МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра математической кибернетики и компьютерных наук

**РАЗРАБОТКА ИНТЕГРАЦИОННОГО РЕШЕНИЯ ДЛЯ АНАЛИЗА
КАЧЕСТВА КОДА И ЕГО РЕФАКТОРИНГА НА ОСНОВЕ БОЛЬШИХ
ЯЗЫКОВЫХ МОДЕЛЕЙ**

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Свинарева Егора Антоновича

Научный руководитель
доцент, к. ф.-м. н.

М. И. Сафрончик

Заведующий кафедрой
к. ф.-м. н., доцент

С. В. Миронов

Саратов 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Теоретические основы анализа качества кода и применения LLM.....	4
1.1 Понятие и критерии качества программного кода	4
1.2 Методы анализа и автоматизированного улучшения программ- ного кода.....	4
1.2.1 Статический и динамический анализ	4
1.2.2 Инструменты автоматического улучшения кода	5
1.3 Взаимодействие с большими языковыми моделями (LLM) в кон- тексте автоматизации анализа и рефакторинга кода	5
1.4 Предложенный метод решения задачи.....	6
2 Реализация системы анализа и улучшения качества кода на основе языковой модели	7
2.1 Анализ требований к системе	7
2.2 Архитектура системы	7
2.3 Реализация модуля отчетов	8
2.4 Реализация модуля рецензирования Merge Request	9
2.5 Реализация модуля рефакторинга	10
2.6 Сборка и развертывание веб-сервиса	10
2.7 Тестирование работы системы.....	11
ЗАКЛЮЧЕНИЕ	12

ВВЕДЕНИЕ

В последние годы особую актуальность приобрела проблема обеспечения высокого качества программного кода, учитывая растущую сложность программных систем. Ручное отслеживание качества кода сопряжено с временными затратами и подвержено человеческому фактору. В связи с этим возрастает интерес к использованию инструментов на основе искусственного интеллекта, в частности, больших языковых моделей (LLM), таких как ChatGPT, продемонстрировавших способность к генерации, интерпретации и рефакторингу программного кода.

Целью данной работы является исследование возможности внедрения интеллектуального решения на основе LLM в процесс анализа и улучшения качества кода, а также разработка программного прототипа такой системы. Основная идея заключается в интеграции анализа качества кода с возможностью автоматической генерации рекомендаций и правок с использованием LLM, при этом обеспечивая возможность развертывания системы во внутреннем контуре организации с соблюдением требований безопасности.

Для достижения цели были поставлены следующие задачи:

- изучить существующие подходы и метрики оценки качества программного кода;
- рассмотреть традиционные и современные инструменты анализа и рефакторинга кода, включая ИИ-решения;
- исследовать потенциал LLM в задачах анализа и улучшения кода;
- разработать архитектуру приложения с интеграцией в процессы разработки;
- реализовать модуль рецензирования кода в рамках Merge Request;
- разработать модуль фонового рефакторинга;
- провести тестирование системы и сформулировать выводы.

Структура и объем работы. Бакалаврская работа состоит из раздела с используемыми определениями и сокращениями, введения, 2 разделов, заключения, списка источников и 3 приложений. Общий объем работы — 62 страницы, из них 54 страниц — основное содержание, включая 14 рисунков, Flash-носитель в качестве приложения, список использованных источников — 22 наименования.

1 Теоретические основы анализа качества кода и применения LLM

1.1 Понятие и критерии качества программного кода

Качество кода — совокупность характеристик, определяющих его читаемость, сопровождаемость, тестируемость и масштабируемость. В условиях Agile и DevOps подходов его обеспечение становится критически важным. Основные критерии:

1. **Читаемость** — понятные имена, логичная структура, единый стиль;
2. **Сопровождаемость** — модульность, наличие тестов, слабая связанность компонентов;
3. **Переиспользуемость** — соблюдение принципов DRY и инкапсуляции;
4. **Надёжность** — обработка ошибок, устойчивость к исключениям;
5. **Расширяемость и тестируемость**

1.2 Методы анализа и автоматизированного улучшения программного кода

Современные программные системы стремительно увеличиваются в объёмах и сложности. Это требует постоянного мониторинга качества исходного кода, особенно в условиях активной командной разработки и быстрого внедрения изменений. В этом контексте критически важную роль играют методы анализа и инструменты автоматического улучшения кода, позволяющие выявлять ошибки, стандартизировать кодовую базу и снижать технический долг.

Методы анализа кода можно условно разделить на два ключевых подхода: статический анализ (до выполнения программы) и динамический анализ (во время выполнения). Также существуют инструменты автоматического рефакторинга и стилистической нормализации, как традиционные, так и основанные на искусственном интеллекте.

1.2.1 Статический и динамический анализ

Статический анализ представляет собой проверку исходного кода без его исполнения. Он позволяет обнаруживать:

- синтаксические ошибки;
- нарушения стиля кодирования;
- потенциальные уязвимости;
- дублирование кода;
- неиспользуемые переменные и функции;

- архитектурные недостатки.

Преимущество статического анализа — его универсальность и применимость на ранних этапах разработки. Он не требует запуска программы и может быть интегрирован в пайплайны CI/CD. Однако статический анализ не способен обнаружить ошибки, связанные с поведением программы при выполнении. Кроме того, возможны ложные срабатывания (false positives), когда корректный код ошибочно помечается как дефектный.

Динамический анализ проводится во время выполнения программы. Он позволяет выявить:

- утечки памяти;
- ошибки многопоточности;
- логические ошибки, проявляющиеся при определённых входных данных;
- проблемы с производительностью.

Он требует полноценной тестовой среды и охватывает только те участки кода, которые были задействованы в тестах. Это ограничивает полноту анализа, но повышает его реалистичность: анализируются ошибки, реально возникающие при выполнении.

Идеальной практикой является совместное использование статического и динамического анализа. Это обеспечивает как широкий охват кода, так и проверку его реального поведения, снижая риски и повышая надёжность системы.

1.2.2 Инструменты автоматического улучшения кода

Автоматизация улучшения кода осуществляется с помощью форматтеров и линтеров, которые выравнивают стиль и устраняют типовые ошибки, но работают по жёстким правилам. Более гибкие и мощные решения построены на больших языковых моделях (например, GitHub Copilot, DeepSeek), которые способны контекстно анализировать и переписывать код, предлагать улучшения, писать тесты и сопровождать разработчика на всех этапах. Такие инструменты интегрируются в IDE и CI/CD, поддерживают запросы на естественном языке и позволяют глубже автоматизировать процесс рефакторинга.

1.3 Взаимодействие с большими языковыми моделями (LLM) в контексте автоматизации анализа и рефакторинга кода

Большие языковые модели, такие как GPT, Codex и DeepSeek, позволяют автоматизировать анализ и рефакторинг кода, генерируя предложения по

улучшению, тесты и документацию. Они интегрируются в процессы разработки через API или как плагины в IDE, обеспечивая интерактивную поддержку в реальном времени. Модель способна учитывать контекст проекта и обрабатывать задачи, сформулированные на естественном языке, тем самым снижая порог входа и ускоряя разработку.

Особое внимание уделяется интеграции LLM с классическими инструментами анализа. Используя механизм tool call, модель может обрабатывать результаты статических анализаторов, таких как SonarQube и Joern, и формировать обоснованные рекомендации. Это делает процесс ревью глубже и понятнее. Возможность локального развертывания модели позволяет применять решение в средах с повышенными требованиями к безопасности и конфиденциальности.

1.4 Предложенный метод решения задачи

Предлагаемый подход сочетает возможности статического анализа и больших языковых моделей (LLM) для автоматизации анализа и улучшения кода. Система отслеживает создание Merge Request в GitLab, извлекает изменённый код, проводит его анализ с использованием инструментов вроде SonarQube и Joern, а затем формирует запрос к LLM, содержащий код, результаты анализа и контекст. Модель генерирует рекомендации по улучшению, которые автоматически добавляются в обсуждение Merge Request.

Ключевое преимущество — возможность локального развёртывания модели на инфраструктуре заказчика, что особенно важно для организаций с ограничениями на использование облачных сервисов. Это обеспечивает соблюдение требований по безопасности и конфиденциальности при сохранении функциональности интеллектуального анализа.

Дополнительно в системе реализован модуль фоновой рефакторинга, который периодически сканирует кодовую базу и предлагает улучшения для устаревших участков. Такой комплексный подход позволяет не только анализировать текущие изменения, но и системно поддерживать качество кода во всём проекте без вмешательства в привычный рабочий процесс команды.

2 Реализация системы анализа и улучшения качества кода на основе языковой модели

2.1 Анализ требований к системе

Перед проектированием программной системы был проведён анализ требований — функциональных и нефункциональных, определяющих цели, поведение и технические ограничения разрабатываемого решения.

Функциональные требования включают:

1. интеграцию с GitLab и отслеживание событий создания Merge Request;
2. запуск анализа изменённого кода с использованием статических анализаторов;
3. генерацию рекомендаций по улучшению с помощью LLM;
4. автоматическое добавление предложений в интерфейс Merge Request;
5. реализацию фонового рефакторинга всего репозитория по расписанию.

Нефункциональные требования касаются архитектуры, производительности и удобства эксплуатации:

1. модульность и расширяемость системы (поддержка новых языков, анализаторов, моделей);
2. способность обрабатывать крупные репозитории и масштабироваться при росте нагрузки;
3. минимальное вмешательство в текущие процессы разработки;
4. устойчивость к ошибкам и логирование всех операций;
5. простота настройки и развёртывания.

Результатом анализа стало формирование технического задания, определившего архитектуру системы, выбор технологий и подходы к интеграции в существующую инфраструктуру.

2.2 Архитектура системы

На основе выявленных требований была спроектирована модульная, распределённая архитектура системы, обеспечивающая гибкость, масштабируемость и устойчивость. Основой взаимодействия между компонентами стал брокер сообщений Apache Kafka, который обеспечивает асинхронную передачу данных и слабую связанность модулей. Это позволяет системе оставаться стабильной даже при временных перебоях или высоких нагрузках.

Центральным элементом системы является API Gateway, реализованный

на NGINX. Он отвечает за маршрутизацию запросов, аутентификацию и обработку ошибок. Взаимодействие с внешними сервисами и пользователями также проходит через этот шлюз.

Архитектура включает три ключевых модуля:

1. Модуль рецензирования Merge Request;
2. Модуль рефакторинга;
3. Модуль формирования отчётов;

Для хранения данных используются две СУБД: PostgreSQL — для MR и результатов анализа, и MongoDB — для отчётов и метаданных. Такое разделение позволило оптимально распределить нагрузку и использовать преимущества каждой базы данных.

Все компоненты системы обмениваются данными через события, публикуемые в Kafka. Например, при создании нового Merge Request модуль рецензирования получает событие, запускает анализ, отправляет результат в модуль отчётов, который обращается к LLM. После получения ответа, рекомендации направляются обратно в GitLab в виде комментариев.

Также предусмотрена возможность фонового рефакторинга: система периодически сканирует репозиторий на наличие потенциальных проблем, таких как дублирование, высокая сложность или неэффективные конструкции. Результаты такого анализа тоже направляются в отчётный модуль и далее к языковой модели, после чего могут быть оформлены как отдельный Merge Request.

Выбранная архитектура делает систему легко расширяемой и адаптируемой под различные среды разработки. Она поддерживает интеграцию новых языков программирования, анализаторов и LLM-провайдеров без значительных изменений в существующих компонентах.

2.3 Реализация модуля отчетов

Модуль отчётов отвечает за сбор, обработку и хранение данных, полученных в результате анализа кода. Он служит связующим звеном между анализирующими компонентами и языковыми моделями, обеспечивая структурированное представление информации.

Основная задача модуля — преобразование сырых данных в удобный формат, который может быть передан для дальнейшей интерпретации. Отчёты содержат информацию о найденных проблемах, предложенных решениях и дополнительных данных, необходимых для понимания контекста изменений.

Данные сохраняются в MongoDB, что обеспечивает гибкость хранения и возможность работы с неструктурированными и полуструктурированными данными. Модуль также обеспечивает выборку и подготовку информации для внешних систем, таких как интерфейс пользователя или LLM.

Таким образом, модуль отчётов выступает как центральный элемент, обеспечивающий согласованность и доступность данных на всех этапах работы системы.

2.4 Реализация модуля рецензирования Merge Request

Модуль рецензирования Merge Request отвечает за автоматический анализ изменений в коде при создании или обновлении запроса на слияние (Merge Request) в системе GitLab. Его задача — выявить потенциальные проблемы, такие как нарушение принципов проектирования, снижение читаемости, ухудшение производительности или возможные ошибки.

Модуль интегрирован с системой контроля версий и активируется при событиях, связанных с Merge Request. Он извлекает контекст изменений, включая сам код и метаданные: имя автора, целевую ветку, историю изменений и другие данные. Эти сведения позволяют провести более точный и контекстуальный анализ.

После сбора информации модуль формирует задание для дальнейшей обработки, которое передаётся через брокер сообщений другим компонентам системы. В частности, данные направляются в модуль отчётов, а затем — на интерпретацию языковой модели.

Одним из ключевых преимуществ модуля является его способность работать в фоновом режиме, не мешая основному процессу разработки. Это позволяет сохранять прозрачность и стабильность рабочих процессов, не блокируя слияние кода даже при наличии предложений по улучшению.

В результате анализа система формирует рекомендации, которые могут быть представлены как комментарии к Merge Request или как отдельное предложение по рефакторингу. Такой подход помогает повышать качество кода уже на ранних этапах разработки, снижает количество ошибок и улучшает общую культуру написания кода в команде.

Таким образом, модуль рецензирования Merge Request играет центральную роль в обеспечении автоматического контроля качества кода и поддерживает непрерывное улучшение проекта.

2.5 Реализация модуля рефакторинга

Модуль рефакторинга отвечает за автоматическое улучшение структуры существующего кода без изменения его поведения. Его основная задача — выявлять участки, требующие улучшения, и предлагать решения, которые делают код более читаемым, поддерживаемым и эффективным.

В отличие от модуля рецензирования, который работает в контексте конкретного Merge Request, модуль рефакторинга может запускаться как по событию, так и в фоновом режиме. Это позволяет периодически анализировать состояние всей кодовой базы и находить потенциальные проблемы, такие как дублирование, высокая сложность или нарушение принципов проектирования.

Модуль взаимодействует с системой контроля версий, извлекая данные о репозитории и файлах, подлежащих анализу. Полученные данные передаются для анализа и последующей интерпретации языковой модели. При этом система учитывает не только текущий файл, но и связанные с ним части проекта, чтобы предложить изменения, согласованные с общей архитектурой.

Результатом работы модуля являются рекомендации по улучшению кода, оформленные в виде отчёта. Эти предложения могут быть представлены как отдельный Merge Request, что позволяет разработчикам легко ознакомиться с изменениями и принять их при необходимости.

Таким образом, модуль рефакторинга обеспечивает постоянное улучшение качества кода вне зависимости от активности разработчиков, снижая риск накопления технического долга и повышая долгосрочную стабильность проекта.

2.6 Сборка и развертывание веб-сервиса

После завершения разработки всех модулей системы был реализован процесс её сборки и последующего развертывания. Основной задачей стало создание устойчивой и воспроизводимой инфраструктуры, которая позволяет легко масштабировать систему и поддерживать её в рабочем состоянии.

Для обеспечения согласованности между средами разработки, тестирования и эксплуатации использован подход на основе контейнеризации. Все компоненты системы упакованы в Docker-образы, что гарантирует одинаковое поведение приложения независимо от окружения. Это позволило избежать проблем, связанных с различиями в конфигурациях серверов, а также упростило управление зависимостями.

Для оркестрации контейнеров применён Docker Compose.

В качестве среды для размещения сервиса могут использоваться как локальные серверы, так и облачные платформы. Это делает систему универсальной и адаптируемой под различные условия эксплуатации.

Таким образом, этап сборки и развертывания обеспечил надёжное функционирование системы в реальных условиях, сделал её устойчивой к изменениям нагрузки и удобной в сопровождении.

2.7 Тестирование работы системы

Тестирование системы проводилось с целью проверки корректности её работы и практической применимости в процессе анализа и улучшения кода. Основное внимание уделялось тому, как система реагирует на создание Merge Request, извлекает изменения, проводит анализ и формирует рекомендации с помощью языковой модели. Была проверена стабильность взаимодействия между модулями, корректность вызовов LLM и точность обработки входных данных.

Особое внимание в тестировании уделялось оценке успешности генерируемого рефакторинга. Результаты предложений от LLM анализировались вручную с точки зрения читаемости, стилистического качества, соответствия стандартам кодирования и реального улучшения архитектуры. Во многих случаях рекомендации модели оказывались релевантными: упрощали сложные конструкции, устраняли дублирование или повышали читаемость. Однако встречались и менее точные предложения, особенно при недостатке контекста или неоднозначной логике исходного кода.

Также была протестирована работа фонового модуля рефакторинга: он корректно сканировал кодовую базу и генерировал предложения для устаревших или перегруженных участков. Итоговое тестирование подтвердило, что система может эффективно использоваться как вспомогательный инструмент при code review и систематическом улучшении кода, при этом не вмешиваясь в привычный рабочий процесс команды.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была достигнута поставленная цель — исследована возможность внедрения решений на основе искусственного интеллекта в процесс анализа и улучшения качества программного кода. Также разработан прототип программной системы, реализующей данный подход. Все задачи исследования успешно выполнены.

Проведён анализ существующих методов оценки качества кода и средств его автоматического анализа. Изучены традиционные подходы, такие как статический и динамический анализ, а также современные решения, основанные на использовании больших языковых моделей (LLM). Особое внимание уделено их интеграции с системами контроля версий и CI/CD-процессами.

На основе проведённого анализа спроектирована модульная и расширяемая архитектура системы, поддерживающая подключение новых языков, анализаторов и LLM-провайдеров. Описаны два ключевых сценария использования: автоматический анализ Merge Request и фоновый рефакторинг кода. Оба построены на событийно-ориентированной модели, что обеспечивает гибкость и масштабируемость.

Реализован модуль отслеживания Merge Request, выполняющий анализ изменений и формирующий рекомендации с помощью LLM. Также создан автономный модуль фонового анализа, способный находить устаревшие, дублирующие или сложные участки кода для последующего улучшения.

Тестирование системы показало эффективность предложенного подхода. Анализ до и после применения рекомендаций продемонстрировал улучшение читаемости, снижение цикломатической сложности, повышение переиспользуемости и соответствия принципам хорошего программирования.

Таким образом, работа вносит вклад в развитие автоматизации программной инженерии, предлагая решение для создания интеллектуальных помощников по улучшению качества кода. Перспективными направлениями являются расширение поддержки языков программирования, добавление динамического анализа и автоматической генерации тестов.