

AI-powered code review assistant utilizing OpenAI/Ollama for best practices



Table of contents

| | |
|----------------------------|-----------|
| Abbreviations | 3 |
| Introduction | 4 |
| Business challenges | 4 |
| Problem statement | 5 |
| Solution | 5 |
| Design overview | 5 |
| Workflow overview | 8 |
| Benefits | 10 |
| Conclusion | 11 |
| References | 11 |
| Author information | 12 |

Abbreviations

| Abbreviation | Full form |
|--------------|--|
| LLM | Large Language Model |
| PR | Pull Request |
| CI/CD | Continuous Integration/Continuous Deployment |
| API | Application Programming Interface |
| GPT | Generative Pre-trained Transformer |

Introduction

In modern software development, code reviews are vital to ensuring code quality and consistency. However, as projects grow in scale and complexity, manual code reviews often struggle to keep up with rapid iterations. To address this challenge, automating code reviews using AI can substantially improve efficiency and accuracy.

The code review engine leverages state-of-the-art language models, like OpenAI's Generative Pre-trained Transformer (GPT) and a locally hosted version of Ollama, to perform intelligent analysis of pull requests (PRs). Integrated with Bitbucket, the system processes code changes, analyses code differences and provides actionable feedback. It also features lifecycle management with callback hooks, allowing seamless integration into development workflows.

This whitepaper outlines the architecture and implementation of an AI-driven code review engine that integrates with Bitbucket and GitHub for processing PRs. It utilizes locally hosted large language models (LLMs) such as Ollama Llama 3.1 with Docker as a fallback mechanism and leverages Azure OpenAI GPT-4 for cloud-based enhancements. Additionally, the system supports callback hooks for triggering actions post-review, such as sending notifications or updating PR status. The design integrates lifecycle hooks to handle pre-review, review and post-review tasks, making the system suitable for Continuous Integration/Continuous Deployment (CI/CD) pipelines.

Business challenges

The business challenges faced by modern software development teams revolve around maintaining high code quality while keeping up with rapid development cycles. As organizations increasingly adopt agile methodologies, code changes grow exponentially, making manual code reviews time-consuming and inconsistent. Reviewers often miss critical performance issues, security vulnerabilities and inefficiencies due to the sheer volume of changes. This leads to technical debt accumulating over time, causing slower application performance, higher maintenance costs and ultimately reduced user satisfaction.

Manual code reviews are further limited by the varying expertise and availability of reviewers, which introduces inconsistencies in feedback. Maintaining up-to-date review standards becomes a significant challenge as new programming languages, frameworks, and technologies emerge. This results in development teams frequently dealing with rework and delayed production releases.

Organizations also face pressure to directly integrate automated testing, security checks and performance analysis into their CI/CD pipelines. However, existing manual code review processes often act as a bottleneck, slowing down the feedback loop and delaying deployments. This is especially problematic in large-scale projects with distributed teams, where timely and efficient code reviews are critical.

Moreover, as teams expand and development accelerates, maintaining uniform review quality across different codebases and projects becomes more difficult. The need for a scalable, consistent and automated solution to detect issues early in the development lifecycle is increasingly evident. Businesses require tools that can provide real-time actionable insights, helping developers resolve issues before they become critical problems in production.

Automating these reviews through AI-driven tools can address these challenges by providing consistent, timely feedback, improving overall software quality and reducing the burden on development teams. This allows organizations to fulfill the demands of contemporary software delivery while maintaining quality and speed.

Problem statement

In contemporary software development, organizations face significant challenges in maintaining code quality and performance standards throughout the DevOps lifecycle. As development teams adopt agile methodologies, the rapid pace of code changes often leads to undetected performance issues that accumulate over time, resulting in increased technical debt, application slowdowns and reduced user satisfaction. Traditional code review processes are frequently insufficient, lacking the ability to proactively identify performance bottlenecks and security vulnerabilities before they escalate into critical problems in production.

To address these challenges, an automated solution that seamlessly integrates with existing DevOps workflows is needed. This would facilitate the early detection of performance issues and ensure adherence to coding standards. By implementing an AI-driven code review engine that leverages advanced natural language processing models, organizations can gain real-time insights into code performance, flagging inefficiencies and noncompliance with best practices at the PRs stage. This proactive approach enhances code quality and supports CI/CD processes, ultimately leading to a more robust and reliable software delivery pipeline.

Solution

The AI-based PR engine is a sophisticated system designed to automate and enhance code reviews. It leverages FastAPI for efficient request handling and integrates advanced LLMs Ollama Llama 3.1 and Azure OpenAI GPT-4. This architecture ensures the engine provides reliable, AI-driven PR analysis of PRs within CI/CD workflows.

One of the system's key aspects is its fallback mechanism, which allows it to switch seamlessly between cloud-based GPT-4 and a local instance of Ollama hosted in Docker containers. This ensures the PR engine remains operational during network outages or cloud API downtimes. Docker's resource-efficient deployment ensures isolated and reliable performance for local inferencing.

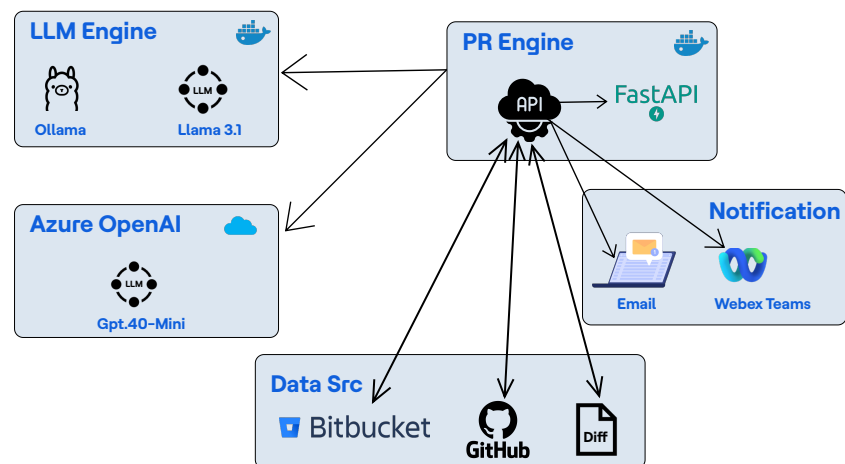
The engine integrates with Bitbucket via webhooks to automatically initiate reviews when PRs are created or updated. Webhooks trigger the system, which then analyzes the code and provides feedback directly to Bitbucket in the form of inline comments, suggestions and review status updates. Callback hooks further enhance this integration, enabling real-time interaction between developers and the AI system.

The engine supports customizable review criteria through templates, allowing teams to tailor the review process to project-specific coding standards, security policies, or other guidelines. This flexibility ensures that the engine adapts to the diverse needs of different development teams.

One of the standout features is the pluggable LLM, which allows organizations to easily swap models based on evolving programming languages, frameworks, or coding standards. The engine's polyglot capabilities ensure it can review code written in multiple languages, making it ideal for handling infrastructure as code, backend development and frontend frameworks in a single system.

By automating routine tasks, the engine reduces the load on individual developers, allowing them to focus on more critical aspects of their work. Its scalability makes it an ideal solution for both small teams and large enterprises, enabling asynchronous processing of multiple PRs simultaneously. The engine ultimately improves consistency, productivity and code quality while reducing bottlenecks in development workflows.

Design overview



The proposed system architecture includes several key components:

1. FastAPI-based API:

The core API is built using FastAPI, which provides fast, asynchronous request handling and allows for easy integration with other tools and services. FastAPI enables the engine to expose multiple endpoints for interacting with the code review process, such as submitting PRs for review or configuring user-specific review preferences.

2. Hosting LLM with Ollama in Docker:

A key aspect of the architecture is the fallback mechanism that ensures the code review engine remains operational even during network

outages or cloud API downtimes. The AI engine primarily uses OpenAI's Azure API for code analysis. Still, when connectivity to the cloud is lost or limited, it seamlessly falls back to Ollama, a local language model hosted in Docker containers.

By leveraging Docker, the system ensures that the local instance of Ollama is isolated and resource-efficient. This Docker-based deployment of Ollama serves as a fallback LLM for local inferencing, enabling AI-driven code reviews even when cloud access is unavailable.

3. Bitbucket integration and callback hooks:

The system integrates with Bitbucket repositories using webhooks, which allow the engine to automatically initiate reviews as part of the CI/ CD lifecycle. When a PR is created or updated in Bitbucket, the system triggers a webhook that sends the code diff to the CodeReviewEngine for analysis. Feedback is provided through a callback hook setup, which updates Bitbucket with comments, inline suggestions and review status.

4. Configurable templates and customization:

The engine's review criteria can be customized using templates that adapt to specific coding standards, security policies, or project guidelines. This allows teams to configure the engine based on their needs, tailoring it to different types of projects, languages, or compliance requirements.

Pluggable LLM: Enhancing the PR engine

Integrating a pluggable LLM via OpenAI configuration or pulling the necessary model to the Ollama container and getting the LLM into the PR engine architecture brings a transformative advantage, addressing both consistency in code reviews and the diverse needs of development teams. Here's how this feature contributes significantly to the code review process:

1. Consistency in quality of reviews:

One of the primary benefits of employing a pluggable LLM is its ability to deliver consistent quality across code reviews. Unlike human reviewers, who may vary in expertise, focus, or workload, an LLM uses a standardized approach to evaluating code. This ensures that every PR is scrutinized against the same criteria, leading to a more uniform code quality and adherence to best practices. By reducing reliance on individual reviewers, teams can avoid inconsistencies in manual reviews.

2. Evolving with organizational needs:

The LLM's pluggable nature means it can easily adapt to an organization's evolving needs. As new programming languages, frameworks, or coding standards emerge, the LLM can be updated or replaced without extensive rework of the underlying architecture. This flexibility enables organizations to keep pace with technological advancements and continuously integrate the latest developments in coding practices.

3. Polyglot capabilities:

Modern LLMs are inherently polyglot or multilingual, meaning they can understand and review code written in multiple programming languages.

This versatility allows a single LLM to handle a variety of technology stacks, including:

- Infrastructure as code: Review configurations and scripts written in Terraform or Docker Compose.
- Backend development: Analyze codebases in Python and Java, providing context-sensitive feedback tailored to each language's specific paradigms.
- Frontend frameworks: Evaluate code written in React, Angular and other popular frontend technologies, ensuring that UI components adhere to best practices.

This capability streamlines the review process and eliminates the need for multiple engines tailored to specific languages or frameworks, making the system more efficient.

4. Reducing the load on individuals:

Manual code reviews can be time-consuming and often place significant pressure on individual developers. By implementing an LLM within the PR engine, routine review tasks are automated, allowing developers to concentrate on more critical aspects of their work, such as architectural decisions or feature development. This workload distribution improves overall productivity and helps prevent burnout among team members, fostering a healthier work environment.

5. Scalable solution for growing teams:

As development teams expand and the volume of PRs increases, maintaining a high standard of code review becomes challenging. The asynchronous processing capabilities of the LLM enable it to handle large numbers of PRs simultaneously, ensuring that feedback is timely and relevant. This scalability makes the PR engine ideal for organizations of all sizes, from startups to large enterprises.

Workflow overview

The PR engine is designed to automate and enhance the code review process within a CI/CD framework. By leveraging AI-driven analysis, it streamlines interactions between developers and the code review system, ensuring high code quality and performance standards throughout the development lifecycle.

1. PR creation

- Developer action: A developer creates a PR in Bitbucket and submits code changes for review.
- Webhook trigger: This action triggers a webhook in Bitbucket, sending the PR details, including the code diff and metadata (e.g., author, branch, commit message), to the PR engine.

2. Code analysis initiation

- **API endpoint call:** The PR engine receives the webhook payload and invokes a designated API endpoint to initiate the code analysis.
- **AI model utilization:** The engine utilizes language models like OpenAI's GPT or the locally hosted Ollama model to analyze the submitted code against established coding standards and performance benchmarks.

3. Analysis and feedback generation

- **Diff chunking:** The PR engine extracts the code diff from the PR and breaks it down into manageable chunks. This allows for a more focused analysis of each segment of code.
- **Contextual information:** To enhance the quality of the analysis, the engine gathers context information from the original code files that are associated with the changes. This includes understanding dependencies, related functions and overall project structure.
- **Automated evaluation:** Each code chunk is sent to the AI model along with the contextual information, allowing the model to evaluate the code for various factors:
 - (i) **Code quality:** Detects anti-patterns, stylistic issues and adherence to best practices.
 - (ii) **Performance issues:** Identifies potential performance bottlenecks, such as inefficient algorithms, excessive resource consumption, or unoptimized queries.
 - (iii) **Security vulnerabilities:** Scans for common security flaws (e.g., SQL injection, improper authentication) and suggests mitigations.
- **Feedback compilation:** The engine combines the reviews from all analyzed chunks, generating a comprehensive feedback report. This report includes:
 - Inline comments and suggestions linked to the respective code changes.
 - A summary of findings, outlining critical issues identified and areas for improvement.

4. Feedback delivery

Callback hook integration: The PR engine utilizes callback hooks to send the generated feedback back to Bitbucket:

- Comments are added directly to the PR, allowing developers to view and respond to the feedback within the Bitbucket interface.
- The review status is updated to reflect the analysis outcome (e.g., approved, changes requested).

5. Iterative review process

- **Developer response:** Developers receive feedback notifications and can address the comments directly within the PR. This might involve making changes to the code and pushing new commits.
- **Re-analysis trigger:** As the PR is updated, the webhook triggers the PR engine again, re-initiating the analysis process for the new code changes.

6. Finalization of PR

- **Approval or rejection:** Once all feedback has been addressed and the code meets the established standards, the PR engine can facilitate the final approval of the PR.
- **Lifecycle hooks:** The system captures the final state of the PR (merged, closed) and logs the outcomes, which can be used for reporting and future audits.

7. Post-review metrics and reporting

- **Metrics collection:** After the PR has been merged, the PR engine collects metrics related to the review process, such as:
 - (i) Time taken for reviews and responses
 - (ii) Common issues identified across multiple PRs
 - (iii) Overall code quality improvements over time.
- **Reporting:** Regular reports can be generated for stakeholders, summarizing review outcomes, improvement areas and code quality trends.

Benefits

The proposed solution offers several advantages, including:

1. **Early detection:** By integrating into the CI/CD pipeline, the PR engine allows for early issue detection, reducing the risk of introducing bugs or performance regressions into production.
2. **Enhanced collaboration:** Automated feedback fosters better collaboration among team members, as developers can quickly address issues and improve their code quality.
3. **Scalability:** The PR engine's asynchronous nature allows it to scale efficiently with increased PR volumes, making it suitable for large teams and projects.
4. **Continuous improvement:** Metrics and reporting mechanisms enable teams to continually refine their coding standards and review processes based on real-world data and trends.
5. **Increased velocity:** By automating code reviews with AI, lead architects and developers can significantly increase development velocity. The system provides real-time feedback, allowing teams to address issues faster and reduce CI/CD pipeline bottlenecks. This streamlined process enables teams to deliver high-quality code more efficiently, speeding up release cycles and reducing manual review time.

Conclusion

In conclusion, as the complexity of software systems increases and the demand for rapid innovation grows, organizations must embrace AI-driven solutions that augment human capabilities rather than replace them. An AI code review assistant represents a technological advancement and a strategic necessity for modern software development. By investing in such an intelligent solution, teams can enhance their coding standards and quality and cultivate an agile, collaborative culture that positions them for success in an increasingly competitive landscape. The future of coding is not merely about writing code; it's about leveraging intelligence to unlock new levels of creativity, efficiency and excellence in software development.

References

- <https://ollama.com/blog/ollama-is-now-available-as-an-official-docker-image>
- <https://learn.microsoft.com/en-us/azure/ai-services/openai/overview>
- <https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/create-resource?pivots=web-portal>
- <https://dev.to/willvelida/building-nlp-applications-with-azure-openai-2637>
- <https://github.com/Azure-Samples/openai>

Author Information



Karthik Subramanian

Karthik has 19 years of experience building enterprise applications, focusing on network management solutions. His expertise includes developing reliable and scalable applications using Java technologies, AWS, Networking, AI and Python. Karthik is dedicated to creating practical, efficient systems that support network operations and management.

HCLTech | Supercharging Progress™

HCLTech is a global technology company, home to more than 220,000 people across 60 countries, delivering industry-leading capabilities centered around digital, engineering, cloud and AI, powered by a broad portfolio of technology services and products. We work with clients across all major verticals, providing industry solutions for Financial Services, Manufacturing, Life Sciences and Healthcare, Technology and Services, Telecom and Media, Retail and CPG, and Public Services. Consolidated revenues as of 12 months ending December 2024 totaled \$13.8 billion. To learn how we can supercharge progress for you, visit hcltech.com.

hcltech.com

