таблица 5: Результаты оценки качества рецензий при фильтрации

Метод	Плохие	Средние (шт, %)	Применимые (шт, %)	Общее
Мягкая фильтрация				
Базовая инструкция	61	26, 24.52	19, 17.92	106
Расширенный контекст	42	21, 26.25	17, 21.25	80
Улучшение требований	**29**	14, 21.87	**21**, **32.81**	64
Фиксированные примеры	34	27, 34.6	17, 21.79	78
Автоподбор примеров	40	**31**, 33.96	**21**, 22.83	92
Жесткая фильтрация				
Базовая инструкция	42	19, 25	15, 19.74	76
Расширенный контекст	22	14, 27.4	15, 29.41	51
Улучшение требований	**21**	13, 24.07	**20**, **37.04**	54
Фиксированные примеры	24	12, 25.53	11, 23.40	47
Автоподбор примеров	26	**22**, 33.3	18, 27.27	66

Проведено тестирование дообучения модели на наборах обратной связи объемами по 50, 100 и 150 положительных и отрицательных примеров. Результаты, представленные на рисунке 4, показывают, что дообучение на 100 примерах вызывают снижение качества, тогда как 200-300 примеров значительно улучшают генерацию, в том числе благодаря более эффективной фильтрации моделью-судьей. При 300 примерах жесткая фильтрация демонстрирует баланс применимых и средних рецензий (по 38,8%) при снижении доли некорректных до 22,4%. Мягкая фильтрация сохраняет больше рецензий (65 против 49) при качестве 30,7% применимых комментариев.

Также показаны показаны результаты сравнения базовой реализации автоматического рецензента с наилучшей из конфигураций на 1000 примерах из набора данных CRer, предоставленной авторами DeepCRCEval. Модель с улучшенной инструкцией генерировала несколько рецензий для каждого фрагмента кода, а базовая — одну. Релевантность оценивалась системой, предложенной авторами DeepCRCEval, ранжирующей комментарии по соответствию коду. Средний обратный ранг (MRR) составил 0.915 против 0.585 у базовой реализации, что свидетельствует о её гораздо более высокой эффективности в генерации релевантных рецензий.

В разделе 4.4 представлены выводы по оценке качества разработанной системы автоматического рецензирования.

В **заключении** приведены основные результаты работы:

- Предложен метод сбора размеченных примеров и набор данных различных типов дефектов кода, извлеченный из истории разработки проектов;
- Предложен метод детектирования антипаттернов исходного кода, основанный на анализе исходного кода и методах машинного обучения.

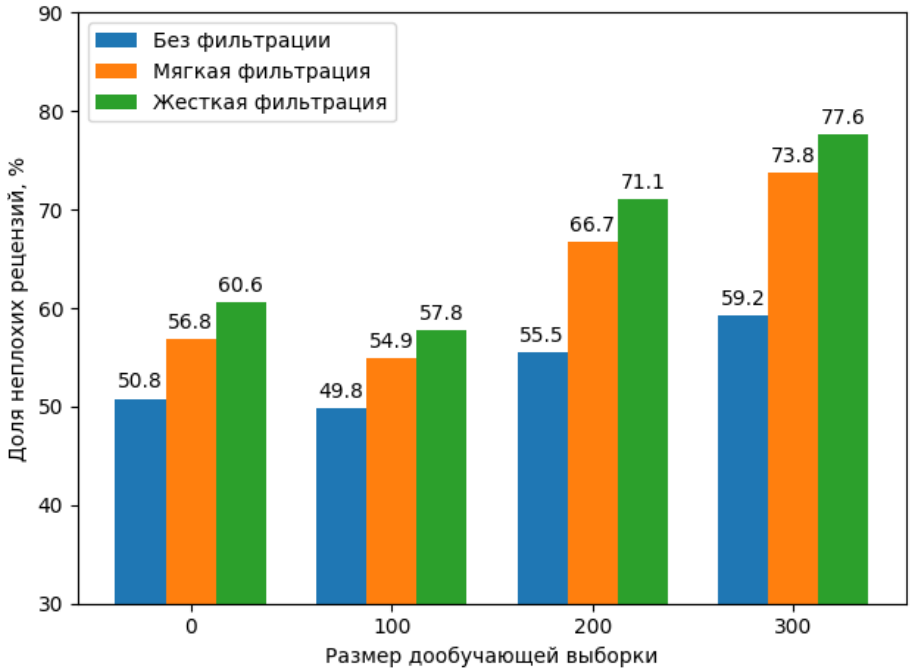

Рисунок 4: Доля неплохих рецензий при дообучении модели

- Предложен метод автоматического рецензирования, основанный на использовании методов глубокого обучения и больших языковых моделей.
- Предложен и реализован программный комплекс, интегрирующий работу детекторов дефектов кода в систему автоматического рецензирования.

Публикации соискателя по теме диссертации

Публикации в журналах ВАК:

1. *Качанов В.В.* Автоматическая генерация рецензий к коду: эволюция инструкций и интеллектуальная фильтрация //Труды Института системного программирования РАН. — 2025. — Т. 37, №. 4, часть 2. — С. 117-132.
2. *Качанов В. В., Хитрова А. С., Марков С. И.* Извлечение именованных сущностей из рецензий к исходному коду // Труды ИСП РАН. — 2023. — Т. 35, № 5. — С. 193–214.
3. *Качанов В. В., Марков С. И., Цурков В. И.* Машинное обучение в проблеме технического долга программного обеспечения // Известия Российской академии наук. Теория и системы управления. — 2023. — № 4. — С. 98–104.
4. *Качанов В.В., Ермаков М.К., Панкратенко Г.А., Спиридонов А.В., Волков А.С., Марков С.И.* Технический долг в жизненном цикле разработки ПО: запахи кода // Труды ИСП РАН. — 2021. — Т. 33, № 6. — С. 95–110.
5. *Савченко В. В., Сорокин К. С., Бронштейн И. Е., Волков А. С., Качанов В. В., Панкратенко Г. А., Ермаков М. К., Марков С. И., Спиридонов А. В., Александров И. В.* Nobrainer: инструмент преобразования C/C++ кода на основе примеров // Программирование. — 2020. — № 5. — С. 33–46.

Публикации в тезисах конференций:

6. *Петрова П.А. Марков С.И. Качанов В.В.* Создание набора данных для комбинированной классификации рецензий исходного кода // Интеллектуализация обработки информации. Тезисы докладов 15-й международной конференции. — 2024.

Свидетельства о государственной регистрации программы для ЭВМ

7. Система рецензирования исходного кода с применением языковых моделей / В.В. Качанов ; ФГБУН Институт системного программирования РАН. — № 2025680617; заявл. 03.07.2025.