iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VII **Month of publication:** July 2026

DOI: https://doi.org/10.22214/ijraset.2026.84056

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 VII July 2026- Available at www.ijraset.com

Auto-Devops GPT: An Agentic AI Framework for Self-Healing CI/CD Pipelines Using LLM-Based Root Cause Analysis and Reinforcement Learning

N. Vijay[1], A. Jitendra[2]
[1, 2]*Department of Computer Science and Engineering, Holy Mary Institute of Technology & Science, Hyderabad, Telangana, India*

Abstract: ***Continuous Integration and Continuous Deployment (CI/CD) pipelines are essential for modern software delivery, but frequent failures in build, test, dependency, configuration and deployment stages reduce reliability and increase recovery time. Traditional CI/CD tools execute predefined workflows but generally require manual log inspection and human intervention after failures. This paper proposes AutoDevOps GPT, an agentic artificial intelligence framework for self-healing CI/CD pipelines. The framework integrates Jenkins webhook events, log-based failure classification, Large Language Model-assisted root cause analysis, multi-agent recovery planning, safe remediation execution, incident storage and reinforcement learning-based recovery optimisation. The proposed system uses an Analyzer Agent to classify failures, a Planner Agent to select recovery actions and an Executor Agent to apply predefined safe remediation. A lightweight Q-learning-inspired policy updates recovery action preferences using success, time and cost feedback. The system is implemented using Python Flask, SQLite, SQLAlchemy, Jenkins, GitHub, Docker, Kubernetes support and a dashboard for incident intelligence. Experimental evaluation shows improvements in detection time, recovery time, recovery accuracy and automation efficiency compared with traditional CI/CD, rule-based recovery and basic AIOps baselines.***

Keywords: AutoDevOps GPT, Agentic AI, CI/CD Pipelines, Self-Healing DevOps, Reinforcement Learning

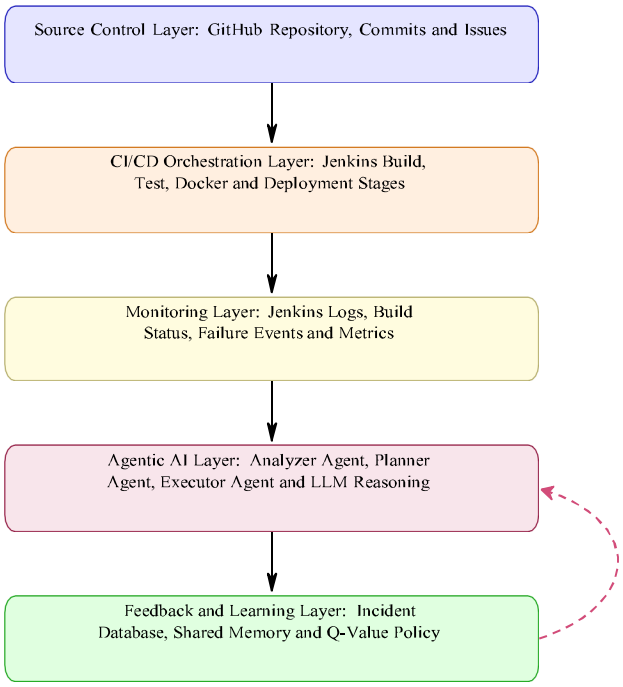


Figure 1: Layered architecture of the proposed AutoDevOps GPT framework.

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

I. INTRODUCTION

Continuous Integration and Continuous Deployment (CI/CD) pipelines have become a fundamental part of modern software engineering. They automate repetitive activities such as source code checkout, dependency installation, compilation, testing, packaging, container image creation and deployment. Con-tinuous delivery practices improve release reliability by mak-ing build, test and deployment activities repeatable and trace-able [1]. DevOps further extends this practice by encouraging collaboration between development and operations teams, sup-ported by automation, monitoring and continuous feedback [2]. Despite these advantages, CI/CD pipelines frequently fail in practice. Common failure sources include missing depen-dencies, build errors, unstable tests, timeout conditions, incor-rect environment variables, expired credentials, container image failures and Kubernetes deployment errors. When such failures occur, conventional CI/CD systems usually stop the pipeline and wait for a developer or DevOps engineer to inspect logs, diagnose the cause and manually apply a corrective ac-tion. This process increases Mean Time to Recovery (MTTR), delays releases and reduces developer productivity.

Site Reliability Engineering (SRE) treats reliability as a soft-ware engineering problem and emphasises automation, moni-toring and rapid incident response [3]. Artificial Intelligence for IT Operations (AIOps) has also introduced machine learn-ing techniques for anomaly detection, event analysis and op-erational decision support [4]. However, many AIOps tools remain alert-oriented and do not provide complete closed-loop recovery for CI/CD failures.

Large Language Models (LLMs) provide a new opportunity for interpreting unstructured logs and explaining error mes-sages. Foundation models have shown broad language under-standing and reasoning capabilities [6], while chain-of-thought and zero-shot reasoning methods demonstrate improved multi-step reasoning performance [7, 8]. Nevertheless, raw LLM outputs cannot be directly executed in DevOps environments because they may be incomplete, unsafe or unsupported by op-erational context.

This paper proposes *AutoDevOps GPT*, an agentic AI frame-work for self-healing CI/CD pipelines. The framework com-bines rule-based failure detection, LLM-assisted root cause analysis, multi-agent recovery planning, safe remediation and reinforcement learning-inspired feedback. The main contribu-tions of this paper are as follows:

- an agentic AI architecture for CI/CD failure diagnosis and recovery;
- a hybrid failure analysis method combining log rules and LLM reasoning;
- a safe recovery mechanism based on predefined remedia-tion actions;
- a Q-learning-inspired policy for improving recovery deci-sions;
- a Flask-based implementation with Jenkins, GitHub, SQLite and dashboard support;
- an experimental comparison against traditional CI/CD, rule-based recovery and AIOps baselines.

II. RELATED WORK

A. CI/CD and DevOps Automation

CI/CD pipelines automate the software delivery lifecycle and reduce manual release effort. Humble and Farley established continuous delivery principles for reliable build, test and de-ployment automation [1]. DevOps practices further integrate development and operations through shared responsibility, au-tomation and feedback loops [2]. However, most CI/CD pipelines are still defined through static scripts or configuration files. These systems are effective during normal execution but have limited capability for autonomous recovery when failures occur.

B. SRE, AIOps and Anomaly Detection

SRE promotes automation and reliability engineering for large-scale systems [3]. AIOps applies artificial intelligence to oper-ational data such as logs, events, alerts and metrics. Dang et al. describe real-world AIOps challenges and research opportuni-ties in operational systems [4]. Anomaly detection methods identify abnormal patterns in logs and metrics and are useful for early failure detection [5]. However, detection alone is in-sufficient for self-healing CI/CD. A complete system must also diagnose, plan, execute and learn from recovery outcomes.

C. LLMs and Agentic AI

LLMs can support software engineering tasks such as sum-marisation, code generation, error explanation and log inter-pretation. However, DevOps automation requires controlled action execution. Agentic AI structures autonomy through spe-cialised agents that observe context, reason about tasks, select actions and interact with tools. In CI/CD recovery, this en-ables separation of responsibilities among monitoring, analy-sis, planning, execution and governance modules.

D. Reinforcement Learning for Operational Optimisation

Reinforcement learning provides a mechanism for improving decisions from feedback. Sutton and Barto describe how agents learn action preferences through reward signals [9]. In a CI/CD recovery context, each failure type can be treated as a state and each recovery operation as an action. Successful, fast and low-cost recovery actions should receive higher values over time.

The research gap identified from existing work is that few systems combine LLM-based log reasoning, agentic planning, safe remediation, incident storage and learning-based optimisation into a single closed-loop CI/CD recovery framework.

Table 1: Failure types and candidate recovery actions

Failure Type	Candidate Recovery Actions
dependency_failure	fix_dependencies, retry_pipeline
build_failure	create_github_issue, no_op
test_failure	retry_pipeline, create_github_issue
timeout_failure	retry_pipeline, restart_service
deployment_failure	rollback_deployment, restart_service
configuration_failure	create_github_issue, no_op
unknown_failure	no_op, create_github_issue

III. PROPOSED AUTODEVOPS GPT FRAME-WORK

AutoDevOps GPT is designed as a closed-loop self-healing framework for CI/CD pipelines. The framework receives Jenk-ins pipeline failure events, analyses logs, identifies root causes, plans safe recovery actions, executes remediation and updates recovery policies.

A. Layered Architecture

The proposed architecture consists of five layers:

1) Source Control Layer: GitHub repository, commits and issue creation.
2) CI/CD Orchestration Layer: Jenkins pipeline execu-tion.
3) Monitoring Layer: Jenkins logs, build status and failure events.
4) Agentic AI Layer: Analyzer Agent, Planner Agent, Ex-ecutor Agent and LLM service.
5) Feedback and Learning Layer: SQLite incident storage and reinforcement learning policy.

B. Agent Responsibilities

The Analyzer Agent classifies the failure type and identifies the probable root cause. The Planner Agent maps the failure con-text to candidate recovery actions and selects the best safe ac-tion. The Executor Agent applies the selected action using con-trolled services. This modular design improves interpretability and prevents uncontrolled automation.

IV. METHODOLOGY

The methodology consists of failure detection, log preprocess-ing, rule-based classification, LLM-assisted root cause analy-sis, recovery planning, safe execution and learning update.

A. Failure Types and Recovery Actions

Table 1 summarises the failure types and recovery actions con-sidered in the proposed framework.

B. Failure Detection and Log Analysis

Jenkins sends a webhook payload containing job name, build number, status, stage and logs. The Monitoring Service ex-tracts relevant error lines and applies pattern matching. For example, ModuleNotFoundError indicates a dependency failure, while CrashLoopBackOff indicates a deployment failure. Unknown or ambiguous logs are forwarded to the LLM service for semantic interpretation.

C. LLM-Based Root Cause Analysis

The LLM service receives preprocessed logs and returns a structured response containing failure type, root cause, confi-dence score and recommended actions. The system treats LLM output as advisory. It does not execute arbitrary commands suggested by an LLM. All recommendations are mapped to a safe action whitelist before execution.

D. Recovery Planning and Execution

The Planner Agent receives the failure state s and candidate actions A. It selects the action with the highest learned value while enforcing safety constraints:

$$a^* = \arg\max_{a \in A} Q(s, a) \quad (1)$$

where $Q(s, a)$ is the current policy value for action a in state s. The Executor Agent performs only predefined actions such as retry_pipeline, rollback_deployment, fix_dependencies and create_github_issue.
High-risk operations require human approval.

E. Reinforcement Learning-Inspired Update

After each recovery attempt, a reward is computed:

$$R = \alpha \times Success - \beta \times Time - \gamma \times Cost \quad (2)$$

where *Success* indicates recovery result, *Time* is recovery du-ration, *Cost* is estimated operational cost, and α, β and γ are weighting parameters.
The policy is updated as:

$$Q(s, a) \leftarrow Q(s, a) + \eta[R - Q(s, a)] \quad (3)$$

where η is the learning rate. This update gradually increases preference for successful and efficient actions.
Figure 2 presents the self-healing workflow.

V. IMPLEMENTATION

The prototype was implemented using Python Flask as the backend framework. Jenkins provides CI/CD pipeline events, while GitHub is used for source control and issue creation sup-port. SQLite with SQLAlchemy stores incidents and policy values. Docker and Kubernetes manifests support containerised deployment.
Table 2 lists the main implementation components.
The dashboard displays total incidents, resolved incidents, failed incidents, pending approval incidents, average reward and recent pipeline failures. The system can run in simulation mode when Jenkins, GitHub or LLM credentials are unavail-able. This design supports reproducible academic evaluation without requiring a production CI/CD environment.

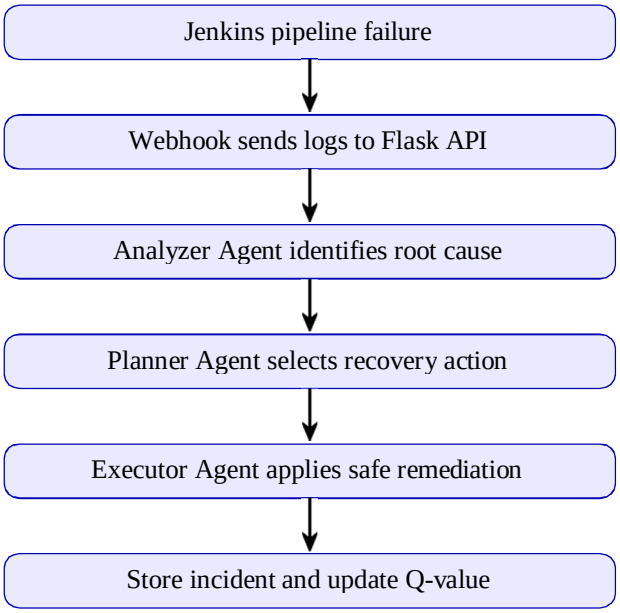


Figure 2: Self-healing workflow followed by AutoDevOps GPT.

Table 2: Implementation components.

Component	Implementation
Backend API	Python Flask REST endpoints
Database	SQLite and SQLAlchemy ORM
CI/CD Input	Jenkins webhook payloads
Source Control	GitHub issue creation support
Agents	Analyzer, Planner and Executor modules
LLM Service	GPT-compatible service with fallback mode
Learning Module	Q-learning-inspired policy table
Dashboard	HTML, CSS and JavaScript templates
Deployment	Docker and Kubernetes support

VI. EXPERIMENTAL SETUP AND RESULTS

The experimental evaluation used representative CI/CD failure scenarios including dependency failure, build failure, test fail-ure, timeout failure, deployment failure, configuration failure and unknown failure. AutoDevOps GPT was compared with three baselines: traditional CI/CD, rule-based recovery and ba-sic AIOps monitoring.

A. Evaluation Metrics

The evaluation used six metrics: Failure Detection Time (FDT), Mean Time to Recovery (MTTR), Pipeline Success Rate (PSR), Resource Utilisation (RU), Recovery Accuracy (RA) and Automation Efficiency (AE).

B. Overall Results

Table 3 presents the overall performance comparison.

Figure 3 compares three representative metrics: failure de-tection time, MTTR and recovery accuracy.

The proposed system reduced failure detection time from 120 seconds to 35 seconds and reduced MTTR from 25 minutes to 8 minutes. It also improved recovery accuracy from 68% to 93%. These improvements are mainly due to webhook-driven detection, agentic analysis, LLM-assisted reasoning and auto-mated safe recovery.

Table 3: Performance comparison of AutoDevOps GPT with baseline systems.

Metric	Traditional CI/CD	Rule-Based Recovery	Basic AIOps	AutoDevOps GPT
Failure Detection Time	120 sec	90 sec	60 sec	35 sec
Mean Time to Recovery	25 min	18 min	12 min	8 min
Pipeline Success Rate	78%	85%	89%	94%
Resource Utilisation	65%	70%	75%	82%
Recovery Accuracy	68%	79%	86%	93%
Automation Efficiency	35%	62%	70%	88%

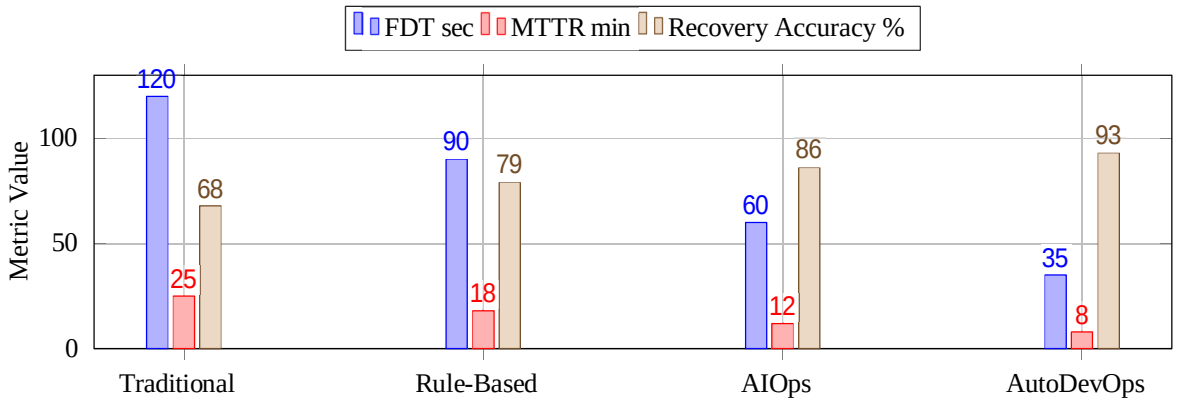


Figure 3: Comparison of detection time, recovery time and recovery accuracy.

Table 4: Ablation study of AutoDevOps GPT components.

Configuration	SR	MTTR	RA
Full system	94%	8	93%
Without RL policy	88%	14	87%
Without Planner Agent	86%	16	84%
Without Analyzer Agent	82%	20	80%
Without LLM Service	84%	18	82%
Without Shared Memory	87%	15	85%

C. Ablation Study

Table 4 shows the contribution of major components. Remov-ing the Planner Agent, Analyzer Agent, LLM Service or RL policy reduces performance.

The ablation results show that the full system performs best. The Analyzer Agent improves classification quality, the Plan-ner Agent improves recovery action selection, the LLM Service helps with ambiguous logs and the RL policy improves future decision-making.

VII. DISCUSSION

The results indicate that closed-loop agentic recovery is more effective than traditional CI/CD failure handling. Traditional CI/CD systems provide automation for normal execution but depend on engineers for recovery. Rule-based recovery im-proves response for known errors but lacks adaptability. Basic AIOps improves detection but is often limited to monitoring and alerting.

AutoDevOps GPT improves failure detection because Jenk-ins webhooks immediately send failure information to the Flask backend. It reduces MTTR because diagnosis, planning and safe execution are performed automatically. Recovery ac-curacy improves because the system combines deterministic rules, LLM-assisted reasoning and policy values. The rein-forcement learning-inspired policy allows the system to favour actions that have produced successful and efficient outcomes in previous incidents.

The framework also addresses safety. Arbitrary commands generated by an LLM are not executed. Recovery is restricted to predefined actions such as pipeline retry, rollback, depen-dency repair recommendation and issue creation. This is nec-essary because CI/CD systems may affect production infras-tructure, credentials and deployment environments. Human ap-proval remains important for high-risk changes, consistent with human-centred machine learning principles [10].

The current prototype has limitations. It uses representa-tive scenarios rather than a large enterprise dataset. SQLite is suitable for demonstration but not ideal for high-concurrency production workloads. LLM-based analysis may introduce la-tency and cost. Incomplete logs can reduce root cause accu-racy. Therefore, future evaluation should include real industrial pipelines, production observability data and stronger security controls.

VIII. CONCLUSION

This paper presented AutoDevOps GPT, an agentic AI frame-work for self-healing CI/CD pipelines. The framework inte-grates Jenkins failure detection, log classification, LLM-based root cause analysis, multi-agent recovery planning, safe reme-diation execution, incident storage and reinforcement learning-inspired policy update. The system is implemented using Flask, SQLite, Jenkins, GitHub, Docker and Kubernetes sup-port.

