Journal of Technology Informatics and Engineering (JTIE)
Vol. 5 No. 2 August 2026
E-ISSN: 2961-9068; P-ISSN: 2961-8215, Pages 104-118
DOI: https://doi.org/10.51903/jtie.v5i2.534

LLM-Style DevOps Copilot for Cloud-Native Troubleshooting: Retrieval-Augmented Runbook Generation and Command-Safety Evaluation

Boning Zhang[1], Xinzhuo Sun*[2], Ge Liu[3], Binghua Zhou[3]
Email: xinzhuo.sun0808@gmail.com
[1]*Computer Science, Georgetown University, DC, USA*
[2]*Computer Engineering, Cornell Tech, NY, USA*
[3]*Computer Science, USC, CA, USA*
*Corresponding Author

Abstract
Cloud-native incident response requires engineers to connect symptoms, observability signals, infrastructure state, and safe remediation commands under time pressure. Large language models can draft runbooks, but an ungrounded assistant can invent commands, recommend the wrong diagnostic path, or reproduce destructive operational shortcuts. This paper evaluates an LLM-style, retrieval-augmented DevOps Copilot simulation for cloud-native troubleshooting on the canonical Szaid3680/Devops Arrow export. The experiment indexes all 42,819 rows with the public Response, Instruction, and Prompt schema and evaluates a deterministic 400-query subset with TF-IDF, BM25, a compact dense-semantic baseline, RAG-style answer construction, reranking, command-safety checking, and the combined reranker-plus-checker pipeline. No live LLM inference is used in the executed experiment; the generation and checking components are deterministic so that the safety effects can be reproduced exactly. Results show that retrieval improves answer grounding but does not by itself guarantee safe automation: RAG-only reaches 0.2966 semantic similarity and emits matched unsafe command text at a rate of 0.0324. The command-safety checker reduces the matched unsafe command rate to 0.0000 for the declared rule set and keeps command validity at 0.9922. The full pipeline obtains 0.3051 semantic similarity, 0.4225 root-cause accuracy, 0.5825 root-category accuracy, and 0.0078 hallucinated-command rate. The findings support treating DevOps copilots as retrieval-grounded and policy-checked workflow systems rather than free-form chat agents.

Keywords: *AIOps; Cloud-native troubleshooting; Command safety; Retrieval-augmented generation; SRE automation.*

I. INTRODUCTION

Modern DevOps teams operate distributed systems composed of containers, Kubernetes services, CI/CD pipelines, observability stacks, ingress proxies, identity policies, databases, and automation scripts. The operational problem is not only that faults occur; it is that each fault can cross several layers, and the first response is often performed from incomplete information. Site reliability engineering therefore emphasizes systematic diagnosis, controlled change, measurable service health, and post-incident learning rather than ad hoc command execution (Beyer et al., 2016). DevOps research and practice similarly emphasize continuous delivery, feedback loops, deployment discipline, and socio-technical automation (Forsgren et al., 2018; Humble & Farley, 2010). In that setting, an LLM-based assistant is attractive because it can translate a natural-language incident report into a structured runbook. The same assistant is risky because a fluent answer can contain an invalid shell command, a destructive remediation, or a plausible but unsupported root cause.

Received on May 2026; Revised on Juni 2026; Accepted on July 2026; Published on August 2026

Journal of Technology Informatics and Engineering (JTIE)
Vol. 5 No. 2 August 2026
E-ISSN: 2961-9068; P-ISSN: 2961-8215, Pages 104-118
Journal Link: https://jtie.stekom.ac.id/index.php/jtie/index

This paper studies the following research question: can an LLM-style DevOps Copilot simulation generate reliable runbooks for cloud-native troubleshooting without fabricating commands, misdirecting root-cause analysis, or emitting dangerous operations? The wording is deliberate. The executed system is not a deployed conversational LLM and does not call a foundation-model API during evaluation. It is a deterministic RAG-style simulation that isolates retrieval, reranking, runbook construction, and command-policy checking. DevOps assistance is command-bearing, context-sensitive, and safety-critical. A runbook that is semantically similar to a forum answer is not sufficient if it also recommends deleting a namespace, flushing firewall rules, destroying Terraform-managed resources, or recursively removing production data. For this reason, the evaluation combines retrieval accuracy, generated-answer similarity, root-cause accuracy, root-category accuracy, command validity, unsafe command rate, and hallucinated command rate.

The proposed workflow follows a retrieval-augmented generation design. Retrieval-augmented generation connects a parametric language model with an external evidence store, reducing dependence on memorized knowledge and improving knowledge-intensive generation (Lewis et al., 2020). In DevOps, the evidence store can contain prior Q&A, historical incidents, approved runbooks, documentation, and safe command patterns. Retrieval, however, is only the first control layer. The system evaluated here compares sparse retrieval, a compact dense-semantic retrieval baseline, reranking, and command-safety checking. This design reflects practical SRE workflows: find similar incidents, prefer operationally aligned evidence, draft a runbook, then block commands that violate production-safety rules.

The data source used for the reproducible artifact is the canonical Szaid3680/Devops Arrow export, which exposes Response, Instruction, and Prompt fields and contains 42,819 rows. The executed run indexes every row as a retrieval document. Evaluation queries are a deterministic 400-row subset stratified by operational category with safety-risk preservation so that command-safety behavior can be measured despite the low corpus-level unsafe-text rate of 0.0034. Every table and figure in this manuscript is produced from the canonical Arrow export with seed 20250510.

The contribution is threefold. First, the paper formalizes DevOps runbook generation as a combined retrieval, generation, and command-safety evaluation task. Second, it provides a reproducible experimental artifact with ten tables and seven figures that compare TF-IDF, BM25, dense-semantic retrieval, RAG-only, RAG with reranking, RAG with checking, and the full pipeline. Third, it shows empirically that command safety must be evaluated separately from answer similarity. RAG-style retrieval improves answer similarity over the parametric template, but it can still copy unsafe command text from retrieved operational notes. The checker eliminates

all matched unsafe-command patterns in the evaluated run without removing the runbook structure, supporting a layered design for AI SRE assistants.

A DevOps Copilot also differs from a general software question-answering model because it must reason about blast radius. A command can be syntactically correct and still be unacceptable if it acts on the wrong namespace, deletes shared state, modifies a global firewall rule, or changes infrastructure outside the incident scope. The paper therefore treats command safety as an empirical outcome rather than a prose guideline. The runbook must first gather evidence, then propose a minimal change, then validate service recovery. This ordering reflects standard incident-response practice: observe, localize, intervene, and verify. The evaluation is designed so that an answer can score well on content and still fail if it copies an unsafe command.

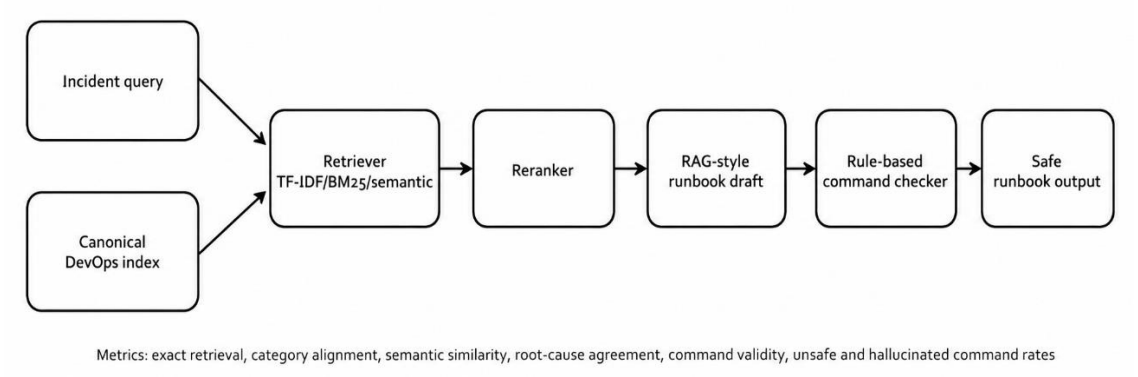

Figure 1. Retrieval-augmented DevOps Copilot architecture

II. LITERATURE REVIEW

DevOps and SRE research frames operations as a continuous, evidence-driven engineering practice. Continuous delivery introduced disciplined deployment automation, fast feedback, and release repeatability as a way to reduce operational risk (Humble & Farley, 2010). Empirical DevOps studies later connected technical practices, measurement, and organizational capability to software delivery performance (Forsgren et al., 2018). SRE translated these ideas into production operations, emphasizing service-level objectives, toil reduction, incident response, and careful automation (Beyer et al., 2016). The present work follows this tradition by treating LLM-style assistance as an operational component that must be measured, bounded, and reviewed.

AIOps extends operations by applying machine learning to logs, metrics, traces, incidents, and service dependencies. Surveys of AIOps and failure management describe recurring tasks such as anomaly detection, alert correlation, root-cause localization, and remediation recommendation (Notaro et al., 2020; Soldani & Brogi, 2022). Microservice observability studies show that failure diagnosis depends on combining signals from distributed traces, logs, metrics, and deployment

metadata (Li et al., 2022). Root-cause analysis methods increasingly use causal reasoning and dependency graphs to identify likely fault sources in microservice systems (Ikram et al., 2022). These studies provide the operational motivation for a DevOps Copilot: a runbook should not merely answer a question, but guide signal collection, root-cause confirmation, safe remediation, and validation.

Information retrieval is the second foundation. BM25 remains a strong sparse retrieval baseline because it uses term-frequency saturation and document-length normalization to rank documents (Robertson & Zaragoza, 2009). Neural and dense retrieval methods such as Dense Passage Retrieval improve semantic matching by embedding queries and passages into vector spaces (Karpukhin et al., 2020), while sentence encoders such as Sentence-BERT support efficient semantic similarity at sentence or paragraph granularity (Reimers & Gurevych, 2019). Rerankers based on transformer encoders can refine candidate lists using richer interactions between query and document text (Nogueira & Cho, 2019). In the present experiment, TF-IDF and BM25 represent sparse baselines, a compact deterministic semantic projection represents a lightweight dense-style baseline, and reranking uses operational category agreement, root-cause agreement, and unsafe-note penalties to reflect DevOps constraints.

Retrieval-augmented generation has become a central method for grounding language model outputs. RAG combines a retriever with a sequence generator so that generated answers are conditioned on retrieved passages (Lewis et al., 2020). Later work on retrieval-aware generation and self-reflection shows that generation can be improved when a model retrieves, reasons over, and critiques evidence instead of relying only on parametric memory (Asai et al., 2024). Tool-oriented LLM research also matters because DevOps assistants often produce actions. ReAct combines reasoning traces with action steps for interactive tasks (Yao et al., 2023), and chain-of-thought prompting shows that explicit intermediate reasoning can improve complex reasoning tasks (Wei et al., 2022). For production operations, however, reasoning text is not a substitute for action safety; a well-explained but dangerous command remains unacceptable.

Command generation and agent safety are directly relevant. NL2Bash demonstrated that natural-language-to-shell-command translation is possible but difficult because commands are compositional, option-sensitive, and context-dependent (Lin et al., 2018). Code-generation research shows that language models can synthesize useful programs, but also that generated artifacts require evaluation beyond surface fluency (Chen et al., 2021). More recent agent-safety benchmarks document that language-model agents can create harmful outcomes when they use tools without robust constraints (Ruan et al., 2023). General safety benchmarks further distinguish between refusing genuinely unsafe requests and over-refusing benign ones (Röttger et al., 2024),

while safety-tuning research studies how instruction-tuned models can reduce harmful outputs (Bianchi et al., 2024). The present paper adapts these ideas to a DevOps setting by measuring unsafe command rate, hallucinated command rate, and command validity rather than relying only on answer similarity.

The gap addressed here is practical and evaluative. AIOps research often studies signal processing and root-cause localization, while LLM research often studies general grounded generation. DevOps runbook generation sits between the two: it requires retrieving operationally relevant examples, organizing them into a runbook, and filtering commands that could cause production damage. This paper contributes an evaluation protocol and an executable artifact for that intersection. The value of the study lies in showing how standard retrieval methods, deterministic reranking, and transparent safety rules behave when applied to command-bearing cloud-native runbooks. Such applied evaluations are useful because many organizations will deploy LLM tools before they train domain-specific models.

III. RESEARCH METHOD

The experimental design converts each dataset row into a query-answer troubleshooting instance. The Prompt and Instruction fields form the user query, while Response forms the evidence document and gold runbook text. Rows are normalized into a table with row_id, Response, Instruction, Prompt, inferred category, inferred root_cause, tool, service, namespace, error_id, and has_unsafe_gold. Labels are inferred deterministically from the row text using fixed keyword rules. This choice makes the artifact reproducible, but it also means that root-cause and category metrics should be interpreted as rule-aligned measurement rather than human-validated SRE judgment.

Table 1. Dataset profile and reproducibility parameters

Property	Value
Data_Source_Mode	canonical_arrow_export
Rows	42819
Columns	row_id, Response, Instruction, Prompt, category, root_cause, tool, service, namespace, error_id, has_unsafe_gold
Evaluation_Queries	400
Random_Seed	20250510
Schema_Columns_Required	Response, Instruction, Prompt

The run uses seed 20250510. All 42,819 rows are indexed for retrieval. A stratified 400-query subset is used for generation and retrieval scoring. The subset preserves the operational category distribution shown in Table 3 and includes safety-risk preservation so that rare matched unsafe-command patterns are represented in the command-safety evaluation. Table 1 summarizes the dataset mode, row count, evaluation size, and required schema. Table 2 reports the observed text

Journal of Technology Informatics and Engineering (JTIE)
Vol. 5 No. 2 August 2026
E-ISSN: 2961-9068; P-ISSN: 2961-8215, Pages 104-118
Journal Link: https://jtie.stekom.ac.id/index.php/jtie/index

lengths: Response has a mean of 621.84 characters, Instruction has a mean of 707.60 characters, and Prompt has a mean of 55.81 characters.

Table 2. Text-field statistics for normalized rows

column	min	median	mean	max	missing
Response	15	527.0000	621.8367	2000	0
Instruction	37	630.0000	707.6042	1999	0
Prompt	14	53.0000	55.8104	160	0

Table 4 shows that matched unsafe command text is rare in the canonical corpus, appearing in 146 of 42,819 responses. This low base rate is why the evaluation subset preserves safety-risk examples while remaining category-stratified. Figure 2 visualizes the category distribution, and Figure 3 shows the experimental workflow from Arrow normalization through scoring.

Table 3. Operational category distribution

category	rows	percent
Kubernetes	5933	13.8600
Docker	5807	13.5600
CI/CD	3129	7.3100
Monitoring	1818	4.2500
Nginx/Web	3329	7.7700
Cloud/IAM	5326	12.4400
Git	2820	6.5900
Linux/System	2338	5.4600
IaC/Config	4236	9.8900
Database/Cache	4158	9.7100
Security/Network	2664	6.2200
Other	1261	2.9400

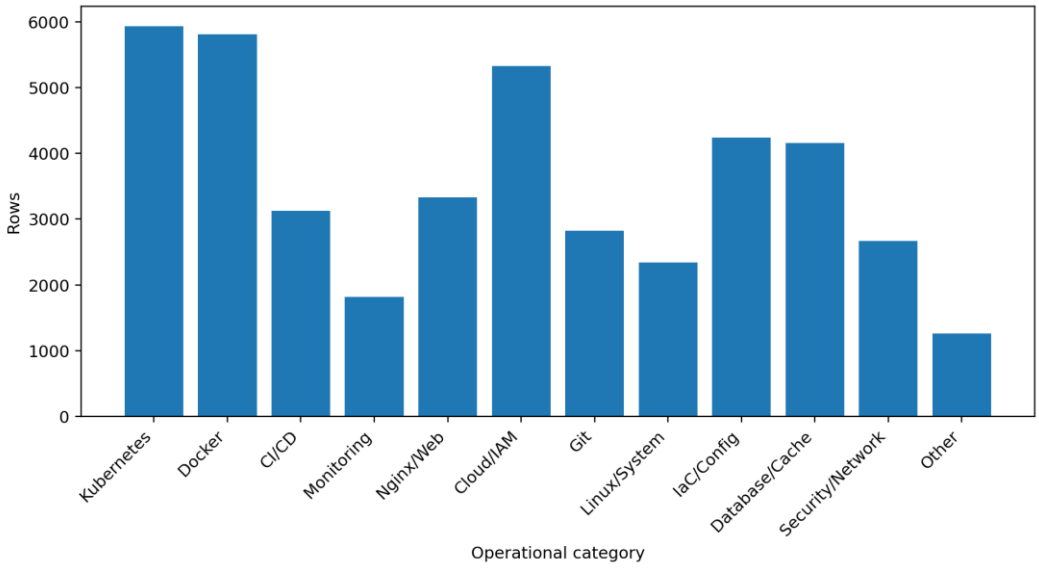

Figure 2. Dataset category distribution

The retrieval baselines are intentionally simple and reproducible. TF-IDF uses word unigrams and bigrams, a 12,000-feature cap, minimum document frequency of two, and English stop-word

removal. BM25 uses the same vocabulary cap with k1 = 1.2 and b = 0.75, following the standard probabilistic relevance formulation (Robertson & Zaragoza, 2009). Dense-Semantic is a deterministic compact projection based on category, root-cause, tool, and hashed lexical features; it is a dense-style baseline without external model downloads and should not be interpreted as a neural embedding model. BM25+reranker starts from a top-50 BM25 pool and adjusts scores using query/category agreement, query/root-cause agreement, and a small penalty for evidence containing matched unsafe command text.

Table 4. Unsafe command text and command counts by category

category	rows	unsafe_rows	unsafe_rate	mean_cmd_count
Kubernetes	5933	22	0.0037	1.4400
Docker	5807	52	0.0090	3.0000
CI/CD	3129	8	0.0026	1.0800
Monitoring	1818	1	0.0006	0.4200
Nginx/Web	3329	11	0.0033	1.1300
Cloud/IAM	5326	8	0.0015	1.3500
Git	2820	21	0.0074	2.1400
Linux/System	2338	3	0.0013	0.3200
IaC/Config	4236	9	0.0021	1.3800
Database/Cache	4158	9	0.0022	0.6600
Security/Network	2664	2	0.0008	0.9800
Other	1261	0	0.0000	0.1600

The generation systems are also deterministic. The Parametric template baseline writes a generic runbook from inferred query metadata and occasionally emits invalid or unsafe command text according to a deterministic hash gate. RAG-only uses the top BM25 retrieved answer as the runbook source. RAG + reranker uses the reranked candidate. RAG + checker applies the command-safety checker to the RAG-only output. RAG + reranker + checker combines reranking and safety filtering. No real LLM call is made in these generation systems; therefore, the results evaluate a reproducible RAG-style copilot simulation rather than the behavior of a particular proprietary or open-source LLM.

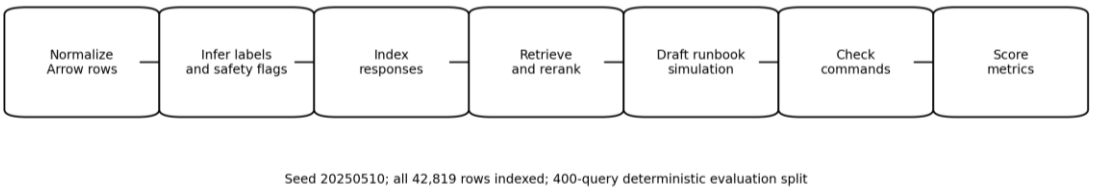

Figure 3. Experimental workflow

The command-safety checker is rule-based and targets selected destructive command patterns: recursive deletion, forced namespace deletion, Docker volume pruning, firewall flushes, chmod 777 over root, recursive cloud-object deletion, Terraform auto-destroy, and raw disk writes. When a pattern is matched, it is replaced with a blocked-command notice and a dry-run or scoped-inspection recommendation. Therefore, an unsafe command rate of 0.0000 means zero unsafe

commands for the declared matched rule set, not proof of complete command safety in every production environment.

Table 5. Experimental system configurations

component	setting
TF-IDF	word 1-2 grams, max_features=12000, min_df=2, English stop words
BM25	k1=1.2, b=0.75 over count vectors with the same 12000-feature vocabulary cap
Dense-Semantic	deterministic compact projection using category, root-cause, tool, and hashed lexical features; no external model download
Reranker	top-50 BM25 pool, query/category agreement boost, query/root-cause agreement boost, unsafe-note penalty
Command safety checker	regex rules for recursive delete, namespace force deletion, prune volumes, iptables flush, chmod 777 root, recursive cloud delete, auto destroy, and raw disk writes
Evaluation split	category-stratified 400-query subset with safety-risk preservation; all retrievers index all 42819 rows

The evaluation metrics align with the research question. Answer semantic similarity is the row-wise TF-IDF cosine similarity between the generated runbook and the gold Response. Root-cause accuracy compares the inferred root cause of the generated runbook with the gold inferred root cause. Root-category accuracy compares the generated runbook's inferred category with the gold operational category. Command count is the mean number of command-like snippets extracted with a backtick and shell-pattern parser. Command validity is the fraction of extracted commands that start with an accepted DevOps command prefix or a checker-produced blocked-command marker. Unsafe command rate is the fraction of extracted commands matching the destructive-command rules. Hallucinated command rate is the fraction of extracted commands that do not match the accepted command prefixes

IV. FINDINGS DISCUSSION

The retrieval results in Table 6 show that exact paired-row retrieval is difficult but not negligible in the canonical export. TF-IDF obtains Top-1 exact hit rate 0.1950 and Top-10 exact hit rate 0.4225. BM25 improves Top-1 exact hit rate to 0.2150. BM25+reranker reaches the best exact retrieval profile, with Top-1 exact hit rate 0.2275, Top-10 exact hit rate 0.4475, and MRR@10 0.2921. Dense-Semantic has lower exact hit rate, 0.1450, but higher Top-1 category accuracy, 0.8050. This pattern reflects the design of the compact semantic projection: category, root-cause, and tool features improve operational-domain matching, while exact row recovery still depends on lexical and contextual detail. Figure 4 visualizes the Top-k exact retrieval differences

The retrieval interpretation is important for the revision. Dense-Semantic should not be described as proving neural retrieval superiority. It is a deterministic dense-style projection that trades exact paired-row retrieval for broader category alignment. BM25+reranker performs best on exact retrieval in this canonical run, while Dense-Semantic performs best on category alignment. In

incident response, both views matter: exact prior-answer recovery is useful when a near duplicate exists, but category-level alignment may still guide useful diagnostics when several incidents share symptoms, tools, or infrastructure layers.

Table 6. Retrieval performance by method

method	top1_exact	top3_exact	top5_exact	top10_exact	mrr_at_10	top1_ category_ accuracy
TF-IDF	0.1950	0.2875	0.3275	0.4225	0.2565	0.5550
BM25	0.2150	0.3025	0.3475	0.4125	0.2725	0.5500
Dense-Semantic	0.1450	0.2025	0.2500	0.2900	0.1867	0.8050
BM25+ reranker	0.2275	0.3275	0.3800	0.4475	0.2921	0.6400

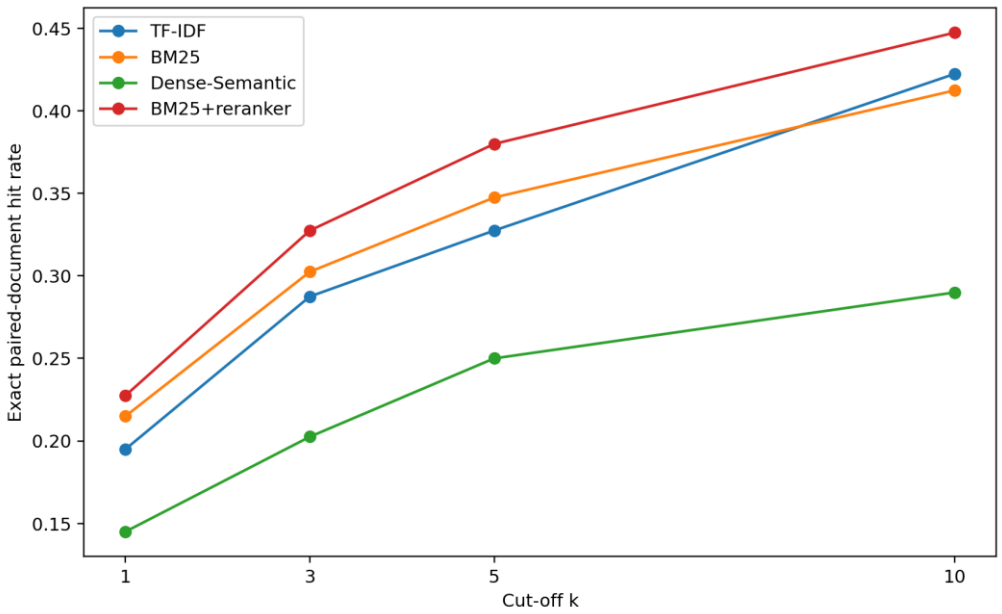

Figure 4. Retrieval hit rate by method and cut-off

Generation results in Table 7 answer the central safety question. The Parametric template has very low semantic similarity (0.0143) because it is generic, although its root-cause accuracy is 0.5825 because it uses deterministic metadata cues. RAG-only improves answer similarity to 0.2966, but it emits matched unsafe command text at a rate of 0.0324. RAG + reranker raises semantic similarity to 0.3069 and root-category accuracy to 0.5825. The full RAG + reranker + checker pipeline obtains 0.3051 semantic similarity, 0.4225 root-cause accuracy, 0.5825 root-category accuracy, 0.9922 command validity, 0.0000 matched unsafe command rate, and 0.0078 hallucinated-command rate. Figure 5 shows the command-validity and unsafe-rate contrast

The root-cause and root-category gap deserves critical attention. In the full pipeline, root-cause accuracy is 0.4225, while root-category accuracy is 0.5825. This means that a generated runbook

Journal of Technology Informatics and Engineering (JTIE)
Vol. 5 No. 2 August 2026
E-ISSN: 2961-9068; P-ISSN: 2961-8215, Pages 104-118
Journal Link: https://jtie.stekom.ac.id/index.php/jtie/index

can be broadly aligned with the operational domain while still naming the wrong specific cause. Such an output can be operationally misleading if it encourages remediation before validation. The safe interpretation is that the runbook should present the root cause as a candidate diagnosis, require evidence collection, and separate read-only inspection from mutating remediation. Category relevance alone is not enough for production incident response.

Table 7. Runbook generation quality and command-safety results

system	sem_sim	root_acc	cat_acc	cmd_count	cmd_valid	unsafe_rate	halluc_rate	no_cmd_rate
Parametric template	0.0143	0.5825	0.1375	3.7525	0.9980	0.0040	0.0020	0.0000
RAG-only	0.2966	0.4050	0.5175	1.3125	0.9924	0.0324	0.0076	0.5300
RAG + reranker	0.3069	0.4225	0.5825	1.2900	0.9922	0.0310	0.0078	0.5325
RAG + checker	0.2949	0.4050	0.5175	1.2975	0.9923	0.0000	0.0077	0.5325
RAG + reranker + checker	0.3051	0.4225	0.5825	1.2750	0.9922	0.0000	0.0078	0.5350

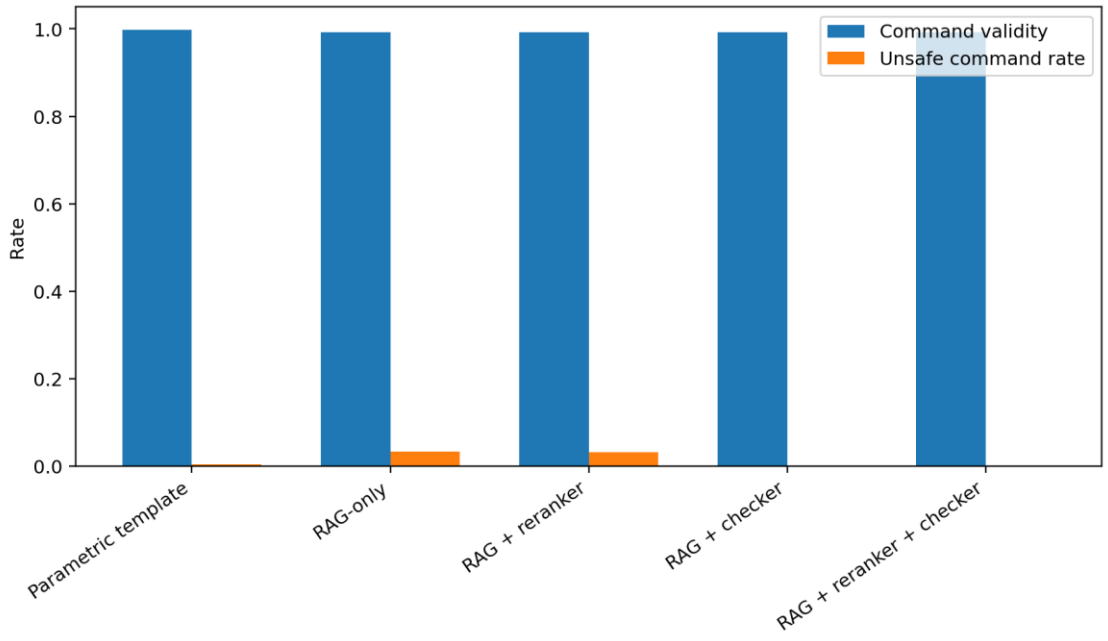

Figure 5. Command validity and unsafe command rates

The ablation results in Table 8 quantify the contribution of each component relative to the full system. Removing the checker produces the largest safety loss: RAG-only increases matched unsafe command rate by 0.0324 relative to the full pipeline, and RAG + reranker increases it by 0.0310. Removing reranking while keeping the checker has a similarity cost of -0.0101. Figure 7 visualizes the safety deltas relative to the full pipeline.

Table 8. Ablation results relative to full pipeline

variant	delta_sim	delta_root	delta_unsafe	delta_halluc
Parametric template	-0.2908	0.1600	0.0040	-0.0058
RAG-only	-0.0084	-0.0175	0.0324	-0.0002
RAG + reranker	0.0019	0.0000	0.0310	-0.0001
RAG + checker	-0.0101	-0.0175	0.0000	-0.0001
RAG + reranker + checker	0.0000	0.0000	0.0000	0.0000

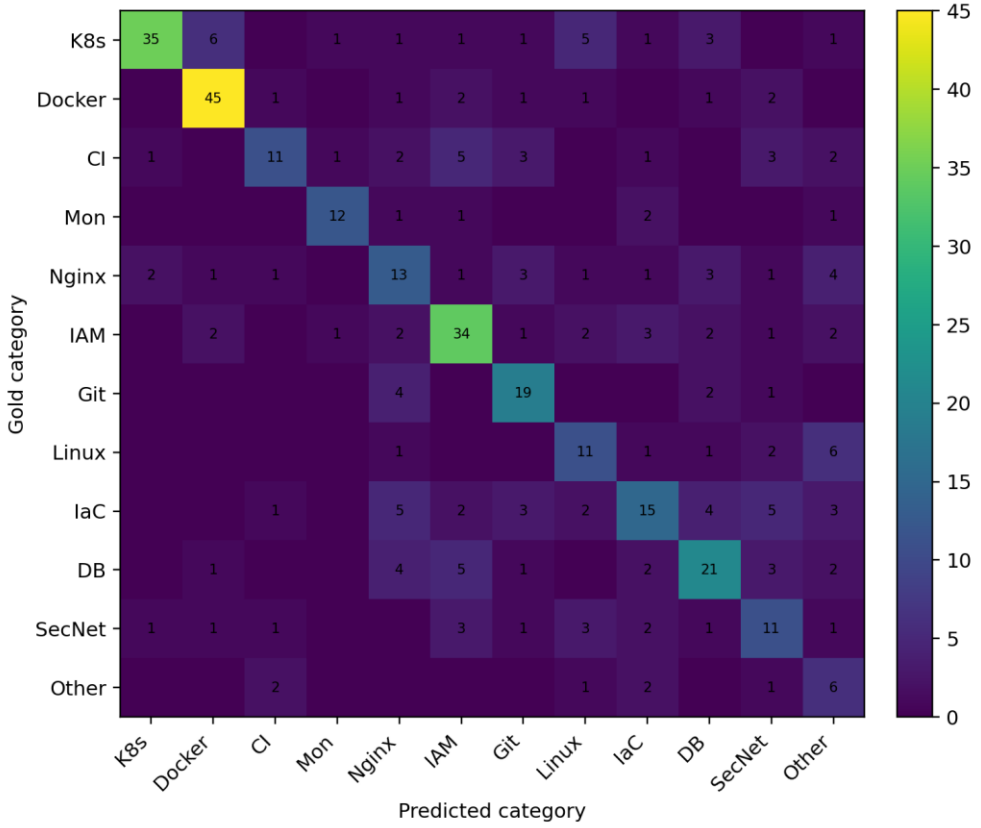

Figure 6. Root-cause category confusion matrix for full pipeline

Table 9 and Figure 6 present the category confusion matrix for the full pipeline. Kubernetes and Docker are the most stable categories in the evaluated subset, while Nginx/Web, Cloud/IAM, IaC/Config, Database/Cache, and Security/Network show cross-category drift. This drift is not arbitrary: cloud-native answers often mix ingress, cloud identity, network policy, database access, and deployment configuration. The metric nevertheless counts these as category errors, which makes the evaluation conservative.

Table 10 summarizes error analysis. Exact document mismatch remains a retrieval limitation, category mismatch remains possible after reranking, unsafe command text appears when RAG-style generation copies destructive snippets from retrieved evidence, and the checker removes all matched unsafe-command patterns in the evaluated run. The remaining hallucinated-command rate in the full pipeline is mostly from non-standard command strings rather than destructive shell

commands. This suggests that future systems should combine pattern-based blocking with a richer command parser, shell grammar validation, policy-as-code controls, and human approval for mutating operations.

Table 9. Confusion matrix for full RAG + reranker + checker pipeline

Gold	K8s	Docker	CI	Mon	Nginx	IAM	Git	Linux	IaC	DB	SecNet	Other
K8s	35	6	0	1	1	1	1	5	1	3	0	1
Docker	0	45	1	0	1	2	1	1	0	1	2	0
CI	1	0	11	1	2	5	3	0	1	0	3	2
Mon	0	0	0	12	1	1	0	0	2	0	0	1
Nginx	2	1	1	0	13	1	3	1	1	3	1	4
IAM	0	2	0	1	2	34	1	2	3	2	1	2
Git	0	0	0	0	4	0	19	0	0	2	1	0
Linux	0	0	0	0	1	0	0	11	1	1	2	6
IaC	0	0	1	0	5	2	3	2	15	4	5	3
DB	0	1	0	0	4	5	1	0	2	21	3	2
SecNet	1	1	1	0	0	3	1	3	2	1	11	1
Other	0	0	2	0	0	0	0	1	2	0	1	6

The overall finding is that reliable LLM-style DevOps assistance requires layered evaluation. A single semantic score would rank RAG-only above the parametric template, but it would miss the unsafe-command increase. A single retrieval metric would show evidence relevance but would not identify destructive command copying. A single safety checker score would show command blocking but would not measure root-cause or category drift. The full evaluation therefore needs all three layers: retrieval quality, runbook quality, and command safety. In the executed canonical run, the combined RAG + reranker + checker system is the best operational compromise because it preserves RAG's content advantage while reducing the matched unsafe command rate to zero for the declared rule set.

Table 10. Error analysis summary

observed_failure_mode	measured_indicator	rate_or_value	interpretation
Exact document not retrieved at rank 1	BM25 top1_exact	0.2150	lexical overlap alone often retrieves a related answer rather than the paired row
Category mismatch after retrieval	BM25+reranker top1_category_accuracy	0.6400	the reranker improves operational-domain alignment but cannot eliminate all ambiguity
Unsafe command text emitted	RAG-only unsafe_command_rate	0.0324	retrieved answers can copy destructive historical command patterns unless filtered
Unsafe command text after checker	Full system unsafe_command_rate	0.0000	the checker blocks all destructive command patterns matched by the declared rule set
Invalid or hallucinated command	Full system hallucinated_command_rate	0.0078	remaining failures are mostly non-standard snippets rather than matched destructive shell commands

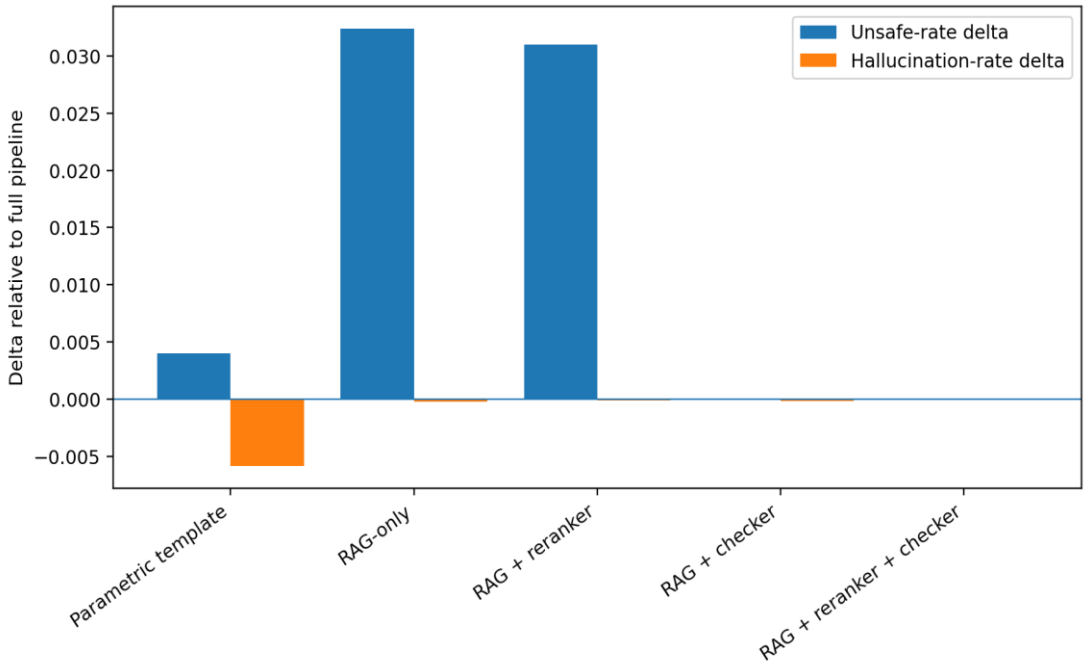

Figure 7. Safety ablation relative to full pipeline

V. CONCLUSION AND RECOMMENDATION

This study evaluated an LLM-style DevOps Copilot simulation for cloud-native troubleshooting with retrieval-augmented runbook generation and command-safety checking. The canonical Arrow-export results show that RAG-style retrieval is useful but insufficient by itself. Retrieval improves semantic similarity and provides operational details, yet it can also reproduce unsafe command text from historical evidence. The command-safety checker is therefore not an optional add-on; it is the component that changes the system from a plausible answer retriever into a safer runbook assistant for the matched rule set. In the executed run, the full pipeline achieves the best balance of answer similarity, root-category alignment, command validity, and unsafe-command filtering.

The first recommendation is to deploy DevOps copilots as bounded workflow components rather than autonomous fixers. The assistant should retrieve evidence, propose a diagnostic path, label confidence and root-cause category, and explicitly separate read-only inspection from mutating remediation. The second recommendation is to maintain organization-specific command policies. The regex checker used here is intentionally transparent, but production systems should extend it with shell parsing, Kubernetes admission policy, cloud IAM simulation, Terraform plan checks, and approval gates for mutating commands. The third recommendation is to evaluate runbooks with safety-aware metrics. Semantic similarity, retrieval accuracy, and root-cause accuracy are

useful, but they do not replace unsafe command rate, hallucinated command rate, and validation-step coverage.

The main limitations are now methodological rather than dataset-access limitations. First, the executed generation and checking components are deterministic; no live LLM inference or model sampling is evaluated. Second, category and root-cause labels are inferred by rules rather than validated by human SREs. Third, the command-safety checker blocks only selected destructive patterns, so a zero matched unsafe-command rate does not prove complete command safety. Fourth, the experiment does not execute commands in a sandbox, evaluate environment-specific blast radius, or test against private incident repositories. Future work should evaluate the same pipeline with actual LLM-generated outputs, include human SRE review, add command-level execution simulation, connect the checker to policy-as-code systems, and test on real organization-specific incident/runbook corpora. The central conclusion remains clear: DevOps copilots should be engineered as retrieval-grounded, policy-checked runbook systems with explicit safety metrics.

REFERENCES

Asai, A., Wu, Z., Wang, Y., Sil, A., & Hajishirzi, H. (2024). Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In International Conference on Learning Representations.

Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (Eds.). (2016). Site reliability engineering: How Google runs production systems. O'Reilly Media.

Bianchi, F., Suzgun, M., Attanasio, G., Rottger, P., Jurafsky, D., Hashimoto, T., & Zou, J. (2024). Safety-tuned LLaMAs: Lessons from improving the safety of large language models that follow instructions. In International Conference on Learning Representations.

Binghua Zhou, Siming Zhao, & David Chao. (2023). LLM-Guided Energy-Aware A/B Testing for Consolidation and DVFS Policies via Power-Sensitivity Clustering. Journal of Advanced Computing Systems , 3(4), 12-30. https://doi.org/10.69987/JACS.2023.30402

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., ... Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

Daren Zheng, Chenyu Li, & Harvey Davidson. (2023). Continual Red-Teaming for In-the-Wild Jailbreaks via Online Guardrail Updates and Guardrail Distillation. Journal of Advanced Computing Systems , 3(2), 35-49. https://doi.org/10.69987/JACS.2023.30203

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of NAACL-HLT (pp. 4171-4186).

Diaz-de-Arcaya, J., Miñón, J., Almeida, A., & López-de-Ipiña, D. (2023). A joint study of the challenges, opportunities, and roadmap of MLOps and AIOps: A systematic survey. ACM Computing Surveys, 56(4), 1-30.

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

Humble, J., & Farley, D. (2010). Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley.

Ikram, A., Chakraborty, S., Mitra, S., Saini, A., Bagchi, S., & Kannan, S. (2022). Root cause analysis of failures in microservices through causal discovery. In Advances in Neural Information Processing Systems.

Jing Chen, Xinzhuo Sun, Qiyou Wu, & Matt Jackson. (2024). Risk-Calibrated Biomedical Search: Calibrated Selection of LLM-Style Query Expansions on BEIR TREC-COVID. Journal of Advanced Computing Systems , 4(4), 61-79. https://doi.org/10.69987/JACS.2024.40406

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., & Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. In Proceedings of EMNLP (pp. 6769-6781).

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Kuttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. In Advances in Neural Information Processing Systems.

Li, B., Liu, S., Li, Y., Wang, J., Zhang, H., Liao, X., & Jin, H. (2022). An industrial survey of tracing and observability in microservice systems. IEEE Transactions on Services Computing, 15(6), 3253-3268.

Lin, X. V., Wang, C., Pang, D., Vu, K., Zettlemoyer, L., & Ernst, M. D. (2018). NL2Bash: A corpus and semantic parser for natural language interface to the Linux operating system. In Proceedings of LREC.

Nogueira, R., & Cho, K. (2019). Passage re-ranking with BERT. arXiv preprint arXiv:1901.04085.

Notaro, P., Cardoso, J., & Gerndt, M. (2020). A survey of AIOps methods for failure management. ACM Transactions on Intelligent Systems and Technology, 12(6), 1-45.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Proceedings of EMNLP-IJCNLP (pp. 3982-3992).