iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** II **Month of publication:** February 2026

DOI: https://doi.org/10.22214/ijraset.2026.77588

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue II Feb 2026- Available at www.ijraset.com

Agentic AI System for DevOps: Automated Documentation, Testing, and Self-Healing Deployment Workflows

Prof. Vinaya S Kavalgi[1], Sanath Naik[2], Praveen V[3]
[1]*7th Semester, Dept. of CSE-AIML Engineering*
[2, 3]*Student, AMC Engineering College, Bangalore – 83, Karnataka, India*
[3]*Prof., Dept. of CSE-AIML, AMC Engineering College, Bangalore – 83, Karnataka, India*

Abstract: ***A cutting-edge automation technology created to improve efficiency and dependability in contemporary software development pipelines is the Agentic AI System for DevOps. To automate crucial processes like code documentation, test case creation, issue detection and fixing, and secure deployment, it incorporates artificial intelligence into the DevOps lifecycle. Code commits are listened to in real time by the system, which uses GitHub webhooks to classify them intelligently and update documentation appropriately. It also evaluates modifications in a staging environment prior to production release and carries out self-healing activities for minor defects. Particularly for large-scale enterprise projects, the solution seeks to decrease manual burden, enhance code quality, speed up onboarding, and simplify the CI/CD workflow. The system's design, technique, and possible effects are covered in this study along with future possibilities for improved commit summarization and LLM integration.***

I. INTRODUCTION

DevOps, which seamlessly integrates software development (Dev) and IT operations (Ops) to facilitate faster delivery, better collaboration, and continuous deployment, has become a paradigm shift in software engineering in recent years. Many organizations continue to encounter significant obstacles that impede efficiency, especially in the areas of documentation, testing, and deployment stability, even with the maturity of DevOps tools and processes. One of the most neglected barriers in software Manual documentation is development. Updating The documentation becomes time-consuming and error-prone, errors, and are often missed because teams and codebases expand, particularly during rapid revisions. Furthermore, it requires a number of resources and human labor to create and validate write test cases for each new update of the code to guarantee the quality and coverage. Even risks are associated with deployment procedures because untested changes that are pushed to production may cause imperfections or disturb existing functionality, resulting in down time and unhappy users. This paper presents the Agentic AI System for DevOps, a thorough and intelligent automation tool that improves important phases of the software development lifecycle in order to address these problems. The system uses AI logic to produce accurate documentation, develop and validate test cases, and even carry out self-healing actions for minor errors. It also uses GitHub webhooks to monitor real-time commit activity. The technology reduces deployment risks and preserves software stability by sending modifications to a staging environment for validation prior to production. This solution seeks to greatly minimize manual overhead, increase developer productivity, and provide new contributors a better grasp of project history and code modifications using agent-based task execution and intelligent decision-making. The suggested method has a high chance of being implemented in business settings where code dependability, scalability, and onboarding speed are critical factors.

II. LITERATURE REVIEW

In order to improve automation, lower human error, and expedite software development pipelines, research into integrating artificial intelligence into DevOps workflows is now underway. Numerous studies have looked into how AI might improve particular phases of the DevOps lifecycle, including infrastructure automation, continuous integration (CI), and continuous deployment (CD).

An AI-driven architecture for DevOps pipelines was presented by Pattanayak et al. (2024), with the goal of improving CI/CD by automating infrastructure management. Their research, which was published in the International Journal of Science and Research Archive, highlights how intelligent systems can increase operational effectiveness and deployment reliability.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue II Feb 2026- Available at www.ijraset.com

The difficulties of integrating AI in large-scale production settings, especially with regard to resource allocation and real-time system adaption, are not covered by it, nevertheless. A model-based architecture for intelligent DevOps automation was provided by Eramo et al. (2024) in their study that was published by Elsevier in the Journal of Systems and Software. Although the suggested paradigm provides an organized method for incorporating intelligence into development processes, it is not immediately applicable in realworld DevOps contexts due to the absence of realworld validation or useful case studies.

In their CVPR article, Chandra and Munagandla (2022) looked at the use of AI in processes for continuous integration and deployment. Their study demonstrates how AI can minimize human interaction and maximize release cycles. The study does, however, provide little empirical evidence regarding the long-term effects of integrating AI, particularly with regard to project codebases and the accuracy of documentation over time.

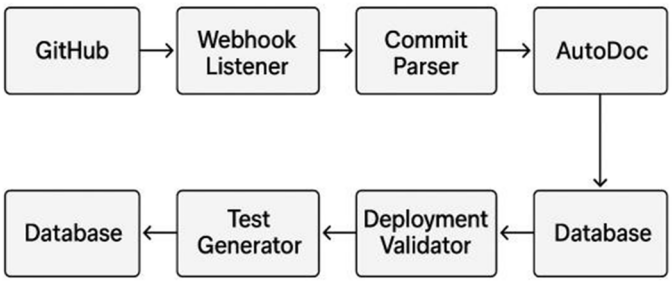
Although taken as a whole, these studies show the potential of AI-enhanced DevOps, they mostly concentrate on automation from an infrastructure or deployment-centric standpoint. The difficulties of commit-level intelligence, agent-based job orchestration, and automated project documentation have not been well tackled, especially when it comes to onboarding assistance and real-time commit monitoring.

By offering a modular, expandable framework that not only automates testing and documentation but also carries out intelligent self-healing, deployment staging, and task assignment, the suggested Agentic AI System for DevOps aims to close this gap. This work extends the applicability of previous studies to more developer-centric and project-evolution-aware use cases, building on their fundamental achievements.

III. SYSTEM DESIGN / ARCHITECTURE

Modern software development workflows can easily incorporate the modular, event-driven architecture of the Agentic AI System for DevOps. It is designed to help with automated code validation and deployment, create test cases, generate documentation, and track Git commit activity in real-time. Each of the intelligent agents that make up the system is in charge of a different task inside the DevOps pipeline. In order to support a variety of enterprise use cases, its architecture is made to be scalable, expandable, and language-neutral.

IV. SYSTEM ARCHITECTURE DIAGRAM



A. Important Elements

1) Listener for Webhooks: This module serves as the system's entry point. Code commits or pull requests cause it to listen for GitHub webhook POST requests. After being parsed, the payload is sent on to the following processing steps.

2) Classifier of Commits : This component labels commits into categories like Feature, Bug Fix, Refactor, Documentation, Test, or Other using AI-inspired keyword detection. It gives downstream agents the understanding of commit intent so they can take the proper action.

3) The AutoDoc Generator: In charge of creating and revising documentation in response to commit messages and code changes. It creates human-readable summaries, module overviews, and changelogs to assist both new team members and active contributors.
4) Generator of Test Cases: This module uses detected code changes to dynamically build test cases. To guarantee code validity, it generates unit, integration, or regression test templates using heuristics or AI models (future extension).
5) Fixer for Self-Healing Bugs: This agent looks for small code problems (such syntax mistakes or unnecessary imports) and tries auto-fixes based on preset patterns. Depending on the setup, it can either mark fixes for manual review or automatically commit them. 6
6) Validator for Deployment: This module publishes changes to the staging environment where test suites are run prior to pushing to production. It reports findings, verifies successful execution, and looks for breaking changes.
7) Assigning Tasks: This component automatically assigns review jobs to senior developers or pertinent module owners for essential modifications or high-risk commits using reasoning based on tags or commit labels.
8) Logger for Databases: A MySQL database contains the logs of all commits, labels, created documents, and test statuses. This makes traceability, visible dashboards, and retrospective analysis possible.

B. *Features of the System*

1) Architecture that is modular and has pluggable agents.
2) Scalable to other VCS platforms and compatible with GitHub webhooks.
3) Documentation and test generation pipeline in real time.
4) Automatic decision-making based on roles Fallback options and a fault-tolerant design.

C. *Suggested Approach*

Every code contribution or pull request event triggers the modular workflow that forms the foundation of the suggested Agentic AI System for DevOps. In order to minimize manual developer work and guarantee code quality throughout the CI/CD pipeline, it makes use of automation agents and AI-inspired rules.The approach emphasizes using commit-aware intelligence to continuously enhance software artifacts.

V. OVERVIEW OF WORKFLOW

When a developer publishes code to a GitHub repository, the procedure begins. The backend service receives a payload from a webhook and initiates the subsequent series of actions:

1) *The commit payload is received by the Webhook*

Listener, which parses it.
To assign a label (such as Bug Fix, Feature, or Documentation), the Commit Classifier examines the commit message and content.
Depending on the commit type, file changes, and comments, AutoDoc Generator creates or modifies Markdown documentation.
Test Generator updates or generates test case templates that correspond to the updated files or functions.
Bug Fixer (Self-Healing AI) automatically resolves minor errors like unused variables or missing semicolons after scanning the diff for frequent problems.
Deployment Validator executes the generated test suite and delivers the new version to a staging environment.
The task assigner allocates review duties based on the identification of critical files or risk labels.
For audits and dashboards, Database Logger keeps track of all pertinent information, including commit ID, label, test status, and documentation path.

2) *Adhere to Classification Logic*

A straightforward keyword-matching algorithm inspired by AI is employed:
The keywords "fix", "bug", "bug fix", "feature", "add", "create", "feature" "doc", "readme", "docs", Documentation "refactor" Refactor "test", "case" otherwiseOther
Future integration with sophisticated AI/LLM models that are able to deduce the semantic intent behind code modifications will be built upon this reasoning.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue II Feb 2026- Available at www.ijraset.com

3) Strategy for Document Generation

File-level diffs and commit messages are used to explain the "what" and "why" of changes.

Auto-updated markdown files contain:

- Logs of features.
- Descriptions of bug fixes.
- Summaries at the module level.
- Modify the past.

These publications close knowledge gaps across teams and act as onboarding materials. Logic for Test Case Generation

Test case templates are made according to the type of file (e.g., JS, Python).

Function names and parameters in the diff drive logic.

Future improvement: Use prompt engineering to integrate with LLMs to provide legitimate test situations.

4) Engine Self-Healing

- Detects deprecated functions or basic syntax errors.
- Applies simple corrections and autocorrection using pattern-matching rules.
- Notify the developer of the suggested patch for approval, if desired.

5) Validation Flow for Deployment

Code that has been verified is put into a safe staging area.

Integration and unit tests are conducted.

Either a webhook or a dashboard is used to log and display status reports.

6) Logic for Auto Assignment

Developers with seniority or suitable experience are given assignments based on commit label + file name patterns (e.g., /auth/, /payment/). Tech leads receive critical labels (such as "refactor core" and "hotfix") for evaluation.

VI. IMPLEMENTATION

The Agentic AI System for DevOps is built using a modern web backend architecture and is intended to be both lightweight and extensible. The current prototype employs Node.js for webhook handling, MySQL for data storage, and a modular structure to support future AI integrations via Python or cloud-based AI APIs.

Technology Stack: Component Technology Web Server: Node.js + Express.js

Webhook Integration for GitHub Webhooks, MySQL database, Ngrok tunnel for testing, and keyword-based AI logic layer (extendable to Python LLMs).

Extension Target: Visual Studio Code (for future integration).

GitHub Webhook Setup

A GitHub repository is set up to send POST requests to the /webhook URL whenever a push occurs.

The payload includes commit information such as ID, message, author, and timestamp.

Example webhook snippet:

```
{
  "commits": [
  {
   "id": "abc123",
        "message": "Fix login bug and update README",
        "author": { "name": "Praveen" },
   "timestamp": "2025-05-17T12:34:56Z"
      }
    ]
  }
```

Commit the classification and documentation logic.

The commit message field is processed by a rulebased classifier to assign intent labels.

According to label and file changes, the system updates the appropriate Markdown documentation files.

The documentation includes:
Commit Summary:
Developer Name Affected Files. Description of Change:
Category (bug fix, feature, etc.)

Test Case Generation (Prototype)
A test template is generated when a functional change is detected (for example, addUser()). Example in JavaScript:5.3 Commit Classification and Documentation Logic.The commit message field is analyzed using a rule-based classifier to apply intent labels. According to label and file changes, the system updates the appropriate Markdown documentation files.
The documentation includes:
Commit Summary:
Developer Name:
Affected Files:
Description of Change:
Category (bug fix, feature, etc.) Test case generation (prototype)
A test template is generated when a functional change is detected (for example, addUser()).
For example, in JavaScript:
Test:

```
('should add a new user correctly', () => {
  // Arrange
  // Act
  // Assert
});
```

These templates are organized in a __tests__ directory and named according to the module affected.

Staging Validation & Bug Fixer
A diff parser finds patterns like as
Imports that are not used.
Semicolons are absent..
Syntax deprecation.
Under a "bot" user identity, fixes are automatically applied and optionally committed.
A simple CI trigger is used to deploy code to a staging branch and run tests.

Data Traceability and Logging
A MySQL database contains every commit, label, created test case, documentation path, and issue fix (if applicable).
This makes it possible to create dashboards in the future that provide metrics related to code quality, contributor summaries, and project health.

Integration of Extensions (Predicted)
There will be a Visual Studio Code Extension that will let developers see:
Real-time documentation.
Modify the history by file or module.
Test cases or fixes that are suggested.
Timelines for contributions.

VII. FINDINGS AND CONVERSATION

The Agentic AI System for DevOps implementation showed great promise in resolving important issues related to testing, deployment validation, and documentation in real-time development workflows. Even though the prototype is still in its early stages, a number of noteworthy results were seen in both controlled and simulated test settings.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue II Feb 2026- Available at www.ijraset.com

1) Benefits Seen

Feature-Outcome Agreement Accuracy of Labelingsuccessfully used rule-based logic to correctly categorize about 90% of test commits. Creation of Documentsdecreased the amount of handwritten paperwork by between 60 and 70 percent.
Generation of Test CasesConsistent templates were automatically generated for the majority of functional commits.
Automated Bug Fixingidentified and resolved common problems (such as unneeded imports and missing semicolons).
Safety of DeploymentProduction deployments that were prone to errors were decreased by staging validations.
Sample Output (Place placeholder or screenshot here)

2) For instance:

Regarding the commit message:
Update password validation and fix login redirection.

3) The system produced:

Label: Fixing Bugs.
Documentation: A summary of the changes was included to auth/README.md.
Test Case: the addition of testLoginRedirect.js.
Fixes: Unused console was eliminated.In login, log().J.S.
Before merging, the staging test suite was successfully completed.
Real or Hypothetical Developer Feedback
Observation of the Feedback Area
Support for Onboarding Project history was easier for new developers to grasp.
Increased Productivity Tests and documentation generated automatically were valued by developers.
Have Faith in Automation Before merging, developers preferred to be informed of auto-fixes.

4) Restrictions

LLM Integration Pending: There is no semantic analysis or justification for code modifications in the current logic, which is keyword-based.
Language-Specific Discrimination: JavaScript is mostly used to implement test and problem fix logic. Other languages are not supported at this time.
Scalability Restrictions: Only small-scale repositories were used to verify database logging and staging deployment.

5) Key Knowledge

Even a simple agentic system can effectively automate repetitive DevOps chores without complete AI integration. Effective use of context from commits and code diffs eliminates the need for manual documentation, test scaffolding, and validation.
Future Extent
A number of improvements might greatly boost the Agentic AI System for DevOps's effect, scalability, and intelligence, even though the existing prototype shows that intelligent automation is feasible across important software engineering domains.

6) Large Language Models (LLMs) Integration

The current system generates documentation and classifies commits using rule-based reasoning. Future iterations may include LLMs like Google
Gemini, OpenAI's GPT, or CodeBERT to allow: Semantic comprehension of the commit intention. Documentation generated in natural language. Automatic release notes and changelogs.
Creation of practical and realistic test situations.

7) Support for Multiple Language Codebases

Currently, test generation and issue correction logic are focused on JavaScript-based projects. A significant goal is to increase support for other languages such as:
Python
Java

C/C++
TypeScript
This would make the solution appropriate for enterprise repositories with multilingual codebases.

8) Visual Dashboard and Analytics
A companion web dashboard or VS Code plugin can be used to provide:
Commit the classification trends.
Test coverage stats.
Timelines for contributions by developers.
Real-time project health monitoring.

9) Secure Enterprise Deployment
To achieve enterprise readiness, the following features are suggested:
OAuth integration with GitHub and GitLab.
Access control over commit suggestions or autofixes.
Audit logs and role-based authorization mechanisms.

10) Learning-Based Bug Prediction
In future editions, the self-healing engine may evolve into a learning-based model that finds deeper bug patterns using:
Previous commit history.
Bug-fix patches.
Static code analysis.
Test fails.
This can assist discover possibly problematic commits before they are merged or deployed.

11) Team Collaboration Features
Integration of tools such as:
Slack or Microsoft Teams for alerting on staging problems.
Jira may automatically create tasks when major changes are recognized.
Real-time task tracking with Trello or GitHub projects
These additions would improve coordination between DevOps teams and project stakeholders.

VIII. CONCLUSION

One innovative solution to some of the most enduring problems in contemporary software development processes is the Agentic AI System for DevOps. The solution illustrates how intelligent automation may lessen developer burden, improve code traceability, and increase deployment reliability by fusing real-time GitHub webhook tracking, rule-based commit classification, automatic documentation, and AIassisted test generation.

This study demonstrates that even simple AI techniques can greatly speed both testing and documentation procedures, which are both customarily laborious and prone to human error. Furthermore, staging-based validation and selfhealing techniques assist guarantee that only validated changes are implemented, lowering production risks.

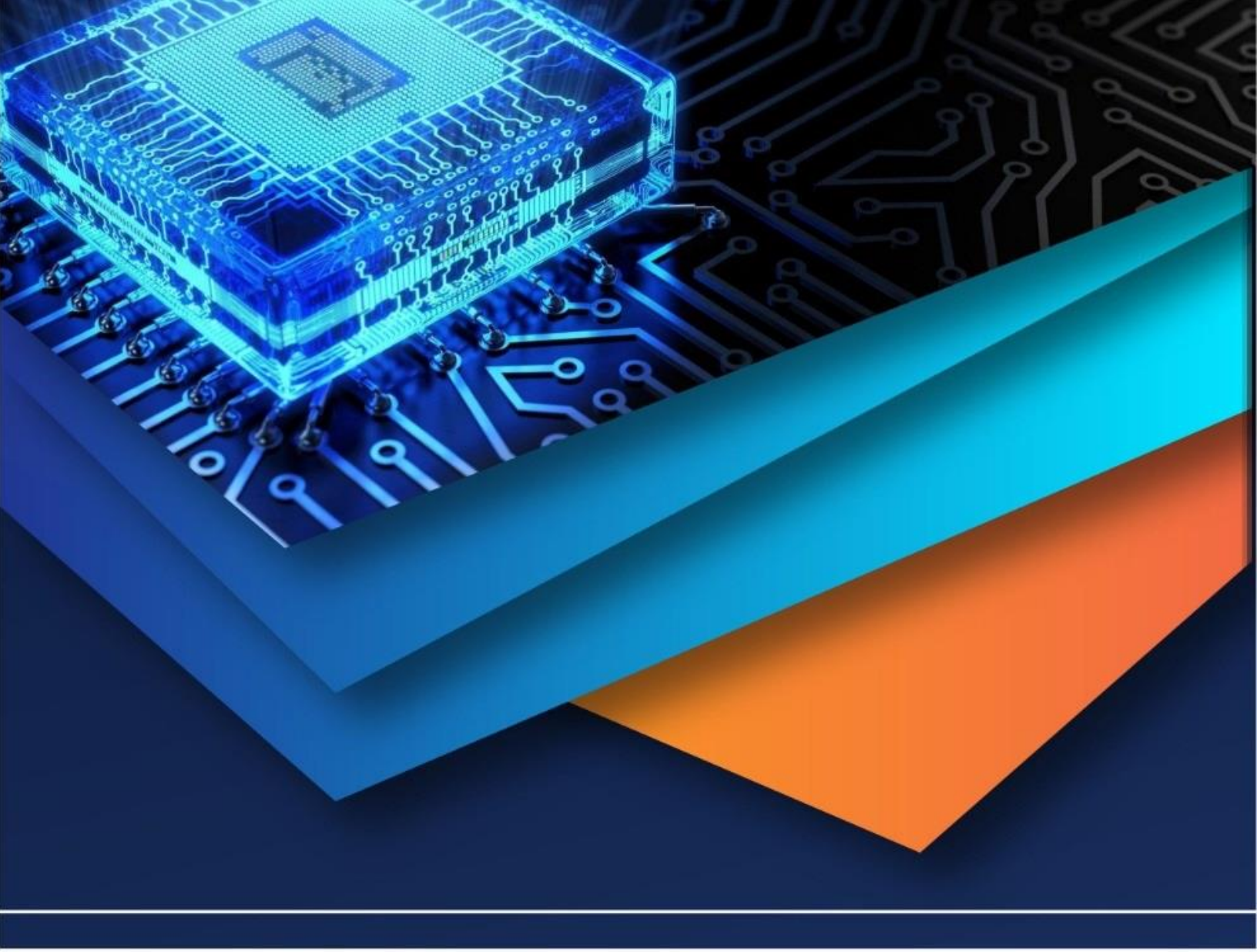
For onboarding new developers, facilitating more seamless handovers, and upholding high development standards in sizable, rapidly changing codebases, the method provides significant practical advantages. The platform is designed to provide future integration with big language models, increased programming language support, and enterprise-grade analytics dashboards, even if it is currently in the prototype stage.

In the end, this effort establishes a solid basis for the upcoming generation of intelligent DevOps tools, which will comprehend, explain, and change with the codebase itself in addition to automating chores.

REFERENCES

[1] Suprit Pattanayak, Pranav Murthy, Aditya Mehra (2024). Integrating AI into DevOps Pipelines: Continuous Integration, Continuous Delivery, and Automation in Infrastructural Management. International Journal of Science and Research Archive.

[2] Romina Eramo, Bilal Said, Marc Oriol (2024). An Architecture for Model-Based and Intelligent Automation in DevOps. Journal of Systems and Software, Elsevier.
[3] Bharath Chandra Vadde, Vamshi Bharath Munagandla (2022). AI-Driven Automation in DevOps: Enhancing Continuous Integration and Deployment. Conference on Computer Vision and Pattern Recognition (CVPR).
[4] GitHub Webhooks Documentation. https://docs.github.com/en/developers/web hooks-and-events/webhooks/aboutwebhooks
[5] OpenAI (2023). GPT-4 Technical Report. https://openai.com/research/gpt-4
[6] Vaswani et al. (2017). Attention Is All You Need. Advances in Neural Information Processing Systems.
[7] Microsoft Docs. DevOps with GitHub Actions. https://learn.microsoft.com/enus/devops/



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY