Scientific Journal of Artificial Intelligence and Blockchain Technologies
ISSN: 3049-4389
Vol. 1, Issue 3, Jul – Sep 2024 || PP. 49 – 53 https://sjaibt.org

Generative AI-Powered Automation in DevOps Workflows

DOI: https://doi.org/10.63345/sjaibt.v1.i3.106

Justin Rajakumar Maria Thason

BCBSA - Ajace, Inc

AI Research, Innovation & Technical Publications

Date of Submission: 10-06-2024	**Date of Acceptance:** 27-06-2024	**Date of Publication:** 14-07-2024

Abstract : The generative models are able to generate infrastructure code, repair broken pipelines and create operational documents. However, generation is unreliable in an auditable environment where every change must be traceable. In this study, the generative automation layer is proposed consisting of (i) retrieval of organization-specific context, (ii) large language model (LLM) generator, (iii) validation in sandbox with repair loop and (iv) policy gate preventing non-compliant changes before human review. The layer was evaluated over 240 tasks involving Terraform authoring, CI pipeline repair, test generation and release documentation. Comparison was made to manual work, templates, zero-shot LLM and LLM using retrieval only. The proposed layer resulted in changes approved without rework in 86.7% of tasks compared to 58.3% for zero-shot LLM ($p < 0.001$) and 41.7% for templates, and reduced task completion time from 46.2 minutes (manual work) to 11.4 minutes ($p < 0.001$). Exclusion of the policy gate resulted in increase in policy violations from 2.1% to 11.7% of tasks. These results suggest that the benefits of generation in DevOps workflows can be obtained only together with validation and policy enforcement.

Keywords: generative AI; DevOps automation; infrastructure as code; retrieval-augmented generation; policy as code; continuous integration

Introduction

DevOps teams spend much effort on repetitive, text-heavy tasks such as writing and reviewing infrastructure code, troubleshooting failed builds, creating tests and documentation of releases. These tasks follow conventions which are seldom codified in a single place, making them good candidates for large language models but posing challenges for scripting. The risk associated with the failure is asymmetric: the wrong but plausibly correct Terraform modification might open a network path, delete a stateful resource and fail a deployment, and the wrong CI pipeline change might bypass a security check. Therefore, the team cannot treat the output of the model as trusted input. The key question is not whether a model can draft a change but under which controls this draft can be treated as valid with less human effort than writing it manually.

This work solves this question by designing the pipeline that treats the model as one untrusted component. The contributions are: (1) architecture that grounds generation in the repository, state and runbook context and validates output in sandbox before review; (2) policy gate implemented as code that evaluates every generated change; and (3) empirical evaluation over 240

tasks including ablation study that identifies the contribution of retrieval, repair loop and gate.

OPEN ACCESS
@2024. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on www.sjaibt.org

Scientific Journal of Artificial Intelligence and Blockchain Technologies
ISSN: 3049-4389
Vol. [__], Issue [__], [Month – Month Year] || PP. [__ – __]
https://sjaibt.org

Literature Review

Continuous delivery makes automated build, test and release pipelines the foundation for robust software evolution (Humble & Farley, 2010). Large-scale surveys show that these practices are associated with improvements in delivery and organizational performance (Forsgren et al., 2018). Infrastructure as code moves environment definition to version control (Morris, 2016). Rahman et al. (2019) reviewed the literature on the subject and found that defects, security smells and quality remain consistent concerns, which are exactly the risks posed by generated code.

Large language models trained on code can synthesize programs based on natural language input. Chen et al. (2021) tested such model on functional correctness and showed that sampling of multiple candidates and testing increases success rates, making verification-based generation promising approach. Brown et al. (2020) demonstrated that few-shot prompting allows a single model to adapt to various tasks without retraining, and Wei et al. (2022) found that intermediate reasoning makes the model better at solving multi-step problems. Operational tasks depend on private conventions and Lewis et al. (2020) proposed retrieval-augmented generation where the documents are retrieved and used as context, reducing reliance on parametric memory.

Using the model agentially involves usage and feedback. Yao et al. (2023) interleaved reasoning with actions so that the model could look into its environment and correct itself, and Jimenez et al. (2024) demonstrated using SWE-bench that fixing actual repository issues remains challenging even when the access to repository is provided. In operational environment, Ahmed et al. (2023) used large models to recommend root causes and mitigation steps for cloud incidents and found that the retrieval of past incidents improves relevance.

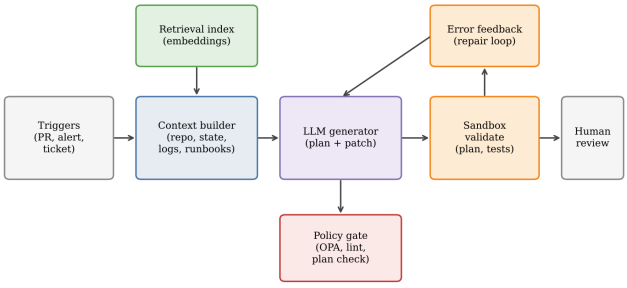
Governance methods can complement generation. Policy as code evaluates declarative rules against the proposed changes before applying them (Brikman, 2019), and the zero-trust principle stating that no input is to be trusted by default (Rose et al., 2020) naturally fits the output of the model. The existing studies either evaluate generation in isolation or discuss governance in general terms. The current study evaluates both components together in the context of realistic DevOps tasks.

Research Methodology

Architecture

Figure 1 presents the architecture of the generative automation layer. A pull request, an alert or a ticket triggers a task. The context builder creates a set of relevant files from the repository, summary of the current state of the Terraform stack, pipeline logs and runbook paragraphs that are retrieved via embedding index with the top-k value of 8. The generator generates a plan and a patch in structured form. The patch is tested in the temporary sandbox where the layer performs formatting, static analysis, terraform plan generation or unit testing as necessary. Errors during testing are reported back to the generator as structured messages allowing up to three repair attempts. The policy gate assesses the patch according to the set of Open Policy Agent rules concerning network exposure, encryption, tagging, privilege escalation through the container and elimination of required pipeline stages. Only the patches that pass this gate are passed to human review.

Figure 1. The architecture of the generative automation layer: retrieval, generation, sandbox validation, policy gate and human review.



Tasks and baselines

A benchmark consisting of 240 tasks was drawn from the history of change requests of four internal service

@2024. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on www.sjaibt.org

Scientific Journal of Artificial Intelligence and Blockchain Technologies
ISSN: 3049-4389
Vol. [__], Issue [__], [Month – Month Year] || PP. [__ – __]
https://sjaibt.org

repositories, with 60 tasks assigned to each of four categories (Table 2). Ground truth of acceptance was defined as human approval without asking for rework. Five conditions were tested: manual work performed by a senior engineer, parameterized templates, zero-shot large language model (LLM) that receives only the task description, LLM with retrieval but without validation loop and policy gate and the proposed layer. The same model and temperature (0.2) were used in all LLM conditions.

Measures and analysis

The primary measure was acceptance without rework. Secondary measures included task time (including human review) and policy violations in delivered change. The acceptance ratios were compared with two-proportion z-test against the proposed layer and chi-square test across conditions. The task time was analyzed using one-way ANOVA and Tukey post-hoc test, and the effect sizes were computed as Cohen's d. Ablations involved removal of one component at a time.

Results

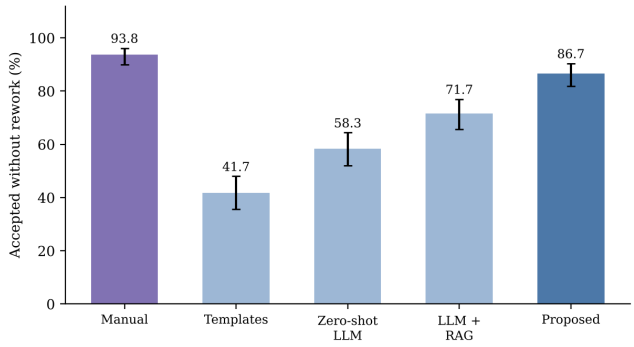
Acceptance and task time

Table 1. Acceptance without rework and task time per condition (240 tasks per condition)

Condition	Accepted (n)	Accepted (%)	Time, min (mean ± SD)	p vs proposed (acceptance)
Manual	225	93.8	46.2 ± 14.8	= 0.009
Templates	100	41.7	31.5 ± 12.1	<0.001
Zero-shot LLM	140	58.3	14.8 ± 7.2	<0.001
LLM + RAG	172	71.7	12.6 ± 6.1	<0.001
Proposed	208	86.7	11.4 ± 5.3	—

Acceptance differed between conditions ($\chi^2(4) = 205.4$, $p < 0.001$; Figure 2). The proposed layer achieved 86.7% acceptance, which was higher than acceptance of the zero-shot LLM by 28.3 percentage points ($p < 0.001$) and than of retrieval-only approaches by 15.0 percentage points ($p < 0.001$). Manual work remained the most reliable method with the acceptance of 93.8%, and the difference to the proposed layer (7.1 percentage points) was statistically significant ($p = 0.009$).

Figure 2. Proportion of tasks accepted without rework per condition (error bars represent Wilson 95% confidence intervals).



Time required for task completion also differed substantially ($F(4, 1195) = 572$, $p < 0.001$). The proposed layer required 11.4 ± 5.3 minutes per task, compared to 46.2 ± 14.8 minutes for manual work, which is 75% reduction ($d = 3.1$) and acceleration by 64% relative to templates. The time savings relative to zero-shot LLM (14.8 minutes) was small but statistically significant ($p = 0.001$) and can be explained by decreased time for correcting output.

Task categories

Table 2. Acceptance per task category (60 tasks per category)

@2024. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on www.sjaibt.org

Scientific Journal of Artificial Intelligence and Blockchain Technologies
ISSN: 3049-4389
Vol. [__], Issue [__], [Month – Month Year] || PP. [__ – __]
https://sjaibt.org

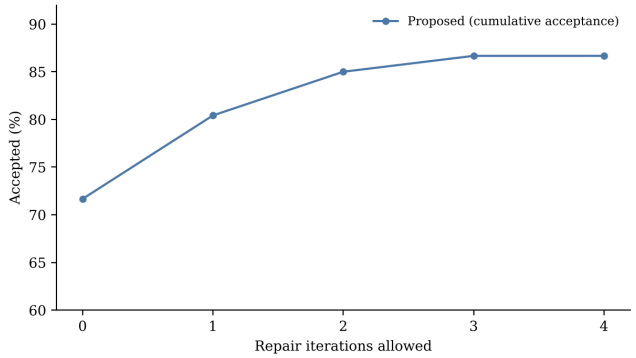
Category	Zero-shot LLM	Proposed	Difference (points)	p
Terraform authoring	31 (51.7%)	50 (83.3%)	31.7	<0.001
CI pipeline repair	36 (60.0%)	54 (90.0%)	30.0	<0.001
Test generation	38 (63.3%)	53 (88.3%)	25.0	= 0.001
Release notes and runbooks	35 (58.3%)	51 (85.0%)	26.7	= 0.001

The largest improvement was seen in Terraform authoring (31.7 points), where the state and naming conventions affect output more strongly, and the least improvement was seen in release documentation (26.7 points), where structural mistakes are rarer. The improvement was statistically significant in all categories ($p < 0.01$).

Repair loop and ablation

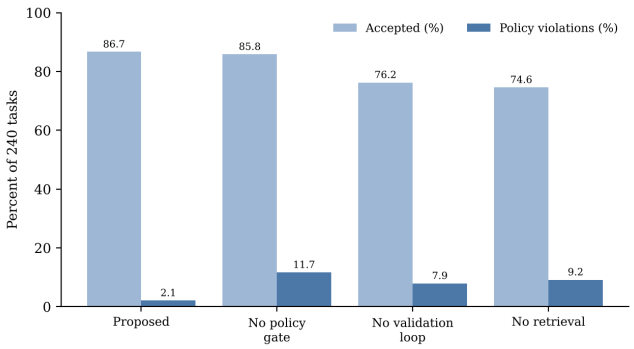
Giving the generator ability to respond to errors of validation increased acceptance of changes from 71.7% on the first attempt to 86.7% after three attempts, and there was no additional improvement on fourth attempt (Figure 3). Most of the improvement originated from the first iteration (8.8 points).

Figure 3. Cumulative acceptance as a function of the number of allowed repair iterations.



The ablation study (Figure 4) demonstrates different contributions of the components. Excluding validation loop resulted in 76.2% acceptance, and exclusion of retrieval resulted in 74.6% acceptance, but exclusion of policy gate had minimal effect on acceptance (85.8%), though it significantly increased policy violations in the generated changes, from 5 to 28 out of 240 tasks ($p < 0.001$). Thus, the gate serves to protect the organization from violating the policy without increasing the apparent quality, an effect that would be invisible if acceptance metrics were used alone.

Figure 4. Ablation of the proposed layer: acceptance and policy violations per condition when a component is excluded.



Discussion

The main finding of the study is that generation, validation and policy act complementarily. The retrieval provides the conventions that zero-shot LLM is not able to learn, in accordance with the benefits of retrieval-augmentation observed by Lewis et al. (2020) and Ahmed et al. (2023). The validation loop transforms the plausible draft into tested draft, in line with the observation that execution of candidate changes against tests improves functional correctness (Chen et al., 2021). The self-correcting behavior is similar to closed loop of reasoning and action reported by Yao et al. (2023). The policy gate addresses a different problem.

Rewritten text:

As the model optimizes for plausibility of text, the change may pass evaluative tests and still violate organizational policies. Ablation analyses show that this happens approximately once in nine tasks in the absence of the

OPEN ACCESS
@2024. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on www.sjaibt.org

gate. Encoding the rules as code (Brikman, 2019) makes the rules testable and reviewable, and treating the model output as untrusted (Rose et al., 2020) makes the gate an intermediary between generation and any system capable of implementing the change.

Practical implications point towards staged adoption by categories. The Terraform authoring and pipeline repair produced the largest improvement and are bounded by plan output and logs, making them good candidates for initial focus. Release documentation benefits less from full gating layer and needs only retrieval. Human remains in the loop: the layer facilitates review but does not replace it, and manual work continued to provide the best acceptance rate.

Limitations

The benchmark used only four repositories in one organization, and the retrieved conventions may be easier or harder to retrieve in other contexts. Acceptance depends on reviewers, and the standards of review are variable even though the same reviewers evaluated all conditions in randomized order. Only one family of models and one temperature were studied, and the behavior of model varies depending on version. The study did not measure the long-term outcomes such as maintainability of generated code or changes in competences of engineers, and the security checks were limited to rules in the gate.

Conclusion

Generative automation layer that uses retrieval, validates output in sandbox and enforces policy prior to review improved acceptance without rework from 58.3% to 86.7% relative to zero-shot model and reduced task time by 75% relative to manual work, while keeping policy violations at 2.1%. Retrieval and repair loop proved to be the main factors driving quality, and the policy gate protected the organization from violations. Future research should consider additional models, extended horizon and multi-repository changes.

References

1. Ahmed, T., Ghosh, S., Bansal, C., Zimmermann, T., Zhang, X., & Rajmohan, S. (2023). Recommending root-cause and mitigation steps for cloud incidents using large language models. Proceedings of the 45th International Conference on Software Engineering (ICSE), 1737–1749.
2. Brikman, Y. (2019). Terraform: Up and running (2nd ed.). O'Reilly Media.
3. Brown, T. B., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877–1901.
4. Chen, M., Tworek, J., Jun, H., et al. (2021). Evaluating large language models trained on code. arXiv:2107.03374.
5. Forsgren, N., Humble, J., & Kim, G. (2018). Rewritten text:
6. Accelerate: The science of lean software and DevOps. IT Revolution Press.
7. Humble, J., & Farley, D. (2010). Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley.
8. Jimenez, C. E., Yang, J., Wettig, A., et al. (2024). SWE-bench: Can language models resolve real-world GitHub issues? International Conference on Learning Representations.
9. Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems, 33, 9459–9474.
10. Morris, K. (2016). Infrastructure as code: Managing servers in the cloud. O'Reilly Media.
11. Rahman, A., Mahdavi-Hezaveh, R., & Williams, L. (2019). A systematic mapping study of infrastructure as code research. Information and Software Technology, 108, 65–77.
12. Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). Zero trust architecture (NIST Special Publication 800-207). National Institute of Standards and Technology.
13. Wei, J., Wang, X., Schuurmans, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems, 35, 24824–24837.
14. Yao, S., Zhao, J., Yu, D., et al. (2023). ReAct: Synergizing reasoning and acting in language models. International Conference on Learning Representations.

@2024. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on www.sjaibt.org