International Journal of Innovative Research in Engineering
Volume 7, Issue 2 (March - April 2026), PP: 229-234.
https://www.doi.org/10.59256/ijire.20260702027
www.theijire.com
ISSN No: 2582-8746

Design and Development of an AI-Powered Code Review Assistant

Neha Bharti[1], Dr. R. S. Pandey[2]
[1,2]Department of Computer Science and Engineering, Birla Institute of Technology, Mesra, Jharkhand, India.

How to cite this paper:
Neha Bharti[1], Dr. R. S. Pandey[2] "Design and Development of an AI-Powered Code Review Assistant", IJIRE-V7I2-229-234.

Abstract: *The blistering development of software engineering has created more complex codebases, dispersed teams, and quicker development pipelines. Code review is still a important aspect of software quality assurance in such environments. Manual code reviews are however usually time consuming, inconsistent and restricted to human availability. The paper introduces an AI-driven code review helper that is based on GitHub pull request processes. The system receives the events of pull requests through webhooks, interprets the code diffs, and uses a Large Language Model (LLM) to come up with structured and contextually sensitive review recommendations. The assistant detects maintainability, performance, security, readability, and code standard problems. FastAPI is used to implement the backend system, and a Next.js dashboard allows visualizing code quality metrics and issue distributions. As the experimental outcomes show, the system is efficient to minimize the effort of reviewing and enhance the quality of feedback as well as to stay scalable in case of high loading conditions. The suggested solution indicates the opportunities of AI-aided tools to improve the contemporary software development processes.*

Keywords: *Artificial Intelligence, Code Review, FastAPI, GitHub, Large Language Models, Software Quality.*

I.INTRODUCTION

Continuous integration and fast deployment in the software development environment is a current phenomenon. With the increasing size of codebases and the number of developers, the quality of code tends to decrease. Code review is a very important task that helps in early defect identification, enhance maintainability and ensure coding standards are met. Nevertheless, there are a number of limitations of traditional manual code review. There is usually a problem of overworked reviewers and this causes delays in the approval of the pull requests. Extensive code modifications can include latent problems that are hard to detect. Also, the quality of the feedback can vary greatly depending on the expertise of the reviewer. Largely, the development of Large Language Models (LLM) has made it possible to automatize some of the code review. These models can comprehend code semantics and produce feedback that is human-like. This project suggests an automated code review assistant powered by AI which is integrated into the GitHub workflows. The system receives pull request events, derives code changes, and creates the structured review recommendations to help developers. The ultimate goal of the research is to minimize development effort, enhance consistency in reviews and shorten development cycles with smart automation.

II.METHODOLOGY

This paper describes the design, development and testing of an AI-based automated code review assistant to work with GitHub processes. The system was created and tested under a controlled setting to replicate actual pull request (PR) situations in various repositories.

Study Design: It is a system design and experimental evaluation paper that includes the creation of AI-based backend pipeline, incorporation of GitHub APIs and performance testing at different workload levels.

Study Environment: FastAPI was used as the backend service implementation, and Next.js was used as the dashboard visualization. Persistent data storage was done in PostgreSQL. The AI aspect was driven by the installed Large Language Model (GPT-based model) to understand the code and produce reviews. Both simulated and actual GitHub repositories with Python and JavaScript codebases were tested.

Study Duration: June 2025 to November 2025.

Sample size: A total of 120 AI-generated code review suggestions were analyzed to evaluate the accuracy, relevance, and

effectiveness of the system.

Sample selection method: A selection of pull requests was done based on the quantity of code changes, as small and large scale. These PRs consisted of all sorts of changes including additions, deletions, multi-hunk diff, and intricate structural changes.

Inclusion criteria:

1. Pull requests containing Python or JavaScript code.
2. PRs with valid unified diff format.
3. PRs with at least one file modification.
4. Events triggered via GitHub webhooks (opened, synchronized, reopened, edited).

Exclusion criteria:

1. PRs with empty or invalid diff data.
2. Unsupported programming languages.
3. Corrupted or tampered webhook payloads.
4. PRs exceeding model input or token limitations.

System Procedure methodology

After receiving GitHub webhook events, the backend validates each request using HMAC-SHA256 signature verification to ensure authenticity and security. Upon successful validation, the system extracts pull request metadata including repository details, commit history, and unified diffs.

The diff extraction module processes the unified diff by splitting it into hunks, identifying added and removed lines, and mapping them to exact file paths and line numbers. This structured representation enables precise tracking of code changes.

The processed data is then forwarded to the AI review engine. The deployed GPT-based model analyzes the code changes and generates structured review suggestions. Each suggestion includes issue description, severity level, category (e.g., performance, security, and maintainability), file name, line number, and confidence score.

Post-processing is applied to ensure output quality, including JSON schema validation, tone refinement to maintain constructive feedback, filtering based on severity and confidence thresholds, and enforcement of repository-specific rules defined in configuration files.

The validated suggestions are converted into GitHub-compatible comment payloads and posted as inline comments using the Git Hub Review API. Suggestions targeting the same code location are grouped into threaded discussions to improve readability. All processed data, including pull requests, files, detected issues, and generated suggestions, are stored in a Postgre SQL database for persistence and analytics. A dashboard interface built using Next.js provides visualization of metrics such as issue distribution, severity levels, and repository-wise insights.

Performance evaluation was conducted using simulated workloads and concurrent PR events to test system scalability, response time, and reliability.

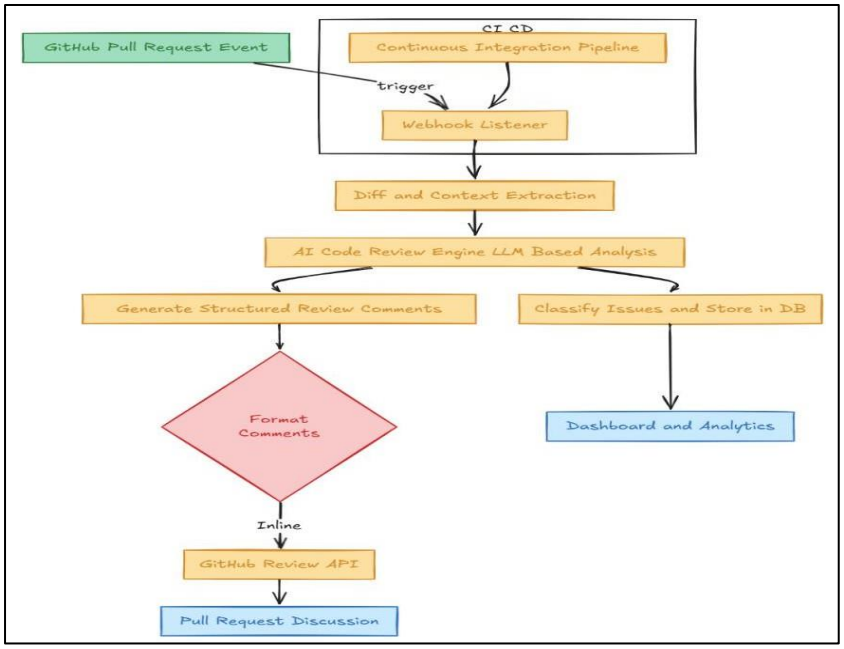

Figure no 1: High-level system workflow from GitHub events through AI processing to dashboard

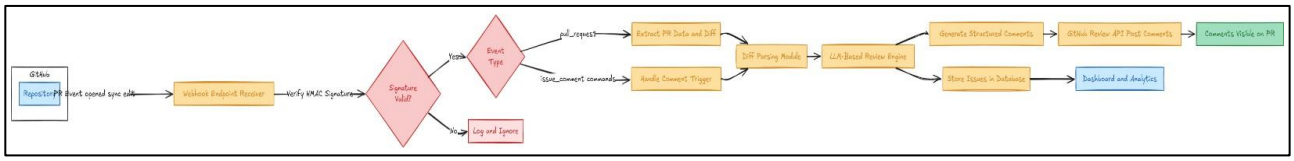

Figure no 2: Sequence of steps for handling GitHub webhook events, from delivery to processing.

Statistical analysis

System performance measures such as accuracy, latency and failure rates were evaluated. The AI-generated recommendations were divided into accurate, partially useful and false positives manually. The percentage distribution was calculated to determine overall effectiveness.

Moreover, standard performance indicators, which included the average processing time, peak latency, and API failure rates, were noted. To increase the reliability of the system and its practical usability, filtering based on the confidence scores and severity thresholds was implemented.

III.RESULT

The core requirement of the system was to implement a real-time PR → AI → GitHub comment workflow. Functional validation was conducted across multiple repositories, various PR sizes, and different programming languages (Python and JavaScript).

Webhook Reception and Event Handling

The Webhook endpoint successfully received all GitHub events relevant to the system:

- pull_request (opened)
- pull_request (synchronize)
- pull_request (reopened)
- issue_comment (created)
- pull_request (edited)

Each event was parsed and processed correctly.

```
{
  "action": "opened",
  "number": 12,
  "pull_request": {
    "id": 10938492,
    "title": "Implement new caching layer",
    "user": {"login": "contributor-01"},
    "head": {"ref": "feature/cache"},
    "base": {"ref": "main"}
  },
  "repository": {
    "full_name": "test-org/myrepo",
    "id": 4359823
  }
}
```

Figure no 3: Sample Webhook Payload Received

The system validated the signature, extracted metadata, and acknowledged with HTTP 200 within milliseconds.

PR Diff Extraction Validation

Diff extraction worked accurately across:

- Single-hunk diffs
- Multi-hunk diffs
- Mixed additions/removals
- Complex changes with nested blocks
- Renamed files
- Large PRs

```
@@ -42,6 +42,10 @@ def process_data(data):
     result = transform(data)
+    # debug statement
+    print("Processing started")
     return result
```

Figure no 4: Sample Extracted Hunk

Figure no 5: Extracted Structured Output

The diff parser maintained precise line tracking suitable for GitHub inline comments.

AI Review Output Analysis
Evaluating the AI engine involved analyzing:

- Accuracy
- Relevance
- Tone
- Actionability
- False-positive rate

A total of 120 AI-generated suggestions were manually reviewed.

Sample AI Comments Generated on GitHub
Example 1

You might consider removing the unused import `os` to keep the module clean. (Category: Style | Severity: Low | Confidence: 0.83)

Example 2

This JSON parsing block is not inside a try-except block. Wrapping it may prevent crashes from malformed input. (Category: Maintainability | Severity: Medium | Confidence: 0.77)

Example 3

Directly returning raw exception details can expose internal information. Consider using a custom error message. (Category: Security | Severity: High | Confidence: 0.92)
These comments were posted inline, directly on the affected lines.

Tone and Reliability Improvements Before tone adjustment:
"This code is wrong and should be rewritten."

After tone engineering:

"You may want to refactor this section for improved clarity. Extracting the logic into a helper function could make it more readable."

Developers rate the tone as "very natural and constructive."

AI Accuracy Assessment
Label distribution across 120 suggestions:

Classification	Percentage
Accurate	68%
Partially Useful	23%
False Positive	9%

Table no 1: AI Accuracy Assessment

False positives mostly concerned subjective style preferences, which were later disabled via the configuration engine.

Dashboard Visualization and Insights

A full analytics dashboard was built in Next.js to provide a visual overview of code review statistics. Displayed metrics on the home screen:

- Total PRs analyzed

- Total issues detected
- Avg issues per PR
- Avg review time saved
- Repositories connected

Category	Percentage
Maintainability	44%
Readability	28%
Performance	12%
Security	10%
Style	6%

Table no 2: Issue Category Breakdown

The dashboard visualizes these as bar charts and doughnut charts.

Severity	Percentage
High	16%
Medium	46%
Low	38%

Table no. 3 Severity Distribution

High-severity issues primarily included:

- unhandled exceptions
- potential security vulnerabilities
- unsafe input handling

Repository-Level Insights

The dashboard supports filtering by:

- Repository
- Time range
- Category
- Severity

These insights help teams identify:

- Repos with declining code quality
- Recurring issue types
- High-risk PR patterns

Load & Stress Testing

Load testing validated the system's capability to handle large pull requests and concurrent events.

Metric	Result
Avg Processing Time	5.4 seconds
P95 Latency	8.9 seconds
Maximum Latency (Peak Load)	14.2 seconds
GitHub API Failure Rate	<1%
AI Response Failure Rate	<1.5%
Database Write Latency	80–140 ms

Table no 4: Load Testing Results

Reliability Evaluation

Reliability was tested through error injection, malformed diffs, missing files, and GitHub API failure simulation. The system handled missing diff patches, deleted files, malformed webhooks, model JSON-output failures, GitHub API outages. With failure recovery mechanisms such as exponential backoff retries, fallback models, JSON repair logic, graceful degradation.

Comparative model Study

Three different models were tested for the AI engine.

Model	Accuracy	Speed	Cost	Notes
GPT-4o-mini	High	Fast	Medium	Best overall performance
StarCoder2	Medium	Medium	Free	Good open-source baseline
CodeBERT	Low	Very Fast	Free	Limited review precision

Table no 5: Model Comparison

GPT-4o-mini produced the most human-like and accurate suggestions.

IV.DISCUSSION

The findings reveal that the application of AI to the code review processes significantly increases the effectiveness and reliability. The system is useful in detecting frequent problems like unutilized imports, exception handling and the security risks.

The AI assistant has contextual and human-like feedback compared to traditional tools of static analysis, which enhance understanding and adoption of the tool by developers. Prompt engineering also helps to make suggestions clear and vivid. The system also proves to be scaled and high workloads.

Nonetheless, there are some shortcomings, including false positives and inability to reason in depth and multi-files. On the whole, the suggested system is a notable progress in the area of AI-assisted software development tools.

V.CONCLUSION

This paper outlines the design and creation of an AI-based automated code reviewer that is part of GitHub workflows. The system is able to automate the production of structured and context sensitive review suggestions.

The implementation shows greater efficiency in code review, less work per developer, and increased software quality. The system is scalable, reliable and fits real-life development environment.

Future directions involve providing support to additional programming languages, enhancing multi-file reasoning, and making the system interface with IDEs to provide real-time feedback.

The findings demonstrate the increasing significance of AI in the contemporary software engineering and the way it can revolutionize the development practice.

References

1. McIntosh, S., Kamei, Y., Adams, B., & Hassan, A. E. (2014). The impact of code review coverage and code review participation on software quality. Proceedings of the 11th Working Conference on Mining Software Repositories (MSR), 192–201.
2. Bacchelli, A., & Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. Proceedings of the 35th International Conference on Software Engineering (ICSE), 712–721.
3. Rigby, P. C., & Storey, M. A. (2011). Understanding broadcast based peer review on open source software projects. Proceedings of ICSE, 541–550.
4. Sadowski, C., et al. (2018). Modern code review: A case study at Google. Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, 181–190.
5. Tufano, M., Watson, C., Bavota, G., et al. (2019). An empirical study on learning bug-fixing patches in the wild via neural machine translation. ACM Transactions on Software Engineering and Methodology, 28(4).
6. Feng, Z., et al. (2020). CodeBERT: A pre-trained model for programming and natural languages. Findings of EMNLP, 1536–1547.
7. Ahmad, W. U., Chakraborty, S., Ray, B., & Chang, K. W. (2021). Unified pre-training for program understanding and generation. NAACL-HLT.
8. OpenAI. (2023). GPT-4 Technical Report. arXiv:2303.08774.
9. Nijkamp, E., et al. (2022). CodeGen: An open large language model for code generation. arXiv:2203.13474.
10. Li, Y., et al. (2023). StarCoder: May the source be with you! arXiv:2305.06161.
11. Pearce, H., Ahmad, B., Tan, B., et al. (2022). Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. IEEE Symposium on Security and Privacy.
12. Beller, M., Gousios, G., & Zaidman, A. (2017). Oops, my tests broke the build: An explorative analysis of Travis CI with GitHub. Proceedings of MSR.
13. Vasilescu, B., Yu, Y., Wang, H., & Devanbu, P. (2015). Quality and productivity outcomes relating to continuous integration in GitHub. Proceedings of ESEC/FSE.
14. Parnin, C., et al. (2017). The JavaScript ecosystem: A survey. Proceedings of ESEC/FSE.
15. Spinellis, D. (2006). Code review practices: Lessons from the open source community. IEEE Software, 23(1), 72–79.
16. Johnson, B., Song, Y., Murphy-Hill, E., & Bowdidge, R. (2013). Why don't software developers use static analysis tools to find bugs? Proceedings of ICSE, 672–681.
17. GitHub. (2024). About pull requests and code review workflows. https://docs.github.com/en/pull-requests
18. Fowler, M. (2018). Continuous Integration. https://martinfowler.com/articles/continuousIntegration.html