The Review of Contemporary Scientific and Academic Studies
An International Multidisciplinary Online Journal
www.thercsas.com
ISSN: 2583-1380 Vol. 6 | Issue No. 3 | March 2026 Impact Factor: 6.53

Autonomous DevOps Platforms: The Role of Generative AI in CI/CD Optimization and Infrastructure Management

Pruthvi Raj Seknametla (pruthviraj9369@gmail.com, pruthviraj.seknametla@ieee.org)
Independent Researcher, USA

 Crossref/DOI: https://doi.org/10.55454/rcsas.6.03.2026.002

Abstract: *The convergence of large language model capabilities with established DevOps toolchains has created a new category of platform engineering: autonomous DevOps systems capable of interpreting natural language intent, generating contextually appropriate infrastructure configurations, diagnosing pipeline failures, and orchestrating remediation responses without requiring engineering specialists to drive every step. This paper proposes and evaluates the Generative DevOps Orchestration Platform (GenDOP), a framework that integrates a fine-tuned instruction-following model into the core decision-making layer of a CI/CD and infrastructure management pipeline. GenDOP introduces four functional capabilities — natural language pipeline synthesis, failure explanation and repair suggestion, context-aware Infrastructure-as-Code generation, and deployment risk scoring — and evaluates them against traditional approaches across twelve engineering teams over fourteen months. Empirical results demonstrate a 58 percent reduction in pipeline configuration time, a 71 percent improvement in mean time to pipeline failure resolution, a 44 percent reduction in infrastructure misconfiguration incidents, and a 63 percent reduction in onboarding time for engineers new to the platform. We characterize the trust, governance, and auditability requirements that distinguish GenDOP deployments producing sustainable improvements from those producing new categories of automation-induced risk.*

Keywords: Autonomous DevOps, CI/CD Optimization, Generative AI; GenDOP, Large Language Models

Article History: **Received: 19 March- 2026;** **Accepted: 25 March- 2026;** **Published/Available Online: 30 March- 2026**

1. Introduction

1.1. Problem Background

Platform engineering has become one of the most expertise-intensive disciplines in software delivery. The average cloud-native DevOps platform in 2026 involves a dozen or more interconnected tools — version control, CI/CD orchestration, container registries, secret management, infrastructure provisioning, policy enforcement, observability, and incident management — each with its own configuration language, operational model, and failure vocabulary. The engineers who configure and maintain these platforms carry an enormous cognitive burden: not just the technical knowledge of each individual tool, but the operational understanding of how they interact, fail, and recover under real production conditions.

This expertise density creates three compounding problems. First, platform capability is bottlenecked by specialist knowledge: when a pipeline fails in a way that only two engineers on a team of twenty understand, incident resolution depends on those two engineers being available, which in a follow-the-sun delivery model is not always the case. Second, platform configuration quality is inconsistent: IaC modules written by engineers with varying levels of cloud provider experience produce infrastructure with inconsistent security posture, resource sizing, and operational instrumentation. Third, developer onboarding onto complex platforms is slow: understanding enough about a Kubernetes-based delivery platform to contribute confidently often requires weeks of ramp-up that reduces the engineering throughput of growing teams.

Generative AI — specifically, large language models capable of instruction-following, code generation, and contextual reasoning over technical documentation and operational logs — offers a mechanism to partially decompress this expertise bottleneck. A platform that can interpret a developer's intent expressed in natural language and produce a syntactically and semantically correct pipeline configuration reduces the barrier between intent and deployment. A platform that can analyze a failing build log and identify the root cause in terms that both junior and senior engineers can act on reduces the specialist dependency in incident resolution. A platform that can generate an IaC module from a natural language description of infrastructure requirements, consistent with the organization's established security and operational patterns, makes the expertise encoded in senior engineers' prior work replicable without requiring their direct involvement in every new provisioning task.

The Review of Contemporary Scientific and Academic Studies
An International Multidisciplinary Online Journal
www.thercsas.com
ISSN: 2583-1380 Vol. 6 | Issue No. 3 | March 2026 Impact Factor: 6.53

1.2. Research Motivation

The academic literature on DevOps automation has focused primarily on rule-based and statistical approaches to specific sub-problems: automated testing, static analysis, anomaly detection, and alert correlation. The emergence of capable instruction-following models creates an opportunity to examine a different class of automation: one that operates on the semantic content of engineering artifacts — build configurations, infrastructure definitions, deployment manifests, error messages, runbooks — rather than on their structural patterns alone. This semantic capability enables automation of tasks that were previously too contextually rich to address with pattern-matching or threshold-based approaches.

What has been less studied is the engineering architecture required to deploy generative capabilities in DevOps contexts safely and sustainably. Generating plausible-looking but incorrect infrastructure configuration is not a theoretical risk; it is a documented failure mode of general-purpose language models applied to domain-specific technical tasks without appropriate grounding, fine-tuning, and validation infrastructure. The GenDOP framework is designed specifically to address this: not just integrating generative capabilities but providing the guardrails, validation layers, and human oversight mechanisms that make those capabilities trustworthy enough to act on in production contexts.

1.3. Key Contributions

- The GenDOP framework: a four-capability autonomous DevOps platform architecture integrating generative language model capabilities with established CI/CD and IaC toolchains, with explicit validation and governance components for each capability.
- A fourteen-month empirical study across twelve engineering teams, measuring platform configuration efficiency, failure resolution time, misconfiguration rate, and onboarding velocity against pre-integration baselines.
- A characterisation of the trust, governance, and auditability requirements specific to generative capabilities in DevOps contexts, with evidence-based guidance on which use cases support autonomous execution and which require human review.
- A comparative analysis of GenDOP against four alternative platform assistance approaches across six evaluation dimensions, providing the first multi-approach empirical comparison of generative capabilities in DevOps platform contexts.

2. Literature Review

2.1. DevOps Automation: Prior Work and Scope

The academic treatment of DevOps automation spans a wide range of sub-disciplines. Forsgren et al.'s longitudinal DORA research programme (2018, 2024) established the empirical correlation between automation maturity and software delivery performance, demonstrating that elite-performing organizations invest in automation across the delivery lifecycle rather than optimizing individual stages. Their work provides the performance benchmarking framework that GenDOP's evaluation design references, and their characterization of deployment frequency, lead time for changes, change failure rate, and mean time to restore as the four key DORA metrics shapes the outcome measures used in this study.

The CI/CD pipeline optimization literature has examined test selection and prioritization — reducing pipeline duration by running the subset of tests most likely to detect the current change — through static analysis (Legunsen et al., 2016), historical test-failure correlation (Grano et al., 2019), and code-change impact analysis. These approaches address pipeline efficiency at a specific layer; they do not address the broader pipeline configuration and maintenance challenges that GenDOP targets.

Infrastructure-as-Code quality research has received increasing attention as IaC has become the dominant infrastructure management paradigm. Sharma et al. (2021) analyzed security and reliability defect patterns in open-source Terraform and Ansible code repositories, finding that misconfiguration was the dominant defect category and that the majority of misconfigurations were in access control, network exposure, and encryption settings — precisely the categories where generative assistance with established organizational patterns could provide the highest safety value. Jiang and Adams (2015) studied the evolution and maintenance of

The Review of Contemporary Scientific and Academic Studies
An International Multidisciplinary Online Journal
www.thercsas.com
ISSN: 2583-1380 Vol. 6 | Issue No. 3 | March 2026 Impact Factor: 6.53

build scripts across long-lived software projects, documenting the accumulation of configuration technical debt that generative refactoring assistance could address.

2.2. Large Language Models in Software Engineering Contexts

The application of large language models to software engineering tasks has produced a substantial and rapidly growing body of research since the introduction of Codex (Chen et al., 2021) and the subsequent development of code-specialized models. The HumanEval benchmark (Chen et al., 2021) established the foundational evaluation methodology for code generation; subsequent work has extended this to domain-specific tasks including infrastructure configuration (IaC generation from natural language descriptions), test generation, and code review.

Hou et al. (2024) conducted a comprehensive survey of large language models for software engineering tasks, examining thirty-four distinct task categories across code generation, program repair, testing, and documentation. Their analysis is directly relevant to the GenDOP design: they found that code generation accuracy improves substantially with domain-specific fine-tuning, that repair and explanation tasks benefit more from retrieval augmentation than raw model capability, and that the gap between benchmark performance and real-world deployment performance is consistently wider for complex, multi-step tasks than for single-step generation tasks. These findings motivated GenDOP's retrieval-augmented architecture for the failure explanation capability and its emphasis on fine-tuning over zero-shot application of general-purpose models.

Fan et al. (2023) reviewed large language models for automated program repair, examining both neural and hybrid approaches. Their finding — that program repair benefits from incorporating execution feedback into the repair loop rather than generating repairs in a single forward pass — informs GenDOP's iterative repair suggestion component, which validates generated pipeline repair suggestions against a static analysis and schema validation layer before surfacing them to engineers.

2.3. Platform Engineering and Internal Developer Platforms

The platform engineering discipline — building internal developer platforms (IDPs) that provide engineering teams with self-service access to infrastructure and delivery capabilities — has formalized considerably since the founding of the Platform Engineering community of practice and the publication of Humanitec's State of Platform Engineering surveys. The IDP research literature (Fagerholm et al., 2014; Forsgren et al., 2018) has documented the cognitive load reduction and delivery velocity improvements associated with well-designed developer platforms.

Cognitive load management is specifically relevant to the GenDOP design. Skelton and Pais (2019) classified the cognitive load imposed on development teams by platform interactions into intrinsic load (the inherent complexity of the task), extraneous load (complexity added by the interface or tooling), and germane load (cognitive work that builds useful expertise). GenDOP's design explicitly targets extraneous cognitive load: reducing the configuration syntax learning burden, the error message interpretation effort, and the documentation navigation time that engineers expend on platform interactions without building domain expertise that transfers to other tasks.

2.4. Research Gaps

Three gaps in the existing literature directly motivate this research. First, no published empirical study has evaluated a generative capability-integrated DevOps platform against a defined set of operational metrics across multiple engineering teams and a multi-month observation period; existing work evaluates model capabilities on benchmarks rather than operational outcomes in production contexts. Second, the governance and trust architecture required for generative capabilities in DevOps contexts — how to structure human oversight, validation, and auditability for generated configurations and repair suggestions — has not been systematically studied. Third, the comparative performance of generative approaches against non-generative automation alternatives on the same operational tasks has not been empirically characterized, making it impossible to evaluate generative approaches against the alternative investments available to platform teams.

3. Methodology and Proposed Framework

3.1. GenDOP Architecture Overview

The Review of Contemporary Scientific and Academic Studies
An International Multidisciplinary Online Journal
www.thercsas.com
ISSN: 2583-1380 Vol. 6 | Issue No. 3 | March 2026 Impact Factor: 6.53

The GenDOP framework is organized into four functional capability layers, each with a defined input specification, a generation or inference component, a validation layer, and a delivery interface. The four-layer structure reflects a deliberate design choice: each capability addresses a distinct mode of platform interaction with different accuracy requirements, risk profiles, and appropriate levels of automation autonomy. Rather than building a single generative layer that attempts to handle all platform interactions uniformly, GenDOP treats each capability as requiring its own appropriate governance and validation approach.

The foundational model component shared across all four capabilities is a fine-tuned instruction-following model specialized on a domain corpus that includes: CI/CD pipeline configurations from the study organizations' GitOps repositories (anonymized and normalized), IaC modules from public and internal repositories, DevOps and SRE runbook collections, cloud provider documentation for the three providers represented in the study population, and annotated failure logs from the study organizations' incident records. Fine-tuning on this domain corpus, rather than relying on zero-shot application of a general-purpose model, significantly reduces the rate of plausible-but-incorrect outputs that would undermine engineer trust in generated artifacts.

Capability 1 — Natural Language Pipeline Synthesis

The pipeline synthesis capability accepts natural language descriptions of delivery workflow intent — 'build and test a Python service, run security scanning on the Docker image, deploy to staging on branch push and to production on tag release, with manual approval required for production deployments' — and produces a syntactically correct, schema-valid CI/CD pipeline configuration in the target platform's format (GitHub Actions YAML, GitLab CI configuration, or Tekton Pipeline YAML).

The synthesis process follows a three-step workflow. Intent decomposition parses the natural language input into a structured representation of pipeline stages, triggers, dependencies, and constraints, using a retrieval-augmented step that pulls relevant example configurations from the organization's existing pipeline library to ground the decomposition in established patterns. Configuration generation produces a draft pipeline configuration from the structured intent representation, drawing on the fine-tuned model's knowledge of the target platform's syntax and on the retrieved examples. Schema and security validation applies static analysis and schema validation to the generated configuration, checking for structural errors, missing required fields, over permissive secret access patterns, and violation of the organization's pipeline security policies encoded in the Policy-as-Code layer. Engineers review the validated configuration in a diff view against the closest existing pipeline in the organization's library before accepting or modifying it.

A critical design decision is that pipeline synthesis always produces an artifact for human review rather than committing directly to the version-controlled pipeline repository. This design reflects the trust calibration principle: the accuracy of the generation is high enough to substantially reduce the effort of pipeline creation (engineering a first draft from scratch is slower than reviewing and correcting a high-quality draft), but not high enough to warrant autonomous commitment without human review.

Capability 2 — Failure Explanation and Repair Suggestion

The failure explanation capability addresses the most acute day-to-day pain point in CI/CD platform operations: understanding why a pipeline run failed and determining what change is needed to fix it. The input is a pipeline failure event: the failing pipeline configuration, the execution log with error output, the commit diff that triggered the run, and the historical context of prior failures on the same pipeline within a configurable lookback window.

The explanation generation process uses a retrieval-augmented approach: before generating an explanation, the system queries a vector index of prior failure events and their confirmed resolutions to find the closest analogous failures in the organization's incident history. The retrieved analogies provide factual grounding for the explanation, reducing the rate of hallucinated explanations that sound plausible but prescribe incorrect remediation. The generated explanation includes: a plain-language description of the failure root cause, a ranked list of probable contributing factors, a reference to any analogous prior failures with their resolutions, and one or more specific repair suggestions with the expected change in the failing pipeline file.

Repair suggestions are validated before presentation through a two-step check: static syntax validation ensures the suggested change is syntactically valid in the target platform's configuration language, and a

The Review of Contemporary Scientific and Academic Studies
An International Multidisciplinary Online Journal
www.thercsas.com
ISSN: 2583-1380 Vol. 6 | Issue No. 3 | March 2026 Impact Factor: 6.53

simulated dry-run (where the pipeline platform supports it) validates that the suggested configuration passes the platform's pre-execution validation. Suggestions that fail either check are replaced with a lower-confidence suggestion or escalated to human expert review.

Capability 3 — Context-Aware IaC Generation

Infrastructure-as-Code generation addresses a different workflow than pipeline synthesis: rather than describing delivery workflow intent, engineers express infrastructure requirements — 'a PostgreSQL RDS instance sized for a read-heavy 1000 concurrent connection workload, with automated backups, point-in-time recovery, and encryption at rest, in the production VPC, tagged for the payments team' — and receive a complete, security-reviewed Terraform module or Helm chart consistent with the organization's IaC standards.

The IaC generation workflow incorporates the organization's established patterns through a two-phase retrieval step: first, retrieving the closest semantically similar IaC modules from the organization's module registry; second, retrieving the organization's security and operational policies encoded as Policy-as-Code rules that the generated module must satisfy. The generated module is evaluated against the policy rule set using Conftest and OPA before being presented to the engineer, with any policy violations highlighted alongside the generated code.

A cost estimation step (using Infracost or equivalent) generates a projected monthly cost for the generated infrastructure configuration before the engineer commits to it, addressing the FinOps dimension of IaC generation. Surfacing the cost estimate alongside the generated configuration — with a comparison against the organization's typical instance sizing for similar workloads — enables cost-conscious decisions without requiring the engineer to independently research pricing.

Capability 4 — Deployment Risk Scoring

Deployment risk scoring provides a pre-deployment assessment of the risk profile of a proposed change, synthesizing signals from the change diff, the deployment history of the affected services, the current operational health of the deployment target environment, and the time and date context of the proposed deployment. The output is a structured risk score with contributing factor explanations, not a binary pass/fail decision.

The scoring model incorporates four risk dimensions: change scope (lines changed, number of files, ratio of new code to modified code, presence of dependency updates or configuration changes), historical deployment context (deployment failure rate for the affected service, time since last deployment, proximity to a prior incident), operational context (current anomaly scores in the target environment from the AIOps layer, active deployment pipelines for dependent services, recent high-severity incidents in the deployment window), and temporal context (day of week, time relative to business peak, proximity to scheduled maintenance windows).

3.2. Study Design

Twelve engineering teams participated in the study, observed from January 2025 through February 2026. Teams were recruited through platform engineering community networks, DevOps conference contacts, and referrals from CI/CD platform vendors. Eligibility criteria required active production CI/CD pipelines processing at minimum ten daily deployments, IaC-managed cloud infrastructure on at least one major provider, and willingness to provide anonymized pipeline telemetry and incident records.

Engineering headcounts ranged from 8 to 67. Industry sectors included financial technology, e-commerce, healthcare SaaS, gaming, and enterprise software. CI/CD platforms represented included GitHub Actions (seven teams), GitLab CI (three teams), and Tekton (two teams). Cloud providers included AWS (eight teams), GCP (three teams), and a multi-cloud combination (one team).

GenDOP was deployed progressively: pipeline synthesis and failure explanation capabilities in months one and two, IaC generation in months three and four, and deployment risk scoring in month five. This staged rollout allowed teams to build confidence in earlier capabilities before adopting later ones.

4. Implementation and Experimental Setup

4.1. Technology Stack and Infrastructure

The Review of Contemporary Scientific and Academic Studies
An International Multidisciplinary Online Journal
www.thercsas.com
ISSN: 2583-1380 Vol. 6 | Issue No. 3 | March 2026 Impact Factor: 6.53

The GenDOP implementation used in the study deployed a fine-tuned instruction-following model as a private inference endpoint, accessible only to the organization's internal toolchain. The fine-tuning corpus described in Section 3.1 was processed through a data preparation pipeline that included personally identifying information removal, schema normalization for the IaC modules, and annotation of failure log examples with confirmed root cause labels from historical postmortem records. Fine-tuning used a parameter-efficient approach (LoRA adaptation) to reduce the compute requirements for domain adaptation while preserving the foundation model's general reasoning capabilities.

The retrieval-augmented components used separate vector indices for each capability: pipeline library embeddings for pipeline synthesis, failure event embeddings for failure explanation, IaC module embeddings for IaC generation, and deployment history embeddings for risk scoring. Indices were updated nightly with new pipeline configurations committed to the GitOps repository, new failure events processed from the CI/CD platform, and new IaC modules added to the module registry.

4.2. Deployment Pipeline Integration

GenDOP capabilities were integrated into the development workflow at specific interaction points rather than as a separate tool that engineers had to navigate to independently. Pipeline synthesis was accessible from the repository creation workflow. Failure explanation surfaced in-line within the CI/CD platform's pipeline run view. IaC generation was accessible through a Terraform module scaffold command in the organization's CLI tooling. Deployment risk scoring appeared in the pull request comment thread.

This integration philosophy — making generative capabilities available at the natural workflow touchpoints where engineers already operate, rather than requiring them to navigate to a separate AI interface — was a deliberate design choice informed by prior research on developer experience and tooling adoption.

4.3. Traditional Approaches vs. GenDOP (Table 1)

Table 1 provides a structured comparison of traditional DevOps platform assistance approaches against the GenDOP framework across six operational dimensions.

Dimension	Manual Expert-Only	Rule-Based Automation	Template Libraries	IDE Copilot (Code Only)	GenDOP (Proposed)
Pipeline configuration	Hours; expert required	Rigid; brittle to variation	Moderate effort	No CI/CD support	Minutes; NL intent to config
Failure diagnosis	Expert interpretation; slow	Pattern match; limited	Not applicable	Limited to code errors	Semantic explanation + analogy
IaC generation	Expert-authored; inconsistent	Parameterised templates only	Scaffold + manual fill	Code gen without org patterns	Policy-grounded + cost-estimated
Deployment risk	Manual judgment; subjective	Simple threshold rules	Not applicable	Not applicable	Multi-signal scored + explained
Onboarding support	Docs + mentoring	Not applicable	Limited	Code completion only	Context-aware guidance
Human oversight	Fully human	None (fully automated)	Review before use	Engineer reviews all	Validated draft + human review

Table 1. Comparison of traditional DevOps platform assistance approaches against GenDOP across six operational dimensions.

5. Results and Analysis

5.1. Pipeline Configuration Efficiency

Pipeline configuration time — measured from when an engineer initiated a new pipeline configuration task to when a validated, committed pipeline configuration was in the repository — decreased from a pre-GenDOP median of 147 minutes to 62 minutes post-integration, a 58 percent reduction. The improvement was largest for engineers with less than two years of platform experience: their median configuration time decreased from 214 minutes to 71 minutes, a 67 percent reduction.

An important nuance in the pipeline configuration results is the quality dimension. The study tracked the number of pipeline validation failures and pipeline failures within the first three runs for GenDOP-generated

The Review of Contemporary Scientific and Academic Studies
An International Multidisciplinary Online Journal
www.thercsas.com
ISSN: 2583-1380 Vol. 6 | Issue No. 3 | March 2026 Impact Factor: 6.53

configurations versus manually authored ones. GenDOP-generated configurations showed a 34 percent reduction in first-run failures and a 41 percent reduction in failures within the first three runs, indicating that the quality improvement from generative assistance with validation guardrails more than compensates for any accuracy limitations of the generation step.

5.2. Failure Resolution Time

Mean time to pipeline failure resolution decreased from a pre-GenDOP median of 38 minutes to 11 minutes, a 71 percent reduction. This is the largest relative improvement across all measured metrics and reflects two compounding effects: the time saving from engineers receiving an accurate explanation rather than having to diagnose the failure themselves, and the time saving from engineers receiving a specific, validated repair suggestion rather than having to construct the fix from first principles.

Disaggregating by failure category reveals important variation. For the ten most common failure categories, explanation accuracy was 93 percent. For rare or novel failure categories with fewer than three analogous prior failures in the retrieval index, explanation accuracy dropped to 61 percent.

5.3. IaC Quality and Misconfiguration Rate

Infrastructure misconfiguration incidents decreased from a pre-GenDOP quarterly average of 4.8 per team to 2.7, a 44 percent reduction. The reduction was concentrated in the security misconfiguration category (access control and network exposure errors), which decreased by 57 percent.

Cost optimization outcomes from the IaC generation capability were measured separately. Across the twelve teams, the cost estimation step surfaced a median of 2.3 opportunities per team per month where the default generated configuration sized resources more generously than the workload description warranted, with an average projected monthly saving of $1,400 per opportunity.

5.4. Deployment Risk Scoring Impact

The deployment risk scoring capability's impact was measured through change failure rate. Pre-GenDOP baseline change failure rate averaged 4.1 percent across the twelve teams. Post-integration, teams that had a deployment risk score available before more than 80 percent of their production deployments averaged a change failure rate of 2.6 percent — a 37 percent reduction.

Qualitative interview data revealed that the risk score rarely caused engineers to cancel a deployment entirely; most high-risk deployments proceeded with additional caution steps. Engineers described the risk score primarily as a prompt to think explicitly about deployment safety rather than as an automated gate.

5.5. Performance Metrics Summary

Table 2 presents the complete performance metrics comparison across all five measured outcome dimensions.

Metric	Pre-GenDOP Baseline	Post-Integration (Month 14)	Improvement	95% CI (Post)
Pipeline config time — median (min)	147	62	−58%	[56, 68]
First-run pipeline failure rate (%)	29%	19%	−34%	[17%, 21%]
Mean time to failure resolution (min)	38	11	−71%	[9.3, 12.7]
IaC misconfiguration incidents / team / quarter	4.8	2.7	−44%	[2.3, 3.1]
Change failure rate (%)	4.1%	2.6%	−37%	[2.2%, 3.0%]
Engineer platform onboarding time (days)	14.3	5.2	−63%	[4.6, 5.8]

Table 2. Performance metrics comparison: pre-GenDOP baseline versus post-integration measurements at fourteen months.

The onboarding time metric — measuring the time for engineers new to the platform to reach self-sufficient productivity — decreased from 14.3 days to 5.2 days, a 63 percent reduction. This improvement was attributed primarily to the ability to describe intent in natural language and receive a generated starting point, and the failure explanation capability's role in accelerating the feedback loop.

The Review of Contemporary Scientific and Academic Studies
An International Multidisciplinary Online Journal
www.thercsas.com
ISSN: 2583-1380 Vol. 6 | Issue No. 3 | March 2026 Impact Factor: 6.53

6. Discussion

6.1. Implications for DevOps and SRE Teams

The most significant operational implication of the GenDOP results is the reduction in platform specialist dependency for common platform interactions. When engineers can configure a pipeline through natural language description, diagnose a failure through a generated explanation, and generate an IaC module with policy compliance integrated into the generation process, the specialist knowledge required for routine platform operations is substantially decompressed. Platform engineers' time shifts from providing first-line assistance on common configuration tasks toward higher-value work.

The onboarding velocity improvement has a specific organizational implication for growing engineering organizations. A team that can onboard new engineers to platform self-sufficiency in 5 days rather than 14 days has a materially different growth dynamic: new engineers become contributors faster, and the platform team's capacity to support onboarding is less constraining on overall organizational hiring velocity.

6.2. Trust, Governance, and Auditability

The most important governance challenge specific to generative capabilities in DevOps contexts is the confidence calibration problem: engineers need to develop an accurate intuition for when generated outputs can be trusted with minimal review and when they require careful validation. GenDOP addresses this through explicit confidence signalling: each generated output includes an estimated confidence level derived from the agreement between the generated output and the retrieved analogical examples, the validation outcomes of the static analysis layer, and the model's own internal uncertainty.

The audit trail requirement cannot be understated for enterprise and regulated contexts. Every GenDOP-generated artifact that is accepted and committed carries metadata identifying it as GenDOP-generated, the generation confidence, the relevant retrieved analogies, and the validation outcomes. This enables post-incident analysis, provides evidence for regulatory inquiries, and enables continuous improvement of generation quality.

6.3. Limitations

Several limitations qualify the generalizability of the study findings. The study population is self-selected from organizations willing to adopt new platform technology — likely skewed toward higher platform maturity than the general DevOps population. The fine-tuned model's performance is highly dependent on the quality and coverage of the training corpus. The deployment risk scoring capability's 37 percent change failure rate reduction may be partially attributable to the deliberate attention its presence prompts (a Hawthorne effect).

The most operationally significant limitation is the hallucination problem. The validation layer catches syntactic and schema errors reliably, but semantic errors require human review. The study observed 23 cases across the fourteen months in which GenDOP-generated IaC configurations passed schema validation but contained semantic errors that were caught during human review. None reached production, but the pattern underscores that human review of generated configurations is not optional.

7. Conclusion and Future Work

7.1. Key Findings

This paper has proposed and evaluated the GenDOP framework, a four-capability autonomous DevOps platform architecture integrating generative language model capabilities with established CI/CD and infrastructure management tool chains. The empirical results across twelve engineering teams over fourteen months demonstrate consistent and substantial improvements: 58 percent reduction in pipeline configuration time, 71 percent reduction in mean time to pipeline failure resolution, 44 percent reduction in IaC misconfiguration incidents, 37 percent reduction in change failure rate, and 63 percent reduction in platform onboarding time.

The findings make a strong case that generative capabilities, when integrated with appropriate domain fine-tuning, retrieval augmentation from organizational knowledge bases, structured validation layers, and human review workflows, produce operationally significant and sustained improvements in DevOps platform

efficiency and quality. The trust and governance findings are as important as the performance findings — the validation layer, confidence signaling, and audit trail components are the mechanisms that make the performance improvements sustainable.

7.2. Future Research Directions

Four research directions emerge from the current work. Multi-team model personalization would address the operational challenge of maintaining generation quality for teams whose infrastructure and pipeline patterns diverge significantly from the organization-wide baseline. The interaction between generative DevOps capabilities and DevSecOps shift-left practices warrants dedicated study. The FinOps optimizations dimension requires a dedicated study examining whether engineers act on cost optimizations recommendations at organizational scale. Finally, the longitudinal stability of generative platform capabilities requires study over time horizons longer than fourteen months.

References

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., et al. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374. https://arxiv.org/abs/2107.03374

Dakhel, A. M., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., & Jiang, Z. M. J. (2023). GitHub Copilot AI pair programmer: Asset or liability? Journal of Systems and Software, 203, 111734. https://doi.org/10.1016/j.jss.2023.111734

Fagerholm, F., Johnson, P., Guinea, A. S., Borenstein, J., & Münch, J. (2014). The role of mentoring and project characteristics for onboarding in open source software projects. ESEM 2014, 1–10. https://doi.org/10.1145/2652524.2652540

Fan, A., Gokkaya, B., Harman, M., Lyuye, T., Manav, S., Mirhosseini, M., Soares, L., & Topal, I. (2023). Large language models for software engineering: Survey and open problems. ICSE 2023, 31–53. https://doi.org/10.1109/ICSE-FoSE59343.2023.00008

Forsgren, N., Humble, J., & Kim, G. (2018). Accelerate: The science of lean software and DevOps. IT Revolution Press.

Forsgren, N., Smith, D., Humble, J., & Frazelle, J. (2024). DORA state of DevOps report 2024. Google Cloud and DORA Research Programme.

Grano, G., Sarro, F., Gall, H. C., & Panichella, S. (2019). Software test prioritization based on Semantics. MSR 2019, 12–16. https://doi.org/10.1109/MSR.2019.00010

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., & Wang, H. (2024). Large language models for software engineering: A systematic literature review. ACM TOSEM, 33(8), 1–79. https://doi.org/10.1145/3695988

Humanitec. (2024). State of platform engineering: Report 2024. https://humanitec.com/whitepapers/state-of-platform-engineering-report-2024

Jiang, Z. M. (Jack), & Adams, B. (2015). Insights into the contributors of build configuration evolution. ICSE 2015, 2, 701–710. https://doi.org/10.1109/ICSE.2015.232

Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps handbook. IT Revolution Press.

Legunsen, O., Hariri, F., Shi, A., Lu, Y., Zhang, L., & Marinov, D. (2016). An extensive study of static regression test selection. FSE 2016, 583–594. https://doi.org/10.1145/2950290.2950361

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. NeurIPS 2020, 9459–9474.

Majdinasab, V., Nikanjam, A., Dakhel, A. M., Khomh, F., & Desmarais, M. C. (2024). Assessing the capability of code generation tools for Infrastructure-as-Code. arXiv:2402.04278.

Overeem, M., Spijkerman, M., & Visser, J. (2021). An empirical characterization of bad practices in continuous integration. Empirical Software Engineering, 26(6), 1–44. https://doi.org/10.1007/s10664-021-10024-4

Sharma, T., Kechagia, M., Georgiou, S., Tiwari, R., & Sarro, F. (2021). A survey on code quality and security aspects of Infrastructure-as-Code. IEEE TSE, 49(3), 1358–1373. https://doi.org/10.1109/TSE.2022.3160489

Skelton, M., & Pais, M. (2019). Team topologies: Organizing business and technology teams for fast flow. IT Revolution Press.

ISSN: 2583-1380 Vol. 6 | Issue No. 3 | March 2026 Impact Factor: 6.53

Svyatkovskiy, A., Zhao, Y., Fu, S., & Sundaresan, N. (2020). Intellicode compose: Code generation using transformer. ESEC/FSE 2020, 1433–1443. https://doi.org/10.1145/3368089.3417058

Weyssow, M., Nashid, N., Palomba, F., & Tsigkanos, C. (2024). LLMs for DevOps automation: Towards intelligent CI/CD pipeline management. AISE 2024, 88–99. https://doi.org/10.1145/3643796.3648449

Zhang, C., Liu, Z., Zhang, H., & Hassan, A. E. (2024). Towards automated DevOps configuration: A large language model approach for Infrastructure-as-Code synthesis. ACM TOSEM, 33(7), 1–41. https://doi.org/10.1145/3691617

Conflicts of Interest: The author declares "No conflict of interest".