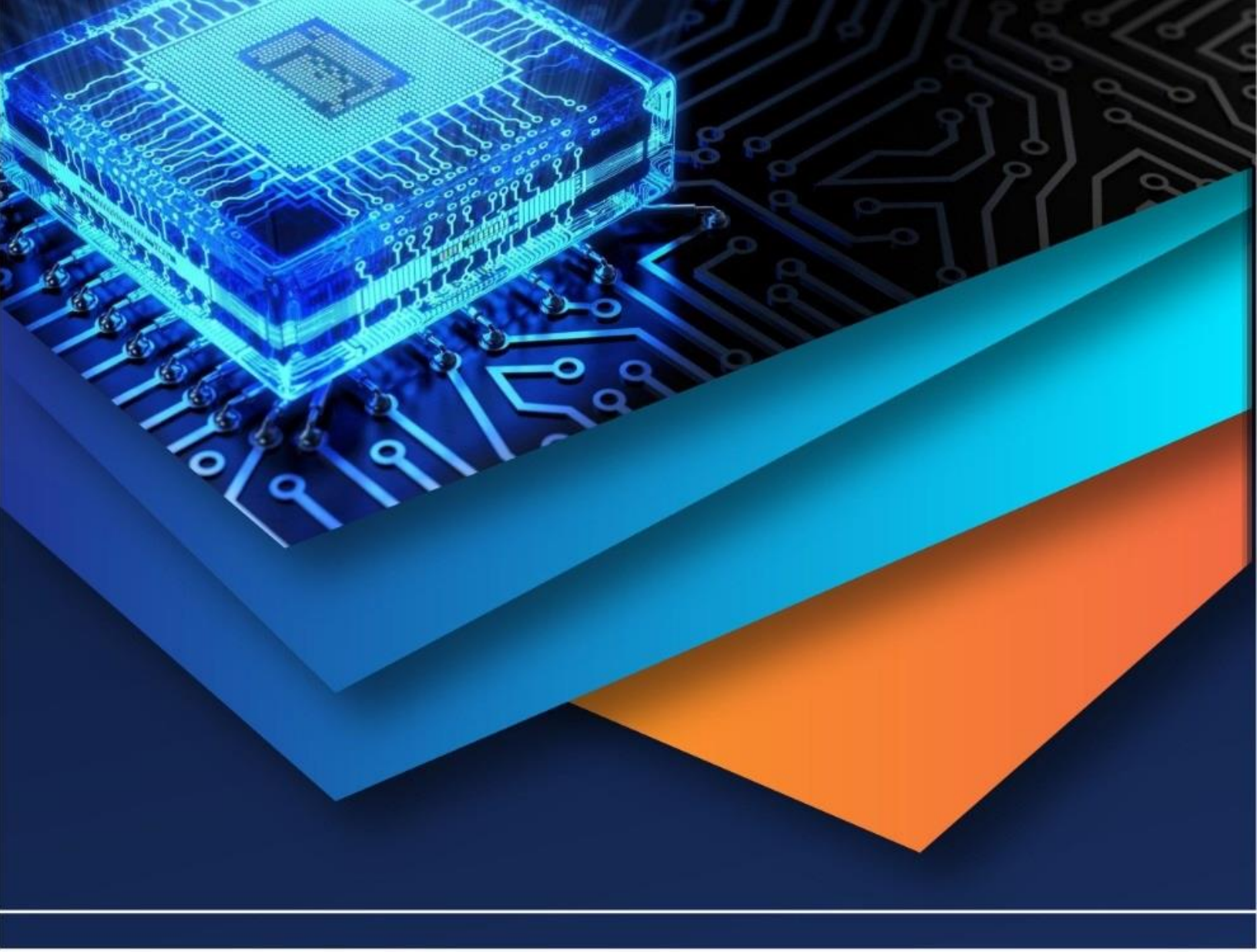
Experimental results show that AutoDevOps GPT improves all evaluated metrics compared with traditional CI/CD, rule-based recovery and basic AIOps baselines. Failure detection time decreased from 120 seconds to 35 seconds, MTTR de-creased from 25 minutes to 8 minutes, pipeline success rate increased from 78% to 94%, recovery accuracy increased from 68% to 93% and automation efficiency increased from 35% to 88%.Future work will extend the framework to GitHub Actions, GitLab CI and Azure DevOps. Additional improvements in-clude Kubernetes-native operators, Prometheus and Grafana integration, automated pull request generation, explainable AI reports, enterprise-grade security policies and large-scale pro-duction evaluation.

IX. ACKNOWLEDGEMENTS

The authors thank the Department of Computer Science and Engineering, Holy Mary Institute of Technology & Science, Hyderabad, India, for providing academic support and facilities to carry out this work.

REFERENCES

[1] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston, MA, USA: Addison-Wesley, 2010.
[2] G. Kim, J. Humble, P. Debois, J. Willis, and N. Forsgren, The DevOps Handbook. Portland, OR, USA: IT Revolu-tion Press, 2016.
[3] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Site Reli-ability Engineering. Sebastopol, CA, USA: O'Reilly Me-dia, 2016.
[4] Y. Dang, Q. Lin, and P. Huang, "AIOps: Real-world chal-lenges and research innovations," in Proc. IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, Montreal, Canada, 2019, pp. 4–5.
[5] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly de-tection: A survey," ACM Computing Surveys, vol. 41, no. 3, pp. 1–58, 2009.
[6] R. Bommasani, D. A. Hudson, E. Adeli, et al., "On the opportunities and risks of foundation models," arXiv preprint arXiv:2108.07258, 2021.
[7] J. Wei, X. Wang, D. Schuurmans, et al., "Chain-of-thought prompting elicits reasoning in large language models," in Advances in Neural Information Processing Systems, vol. 35, 2022, pp. 24824–24837.
[8] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwa-sawa, "Large language models are zero-shot reasoners," in Advances in Neural Information Processing Systems, vol. 35, 2022, pp. 22199–22213.
[9] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.
[10] S. Amershi, M. Cakmak, W. B. Knox, and T. Kulesza, "Power to the people: The role of humans in interactive machine learning," AI Magazine, vol. 35, no. 4, pp. 105–120, 2014.
[11] Jenkins, "Jenkins documentation," Available: https://www.jenkins.io/doc/, accessed 21 June 2026.
[12] Kubernetes, "Kubernetes documentation," Available: https://kubernetes.io/docs/, accessed 21June 2026.
[13] Docker, "Docker documentation," Available: https:// docs.docker.com/, accessed 21 June 2026.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY