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.2026.84057

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

AutoInfraOps: Infrastructure Deployment via Multi-Step Prompt Pipelines

G. Vaishnavi, Mrs. G. Devi
Department of Computer Science and Engineering Holy Mary Institute of Technology & Science Hyderabad, India

Abstract: ***Cloud-native software delivery increasingly depends on DevOps pipelines, Infrastructure-as-Code (IaC), container orchestration, and continuous integration and deployment (CI/CD) workflows. However, many conventional deployment processes remain static, script-driven, and highly dependent on manual configuration. Such approaches provide repeatability but of-fer limited contextual reasoning, weak adaptation to changing infrastructure requirements, and insufficient feedback-driven optimization. This paper presents AutoInfraOps, a simulation-safe intelligent infrastructure deployment framework based on multi-step prompt pipelines. The proposed framework converts high-level natural language deployment requests into structured requirements, enriched prompts, LLM-assisted reasoning outputs, validated deployment plans, simulated execution logs, monitoring metrics, and feedback suggestions. AutoInfraOps integrates prompt generation, context enrichment, validation and governance, execution simulation, deployment history, and feedback refinement into a unified workflow. The prototype is implemented as a full-stack web application with a React/Vite frontend, Python backend services, SQLite storage, a mock LLM reasoning module, validation services, and a safe execu-tion simulator. Experimental evaluation using four deployment scenarios produced three successful simulations, one validation-blocked deployment, a 75% success rate, 75% prompt efficiency, 65.08 seconds average deployment time, 52.88% average resource utilization, and $31.44 average cost estimate. The results demon-strate that multi-step prompt pipelines can improve transparency, safety, and performance awareness in LLM-assisted DevOps automation while avoiding direct execution of real infrastructure commands.***

Index Terms: Large Language Models, Prompt Engineering, Multi-Step Prompt Pipelines, DevOps, CI/CD, Infrastructure Automation, AIOps, Agentic AI, Cloud Computing, Validation, Feedback Refinement.

I. INTRODUCTION

Modern software systems are increasingly delivered through cloud-native architectures, container platforms, microservices, DevOps practices, and automated CI/CD workflows. Orga-nizations rely on tools such as Docker, Kubernetes, Ter-raform, Ansible, Jenkins, GitHub Actions, and GitLab CI to improve deployment speed, reproducibility, and operational consistency. Infrastructure-as-Code further enables infrastruc-ture definitions to be version-controlled and reused across environments. However, despite these improvements, many deployment processes remain static and configuration-heavy. Deployment engineers often need to manually prepare scripts, update configuration files, resolve dependency conflicts, and debug failed pipelines.

Traditional CI/CD and IaC workflows are effective for known and repetitive deployment tasks, but they offer limited contextual reasoning. A static pipeline can execute predefined instructions but cannot easily interpret high-level intent, adapt to new infrastructure states, reason about ambiguous require-ments, or generate optimized deployment plans from natu-ral language input. This limitation becomes more significant in distributed cloud, edge, and hybrid environments, where deployment decisions depend on resource availability, policy constraints, latency requirements, security levels, cost, and application-specific requirements.

Large Language Models (LLMs) provide a promising di-rection for intelligent automation because they can understand natural language, generate structured text, reason over con-text, and interact with tool-like workflows [7], [8]. Recent research on prompt engineering, chain-of-thought reasoning, tool learning, and autonomous agents shows that LLMs can support planning and task decomposition in complex environ-ments [9]–[12]. However, direct execution of LLM-generated infrastructure code is unsafe because model outputs may be incomplete, hallucinated, non-compliant, or operationally dangerous.

This paper proposes AutoInfraOps: Infrastructure Deploy-ment via Multi-Step Prompt Pipelines. Instead of relying on a single LLM response, AutoInfraOps decomposes infrastruc-ture deployment into controlled stages: requirement structur-ing, prompt generation, context enrichment, LLM reasoning, draft plan generation, validation and governance, execution simulation, monitoring, deployment history, and feedback re-finement. The framework is intentionally simulation-safe: it does not execute real shell, Docker, Kubernetes, Terraform, or cloud commands. It demonstrates the logic of LLM-assisted DevOps automation in a controlled academic prototype.

The main contributions of this paper are:

1) A multi-step prompt pipeline architecture for LLM-assisted infrastructure deployment.
2) A context-enriched reasoning approach for converting natural language deployment intent into structured de-ployment plans.
3) A validation and governance layer for syntax, resource, security, policy, and hallucination-risk checks.
4) A simulation-safe execution engine that generates deploy-ment logs and metrics without running real infrastructure commands.
5) A monitoring and feedback refinement module for performance-aware evaluation.
6) A full-stack prototype evaluated using deployment time, cost estimate, resource utilization, success rate, and prompt efficiency.

II. RELATED WORK

A. DevOps, CI/CD, and Infrastructure-as-Code

DevOps practices aim to improve collaboration between de-velopment and operations teams and accelerate software deliv-ery [1], [2]. CI/CD pipelines automate build, test, release, and deployment stages. Tools such as Jenkins, GitHub Actions, and GitLab CI have become widely used for pipeline automation. IaC tools such as Terraform and Ansible enable infrastructure provisioning through declarative configuration files [3]. These technologies improve consistency and repeatability but are commonly limited by static templates, manual maintenance, and limited adaptive reasoning.

Cloud-native deployment further increases operational com-plexity because applications may be deployed as distributed microservices across container clusters and dynamic cloud resources. Kubernetes supports container orchestration, scal-ing, service discovery, and rolling updates [4]. However, it still requires correct manifests, policies, resource limits, and operational expertise. Misconfigurations in IaC and container orchestration can cause deployment failures, security risks, and inefficient resource usage.

B. AIOps and Intelligent Operations

AIOps applies artificial intelligence to IT operations for monitoring, anomaly detection, alert correlation, root cause analysis, and automated remediation [6]. Existing AIOps sys-tems are valuable for operational visibility, but many are reactive and focus on monitoring after deployment rather than intelligent planning before deployment. AutoInfraOps extends the AIOps direction by integrating reasoning, validation, sim-ulated execution, and feedback refinement into the deployment workflow itself.

C. LLMs, Prompt Engineering, and Agentic AI

LLMs have demonstrated strong capabilities in natural language understanding, code generation, reasoning, and tool-assisted workflows [7], [8]. Prompt engineering techniques guide LLM behavior through structured instructions, exam-ples, and task decomposition. Chain-of-thought prompting im-proves multi-step reasoning by encouraging intermediate rea-soning paths [9]. ReAct-style prompting combines reasoning and acting, allowing language models to interact with external tools and observations [10]. Tool learning and function calling further extend LLMs beyond text generation by allowing structured interaction with external systems [11]. Agentic AI systems use LLMs as planning and decision-making components that can manage goals, memory, tools, and feedback [12]. However, LLM agents also introduce safety concerns. They may generate incorrect actions, hallucinated configurations, or unsafe commands. Therefore, LLM-assisted infrastructure automation requires validation, governance, and constrained execution.

D. Research Gap

Existing deployment systems provide automation but generally lack natural language reasoning and feedback-driven adaptation. Existing LLM systems provide reasoning but often lack deterministic validation and safe execution. Existing AIOps tools provide operational monitoring but do not fully integrate deployment planning, validation, simulation, and prompt refinement. Therefore, there is a need for a unified framework that combines natural language deployment intent, multi-step prompt pipelines, LLM reasoning, validation, simulation-safe execution, monitoring, and feedback refinement. AutoInfraOps addresses this gap.

III. PROPOSED SYSTEM

AutoInfraOps is designed as a modular, simulation-safe framework for LLM-assisted infrastructure deployment. The system accepts a deployment request through a web interface and processes it through multiple pipeline stages. The major modules are:

1) User Interface Module: Provides pages for dashboard, deployment request, prompt pipeline, validation, monitor-ing, history, feedback, and project overview.
2) Deployment Request Module: Captures application name, type, environment, deployment target, CPU, mem-ory, storage, replicas, port, security level, and natural language request.

3) Prompt Generation Module: Converts the structured request into an LLM-ready deployment prompt.
4) Context Enrichment Module: Adds environment de-tails, policies, resource constraints, and deployment his-tory.
5) LLM Reasoning Module: Generates reasoning output and a draft deployment plan using a mock LLM or optional real API.
6) Validation and Governance Module: Performs syntax, resource, security, policy, and hallucination-risk checks.
7) Execution Simulation Engine: Simulates deployment steps and generates logs without executing real com-mands.
8) Monitoring and Metrics Module: Computes deploy-ment time, utilization, cost estimate, success rate, and error rate.
9) Feedback Refinement Module: Generates improved prompt suggestions and recommendations.
10) Database and History Module: Stores deployments, prompts, validation reports, execution logs, and feedback records.

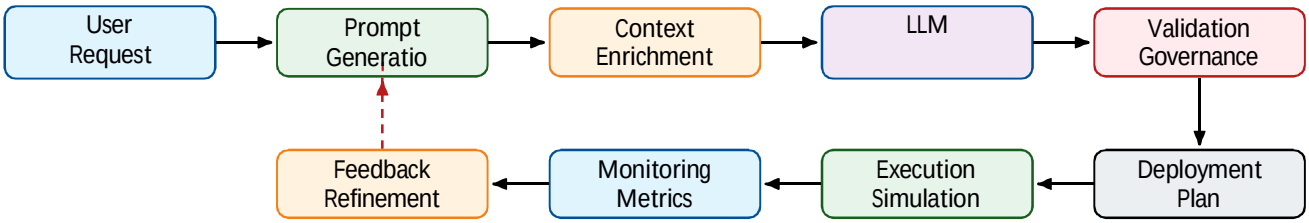


Fig. 1. Architecture of AutoInfraOps multi-step prompt pipeline.

Algorithm 1 Multi-Step LLM-Driven Infrastructure Deployment

Require: Deployment request R, infrastructure context C, policy rules P
Ensure: Validated deployment result D

Convert user request R into structured requirement S
Generate initial prompt $P_0 \leftarrow$ PROMPTGENERATOR(S)
Enrich prompt $P_1 \leftarrow$ CONTEXTENRICHMENT(P_0, C)
Generate reasoning output $L \leftarrow$ LLMREASONING(P_1)
Generate draft deployment plan D_{draft} from L
Validate plan $V \leftarrow$ VALIDATE(D_{draft}, P)
if V = false **then**
    Generate validation feedback F_x
    Store failed validation record and feedback
    return blocked deployment result
else
    Format final deployment plan D_{plan}
    Simulate deployment execution $E \leftarrow$ SIMULATE(D_{plan})
    Collect logs and metrics $M \leftarrow$ MONITOR(E)
    Store deployment history (D_{plan}, E, M)
    Generate improved prompt suggestion F_p
    return complete deployment result D
end if

Fig. 1 shows the closed-loop pipeline. The feedback path is important because it converts validation outcomes and execution metrics into improved prompt suggestions for later deployments.

IV. METHODOLOGY

The methodology begins with requirement acquisition. A user submits structured values and a natural language deploy-ment request. The system converts this input into a structured requirement object. A prompt is generated using deployment context and constraints. The context enrichment stage adds environment, policy, history, and validation requirements. The LLM reasoning module generates a draft plan. The validation module then checks whether the plan is safe and compliant. If validation fails, the plan is blocked and feedback is generated. If validation passes, the execution simulator produces logs, status, and metrics. Finally, the system stores history and generates feedback for prompt refinement.

The validation module applies several deterministic rules. CPU, memory, and storage must be greater than zero. Replicas

TABLE I
PROTOTYPE IMPLEMENTATION STACK

Layer	Technology / Component
Frontend	React.js, Vite, sidebar navigation, dashboard pages
Backend	Python REST API services
Database	SQLite deployment history storage
LLM Layer	Mock LLM engine with optional API integration
Validation	Rule-based syntax, resource, security, and policy checks
Execution	Simulation-only deployment engine
Monitoring	Deployment time, cost, utilization, error rate, prompt efficiency

must be within an acceptable limit. Port values must fall within the allowed range. Production deployments require medium or high security. Unsafe command patterns such as destructive file operations or privileged execution are blocked. This separation between LLM reasoning and deterministic validation is a major safety design feature.

V. IMPLEMENTATION

The AutoInfraOps prototype was implemented as a full-stack web application. The frontend was developed using React.js with Vite. It includes a sidebar-based interface with pages for dashboard, new deployment, prompt pipeline, vali-dation, monitoring, history, feedback, and about. The backend was implemented using Python-based REST services. SQLite was used for local storage. The LLM layer uses a mock reasoning engine by default, with optional support for external API integration through environment variables. The backend contains separate services for prompt genera-tion, context enrichment, LLM reasoning, validation, execution simulation, monitoring, and feedback. The execution simulator generates realistic logs such as initialization, configuration reading, resource validation, image building, network con-figuration, resource provisioning, scaling policy application, service startup, health checks, and deployment completion. No real shell, Docker, Kubernetes, Terraform, or cloud command is executed.

VI. EXPERIMENTAL SETUP

The prototype was evaluated locally through the frontend interface at http://127.0.0.1:5173. Four deployment scenarios were used: ResearchPortal, MetricsAPI, LegacyWorker, and ExperimentUI. The deployment tar-gets included Kubernetes, Docker, and Terraform-style sim-ulation. The environments included staging, production, and

TABLE II
Database Tables in AutoInfraOps

Table	Purpose
deployments	Stores request, status, metrics, and configuration
prompt_records	Stores generated prompt, context, LLM output, and plans
validation_reports	Stores syntax, resource, security, policy, and risk checks
execution_logs	Stores step-by-step simulation logs
feedback_records	Stores feedback summaries and improved prompts

TABLE III
Deployment Results from AutoInfraOps Prototype

Application	Env.	Target	Status	Time
ResearchPortal	Staging	Kubernetes	Success	92.7s
MetricsAPI	Production	Docker	Success	84.6s
LegacyWorker	Production	Terraform	Validation Failed	0s
ExperimentUI	Development	Docker	Success	83.0s

development. The monitored metrics were deployment time, success count, failure or blocked count, cost estimate, resource utilization, prompt efficiency, and validation status.

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

The evaluation is based on simulated deployment records generated by the prototype. The purpose of the evaluation is not to measure real cloud deployment performance but to demonstrate the functional correctness, safety design, and performance-awareness of the AutoInfraOps workflow.

VII. RESULTS AND DISCUSSION

A. *Deployment Results*

Table III summarizes the deployment records produced by the prototype. Three deployment simulations succeeded and one deployment was blocked by validation.

The validation failure of LegacyWorker should not be in-terpreted as a weakness of the system. Instead, it demonstrates the role of the governance layer. A deployment that fails safety or policy checks is intentionally blocked before execution. This behavior is desirable for LLM-assisted infrastructure automation because it reduces the risk of executing unsafe or unsupported actions.

B. *Dashboard Metrics*

The dashboard summarized the four deployment attempts using aggregate metrics. Table V presents the observed values.

The average deployment time is computed as:

$$T_{avg} = \frac{92.7 + 84.6 + 0 + 83.0}{4} = 65.08 \text{ seconds.} \quad (1)$$

The average resource utilization is:

$$U_{avg} = \frac{76 + 70.5 + 0 + 65}{4} = 52.88\%. \quad (2)$$

The average cost estimate is:

$$C_{avg} = \frac{55.55 + 40.95 + 8.91 + 20.35}{4} = 31.44. \quad (3)$$

TABLE IV
COST AND RESOURCE UTILIZATION RESULTS

Application	Cost Estimate	Utilization
ResearchPortal	$55.55	76%
MetricsAPI	$40.95	70.5%
LegacyWorker	$8.91	0%
ExperimentUI	$20.35	65%

TABLE V
DASHBOARD SUMMARY METRICS

Metric	Value
Total deployments	4
Successful deployments	3
Failed or blocked deployments	1
Average deployment time	65.08 seconds
Average resource utilization	52.88%
Average cost estimate	$31.44
Prompt efficiency	75%
Success rate	75%
Failure or block rate	25%

Prompt efficiency is calculated as:

$$P_{eff} = \frac{3}{4} \times 100 = 75\%. \quad (4)$$

These results show that the system can store and aggregate deployment records into meaningful metrics. The monitoring module provides visibility into performance and supports comparison across deployment scenarios.

C. Prompt Pipeline and Validation Behavior

The prompt pipeline output demonstrates that the system exposes intermediate reasoning stages rather than hiding LLM processing as a black box. For a selected deployment, the system displays user intent, generated prompt, structured requirement, reasoning output, validation status, final plan, and feedback. This improves transparency and makes the deployment decision easier to audit.

The validation report contains five checks: syntax validation, resource validation, security validation, policy validation, and hallucination-risk check. The selected successful deployment passed all governance checks. This confirms that AutoInfraOps separates generative reasoning from deterministic validation, which is important for safety in infrastructure automation.

D. Feedback Refinement

The feedback refinement module generates improved prompt suggestions based on validation and execution out-comes. For a successful deployment, the system recommended simulation-safe planning, validation gates, monitoring metrics, rollback notes, and cost optimization. Thus, feedback is not limited to failure handling; it also supports continuous im-provement after successful deployment.

VIII. COMPARATIVE ANALYSIS

Table VI compares traditional static deployment with Au-toInfraOps. Traditional deployment relies on scripts and manual configuration, whereas AutoInfraOps supports natural language input, prompt reasoning, context enrichment, validation, monitoring, and feedback refinement.

TABLE VI
Comparison with Traditional Static Deployment

Feature	Traditional Deployment	AutoInfraOps
Input	Scripts/forms	Natural language plus structured input
Reasoning	Limited	Multi-step prompt reasoning
Context	Manual	Context enrichment
Validation	Tool-specific or manual	Integrated governance checks
Execution	Direct commands	Simulation-safe execution
Monitoring	Separate tools	Integrated metrics dashboard
Feedback	Limited	Improved prompt suggestions
Traceability	Log-dependent	Prompt, validation, logs, metrics, history
Safety	Command risks	No real command execution

The comparison indicates that AutoInfraOps offers a safer and more transparent workflow for academic demonstration of intelligent deployment. It is especially useful for explaining how LLM reasoning can be combined with validation and feedback before any real infrastructure execution is considered.

IX. LIMITATIONS

The current version of AutoInfraOps has several limitations. First, execution is simulation-only and does not deploy real cloud infrastructure. Second, the LLM reasoning module uses a mock engine by default, so the reasoning quality may differ from advanced LLM APIs. Third, the experimental dataset contains only four sample deployment records. Fourth, vali-dation is rule-based and can be extended with policy-as-code engines. Fifth, the prototype does not include production-grade authentication, role-based access control, provider-specific cloud APIs, or real-time cloud monitoring.

These limitations are intentional in the current stage because the objective is to develop a safe academic prototype that demonstrates the core workflow of LLM-assisted infrastructure deployment.

X. FUTURE WORK

Future work will focus on extending AutoInfraOps from a simulation-based prototype to a practical deployment as-sistant. The execution engine can be integrated with real cloud platforms such as AWS, Microsoft Azure, and Google Cloud. Kubernetes cluster integration can be added to create deployments, services, ingress rules, autoscaling policies, and health checks. Terraform plan generation and policy-as-code validation can be incorporated to improve correctness and compliance. Retrieval-Augmented Generation can be used to enrich prompts with infrastructure documentation, organizational policies, and previous deployment logs. Reinforcement learn-ing can be explored for optimizing resource allocation, cost, and deployment time. Self-healing infrastructure can be im-plemented by analyzing failed logs and generating validated remediation actions. Multi-cloud and hybrid-cloud support, role-based access control, explainable deployment decisions, and CI/CD platform integration are also important future directions.

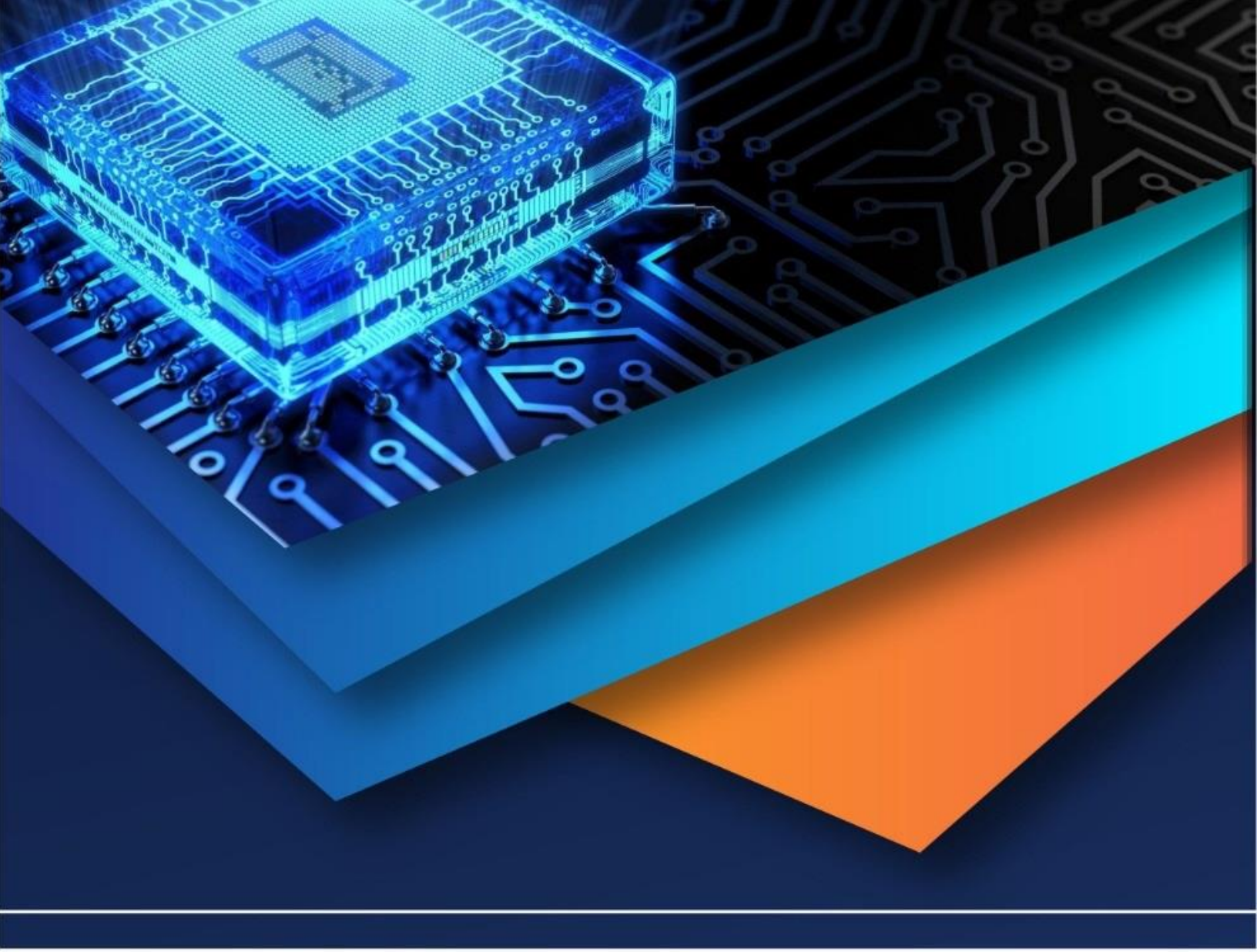
XI. CONCLUSION

This paper presented AutoInfraOps, an intelligent infrastructure deployment framework based on multi-step prompt pipelines. The proposed system converts high-level deploy-ment intent into structured prompts, enriched context, LLM-assisted reasoning outputs, validated deployment plans, simulated execution logs, monitoring metrics, and feedback sugges-tions. The framework emphasizes safety by preventing direct execution of real shell, Docker, Kubernetes, Terraform, or cloud commands.

The prototype evaluation demonstrated that AutoInfraOps can process deployment requests, maintain deployment history, validate generated plans, simulate execution, compute per-formance metrics, and generate prompt refinement feedback. The system achieved a 75% success rate and 75% prompt efficiency across four deployment scenarios. The validation-blocked deployment demonstrates the importance of gover-nance in LLM-assisted DevOps workflows. Overall, AutoIn-fraOps provides a safe, transparent, and performance-aware foundation for next-generation DevOps automation using LLM reasoning and multi-step prompt pipelines.

REFERENCES

[1] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010.
[2] L. Bass, I. Weber, and L. Zhu, DevOps: A Software Architect's Perspec-tive. Addison-Wesley, 2015.
[3] K. Morris, Infrastructure as Code: Managing Servers in the Cloud. O'Reilly Media, 2016.
[4] The Kubernetes Authors, "Kubernetes Documentation," 2024. [Online]. Available: https://kubernetes.io/docs/
[5] HashiCorp, "Terraform Documentation," 2024. [Online]. Available: https://developer.hashicorp.com/terraform/docs
[6] P. Notaro, J. Cardoso, and M. Gerndt, "A survey of AIOps methods for failure management," ACM Computing Surveys, vol. 54, no. 11, pp. 1–36, 2021.
[7] T. Brown et al., "Language models are few-shot learners," in Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901, 2020.
[8] OpenAI, "GPT-4 technical report," arXiv preprint arXiv:2303.08774, 2023.
[9] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large lan-guage models," in Advances in Neural Information Processing Systems, vol. 35, pp. 24824–24837, 2022.
[10] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in International Conference on Learning Representations, 2023.
[11] T. Schick et al., "Toolformer: Language models can teach themselves to use tools," in Advances in Neural Information Processing Systems, vol. 36, 2023.
[12] L. Wang et al., "A survey on large language model based autonomous agents," Frontiers of Computer Science, vol. 18, no. 6, 2024.
[13] M. Chen et al., "Evaluating large language models trained on code," arXiv preprint arXiv:2107.03374, 2021.
[14] A. Madaan et al., "Self-refine: Iterative refinement with self-feedback," in Advances in Neural Information Processing Systems, 2023.
[15] G. Li et al., "CAMEL: Communicative agents for mind exploration of large language model society," arXiv preprint arXiv:2303.17760, 2023.
[16] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. MIT Press, 2018.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY