Муниципальное бюджетное общеобразовательное учреждение средняя общеобразовательная школа №79 г. Пензы

VIII открытый региональный конкурс исследовательских и проектных работ школьников
«Высший пилотаж – Пенза» 2026

Секция: Computer science

Проектная работа
«Разработка AI-сервера автоматизированного Code Review: интеграция GitLab, векторной БД и LLM для семантического анализа коммитов»

Автор: Свищев Илья Сергеевич
11Б класс
Руководитель: Пономарева Галина Николаевна
учитель информатики и ИКТ
высшей квалификационной категории

Пенза
2026 г.

Оглавление

Паспорт проекта

Название проекта	«Разработка AI-сервера автоматизированного Code Review: интеграция GitLab, векторной БД и LLM для семантического анализа коммитов»
Участники проекта	Обучающийся 11Б класса МБОУ СОШ №79 г. Пензы Свищев Илья Сергеевич
Целевая аудитория	✓ Корпоративные инженеры и разработчики, которым необходим качественный мониторинг изменений в коде. ✓ Программисты Senior, участвующие в процессе контроля качества. ✓ Менеджеры проектов, следящие за соблюдением стандартов разработки.
Аннотация проекта	Проект направлен на автоматизацию процесса анализа изменений в коде и предоставление детальных рекомендаций по улучшению качества программного обеспечения. Использование современных технологий позволяет минимизировать временные затраты и снизить риски пропусков важных изменений.
Проблема	Разработчики ежедневно сталкиваются с большим количеством изменений в коде, проверка которых требует больших трудовых и временных ресурсов. Традиционная практика ручного ревью неэффективна и создаёт высокий риск допущения ошибок.
Решение проблемы	Предлагается автоматизированная система, которая индексирует файлы, производит сравнение изменений и предоставляет рекомендации на основе искусственного интеллекта.
Цель проекта	Разработка AI-сервера автоматизированного Code Review.
Задачи проекта	✓ Изучить существующие подходы и инструменты для автоматизированного Code Review. ✓ Разработать архитектуру AI-сервера, включая интеграцию с GitLab, векторной БД и LLM. ✓ Реализовать семантический анализ коммитов с использованием LLM. ✓ Провести тестирование и оценку эффективности разработанного AI-сервера.
Сроки реализации проекта	Проект долгосрочный: *1)Подготовительный этап (май 2025)* ✓ выявление проблемы, ✓ поиск информации для решения выявленной проблемы, ✓ определение плана работы. *2)Основной этап (июнь-ноябрь 2025)* ✓ изученние существующих подходов и инструментов ✓ выбор необходимых инструментов для создания проекта ✓ проектирование архитектуры AI-сервера ✓ определение модулей *3) Заключительный этап (ноябрь 2025)* ✓ защита проекта,

	✓ определение направлений для дальнейшего развития проекта.
Продукт проекта	Автоматизированная система, предназначенная для анализа изменений в программном коде с использованием технологий искусственного интеллекта и векторных поисковиков.
Ожидаемые результаты	Продукт: AI-сервера автоматизированного Code Review Эффект: *Образовательный эффект:* углубление своих знаний в области разработки программного обеспечения, включая работу с AI и машинным обучением. *Технологический эффект:* разработка AI-сервера для автоматизированного Code Review позволяет повысить эффективность процесса проверки кода, снизить вероятность ошибок и улучшить качество кода. *Практический эффект:* созданный AI-сервер может быть использован в реальных проектах разработки программного обеспечения для автоматизации процесса Code Review. *Инновационный эффект:* интеграция GitLab, векторной БД и LLM для семантического анализа коммитов представляет собой инновационный подход к автоматизации Code Review. *Профессиональный эффект:* применение полученных знаний и навыков на практике, в будущих проектах или в своей будущей профессии.

Введение

В современном мире разработка программного обеспечения является одной из ключевых отраслей экономики, и качество кода играет важную роль в успехе проектов. Однако даже опытные разработчики могут допускать ошибки, которые могут привести к серьёзным проблемам в работе программ. Поэтому процесс проверки кода является неотъемлемой частью разработки программного обеспечения.

Традиционно Code Review проводится вручную, что требует значительных временных и человеческих ресурсов. Однако с развитием технологий искусственного интеллекта и машинного обучения появилась возможность автоматизировать этот процесс.

Актуальность темы обусловлена необходимостью повышения качества кода и снижения затрат на его проверку. Разработка такого AI-сервера позволит ускорить процесс Code Review, снизить вероятность пропуска ошибок и повысить качество кода.

Целью проектной работы является разработка AI-сервера автоматизированного Code Review.

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

1. Изучить существующие подходы и инструменты для автоматизированного Code Review.
2. Разработать архитектуру AI-сервера, включая интеграцию с GitLab, векторной БД и LLM.
3. Реализовать семантический анализ коммитов с использованием LLM.
4. Провести тестирование и оценку эффективности разработанного AI-сервера.

Практическая значимость работы заключается в возможности использования разработанного AI-сервера для автоматизации процесса Code Review в реальных проектах разработки программного обеспечения.

Продолжительность проекта: долгосрочный, май 2025 года – декабрь 2025 года, планируем работать над проектом дальше.

Тип проекта: информационно-цифровой.

Оценка ресурсов проекта:

Ресурсы	Источник
Информационные	• интернет-ресурсы (сайты технической документации, научные статьи и тематические форумы). • книги и справочники по языку программирования Python, системам контроля версий (GitLab), RESTful API, библиотекам анализа кода (например, javalang и Qdrant).
Кадровые	• инициативная группа, состоящая из родителей и учителя информатики школы МБОУ СОШ №79 г. Пензы.
Материальные	• компьютер с операционной системой Windows/MacOS/Linux и установленным программным обеспечением (редакторы кода, компиляторы, интерпретаторы). • доступ к интернету для загрузки необходимой информации и участия в онлайн-конференциях и семинарах. • серверное оборудование или облачная инфраструктура для развёртывания векторного хранилища и выполнения расчётов.

Результативность проекта мы определяем следующим образом:

Результат-продукт: автоматизированная система, предназначенная для анализа изменений в программном коде с использованием технологий искусственного интеллекта и векторных поисковиков.

Результат-эффект:

- *Образовательный эффект:* углубление своих знаний в области разработки программного обеспечения, включая работу с AI и машинным обучением.
- *Технологический эффект:* разработка AI-сервера для автоматизированного Code Review позволяет повысить эффективность процесса проверки кода, снизить вероятность ошибок и улучшить качество кода.
- *Практический эффект:* созданный AI-сервер может быть использован в реальных проектах разработки программного обеспечения для автоматизации процесса Code Review.
- *Инновационный эффект:* интеграция GitLab, векторной БД и LLM для семантического анализа коммитов представляет собой инновационный подход к автоматизации Code Review.
- *Профессиональный эффект:* применение полученных знаний и навыков на практике, в будущих проектах или в своей будущей профессии.

Глава 1. Теоретическая часть

1.1. Понятие Code Review и его значение в разработке программного обеспечения

Code review — это процесс анализа исходного кода программы с целью выявления ошибок, улучшения качества кода и обеспечения соответствия стандартам и лучшим практикам программирования. Он является неотъемлемой частью разработки программного обеспечения и позволяет повысить качество кода, снизить количество ошибок и улучшить понимание кода разработчиками.

Code review может проводиться как вручную, так и с помощью автоматизированных инструментов. В ручном режиме код анализируется и оценивается разработчиками, которые могут вносить предложения по улучшению кода. В автоматизированном режиме используются специальные инструменты, которые анализируют код и предоставляют разработчикам информацию о возможных ошибках и проблемах.

1.2. Существующие подходы и инструменты для автоматизированного Code review

Существует множество подходов и инструментов для автоматизированного Code review. Они могут быть разделены на несколько категорий:

- Статический анализ кода: инструменты статического анализа кода анализируют исходный код программы без её выполнения и выявляют потенциальные ошибки и проблемы. Например, SonarQube, ESLint.
- Динамический анализ кода: инструменты динамического анализа кода анализируют код во время его выполнения и выявляют ошибки, которые могут быть пропущены при статическом анализе. Например, JUnit, Selenium.
- Инструменты для ревью кода: инструменты для ревью кода позволяют разработчикам анализировать изменения в коде и давать рекомендации по их улучшению. Например, GitHub Codespaces, Crucible.
- Системы непрерывной интеграции (CI): системы непрерывной интеграции автоматически собирают и тестируют код при каждом внесении изменений. Например, Jenkins, Travis CI.

1.3. Основы работы GitLab и его интеграция в процессы разработки

GitLab — это комплексная платформа для разработки программного обеспечения, которая включает в себя систему контроля версий Git, инструменты для непрерывной интеграции и развёртывания, а также другие полезные инструменты для разработки (рис. 1).

На GitLab хранятся репозитории — места, где содержится исходный код программы. Для каждого проекта можно создать свой репозиторий. Это позволяет организовать код и управлять им более эффективно.

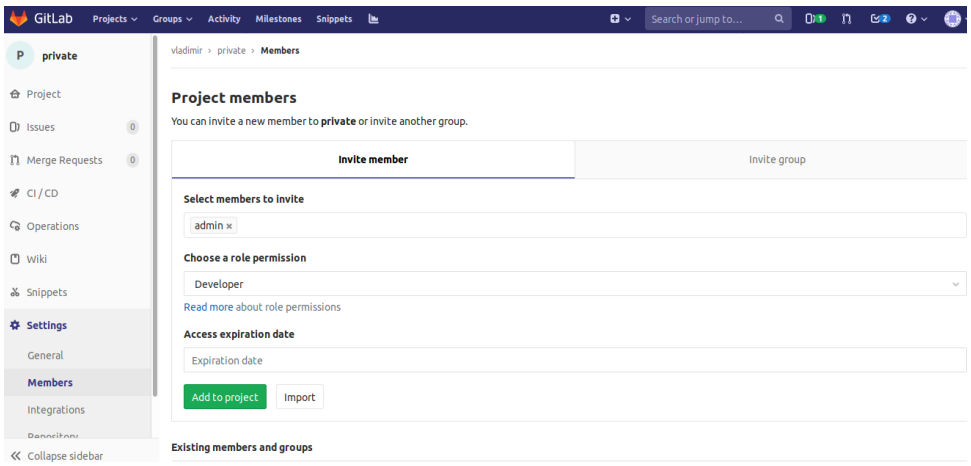

Рисунок 1.

В GitLab разработчики могут вносить изменения в код через ветви — копии репозитория (рис. 2). В системе можно создавать новые ветви и управлять ими. Ветви позволяют разработчикам работать над разными версиями кода параллельно, что ускоряет процесс разработки.

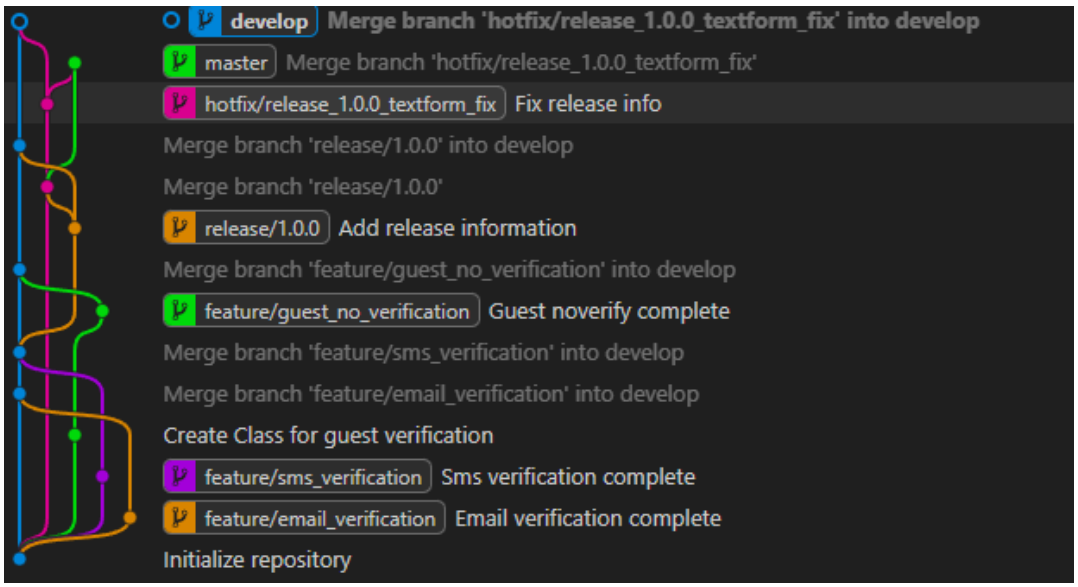

Рисунок 2.

Чтобы объединить изменения из одной ветви в другую, используют мерджи. Они помогают интегрировать изменения в основной код проекта. Это особенно полезно, когда несколько разработчиков работают над одним проектом и вносят изменения в разные ветви.

Теги — это метки, которые можно присваивать репозиториям и ветвям (рис. 3). В GitLab можно использовать теги для идентификации определённых версий кода. Это упрощает поиск и управление версиями. Теги помогают быстро находить нужные версии кода и отслеживать изменения.

GitLab можно настроить для автоматической сборки и тестирования кода при каждом внесении изменений. Также ее можно настроить для автоматического развёртывания кода на серверах или других устройствах. Выбранная нами платформа позволяет сразу нескольким разработчикам одновременно работать над одним и тем же кодом. Это способствует более эффективной и быстрой разработке программ и приложений.

1.4. Векторные базы данных: принципы работы и применение

Векторные базы данных — это специализированные базы данных, предназначенные для хранения и обработки векторных данных. Векторные данные представляют собой многомерные массивы чисел, которые могут быть использованы для представления различных типов информации, таких как изображения, аудио и видео.

Принципы работы векторных баз данных:

- хранение данных
- поиск данных
- обработка данных

Векторные базы данных могут хранить большие объёмы данных, что делает их идеальными для использования в больших данных (Big Data) и других приложениях, требующих хранения больших объёмов информации. В них могут обрабатывать запросы, которые требуют поиска и извлечения данных на основе их содержания. Это может быть полезно для приложений, требующих быстрого поиска и извлечения информации. Также они используются для анализа данных, например, для кластеризации, классификации и регрессии. Это может быть полезно для приложений, требующих анализа больших объёмов данных.

Qdrant — это современная система для векторного поиска, идеально подходящая для задач, связанных с поиском ближайших соседей в пространстве признаков. Ниже описано почему мы выбрали именно Qdrant для своего проекта:

- Qdrant обладает отличной производительностью при поиске в больших наборах данных, обеспечивая минимальные задержки и высокое качество результатов.
- Поддерживает горизонтальное масштабирование, что позволяет легко увеличивать мощность системы по мере роста данных.
- Простой API и доступность в Docker-контейнерах делают Qdrant удобным выбором для любого стека технологий.
- Отсутствие лицензионных платежей позволяет свободно использовать Qdrant в любых проектах, будь то коммерческие или образовательные.
- Наличие гибких настроек для фильтрации и сортировки результатов поиска позволяет настроить оптимальный баланс между скоростью и точностью.
- Полноценная поддержка REST API и клиентских библиотек для разных языков программирования облегчает интеграцию с другими сервисами.
- Возможность асинхронного добавления и удаления данных улучшает устойчивость системы и снижает нагрузку на основной поток выполнения.
- Подходит для многих задач, таких как рекомендательные системы, кластеризация, классификация, поиск образов и многое другое.
- Большое сообщество пользователей и разработчиков гарантирует постоянный рост экосистемы и регулярное обновление библиотеки.

При работе с данной векторной базой, мы выделили для себя основные преимущества. Она быстро находит близкие точки в векторном пространстве, что очень важно для нашей задачи поиска схожего кода. Позволяет накладывать дополнительные условия на поиск, такие как ограничение диапазона расстояний или условий совпадения полей. Легко масштабирует поисковую инфраструктуру, адаптируясь к росту объёма данных. Поддерживает эффективные механизмы сжатия данных, что снижает требования к

ресурсам и повышает производительность. Работает с произвольным числом измерений, что делает его идеальным решением для задач с высокими размерностями данных.

Таким образом, Qdrant идеально подходит для создания нашего проекта и является оптимальным выбором для построения качественной и производительной системы поиска векторов.

Глава 2. Разработка архитектуры AI-сервера

2.1. Выбор технологий и инструментов

Для разработки AI-сервера автоматизированного Code Review необходимо выбрать подходящие технологии и инструменты. Выбор языка программирования определяется спецификой задачи и предпочтениями разработчиков. Для создания нашего проекта был выбран язык программирования Python, так как я давно его изучаю, к тому же он обладает богатой экосистемой библиотек и фреймворков, предназначенных для работы с большими данными, машинообучением и аналитическими задачами. Язык программирования Python удобен для обработки текстов, а простота синтаксиса и широкая поддержка сторонних библиотек способствуют быстрому прототипированию и масштабированию.

Для интеграции с системой контроля версий GitLab мы использовали его собственный REST API, что позволило получать информацию о коммитах, ветках и других событиях, происходящих в репозитории, и проводить анализ изменений непосредственно из кода.

Как уже было сказано ранее, для хранения и поиска векторных представлений исходного кода мы выбрали Qdrant. В нашем проекте используется QdrantClient для Python, который предоставляет удобный интерфейс для взаимодействия с векторным хранилищем.

Для интеграции с лингвистическими моделями мы использовали API OpenAI, который предоставляет доступ к мощнейшим языково-моделям, таким как GPT серии. Через этот инструмент была осуществлена обработка естественного языка, генерация рекомендаций и помощь в принятии решений на основе полученного текста.

Для анализа кода мы взяли стандартную библиотеку ast для синтаксического анализа, библиотеку javalang для парсинга Java кода и игнорируемые форматы, а именно .txt, .doc, .md, .ini, .log

Таким образом, все перечисленные инструменты позволили создать надежное решение для анализа изменений в программном коде, сочетая легкость использования Python, мощные ML-фреймворки, простую интеграцию с GitLab и эффективные инструменты для работы с вектором признаков и языково-моделями.

2.2. Проектирование архитектуры AI-сервера

Проектирование архитектуры AI-сервера включает в себя определение компонентов системы, их взаимодействия и функций (рис. 3). Архитектура системы включает в себя четыре модуля: модуль, отвечающий за получение изменений в коде из GitLab, модуль, отвечающий за хранение и обработку изменений в коде в векторной базе данных, модуль анализа изменений и предоставление разработчикам информации о них и модуль, отвечающий за интеграцию с LLM для семантического анализа коммитов.

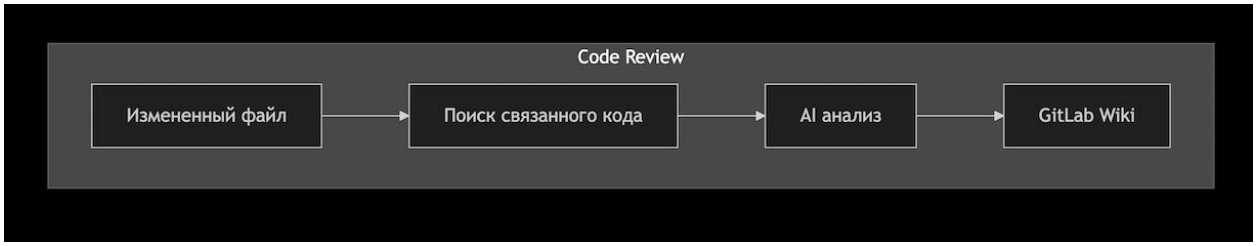

Рисунок 3. Общая архитектура системы

2.3. Интеграция с GitLab

Интеграция AI-сервера с GitLab позволяет получать изменения в коде из GitLab и анализировать их. Для интеграции с GitLab мы использовали GitLab API, который предоставляет инструменты для работы с GitLab.

GitLab API позволяет получать информацию о репозиториях, ветвях, коммитах и других элементах GitLab, что позволяет AI-серверу получать информацию об изменениях в коде и анализировать их (рис.4).

```
def get_files_from_commit(commit_sha: str, project_id: str, gitlab_url: str, private_token:
str) -> list:
    gl = gitlab.Gitlab(gitlab_url, private_token=private_token)
    project = gl.projects.get(project_id)
    commit = project.commits.get(commit_sha)
    diff = commit.diff()
    changed_files = [item['new_path'] for item in diff]
    return changed_files

def get_all_files_in_project(project_id: str, gitlab_url: str, private_token: str) -> list:
    gl = gitlab.Gitlab(gitlab_url, private_token=private_token)
    project = gl.projects.get(project_id)
    repo_tree = project.repository_tree(all=True, recursive=True, ref=BRANCH)
    files = [entry['path'] for entry in repo_tree if entry['type'] == 'blob']
    return files
```

Рисунок 4. Функции получения файлов из GitLab

2.4. Взаимодействие с векторной базой данных

В нашем проекте мы используем Qdrant — современную векторную базу данных, оптимизированную для работы с большими объемами векторов признаков (рис. 5). Qdrant предоставляет удобные инструменты для поиска ближайшего соседа в многомерном пространстве признаков, что позволяет эффективно сравнивать и классифицировать объекты.

```
client = QdrantClient(host=qdrant_host)
client.create_collection(
    collection_name=COLLECTION_NAME,
    vectors_config=VectorParams(size=len(embeddings[0]), distance=Distance.COSINE)
)

points = [
    PointStruct(
        id=str(uuid.uuid4()),
        vector=embedding,
        payload={"chunk": chunk, "file_path": file_path}
    )
    for chunk, embedding in zip(chunks, embeddings)
]
client.upsert(collection_name=COLLECTION_NAME, points=points)
```

Рисунок 5. Код инициализации Qdrant и сохранения векторов

2.5. Генерация векторов с использованием OpenAI

Векторные представления исходного кода формируются с помощью OpenAI API. Модель text-embedding-ada-002 преобразует текстовые фрагменты кода в векторы размерностью 1536 для семантического поиска (рис.6).

```python
def generate_embedding(text: str) -> list:
    response = requests.post(
        os.environ['OPENAI_API_URL'],
        headers={"Authorization": f"Bearer {os.environ['OPENAI_API_KEY']}"},
        json={"input": text, "model": "text-embedding-ada-002"}
    )
    return response.json()["data"][0]["embedding"]
```

Рисунок 6. Функция генерации эмбеддингов через OpenAI API

2.6. Индексирование файлов и поиск схожих изменений

Проанализированный код разбивается на логические фрагменты (чанкинг), каждый из которых проходит через операцию индексирования и сохраняется в векторной базе данных (рис. 7). После этого проводится поиск схожих изменений, что позволяет обнаруживать повторяющийся код и возможные ошибки (рис. 8).

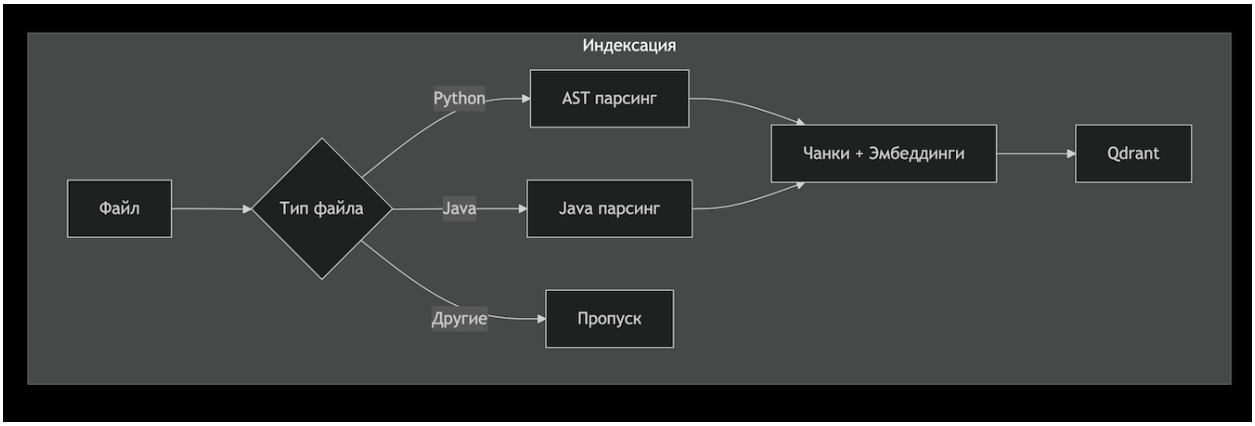

Рисунок 7. Процесс индексации файлов

```python
def process_python_file(code: str, file_path: str) -> List[str]:
    tree = ast.parse(code)
    chunks = []
    for node in ast.walk(tree):
        if isinstance(node, (ast.FunctionDef, ast.AsyncFunctionDef, ast.ClassDef)):
            chunks.append(ast.unparse(node))
    return chunks

def process_java_file(code: str, file_path: str) -> List[str]:
    tree = javalang.parse.parse(code)
    chunks = []
    for path, node in tree.filter(javalang.tree.ClassDeclaration):
        chunks.append(get_source_from_node(node))
    for path, node in tree.filter(javalang.tree.MethodDeclaration):
        chunks.append(get_source_from_node(node))
    return chunks
```

Рисунок 8. Функции чанкинга Python и Java кода

2.7. Отображение результатов анализа

Заключительным этапом является оформление результатов анализа в виде отчета и размещение его в GitLab Wiki (рис. 9). Мы создали механизм автоматического создания и обновления страниц Wiki, что позволяет разработчикам оперативно видеть рекомендации и обратную связь по изменениям в коде.

```
def send_comments_to_review_files(gitlab_client, project_id_param, comment_text, path):
    updated_path = path.replace('/', '_')
    timestamp = time.strftime("%Y%m%d_%H%M%S")
    filename = os.path.join(f"{updated_path}_{timestamp}.md")

    review_content = io.StringIO()
    review_content.write(f"# Review for file: {path}\n")
    review_content.write(f"{comment_text}\n")

    project = gitlab_client.projects.get(project_id_param)
    content = review_content.getvalue()

    project.wikis.create({
        'title': f' {filename}',
        'content': content
    })
```

Рисунок 9. Функция отправки комментариев в GitLab Wiki

Календарный план реализации проекта

№	Мероприятие	Период проведения	Результаты (количественные и качественные)	Отметка о выполнении
1.	Выявление проблемы, обсуждение с учителем выбора темы.	Май, 2025 г.	-обозначена проблема и конечный результат проекта, -собрана информация по проблеме, -составлен план и сроки реализации проекта	✓
2.	Изучение подходов и инструментов для автоматизированного Code Review	Июнь-август, 2025 г.	-изучены существующие подходы и инструменты -выбраны необходимые инструменты для создания нашего проекта	✓
3.	Разработка архитектуры AI-сервера	Сентябрь-декабрь, 2025 г.	-проектирование архитектуры AI-сервера -определение модулей	✓
4.	Защита проекта	Январь, 2026 г.	-презентация на научно-практической конференции школьников «Открытие»	✓

Список интернет-источников

1. https://ru.wikipedia.org
2. https://about.gitlab.com/
3. https://tproger.ru/
4. https://www.ultralytics.com/ru/glossary/vector-database