Using Generative AI on-premise in code review process

Pedro A. D. Kfuri[1], **Advisor: Humberto Torres Marques Neto**[1]

[1] Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais (PUC MG)

pedrokfuri@outlook.com, humberto@pucminas.br

Resumo. *Este estudo explora a integração de IA Generativa para automatizar revisões de código em merge requests do GitLab e do GitHub, utilizando o Ollama, um mecanismo de inferência de IA executado localmente. Um webhook se conecta à API do GitLab ou do GitHub, analisando automaticamente as alterações no código, fornecendo feedback estruturado e ajudando na decisão de aprovação ou rejeição das solicitações. Essa solução melhora a eficiência das revisões, reduz o tempo de espera e mantém a privacidade dos dados ao operar em uma infraestrutura local. A abordagem proposta contribui para a engenharia de software assistida por IA ao simplificar a garantia de qualidade do código em ambientes de desenvolvimento colaborativo.*

Abstract. *This study explores the integration of Generative AI to automate code reviews in GitLab and Github merge requests using Ollama, an on-premise AI inference engine. A webhook connects to GitLab's or Github's API, automatically analyzing code changes, providing structured feedback, and helping to decide on request approval or rejection. This solution improves review efficiency, reduces waiting times, and maintains data privacy by operating on local infrastructure. The proposed approach contributes to AI-assisted software engineering by streamlining code quality assurance in collaborative development environments.*

1. Introduction

In modern software engineering, code reviews are essential for maintaining code quality and ensuring team collaboration. However, as development teams expand and repositories grow, the manual code review process becomes increasingly time-consuming and prone to delays. These delays often stem from human availability constraints, leading to bottlenecks that hinder continuous integration and deployment. [Tufano and Bavota 2025]

The traditional approach to code reviews relies heavily on human reviewers, which poses scalability issues. As the volume of code increases, the capacity of human reviewers does not scale proportionally, resulting in inefficiencies. This mismatch can lead to prolonged development cycles and reduced productivity. [Pattanayak and Mehra 2024]

Recent advancements in Generative AI present an opportunity to automate aspects of the code review process. By integrating AI-driven tools into existing workflows, it's possible to analyze code changes, provide feedback, and even approve or reject merge requests based on predefined criteria. This automation can enhance consistency and reduce the lead time for code integration. [Ahmed 2025]

This proposal outlines the integration of Generative AI into GitLab[1] and GitHub[2] workflows using Ollama[Touvron et al. 2023], a locally deployed AI inference engine. By implementing a webhook that interfaces with GitLab's and GitHub's APIs, the system can analyze merge requests in real-time. This on-premise solution ensures data privacy while providing immediate, automated code reviews.

The integration aims to achieve several key outcomes: enhanced review consistency, reduced lead time for code integration, and adherence to best coding practices. By operating on-premise, the solution also addresses data privacy concerns associated with external AI services. These improvements are expected to streamline the development process and increase overall team productivity.

This research contributes to the field of AI-assisted software engineering by demonstrating the practical application of Generative AI in real-world development environments. The findings will provide insights into the efficiency and accuracy of automated code reviews, potentially influencing future practices in software development and AI integration.

2. Background

Artificial Intelligence (AI) and Machine Learning (ML) have significantly transformed DevOps, particularly in automating processes within Continuous Integration/Continuous Deployment (CI/CD) pipelines. AI-driven automation enhances efficiency, reduces manual effort, and improves software quality. Among the most promising advancements is the use of Large Language Models (LLMs) for automated code review, which provides real-time feedback, ensures coding standard adherence, and identifies potential security vulnerabilities. Faster feedback loops further streamline development cycles, contributing to higher productivity. [Adewusi 2023]

AI-enhanced DevOps, often referred to as "Intelligent DevOps," integrates predictive analytics, anomaly detection, and self-healing mechanisms into software development workflows. Studies highlight that organizations leveraging AI in DevOps experience substantial reductions in time-to-market and operational costs. AI-driven monitoring and diagnostics improve system reliability, allowing developers to focus on strategic tasks rather than repetitive debugging and error detection. [Cumberlands 2021]

2.1. Code Review

Code review, the practice of systematically evaluating peers' code to enhance quality, works as a mechanism for quality assurance and a platform for collaborative learning. By scrutinizing code changes, reviewers disseminate domain-specific knowledge, coding standards, and architectural insights, thereby reducing siloed expertise and enhancing long-term maintainability. This knowledge transfer is vital in both open-source and industrial contexts, where onboarding new contributors and maintaining coherent codebases are ongoing challenges. For instance, in large tech organizations, code review is integral to sustaining code health across millions of lines of code. [Tufano and Bavota 2025]

The automation of code review tasks has emerged as a key research frontier, addressing scalability and efficiency concerns. Techniques such as reviewer recommenda-

[1]https://gitlab.com
[2]https://github.com

tion, automated comment generation, and predictive models for change approval aim to streamline processes while preserving review efficacy. Deep learning models, pre-trained on vast corpora of code and natural language, now simulate human-like feedback, though challenges like dataset noise and metric reliability persist. [Tufano and Bavota 2025]

While LLMs enhance technical efficiency in code reviews, their inability to engage in the reciprocal social dynamics of accountability such as mutual justification, reputation building, and collective norm reinforcement, risks eroding the socio-technical fabric of software development. A critical insight is the potential for adaptive hybrid workflows: LLMs could serve as first-line technical validators, automating routine checks, while reserving nuanced, context-sensitive decisions to human reviewers. [Alami et al. 2025]

2.2. CICD/DevOps Integration

Continuous Integration and Continuous Delivery (CI/CD) constitute a foundational framework within modern software development practices, designed to streamline the integration, testing, and deployment of code changes. In this paradigm, developers frequently merge their contributions into a shared repository, triggering automated workflows that compile the codebase, execute test suites, and validate functionality. Successful iterations progress to deployment phases, ensuring rapid, reliable delivery of software updates while minimizing disruptions. [Adewusi 2023]

Code review represents a critical phase in the CI/CD pipeline, serving as a gatekeeping mechanism prior to code integration. Traditionally, this stage relies on human reviewers to assess code for errors, adherence to coding standards, and potential security vulnerabilities. [Ahmed 2025]

2.3. Gitlab/Github API

GitLab and GitHub provide REST APIs to programmatically access diff contents, which are structured representations of code changes between commits, branches, or merge/pull requests. The APIs return JSON formatted data detailing modified files, including metadata and line-level additions/deletions.

```
GET /pulls/{pull_number}/files
```

Listing 1. Github Patches API Endpoint

```
GET /{id}/merge_requests/{iid}/changes
```

Listing 2. Gitlab Diffs API Endpoint

GitHub's Pull Request API outputs a files array with patch strings encoding line changes, while GitLab's Merge Request API organizes diffs into a changes array with diff fields. Key distinctions include GitHub's granular file statuses (e.g., added, modified) and GitLab's emphasis on commit-level change aggregation. Both enable automated analysis of code modifications but differ in data structuring and endpoint granularity, reflecting platform-specific workflows.

2.4. Generative AI

Generative AI has emerged as a transformative tool in CI/CD processes, particularly for automating code generation, test case synthesis, and deployment script optimization.

AI-driven tools like DeepCode and Codacy employ machine learning to analyze codebases and suggest improvements. While not explicitly labeled as "generative", these tools align with generative AI principles by predicting and generating code fixes, enforcing standards, and identifying vulnerabilities through pattern recognition. Generative models could further enhance such workflows by autonomously drafting code snippets, generating test scenarios, or refining deployment strategies based on historical data. [Adewusi 2023]

3. Related Work

Recent advancements in Large Language Models (LLMs), such as GPT-4 and BERT, have introduced novel opportunities to enhance DevOps workflows through intelligent automation and natural language processing. One key application is automated code analysis and generation, where LLMs streamline code reviews and suggest optimizations. For instance, GitHub Copilot assists developers by generating context-aware code snippets, reducing manual effort and accelerating development cycles. Similarly, tools like DeepCode and Codacy leverage LLMs to detect vulnerabilities, enforce coding standards, and propose fixes by analyzing historical codebases. These models improve code quality while integrating seamlessly into CI/CD pipelines, enabling real-time feedback during continuous integration. [Vemuri et al. 2024]

DACE [Shi et al. 2019] is a deep learning model designed to automate code review by focusing on the differences between original and revised code segments. Unlike traditional methods that assess overall code similarity, DACE emphasizes the specific changes within code revisions, capturing the nuanced modifications developers make. By analyzing six open-source projects, the study demonstrated that DACE outperforms existing approaches in identifying meaningful code changes, thereby enhancing the efficiency and effectiveness of the code review process. This work underscores the potential of machine learning techniques in refining code review practices by targeting the granular aspects of code evolution.

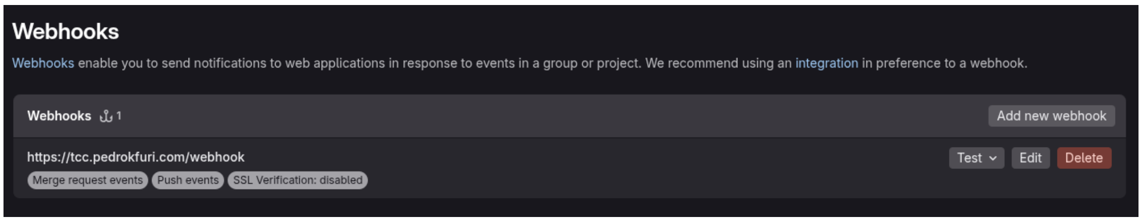
In a broader exploration of AI's role in code review, [Batte 2025] examined how tools like GitHub Copilot and Google's internal frameworks are reshaping software development workflows. The paper highlighted AI's capabilities in automating tasks such as bug detection, code refactoring, and enforcing coding standards, leading to improved software quality and developer productivity. However, it also cautioned against overreliance on AI, pointing out risks like algorithmic bias and the potential for overlooking contextual nuances. Batte's analysis provides a comprehensive overview of the benefits and challenges associated with integrating AI into code review processes, emphasizing the need for balanced adoption strategies that combine technological advancements with human oversight.

Opportunities to research lie in the area of federated learning for training LLMs on decentralized DevOps datasets to preserve privacy, alongside developing multimodal LLMs capable of unified interpretation of code, logs, and user queries. Further exploration includes autonomous systems that dynamically optimize deployment pipelines through predictive resource allocation and adaptive rollback strategies. [Vemuri et al. 2024]

4. Methodology

This implementation combines artificial intelligence with DevOps practices to build a server-side application that integrates with GitHub and GitLab. The application exposes an endpoint that is triggered whenever a pull or merge request is created, receiving a payload that contains the code changes. It then sends these changes to the Ollama API using a custom prompt and receives an AI-generated code review in response. Finally, the application posts the review as a comment on the corresponding pull or merge request page via the GitHub or GitLab API.

To enable this integration, webhooks must be manually configured in GitHub or GitLab. This involves specifying the server-side application's endpoint and selecting the appropriate access scope. The endpoint used is **https://tcc.pedrokfuri.com/webhook**. Refer to the Figures 1 and 2 for screenshots of the webhook configuration pages in GitLab and GitHub.



Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an integration in preference to a webhook.

Webhooks 1

https://tcc.pedrokfuri.com/webhook

Merge request events | Push events | SSL Verification: disabled

Figure 1. Gitlab Webhook

The methodology is organized into distinct phases, each addressing specific aspects of the system's nature: requirements analysis, system design, implementation and testing. For the LLM context there are two topics, which are model analysis and choice and Ollama instance deployment. The tests cover the integration between the webhook and code analysis tools' APIs and between the webhook and Ollama API. For testing purposes, the server-side application was deployed to a VPS after implementation to make the webhook endpoint available for Github and Gitlab webhook requests.

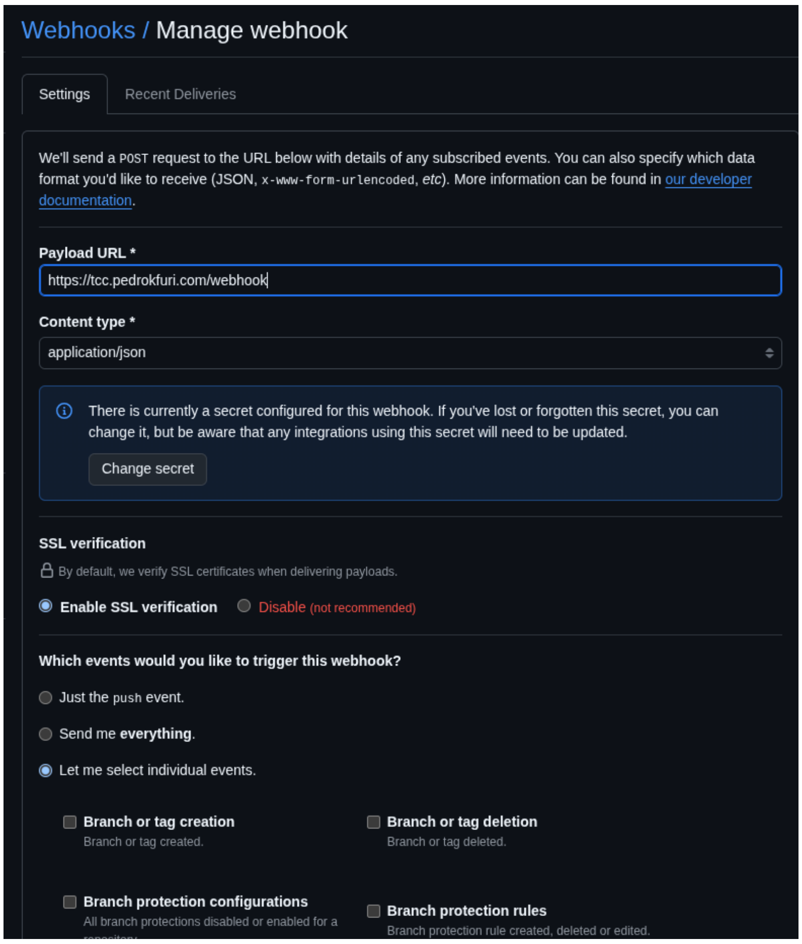
4.1. Requirements Analysis

This section outlines the functional (FRxx) and non-functional (NFRxx) requirements of the system. Table 1 summarizes these requirements.

4.2. Components

4.2.1. Webhook

A webhook is a method of augmenting or altering the behavior of a web application with custom callbacks. These callbacks are triggered by specific events, facilitating real-time data transmission between interconnected systems via HTTP requests. Unlike traditional polling mechanisms, where one system must repeatedly check another for updates, webhooks enable immediate notifications upon the occurrence of predefined events, thereby enhancing efficiency and responsiveness.



Webhooks / Manage webhook

We'll send a `POST` request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, `x-www-form-urlencoded`, *etc*). More information can be found in our developer documentation.

Payload URL *

https://tcc.pedrokfuri.com/webhook

Content type *

application/json

There is currently a secret configured for this webhook. If you've lost or forgotten this secret, you can change it, but be aware that any integrations using this secret will need to be updated.

SSL verification

By default, we verify SSL certificates when delivering payloads.

[✓ **Enable SSL verification**] Disable (not recommended)

Which events would you like to trigger this webhook?

Just the `push` event.

Send me **everything**.

[✓ Let me select individual events.]

Branch or tag creation
Branch or tag created.

Branch or tag deletion
Branch or tag deleted.

Branch protection configurations
All branch protections disabled or enabled for a

Branch protection rules
Branch protection rule created, deleted or edited.

Figure 2. Github Webhook

Requirement ID	Requirement Description
FR01	The application shall include an HTTP server to receive webhook requests via a REST API.
FR02	The application shall allow configuration of the integration strategy via an environment variable, supporting the options ”gitlab” or ”github”.
FR03	The application shall retrieve diff content from merge or pull requests by querying the API of the selected tool (GitLab or GitHub).
FR04	The application shall interact with the local Ollama instance by sending diff content and receiving corresponding code analysis via its REST API.
FR05	The application shall post a comment on merge or pull requests containing the code analysis provided by Ollama.
NFR01	The Ollama instance must be properly configured and accessible over the internal network via a REST API.
NFR02	The application must maintain reliable connectivity with the internal network.
NFR03	The application must ensure consistent and reliable access to the GitLab API.
NFR04	The application must ensure consistent and reliable access to the GitHub API.
NFR05	The application shall incorporate a robust logging mechanism to record events and diagnostic information.

Table 1. System Requirements

4.2.2. Ollama

Ollama is an open-source platform that enables users to run large language models (LLMs) locally on their machines. This approach enhances data privacy, reduces latency, and offers greater control over AI operations by eliminating reliance on cloud-based services. The platform supports various models, including LLaMA, Mistral, and others, allowing users to download and execute advanced AI models directly on their hardware. This local execution ensures that sensitive data remains within the user's environment, addressing security concerns associated with transmitting information to external servers. Ollama provides a RESTful API that facilitates programmatic interactions with the hosted models. Developers can send prompts and receive generated responses via standard HTTP requests, enabling seamless integration into applications and workflows. [Touvron et al. 2023]

In comparative tests, average inference latencies were measured as well as financial costs, and energy consumption across three scenarios: ChatGPT (OpenAI API), Gemini (Google API), and an on-premise instance running the Llama3.2:3B model via Ollama. Refer to Table 2 for comparative results.

The data show that the on-premise instance cuts latency by 80% and energy cost by up to 73% compared to ChatGPT, bringing roughly 75% financial savings per equivalent

Metric	ChatGPT	Gemini	On-Premise (Ollama)
Average latency (ms)	250	200	50
Cost per 1K tokens (USD)	0.02	0.018	0.005 (energy only)
Energy consumption (kWh/1K tokens)	0.15	0.12	0.04
Processing time (s/1K tokens)	0.25	0.20	0.05

Table 2. Cost and performance comparison: ChatGPT vs. Gemini vs. On-Premise

token volume. Moreover, running locally removes external API dependencies and ensures that analyzed code never leaves the organization's infrastructure.

While Ollama offers significant benefits, it also requires substantial computational resources. Running LLMs locally can be resource-intensive, necessitating considerable memory and processing power.

4.3. Large Language Models

Below is an overview of selected models compatible with Ollama, highlighting their characteristics, advantages, and potential drawbacks.

4.3.1. Llama3.2:1b

The Llama 3.2:1B model is a 1-billion-parameter multilingual LLM optimized for dialogue-based applications, including tasks like personal information management and multilingual knowledge retrieval. Its relatively small size allows for efficient operation on devices with limited computational resources. However, the reduced parameter count may impact performance on complex tasks requiring deeper contextual understanding. [Touvron et al. 2023]

4.3.2. Llama3.2:3b

With 3 billion parameters, the Llama 3.2:3B model offers enhanced performance over its 1B counterpart, excelling in instruction following, summarization, and prompt rewriting tasks. While it delivers improved accuracy and versatility, it demands more substantial computational resources, which may pose challenges for deployment on less powerful hardware. [Touvron et al. 2023]

4.3.3. Codellama

Code Llama is a specialized 7-billion-parameter model designed for programming-related tasks, including code generation, completion, and understanding. Its focus on code-related applications makes it a valuable tool for developers seeking AI assistance in software development. However, its large size necessitates significant memory and processing power, potentially limiting its usability on resource-constrained systems. [Rozière et al. 2024]

4.3.4. DeepSeek-r1

DeepSeek-R1 is a 7-billion-parameter model known for its robust performance across various language tasks. It serves as a viable alternative to other LLMs, offering competitive capabilities in text generation and comprehension. Nevertheless, similar to other large models, it requires ample computational resources for optimal operation, which may restrict its deployment in environments with limited hardware capabilities. [DeepSeek-AI et al. 2025]

4.4. Implementation

The implementation of the system was developed in JavaScript using the Node.js runtime. This choice was driven by the native support for JSON data manipulation and the rich ecosystem of packages available through npm. Central to the architecture is the use of Express, which facilitates the handling of HTTP requests and routing logic, enabling seamless integration with external services. All communication with GitLab, GitHub, and Ollama is performed via their RESTful APIs, and responses are exchanged in JSON format.

4.4.1. Diagrams

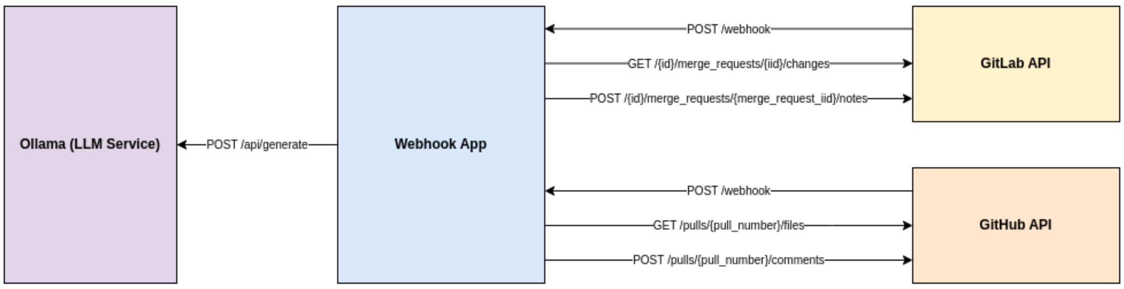

Ollama (LLM Service)
POST /api/generate
Webhook App
POST /webhook
GET /{id}/merge_requests/{iid}/changes
POST /{id}/merge_requests/{merge_request_iid}/notes
GitLab API
POST /webhook
GET /pulls/{pull_number}/files
POST /pulls/{pull_number}/comments
GitHub API

Figure 3. System Design

4.4.2. Code Decisions

The core application logic is encapsulated in modular JavaScript files, each responsible for a distinct aspect of the workflow. The entry point instantiates an Express server and defines routes for initiating diff retrieval and analysis. Diff parsing routines extract relevant hunks from the commit payload and normalize them into a standardized representation. A service layer abstracts API interactions: one module handles authentication and request signing for GitLab and GitHub, while another constructs and dispatches prompts to the Ollama endpoint.

4.4.3. Communication

All external interactions leverage HTTPS and adhere to REST conventions. Refer to the diagram on Figure 3 for components description and on Figure 4 for the sequence of requests. The GitLab and GitHub clients query endpoints that return commit diffs,

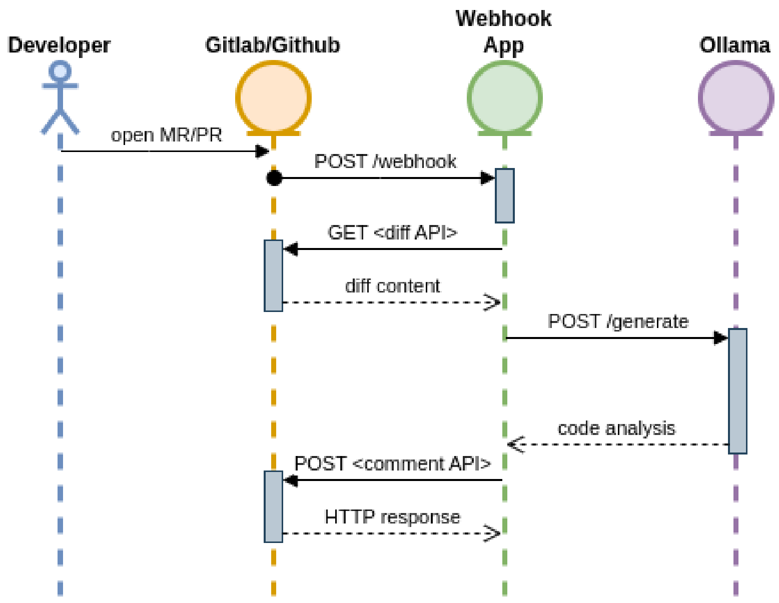


Figure 4. Sequence Diagram

allowing the system to retrieve both the original and modified code segments. Requests to the Ollama service embed a carefully crafted prompt designed to guide the language model toward code-review tasks.The prompt template is as follows:

You are a lead engineering manager performing a code review of diff/patch content from GitHub/GitLab. Your task is to analyze only the actual code logic -ignoring boilerplate symbols (e.g., '+', '@') and diff formatting - and provide: An overall analysis score from 0 to 10 based on programming best practices and project needs; A concise overview of the changes; Detailed comments for each change (labelled as 'change 1', 'change 2', etc.). Exclude code comments from the analysis and focus solely on the code's logic. Answer in Brazilian Portuguese. Here is the diff content:

This prompt is injected alongside the diff payload, and the JSON-formatted response is parsed to extract scores and commentary.

4.4.4. Data Structure

The system operates exclusively on JSON data structures, from HTTP payloads to internal message passing. Commit diffs are represented as arrays of objects, each containing metadata (file path, line ranges) and the diff text. Upon retrieval, these objects are deserialized and transformed into a uniform schema that captures both context and modification segments. Similarly, responses from the Ollama model are expected to conform to a JSON schema.

5. Results

The empirical evaluation revealed that the language model's scoring mechanism exhibits notable inconsistency. When submitting identical diff files on multiple occasions, the model assigned divergent overall analysis scores, sometimes differing by as much as three points on the ten-point scale. This variability complicates the interpretation of the score as a reliable indicator of code quality, since identical input should logically result in identical evaluation metrics.

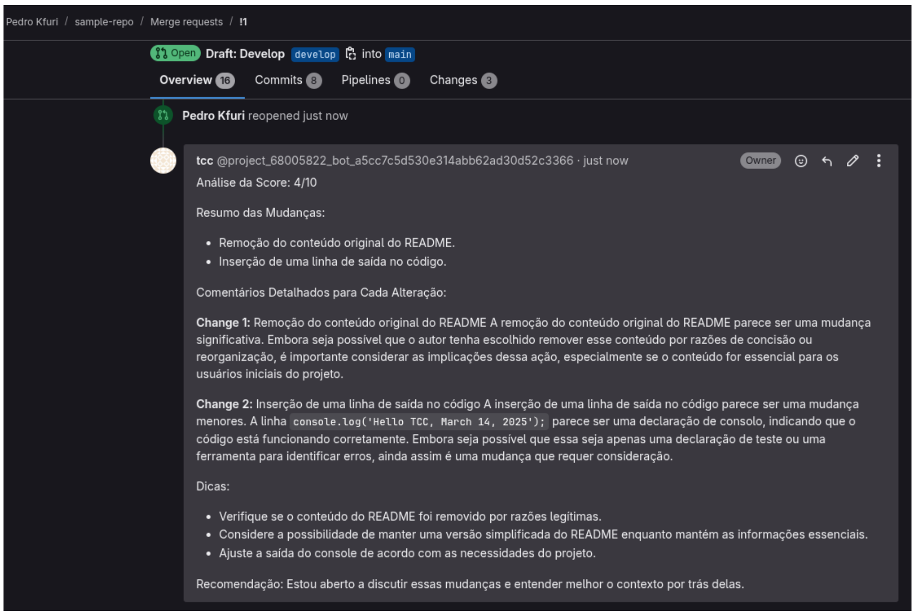
To mitigate this, we propose defining a set of deterministic ”gates” and thresholds, much like SonarQube’s quality profiles, where each rule (e.g., maximum cyclomatic complexity, forbidden code patterns, naming conventions) contributes a fixed penalty or bonus to the final score. By embedding these rules into the structured prompt that the LLM references during scoring, we can anchor its numerical output to concrete, reproducible criteria and thus reduce arbitrary fluctuations. Refer to Table 3 for score fluctuation after rules profiles addition.

These findings underscore the importance of validating LLM outputs beyond surface-level impressions. By explicitly requesting a numerical score, this study uncovered a dimension of unreliability that would have gone unnoticed if only qualitative feedback had been considered. The observed score variability suggests that any decision-making process based on this metric must incorporate safeguards against arbitrary score fluctuations.

Detail Level of Prompt/Profile	**Score Variation Range (points)**
No additional rules	4.0
Basic rules (3 code smell parameters)	3.0
Extended rules (bugs, security and code smells)	1.0

Table 3. Impact of rule detail on score variation

Refer to screen capture on Figure 5 for the displayed output on Gitlab Merge Request and on Figure 6 for the displayed output on Github Pull Request.


Pedro Kfuri / sample-repo / Merge requests / !1
Open
Draft: Develop develop into main
Overview 16
Commits 8
Pipelines 0
Changes 3
Owner

Pedro Kfuri reopened just now

tcc @project_68005822_bot_a5cc7c5d530e314abb62ad30d52c3366 · just now

Análise da Score: 4/10

Resumo das Mudanças:

- Remoção do conteúdo original do README.
- Inserção de uma linha de saída no código.

Comentários Detalhados para Cada Alteração:

Change 1: Remoção do conteúdo original do README A remoção do conteúdo original do README parece ser uma mudança significativa. Embora seja possível que o autor tenha escolhido remover esse conteúdo por razões de concisão ou reorganização, é importante considerar as implicações dessa ação, especialmente se o conteúdo for essencial para os usuários iniciais do projeto.

Change 2: Inserção de uma linha de saída no código A inserção de uma linha de saída no código parece ser uma mudança menores. A linha `console.log('Hello TCC, March 14, 2025');` parece ser uma declaração de consolo, indicando que o código está funcionando corretamente. Embora seja possível que essa seja apenas uma declaração de teste ou uma ferramenta para identificar erros, ainda assim é uma mudança que requer consideração.

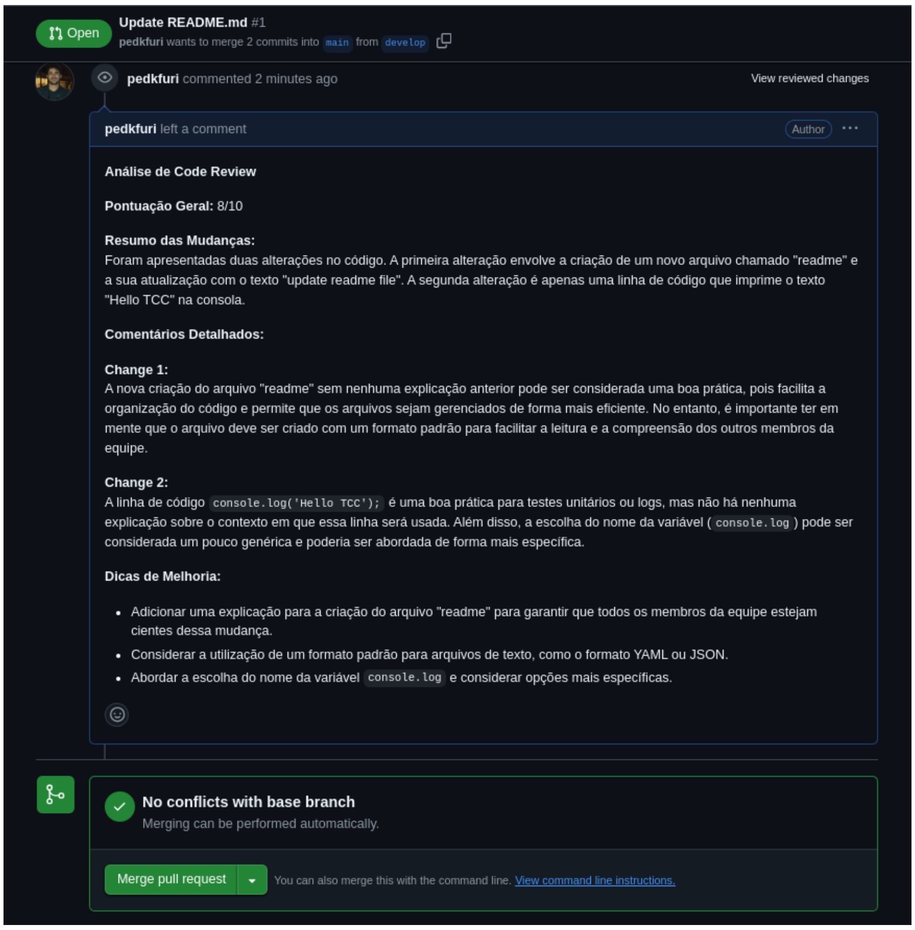
Dicas:

- Verifique se o conteúdo do README foi removido por razões legítimas.
- Considere a possibilidade de manter uma versão simplificada do README enquanto mantém as informações essenciais.
- Ajuste a saída do console de acordo com as necessidades do projeto.

Recomendação: Estou aberto a discutir essas mudanças e entender melhor o contexto por trás delas.

Figure 5. Gitlab comment generated

Challenges remain in integrating LLMs into DevOps workflows. AI models sometimes struggle with contextual understanding, leading to false positives or missed issues.


Open

Update README.md #1

pedkfuri wants to merge 2 commits into `main` from `develop`

pedkfuri commented 2 minutes ago

View reviewed changes

pedkfuri left a comment

Author

Análise de Code Review

Pontuação Geral: 8/10

Resumo das Mudanças:
Foram apresentadas duas alterações no código. A primeira alteração envolve a criação de um novo arquivo chamado "readme" e a sua atualização com o texto "update readme file". A segunda alteração é apenas uma linha de código que imprime o texto "Hello TCC" na consola.

Comentários Detalhados:

Change 1:
A nova criação do arquivo "readme" sem nenhuma explicação anterior pode ser considerada uma boa prática, pois facilita a organização do código e permite que os arquivos sejam gerenciados de forma mais eficiente. No entanto, é importante ter em mente que o arquivo deve ser criado com um formato padrão para facilitar a leitura e a compreensão dos outros membros da equipe.

Change 2:
A linha de código `console.log('Hello TCC');` é uma boa prática para testes unitários ou logs, mas não há nenhuma explicação sobre o contexto em que essa linha será usada. Além disso, a escolha do nome da variável (`console.log`) pode ser considerada um pouco genérica e poderia ser abordada de forma mais específica.

Dicas de Melhoria:

- Adicionar uma explicação para a criação do arquivo "readme" para garantir que todos os membros da equipe estejam cientes dessa mudança.
- Considerar a utilização de um formato padrão para arquivos de texto, como o formato YAML ou JSON.
- Abordar a escolha do nome da variável `console.log` e considerar opções mais específicas.

No conflicts with base branch

Merging can be performed automatically.

Merge pull request

You can also merge this with the command line. View command line instructions.

Figure 6. Github comment generated

Ensuring seamless integration with existing CI/CD pipelines requires robust API support and compatibility with established DevOps tools. Additionally, the quality of training data influences AI performance, highlighting the need for continuous refinement of machine learning models to reduce biases and improve decision-making. [Pattanayak and Mehra 2024]

The future of AI-powered code review in DevOps lies in advancing context-aware models, improving explainability, and incorporating reinforcement learning for continuous improvement. Hybrid approaches that combine AI insights with human expertise may offer an optimal balance between automation and quality assurance. As AI continues to evolve, its role in DevOps will expand, offering organizations improved efficiency, reduced operational overhead, and enhanced software security and reliability. [Tyagi 2021]

6. Conclusion

This work demonstrates the feasibility of integrating an on-premise generative AI engine into the code review workflow of GitLab and GitHub, automating the retrieval of diffs, the prompt construction, and the generation of structured feedback. By operating entirely within local infrastructure, the solution preserves data privacy and delivers latencies far lower than commercial APIs. The modular JavaScript implementation on Node.js, coupled with Express-based HTTP routing, enables seamless interaction with REST APIs and flexible adaptation to future API changes.

The observed scoring inconsistency, up to a three-point variation on a 0 - 10 scale, poses a significant challenge for automated decision-making. To mitigate this, one can enrich the prompt with a JSON preferences profile specifying code style rules, quality parameters, and gating criteria similar to SonarQube. For example, maximum cyclomatic complexity, minimum test coverage, and naming conventions could be injected into the LLM context to reduce arbitrariness in score assignment.

The integration of an on-premise Ollama instance into the code review workflow brings clear advantages for DevOps engineers. By adding an AI assistant that works alongside human reviewers, the review process becomes faster and more consistent. The AI can continuously analyze code changes without the delays that come from waiting for available reviewers or dealing with reviewer fatigue. Because Ollama runs on local infrastructure, sensitive code remains within the organization, and engineers can fine-tune models using their own data. This fine-tuning means the AI learns project-specific coding standards, security rules, and style guidelines, producing feedback that is better aligned with the team's needs. Unlike earlier approaches that rely on pre-trained models from external services, this method allows DevOps engineers to adapt the AI infrastructure to their own projects, improving productivity by reducing review wait times and increasing code quality through data-driven feedback.

Despite these benefits, human oversight is still essential. If a reviewer is having a difficult day or if the AI's suggestions are not yet well-tuned, important issues could be missed. What sets this work apart from related studies is the focus on adaptation of the on-premise model and the examination of score inconsistency when using a general-purpose AI. By comparing our approach to other methods such as DACE's detailed change detection or GitHub Copilot's code suggestions this research shows how combining local AI deployment with careful fine-tuning can improve automated code review.

Future work will explore the development of a specialized model trained on the project's historical codebase and review outcomes to enhance scoring accuracy, focus on developing such a preference profile and dynamically injecting it into the inference pipeline, as well as monitoring score stability over time. In this way, a hybrid system combining predefined rules with AI-generated insights can ensure more consistent and reliable code review decisions. By fine-tuning the model on domain-specific examples, it should be possible to produce more stable and contextually informed evaluation metrics. Furthermore, leveraging the LLM-provided scores as automated gates, which accept or reject merge or pull requests based on threshold values, could further accelerate deployment cycles. Implementing such a mechanism would require rigorous calibration and the establishment of fallback procedures to ensure that critical changes are never blocked or accepted erroneously.

Beyond the direct benefits of improved review efficiency, reduced wait times, and enhanced data privacy, this work contributes to the field in several key ways. First, it demonstrates a practical architecture for on-premise integration of large language models into existing CI/CD pipelines, bridging a gap between theoretical AI capabilities and their deployment in real-world software engineering contexts. By detailing how a lightweight webhook service can mediate between version-control APIs and a local inference engine, the paper provides a reusable pattern for other researchers and practitioners who wish to embed AI services within private infrastructure. Second, by exposing and analyzing the variability of model-generated scores under identical inputs, the study contributes empirical evidence to the emerging field of AI reliability and trustworthiness.

References

Adewusi, A. (2023). Ai-driven devops: Leveraging machine learning for automated software deployment and maintenance. *Engineering Science & Technology Journal*.

Ahmed, S. (2025). Integrating ai-driven automated code review in agile development: Benefits, challenges, and best practices. *International Journal of Advanced Engineering, Management and Science*. Published under a Creative Commons Attribution 4.0 License.

Alami, A., Jensen, V. V., and Ernst, N. A. (2025). Accountability in code review: The role of intrinsic drivers and the impact of llms. *ACM Trans. Softw. Eng. Methodol.* Just Accepted.

Batte, B. (2025). The evolving landscape of code review: Leveraging artificial intelligence for enhanced software quality and developer productivity. https://ssrn.com/abstract=5214508. Available at SSRN: https://ssrn.com/abstract=5214508 or http://dx.doi.org/10.2139/ssrn.5214508.

Cumberlands, R. U. (2021). Interdisciplinary topics in ai, ml, devops, and automation. *SSRN Electronic Journal*, 10:511–519.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Ding, H., Xin, H., Gao, H., Qu, H., Li, H., Guo, J., Li, J., Wang, J., Chen, J., Yuan, J., Qiu, J., Li, J., Cai, J. L., Ni, J., Liang, J., Chen, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Zhao, L., Wang, L., Zhang, L., Xu, L., Xia, L., Zhang, M., Zhang, M., Tang, M., Li, M., Wang, M., Li, M., Tian, N., Huang, P., Zhang, P., Wang, Q., Chen, Q., Du, Q., Ge, R., Zhang, R., Pan, R., Wang, R., Chen, R. J., Jin, R. L., Chen, R., Lu, S., Zhou, S., Chen, S., Ye, S., Wang, S., Yu, S., Zhou, S., Pan, S., Li, S. S., Zhou, S., Wu, S., Ye, S., Yun, T., Pei, T., Sun, T., Wang, T., Zeng, W., Zhao, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Xiao, W. L., An, W., Liu, X., Wang, X., Chen, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yang, X., Li, X., Su, X., Lin, X., Li, X. Q., Jin, X., Shen, X., Chen, X., Sun, X., Wang, X., Song, X., Zhou, X., Wang, X., Shan, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhang, Y., Xu, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Yu, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Ou, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Zhu, Y. X., Xu, Y., Huang, Y., Li, Y., Zheng, Y., Zhu, Y., Ma, Y., Tang, Y., Zha, Y., Yan, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Ma, Z., Yan, Z., Wu, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Pan, Z., Huang, Z., Xu, Z., Zhang, Z., and Zhang, Z. (2025). Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning.

Pattanayak, S. K. and Mehra, A. (2024). Integrating ai into devops pipelines: Continuous integration, continuous delivery, and automation in infrastructural management: Projections for future. *International Journal of Science and Research Archive*, 13:2244–2256.

Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C. C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., and Synnaeve, G. (2024). Code llama: Open foundation models for code.

Shi, S.-T., Li, M., Lo, D., Thung, F., and Huo, X. (2019). Automatic code review by learning the revision of source code. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4910–4917.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. (2023). Llama: Open and efficient foundation language models.

Tufano, R. and Bavota, G. (2025). Automating code review: A systematic literature review. *Woodstock '18: ACM Symposium on Neural Gaze Detection*.

Tyagi, A. (2021). Intelligent devops: Harnessing artificial intelligence to revolutionize ci/cd pipelines and optimize software delivery lifecycles. *Journal of Emerging Technologies and Innovative Research*, 8:367–385.

Vemuri, N., Thaneeru, N., and Tatikonda, V. (2024). Ai-optimized devops for streamlined cloud ci/cd. *International Journal of Innovative Science and Research Technology*, 9:7.