

When LLM Plans Fail: Solver-Verified Remediation Under Operational Constraints

Priyaranjan Pattnayak

Oracle America Inc.

priyaranjanpattnayak@gmail.com

Abstract

We study postmortem remediation planning for large-scale cloud incidents, focusing on whether generated actions comply with explicit operational constraints and are executable in practice. While LLM-based systems can produce structured corrective actions, we find that 25–35% of generated plans across 8,200 real-world incidents violate at least one encoded feasibility constraint due to hidden dependencies, policy limits, or risk budgets.

We introduce *Solver-Verified Remediation Planning (SVRP)*, a decision-centric framework that separates action proposal from feasibility verification. An LLM proposes candidate actions, which are checked against an explicit constraint system using satisfiability solving; inclusion-minimal diagnostic cores guide targeted repair or abstention. SVRP raises feasibility from 62.4% to 93.1%, reduces encoded risk and permission violations by more than 50%, and improves expert-rated executability from 3.7 to 4.7 while preserving semantic intent. These results suggest that remediation failures often reflect structural constraints that benefit from solver-backed verification in addition to improved prompting.

1 Introduction

Cloud outages are costly and operationally complex. While postmortem analyses help teams understand failures, a final step of producing *Corrective and Preventive Measures (CAPM)* that specify what actions should be taken, remains manual and difficult to scale (Ahmed et al., 2023; Chen et al., 2024). In practice, CAPMs must be actionable, role-appropriate and safe, yet are often inconsistent or infeasible under operational policies.

Recent work shows that LLMs can generate structured remediation actions from incident narratives (Lin et al., 2025; Huang et al., 2024). However, these systems are evaluated primarily as text generators, creating a mismatch: plans may be

fluent yet *jointly infeasible* with real operational constraints due to hidden dependencies, role policies, or risk limits (Hao et al., 2025; Schick et al., 2023; Yao et al., 2023). Across 8,200 real-world incidents, 25–35% of LLM-generated plans violate at least one feasibility constraint, indicating failures driven by unmodeled action interactions rather than language errors.

We introduce *Solver-Verified Remediation Planning (SVRP)*, a decision-centric framework that separates action proposal from feasibility verification. An LLM proposes candidate actions, which are verified against an explicit operational constraint system using satisfiability solving. When plans are infeasible, minimal unsatisfiable cores guide targeted repair or principled abstention. SVRP avoids relying on LLMs to judge feasibility or safety.

Evaluated on 8,200 real-world incidents and a synthetic stress test, SVRP raises feasibility from 62.4% to 93.1%, reduces unsafe actions by more than 50%, and improves expert-rated executability from 3.7 to 4.7 (5-pt scale). These results show that failures in LLM-generated remediation are structural rather than linguistic, and benefit from explicit constraint reasoning beyond prompting alone.

Our Contributions.

- We identify and quantify a systematic gap between linguistically plausible remediation plans and plans that are feasible under real operational constraints.
- We introduce *Solver-Verified Remediation Planning (SVRP)*, that separates action proposal from decision-level feasibility using solver-based checking and unsatisfiable-core guided repair.
- We provide rigorous evaluation on 8,200 real-world incidents, demonstrating consistent improvements in feasibility, safety, and exe-

cutability across frozen and fine-tuned generators.

- We release a synthetic dataset (≈ 950 incidents) to enable reproducible analysis of interaction-level feasibility violations.

2 Related Work

Incident Management and Root Cause Analysis. Automating cloud incident management has focused primarily on *RCA*, identifying why failures occur rather than what actions should follow. Prior work uses fault patterns from incident tickets (Huang et al., 2024), causal inference (Chen et al., 2019), and fine-tuned LLMs (Ahmed et al., 2023; Chen et al., 2024); systems such as GPT-4-based *RCA* (Zhang et al., 2024) and RCACopilot (Chen et al., 2024) further demonstrate LLM-based diagnosis. However, these approaches typically stop at diagnosis or provide remediation suggestions without explicit feasibility or safety guarantees.

LLM-Based Action Generation for Operations. Prior work explores LLM-based action generation, including workflow synthesis (Qiao et al., 2025), retrieval-augmented dialogue generation (Shen et al., 2023), and cloud operations copilots (Lin et al., 2025). While these methods generate fluent remediation steps, they treat planning as text generation and do not ensure that action sets are jointly executable under operational constraints such as risk limits, permissions, and dependencies.

Verification and Constraint Checking with LLMs. LLM reliability work combines models with symbolic reasoning, formal solvers, self-critique, and verify-and-edit loops (Ye et al., 2023; Hao et al., 2025; Lee et al., 2025; Madaan et al., 2023; Zhao et al., 2023). Program repair similarly uses diagnostics to localize inconsistencies (Weimer et al., 2009). However, these approaches primarily target code or synthetic planning benchmarks and often reject or regenerate infeasible outputs rather than repairing operational action sets.

Positioning of This Work. We frame postmortem remediation as a *decision-level feasibility problem over action sets*, rather than a generation or validation task. LLMs are used only to propose candidate actions, while solver-based verification determines joint executability under explicit operational constraints. Unlike prior works, when plans

are infeasible, unsatisfiable cores localize conflicts and enable targeted, constraint-preserving repair or abstention. This design supports reliable post-incident decision-making in settings where failures are structural rather than surface-level text quality.

3 Methodology

We address postmortem remediation as a decision problem over action sets, separating action proposal from feasibility verification. Our method, *Solver-Verified Remediation Planning (SVRP)*, uses LLMs to propose candidate actions and applies solver-based verification to ensure joint feasibility, safety, and executability under real constraints.

3.1 Problem Setting

Given an incident summary x , the goal is to produce a set of corrective and preventive actions $A = \{a_1, \dots, a_n\}$ that can be jointly executed. A plan is valid only if it satisfies a fixed set of explicit operational constraints, including role permissions, risk limits, action dependencies, and verification requirements. All constraints are derived from standard postmortem practices, fixed prior to evaluation, reviewed by independent SRE and technical-program-management experts, and applied uniformly across all incidents, services, and baselines.

3.2 Action Proposal

For each incident, an LLM proposes candidate remediation actions conditioned on the incident summary and retrieved historical context. Actions are normalized into structured tuples (e.g., *type, role, scope, risk, dependencies*). The proposal stage is permissive and does not enforce feasibility. Each incident yields 6–12 actions (median 9), capped at 15 as a fixed guardrail on proposal length, normalization cost, and end-to-end latency. Details in Appendix A.2.1.

3.3 Feasibility Constraints

SVRP verifies action sets against four declarative constraint classes: (1) **Role**: actions must be assigned to permitted roles; (2) **Risk**: actions have fixed risk levels (low, medium, high) with severity-conditioned budgets; (3) **Dependency**: prerequisite and ordering requirements must hold; (4) **Verification**: each plan must include at least one verification step. All constraints are deterministic, LLM-independent, and specified in Appendix A.2.3.

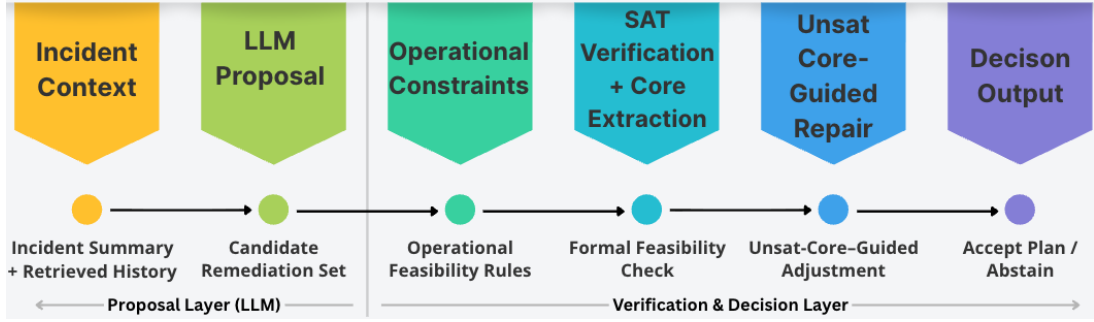


Figure 1: Overview of Solver-Verified Remediation Planning (SVRP)

3.4 Solver-Based Verification

Proposed actions and constraints are compiled into a propositional satisfiability (SAT) formulation, where each action a_i corresponds to a Boolean inclusion variable $z_i \in \{0, 1\}$. Constraints are encoded as hard clauses, including severity-conditioned constraints for high-risk actions.

We check feasibility with MiniSat 2.2, passing all proposed actions as positive assumptions so none can be silently dropped. A SAT result therefore confirms the selected plan against the encoded constraints. For UNSAT plans, we shrink the returned conflict to an inclusion-minimal diagnostic core, which is not necessarily the smallest possible core. Action ordering is checked separately using a deterministic validator. Solver settings and encodings are fixed across datasets (Appendix A.2.4).

3.5 Unsat-Core-Guided Repair and Abstention

When a plan is non-compliant, SVRP uses the diagnostic core to perform targeted, deterministic repair rather than discarding the plan. Repairs follow a fixed priority: (i) insert an available prerequisite from the proposal set, (ii) enforce ordering, (iii) substitute a lower-risk proposed alternative, and (iv) greedily remove a core action. Repair is limited to two iterations to guard against semantic drift; if feasibility cannot be restored, SVRP abstains. Details in Appendix A.2.5.

3.6 Experimental Setup

We evaluate SVRP on 8,200 real-world incidents spanning 36 months and a synthetic dataset of 950 incidents designed to induce interaction-level feasibility violations. We compare against generation-only, rule-based filtering, LLM-based validation, constrained LLM generation, constrained LLM rewriting, and a fine-tuned generator baseline.

Cohere Command-R+ is used for proposal, vali-

dation, and constrained prompting (identical decoding), and a fine-tuned Mixtral-8x22B model serves as a stronger generator baseline. Human evaluation is conducted on disagreement cases.

Deterministic vs. Learned Components. All constraints, solver checks, diagnostics, and repair operations are deterministic. In SVRP, LLMs are used only for proposing actions and are never used for feasibility validation or repair.

4 Evaluation

We evaluate SVRP as a *decision system*, not a text generator, focusing on feasibility, safety, executability, and repair under explicit operational constraints. Metrics are computed deterministically from structured action representations.

We test robustness across frozen (Cohere Command-R+) and fine-tuned (Mixtral-8x22B) generators. Surface text metrics (e.g., BLEU, BERTScore, groundedness) are reported in the appendix but are secondary, as they do not measure executability or constraint satisfaction. Human evaluation assesses real-world decision quality.

4.1 Baselines

All prompts are provided in Appendix A.4.

Generation-only. Accepts the LLM-proposed action set without feasibility checking or repair.

Rule-based filtering A deterministic baseline that greedily removes actions violating local constraints (unauthorized roles, severity-dependent risk limits, duplicate scope, or missing verification). Rules are applied until no violations remain or all actions are removed. It performs no global reasoning, dependency insertion, or minimal repair.

LLM-based validation and rewriting. Uses Cohere Command-R+ (temperature 0.2, top- $p = 0.75$) to rewrite proposed plans without access to

explicit constraints, solver diagnostics, or unsatisfiable cores, reflecting common prompt-based validation approach using implicit reasoning.

Constrained LLM generation. Uses the same LLM and configuration, but provides the model with the operational rules used by SVRP: the action vocabulary, role permissions, severity-conditioned risk budgets, dependency requirements, and mandatory verification requirement. The model generates a revised plan directly, without solver feedback or symbolic repair.

Constrained LLM rewrite. Uses the same explicit operational rules as constrained generation, but applies them as a post-processing rewrite over the original proposed plan. This baseline tests whether prompt-level access to the constraints is sufficient to repair infeasible plans without unsatisfiable cores or deterministic solver-guided edits.

Fine-tuned generator baseline: A Mixtral-8x22B-Instruct model fine-tuned with LoRA on a separate historical pool of 8,200 incident-CAPM pairs, split 6,560/820/820 by incident. No pair overlaps the 8,200 evaluation incidents; Appendix A.2.2 details deduplication and retrieval separation. The model generates plans from incident summaries without retrieval or feasibility enforcement.

4.2 Feasibility and Verification Metrics

Let $\mathcal{P} = \{P_i\}_{i=1}^N$ denote remediation plans and $\mathcal{C}$ the fixed constraint set. All feasibility metrics are computed using the same constraint checker:

SAT Rate (%): fraction of plans satisfying all constraints.

$$\text{SAT} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}[P_i \models \mathcal{C}],$$

where $\mathbb{I}[\cdot]$ indicates all constraints satisfied.

Repair-SAT Rate (%): % of initially infeasible plans repaired to satisfiable with bounded repair.

UNSAT Rate (%): fraction of plans remaining infeasible after repair.

Unsat Core Size: mean size of deletion-shrunk, inclusion-minimal diagnostic cores for infeasible plans.

Constraint Coverage: average number of distinct constraint classes (role, risk, dependency, verification) exercised during checking or repair.

Higher SAT rates and constraint coverage indicate better handling of interactions represented in

the declared policy; they are not universal safety estimates.

4.3 Safety and Executability Violations

We quantify specific failure modes prior to verification using deterministic checks on structured actions: 1) **Risk Violations (%)**: % of plans containing actions whose cumulative risk exceeds severity-dependent budgets. 2) **Permission Violations (%)**: % of plans containing actions assigned to unauthorized roles. 3) **Dependency Violations (%)**: % of plans with missing prerequisites or invalid action ordering. 4) **Missing Verification (%)**: % of plans lacking any explicit validation or verification step. These metrics capture policy violations relevant to safety that cannot be detected via surface-level text similarity.

4.4 Repair Behavior

To rule out trivial deletion, we analyze repair behavior on initially infeasible plans:

Average Repair Edits: mean number of atomic edits (insert, delete, reorder, substitute).

Non-Deletion Rate (%): fraction of repairs achieving feasibility without deleting actions.

Semantic Retention: cosine similarity between sentence embeddings of original and repaired plans, computed using all-MiniLM-L6-v2.

4.5 Coverage and Abstention

Because SVRP may abstain when no feasible repair exists, we report: **Coverage (%)**, the fraction of incidents with a feasible plan, and **Quality-at-Coverage**, executability and safety measured at fixed coverage levels. This prevents inflating quality through excessive abstention and reflects realistic deployment tradeoffs.

4.6 Human Evaluation

We conduct human evaluation on 300 cases sampled from the 2,180-incident disagreement pool (26.6% of evaluation incidents), where SVRP differs from generation-only, LLM-based validation, or constrained LLM rewriting. The study estimates comparative quality when methods differ, not full-population quality. Two domain experts (senior SREs) independently rate every anonymized plan (Cohen’s κ) given the incident summary and retrieved context for:

Technical Correctness (1–5): Is the plan valid and safe given the incident context?

Table 1: Encoded-policy compliance and violation rates across mechanisms.

Method	SAT%	Risk	Perm.	Dep.	Miss.	Ver.
Gen-only	62.4	18.9	14.3	21.7	26.4	
Rule-based	81.2	6.4	5.9	13.2	11.8	
LLM validation	76.8	8.7	7.1	15.6	14.2	
Constr. LLM gen.	82.1	5.9	4.8	12.7	9.6	
Constr. LLM rewrite	86.3	4.9	3.9	10.7	8.1	
SVRP (Ours)	93.1	1.8	2.3	4.9	3.7	

Executability (1–5): Can the plan be realistically implemented as written?

Adoption Likelihood (1–5): Would the expert adopt this plan in a real postmortem?

Preference: Which plan would the expert choose overall?

5 Results

We organize results around five questions: (i) do plans satisfy operational constraints, (ii) how are infeasible plans repaired, (iii) do gains arise from abstention, (iv) human preference, and (v) are gains robust to generator strength?

5.1 Feasibility and Safety of Remediation Plans

Table 1 compares SVRP against generation-only, rule-based filtering, unconstrained LLM validation, and constraint-aware LLM baselines. Generation-only plans are frequently infeasible (62.4% SAT), with high risk, permission, dependency, and missing-verification violations. Rule-based filtering improves feasibility (81.2% SAT) primarily through deletion, while unconstrained LLM validation reaches 76.8% SAT.

Providing the LLM with explicit operational rules improves performance substantially: constrained generation reaches 82.1% SAT, and constrained rewriting reaches 86.3% SAT. SVRP achieves the highest encoded-policy compliance (93.1%) and the lowest violation rates across all categories. Dependency and missing-verification violations show similar reductions (Appendix A.5.1), confirming that many failures involve structural interactions. Over 80% of initially infeasible SVRP plans are repaired, with small diagnostic cores indicating localized conflicts (Appendix Table 12).

Constraint Coverage. Generation-only plans trigger 1.3 constraint types per incident on average, compared to 3.8 under SVRP (Table 2), indicating that gains arise from resolving multi-constraint interactions rather than trivial single-constraint cases.

Table 2: Repair behavior (on initially infeasible plans) and constraint coverage across methods. Repair metrics apply only to methods that perform repair.

Method	Avg. Edit	Non-Del.	Sem. Ret.	Const. Cov.
Gen-only	—	—	—	1.3
Rule-based	2.6	9.4	0.71	2.1
LLM rewrite	3.1	54.2	0.78	2.4
Constr. rewrite	2.4	68.7	0.83	3.2
SVRP (ours)	1.4	82.7	0.89	3.8

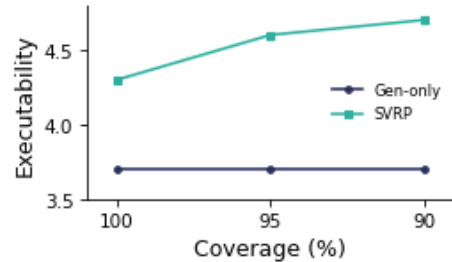


Figure 2: Coverage–executability tradeoff

5.2 Repair Behavior and Minimality

Table 2 shows that rule-based filtering achieves feasibility mainly through deletion, while LLM rewriting preserves more content but requires more edits and can introduce semantic drift. While constrained rewrite improves in both, it still lags SVRP. SVRP performs the fewest edits (1.4), achieves the highest non-deletion rate (82.7%), and best preserves intent (0.89 semantic retention). Robustness to embedding is also seen as results remain consistent with MPNet embeddings (Appendix A.5.2).

5.3 Coverage–Quality Tradeoff

Because SVRP may abstain when no feasible repair exists, we analyze quality as a function of coverage. Fig. 2 shows that even at 100% coverage, SVRP outperforms generation-only in executability. Allowing modest abstention (5–10%) yields further gains without major coverage loss, indicating that improvements stem from feasibility enforcement rather than aggressive abstention.

5.4 Human Preference and Adoption

In 300 disagreement cases, experts preferred SVRP plans 54% of the time, versus 24% for constrained LLM rewrite and less for other baselines (Table 4). SVRP also achieved the highest correctness and executability scores, with substantial agreement ($\kappa = 0.69$). All human-evaluation metrics had 95% confidence intervals within ± 0.12 (Appendix A.5.5).

Table 3: SVRP feasibility and safety across service domains.

Service Category	Incidents	SAT (%)	Risk Viol. (%)	Perm. Viol. (%)	Non-Deletion (%)
Compute & Infrastructure	1404	93.8	1.6	2.1	84.2
Database Services	895	92.9	1.9	2.4	82.7
AI & ML Services	1105	92.1	2.3	2.8	80.4
API & Identity Services	997	93.4	1.7	2.2	83.1
Networking Services	851	91.8	2.6	3.1	79.6
Storage Services	1101	93.6	1.5	2.0	84.9
Logging	903	92.4	2.1	2.6	81.3
Artifacts & Key Mgmt	499	90.7	2.9	3.4	78.5
Marketplace	445	90.1	3.2	3.8	76.9
Overall (Weighted)	8,200	93.1	1.8	2.3	82.7

Table 4: Human evaluation on disagreement cases.

Method	Correctness	Executability	Preferred %
Gen-only	3.9	3.7	10
LLM validation	4.1	4.0	12
Const. LLM rewrite	4.3	4.2	24
SVRP (Ours)	4.6	4.7	54

Table 5: Ablation results isolating SVRP components.

Variant	SAT	Avg. Edits	Non-Del.	Execut.
SVRP (Full)	93.1	1.4	82.7	4.7
w/o Solver Verif.	62.8	0.0	100.0	3.8
w/o Diagnostic Repair	88.6	2.9	56.4	4.1
Deletion-Only Repair	90.2	2.6	9.4	3.9

5.5 Robustness Across Generators and Domains

SVRP’s gains are consistent across generator strengths. As shown in Table 6, stronger generators improve proposal quality, and explicit constraint prompting further improves feasibility. However, both frozen and fine-tuned generators remain below SVRP after constraint-aware prompting.

Table 3 shows similar gains across heterogeneous service domains, with SAT rates above 90% even in smaller or specialized categories. This suggests that solver-verified planning generalizes across diverse operational constraints rather than relying on domain-specific artifacts.

5.6 Ablation Studies

We ablate core SVRP components to identify what drives feasibility and executability gains (Table 5).

Removing solver verification reduces feasibility from 93.1% to 62.8%, revealing widespread interaction-level violations. Deletion-only repair restores feasibility but causes major content loss (9.4% non-deletion; executability 3.9), whereas full SVRP achieves high feasibility, minimal edits, and

Table 6: Feasibility across generator strengths.

Generator	Gen-only SAT%	Constrained Gen. SAT%	SVRP SAT%
Frozen LLM	60.8	82.1	92.7
Fine-tuned Mixtral	68.9	87.4	94.1

superior executability. Thus, global enforcement provides the principal compliance gain, while diagnostic cores improve localization and preservation (Appendix A.5.4).

5.7 Synthetic Dataset Validation and Generation Quality

We evaluate SVRP on a released synthetic dataset of 950 incidents designed to induce dense multi-constraint conflicts. SVRP maintains high feasibility and non-deletion rates, while generation-only and heuristic baselines degrade substantially (Appendix A.5.7), indicating that gains are not driven by dataset-specific regularities.

Generation Quality. For completeness, we report surface-level proposal metrics (BLEU, BERTScore, etc.) in Appendix A.5.6. They show no degradation relative to generation-only baselines, confirming that improvements stem from decision-level verification rather than generation.

5.8 Error Analysis

Across real-world incidents, 6.9% of plans remain non-compliant after bounded repair and are routed to manual review, mainly because of (i) proposal incompleteness, (ii) unavailable admissible substitutions, and (iii) missing or unparseable verification steps. Solver-related failures are rare (<1.2%), indicating that most residual errors originate in proposal generation rather than verification or repair (Appendix A.6). Pilot deployment showed high feasibility and adoption (Appendix A.1).

6 Conclusion

We introduced Solver-Verified Remediation Planning (SVRP), a decision-centric framework that separates action proposal from feasibility verification in postmortem remediation. By combining LLM-based generation with solver-backed constraint checking and diagnostic-core-guided repair, SVRP produces plans with substantially higher encoded-policy compliance and expert-rated executability than generation-only or heuristic baselines.

Across 8,200 real-world incidents and a synthetic test set, SVRP raises encoded-policy compliance from 62.4% to 93.1%, minimizes destructive edits, and is preferred by experts in 54% of disagreement cases. Results suggest that many failures in LLM-generated remediation stem from unmet operational constraints rather than prompting alone. Broadly, SVRP demonstrates how solver-backed decision mechanisms can complement LLMs in safety-critical planning while retaining human approval.

Limitations

SVRP has several limitations. First, it relies on a fixed action vocabulary and predefined constraint set. While this enables rigorous feasibility verification, it limits coverage for incidents requiring novel or specialized remediation; SVRP does not invent new actions and may abstain when proposals are incomplete.

Second, risk modeling is intentionally coarse. Action-level risk labels and severity-conditioned budgets abstract contextual nuances (e.g., partial rollbacks or scoped deployments), which can block feasible plans in rare cases.

Third, the SAT result certifies consistency only with the 47-action vocabulary and encoded role, risk, dependency, and verification rules. It does not cover omitted topology, maintenance-window, regulatory, causal-sufficiency, or blast-radius constraints. The normalization audit in Appendix Table 7 quantifies this model boundary; organizations must validate and maintain their own policy before deployment.

Finally, solver-based verification introduces computational overhead, though empirical solve times remain well within practical limits for offline postmortem analysis.

SVRP is intended as decision support rather than autonomous execution; human review remains nec-

essary before adopting remediation plans, especially when incident context or operational constraints are incomplete.

Future work includes richer action vocabularies, hierarchical risk models, and hybrid symbolic-learned constraint extraction.

Ethical Considerations

SVRP is decision support, not autonomous execution. Operators see the proposed plan, violated constraints, repair, and abstention status; every emitted plan requires human approval, and unresolved cases return to manual review. This is important because encoded-policy compliance can create automation bias despite omitted context. Real incident narratives remained access-restricted; identifiers and sensitive details were removed before processing, and only deidentified text was sent to the external LLM provider. We report aggregate results and release only synthetic incidents. Deployment should include access controls, immutable decision logs, coverage monitoring, and periodic expert review of the action vocabulary and policy mappings.

References

- Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending root-cause and mitigation steps for cloud incidents using large language models. In *Proceedings of the 2023 IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 1737–1749.
- Pengfei Chen, Yong Qi, and Di Hou. 2019. [Causeinfer: Automated end-to-end performance diagnosis with hierarchical causality graph in cloud environment](#). *IEEE Transactions on Services Computing*, 12(2):214–230.
- Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. 2024. [Automatic root cause analysis via large language models for cloud incidents](#). In *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys '24*, page 674–688, New York, NY, USA. Association for Computing Machinery.
- Yilun Hao, Yongchao Chen, Yang Zhang, and Chuchu Fan. 2025. [Large language models can solve real-world planning rigorously with formal verification tools](#). In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language*

- Technologies (Volume 1: Long Papers)*, pages 3434–3483, Albuquerque, New Mexico. Association for Computational Linguistics.
- Or Honovich, Roei Aharoni, Jonathan Herzig, Hagai Taitelbaum, Doron Kukliansky, Vered Cohen, Thomas Scialom, Idan Szpektor, Avinatan Hassidim, and Yossi Matias. 2022. [TRUE: Re-evaluating factual consistency evaluation](#). In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3905–3920, Seattle, United States. Association for Computational Linguistics.
- Junjie Huang, Jinyang Liu, Zhuangbin Chen, Zhihan Jiang, Yichen Li, Jiazhen Gu, Cong Feng, Zengyin Yang, Yongqiang Yang, and Michael R. Lyu. 2024. [Faultprofit: Hierarchical fault profiling of incident tickets in large-scale cloud systems](#). In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP '24*, page 392–404, New York, NY, USA. Association for Computing Machinery.
- Christine P. Lee, David Porfirio, Xinyu Jessica Wang, Kevin Chenkai Zhao, and Bilge Mutlu. 2025. [Veriplan: Integrating formal verification and llms into end-user planning](#). In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, CHI '25*, New York, NY, USA. Association for Computing Machinery.
- Xihuan Lin, Jie Zhang, Gelei Deng, Tianzhe Liu, Tianwei Zhang, Qing Guo, and Riqing Chen. 2025. [Ircopilot: Automated incident response with large language models](#). *Preprint*, arXiv:2505.20945.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: iterative refinement with self-feedback. In *Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23*, Red Hook, NY, USA. Curran Associates Inc.
- Potsawee Manakul, Adian Liusie, and Mark Gales. 2023. [SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9004–9017, Singapore. Association for Computational Linguistics.
- Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2025. [Benchmarking agentic workflow generation](#). *Preprint*, arXiv:2410.07869.
- Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. [SQuAD: 100,000+ questions for machine comprehension of text](#). In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23*, Red Hook, NY, USA. Curran Associates Inc.
- Weizhou Shen, Yingqi Gao, Canbin Huang, Fanqi Wan, Xiaojun Quan, and Wei Bi. 2023. [Retrieval-generation alignment for end-to-end task-oriented dialogue system](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8261–8275, Singapore. Association for Computational Linguistics.
- Zeyu Wang. 2024. [CausalBench: A comprehensive benchmark for evaluating causal reasoning capabilities of large language models](#). In *Proceedings of the 10th SIGHAN Workshop on Chinese Language Processing (SIGHAN-10)*, pages 143–151, Bangkok, Thailand. Association for Computational Linguistics.
- Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. 2009. [Automatically finding patches using genetic programming](#). In *Proceedings of the 31st International Conference on Software Engineering, ICSE '09*, page 364–374, USA. IEEE Computer Society.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: deliberate problem solving with large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23*, Red Hook, NY, USA. Curran Associates Inc.
- Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2023. Satlm: satisfiability-aided language models using declarative prompting. In *Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23*, Red Hook, NY, USA. Curran Associates Inc.
- Xuchao Zhang, Supriyo Ghosh, Chetan Bansal, Rujia Wang, Minghua Ma, Yu Kang, and Saravan Rajmohan. 2024. [Automated root causing of cloud incidents using in-context learning with gpt-4](#). In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, FSE 2024*, page 266–277, New York, NY, USA. Association for Computing Machinery.
- Ruochen Zhao, Xingxuan Li, Shafiq Joty, Chengwei Qin, and Lidong Bing. 2023. [Verify-and-edit: A knowledge-enhanced chain-of-thought framework](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume*

A Appendix

A.1 Deployment and Operational Characteristics

SVRP was piloted within our enterprise ticketing workflow beginning May 2025, automatically generating remediation plans for approximately 200 Severity-1/2 incidents per month during evaluation.

End-to-end processing time remained under three minutes, including proposal, verification, and repair. During the pilot, 56% of plans were adopted unchanged, 29% after edits, 10% were rejected, and 5% were system abstentions. Among solver-SAT plans, 1.9% were rejected because of omitted operational context, illustrating the boundary of encoded-policy compliance. Abstentions and rejections returned to manual CAPM review.

In the offline evaluation, 6.9% remained non-compliant after bounded repair (7.1% of Severity-1 and 5.6% of Severity-2 cases) and were likewise routed to manual review rather than labeled feasible. The separate 4.8% expert false-acceptance audit (Appendix A.5.5) measures omitted context among solver-SAT plans and is not included in SAT.

Operational monitoring tracked compliance rates and validator outputs to detect drift. During the pilot, five vocabulary updates and two policy-mapping updates required approximately three expert-hours per month. These figures measure ongoing maintenance; initial policy setup for a new organization was not measured, and we do not claim zero-shot transfer.

A.2 Implementation Details of Solver-Verified Remediation Planning

This appendix provides implementation-level details for each component of SVRP, corresponding directly to the methodology described in Section 3. All components described here are deterministic unless explicitly stated otherwise.

A.2.1 Action Proposal and Normalization

SVRP uses large language models (LLMs) only to propose candidate remediation actions. The proposal stage is intentionally permissive and does not enforce feasibility, safety, or executability constraints.

LLM Generators. Unless otherwise stated, candidate actions are generated using **Cohere Command-R+**, accessed via API with fixed decoding parameters: temperature $T = 0.2$, top- $p = 0.75$, and maximum output length of 512 tokens. These settings favor factual consistency and conservative action proposals and are identical to those used in our prior PAG system.

Identifiers and sensitive operational fields were removed internally before model access; only de-identified incident text was sent to Cohere, while raw records remained in the access-restricted enterprise environment.

For experiments involving a fine-tuned generator baseline (Section 4.1), we use **Mixtral-8×22B-Instruct**, fine-tuned via parameter-efficient LoRA adaptation on incident→CAPM pairs. This model directly generates remediation plans end-to-end and does not use retrieval or feasibility verification.

Prompting and Output Format. LLMs are prompted to generate a list of corrective and preventive actions given: (i) the incident summary and (ii) a small number of retrieved historical postmortems for contextual grounding. The prompt requests concise, imperative action statements but imposes no structural constraints. No prompts instruct the model to reason about feasibility, risk budgets, or dependencies.

Candidate Set Size. Each incident yields between 6–12 candidate actions (median 9).

The number of proposed actions is capped at 15 as a fixed guardrail on proposal length, normalization cost, and worst-case end-to-end latency, not because of MiniSat capacity. This cap is fixed across all experiments and baselines. Median/p95 SAT latency is 4.1/19.6 ms; fewer than 0.5% of instances reach the 200 ms timeout.

Normalization. LLM outputs are converted into structured action representations using deterministic post-processing. Each action is normalized into a tuple:

$$a_i = (\text{type}, \text{role}, \text{scope}, \text{risk}, \text{deps})$$

Parsing Pipeline. We apply a rule-based normalization pipeline consisting of: (i) sentence segmentation, (ii) keyword-based action type matching, (iii) role extraction using a fixed role lexicon (e.g., SRE, on-call engineer, service owner), (iv) scope normalization using service identifiers, and (v) de-

Table 7: Double-annotated normalization audit (500 actions). The 2.1% action-level parse-failure rate is distinct from the corpus-wide 2.3% proposal-level rate.

Audit metric	Result (%)
Mapped / OOV or ambiguous / parse failure	93.1 / 4.8 / 2.1
Type / role / scope accuracy	95.3 / 96.1 / 90.6
Dependency F1 / tuple exact match	88.2 / 87.3
Gold-vs.-parsed SAT disagreement	4.6

pendency detection via trigger phrases such as “after”, “requires”, and “once deployed”.

Actions that cannot be parsed into a valid tuple are discarded. Across the full dataset, parse failures occurred in 2.3% of proposals.

Action Vocabulary. Action types are drawn from a fixed vocabulary of 47 remediation primitives (e.g., add_monitoring, rollback, increase_quota, update_config, restart_service). This vocabulary was derived from historical postmortems and fixed prior to evaluation.

Separation from Verification. All normalization, parsing, and action vocabulary mappings are implemented as deterministic code. LLMs are not used to assign risk labels, detect dependencies, validate permissions, or enforce feasibility. These decisions are deferred entirely to solver-based verification (Section A.2.4).

A.2.2 Data Separation and Governance

The Mixtral LoRA corpus is a separate historical pool of 8,200 incident-CAPM pairs, split 6,560/820/820 by incident; none overlaps the 8,200 evaluation incidents. Normalized-text MinHash removes near duplicates at Jaccard similarity > 0.85 . Retrieval is restricted to records preceding the target incident and cannot retrieve the target postmortem or CAPM. Confidential narratives cannot be released; the public artifact contains 950 synthetic incidents (505 Severity-1 and 445 Severity-2) across nine service categories and 7,589 CAPM actions, together with the schema and prompts.

A.2.3 Constraint Definitions and Exact Encoding

SVRP enforces a fixed, deterministic set of feasibility constraints. All constraints are applied uniformly across datasets, services, and baselines and are independent of the LLM used for action proposal.

Table 8: Action-to-role permission mapping (fixed across all experiments).

Action Type	Permitted Roles
add_monitoring	{SRE, ServiceOwner}
update_config	{SRE, ServiceOwner}
rollback	{SRE}
restart_service	{SRE, OnCall}
increase_quota	{ServiceOwner, Manager}
deploy_fix	{Developer, ServiceOwner}
approve_deployment	{Manager}
security_audit	{Security}
post_incident_review	{Manager, SRE}

Table 9: Action-to-risk mapping.

Risk Level	Action Types
Low	add_monitoring, alert_tuning, logging
Medium	update_config, restart_service, increase_quota
High	deploy_fix, rollback, data_migration

Role Universe. We define a closed set of operational roles:

$$\mathcal{R} = \{\text{SRE, ServiceOwner, Developer, Manager, Security, OnCall}\}$$

No other roles are permitted in any plan.

Role Constraints (Exact Mapping). Each action type is associated with a fixed, predefined set of permitted roles. Let $\text{role}(a_i)$ denote the assigned role of action a_i . An action is valid only if:

$$\text{role}(a_i) \in \mathcal{R}_{\text{type}(a_i)}$$

Table 8 shows the complete action-to-role mapping used in all experiments.

Assignments outside the permitted role set are forbidden and encoded as hard SAT clauses.

Risk Constraints. Each action type is deterministically mapped to a risk level:

$$\text{risk}(a_i) \in \{\text{low, medium, high}\}$$

Risk levels are fixed and defined by action semantics, not learned. The complete mapping is shown in Table 9.

Incident severity determines the maximum allowable number of high-risk actions:

$$\sum_{a_i \in A} \mathbb{I}[\text{risk}(a_i) = \text{high}] \leq B(s)$$

where $B(s)$ is a fixed budget:

$$B(s) = \begin{cases} 0 & \text{low severity} \\ 1 & \text{medium severity} \\ 2 & \text{high severity} \end{cases}$$

These constraints are encoded as cardinality constraints in SAT using sequential-counter encodings.

Dependency Constraints. Dependencies are represented as directed prerequisites. If action a_i depends on a_j , then:

$$z_i \rightarrow z_j$$

which enforces prerequisite inclusion. Plans also retain sequence positions, and a deterministic validator enforces execution precedence:

$$\text{pos}(a_j) < \text{pos}(a_i).$$

This precedence check is separate from the CNF implication; the implication alone does not encode temporal order. Dependencies are extracted deterministically during normalization using fixed trigger patterns (e.g., “after deployment”, “requires rollout”).

Verification Constraints. Each plan must include at least one verification action:

$$\sum_{a_i \in A} \mathbb{I}[a_i \in \mathcal{V}] \geq 1$$

where:

$$\mathcal{V} = \{\text{add_monitoring}, \text{validation_check}, \text{canary_test}, \text{rollback_plan}\}$$

Plans violating this constraint are infeasible.

Summary. All constraints are: (i) explicitly defined, (ii) deterministic, (iii) fixed prior to evaluation, (iv) uniformly applied across all datasets and baselines, and (v) enforced solely via solver logic rather than learned models.

The schema and mappings were independently reviewed by expert SREs and technical program managers; this validates policy fidelity, not completeness under unmodeled context.

A.2.4 SAT Encoding and Solver Implementation

SVRP formulates feasibility verification as a propositional satisfiability (SAT) problem. All encodings are deterministic and symbolic; no learned models or neural components are used for constraint checking, diagnosis, or repair.

Variables. Each candidate action a_i is associated with a Boolean variable $z_i \in \{0, 1\}$, where $z_i = 1$ indicates that action a_i is included in the final remediation plan. No additional latent variables are introduced beyond selector and cardinality-encoding variables. During the initial check, every proposed action is passed as a positive assumption literal z_i ; therefore, the solver cannot obtain SAT by silently dropping an action. No language models, embeddings, or learned predictors are involved at this stage.

Constraint Encoding. All feasibility constraints are compiled into conjunctive normal form (CNF) as follows:

- **Role Constraints.** Each action type is associated with a fixed set of permitted roles (SRE, ServiceOwner, Manager, Security, Developer and OnCall). For any action a_i whose extracted role is not in its permitted role set, a hard clause $\neg z_i$ is added, forbidding inclusion of that action.
- **Risk Constraints.** Actions are partitioned into disjoint risk classes $\{\mathcal{A}_{\text{low}}, \mathcal{A}_{\text{medium}}, \mathcal{A}_{\text{high}}\}$ using a fixed mapping from action type to risk level. For each incident severity level, a maximum allowable count k of high-risk actions is specified. The constraint $\sum_{a_i \in \mathcal{A}_{\text{high}}} z_i \leq k$ is encoded using a sequential-counter cardinality encoding and translated into CNF.
- **Dependency Constraints.** For each dependency relation $a_i \rightarrow a_j$ (meaning a_i requires a_j to be executed first), an implication clause $(\neg z_i \vee z_j)$ is added. This clause enforces inclusion of a_j , not precedence. A deterministic sequence validator separately checks $\text{pos}(a_j) < \text{pos}(a_i)$ and attaches any ordering violation to the same diagnostic and repair workflow.
- **Verification Constraints.** A fixed set $\mathcal{V}$ of verification actions (e.g., add_monitoring, add_alert, run_validation) is defined. To ensure that every plan includes at least one verification step, a single disjunctive clause $\bigvee_{a_i \in \mathcal{V}} z_i$ is added.

All constraints are hard constraints; no soft clauses or optimization objectives are used.

Solving and Unsatisfiable Cores. We use MiniSat 2.2 under selector assumptions. For each incident, the solver checks the constructed CNF while assuming all currently selected action literals are true.

- If the instance is satisfiable, the selected remediation plan is compliant with the encoded constraints; unmodeled operational safety remains outside this guarantee.
- If unsatisfiable, MiniSat returns a conflict subset over selector assumptions. We apply deterministic deletion-based shrinking until no remaining selector can be removed while preserving UNSAT. The result is inclusion-minimal, not guaranteed to have minimum cardinality.

The solver is invoked with a per-instance timeout of 200 ms. Across all experiments, fewer than 0.5% of instances exceed the timeout and are conservatively treated as infeasible. Median/p95 SAT latency is 4.1/19.6 ms, and median/p95 diagnostic-repair latency is 23.6/88.3 ms. SAT encoding, solver calls, conflict shrinking, sequence validation, and repair are deterministic; no language models, embeddings, or learned predictors are used in these stages.

A.2.5 Unsatisfiable-Core-Guided Repair

SVRP repairs infeasible remediation plans using unsatisfiable cores returned by the SAT solver. All repair operations in SVRP are performed by a deterministic repair module implemented as program logic.

Unsatisfiable Core Representation. For an infeasible plan, MiniSat’s assumption conflict is deletion-shrunk to a small, inclusion-minimal diagnostic core

$$C = \{c_1, \dots, c_m\},$$

where each element c_k corresponds to either: (i) an action inclusion variable z_i , or (ii) a constraint clause involving one or more actions. Each core element is traceable to the originating action(s) via solver bookkeeping.

Repair Candidates. From the unsatisfiable core C , we derive a candidate action set

$$A_C = \{a_i \mid z_i \in C \text{ or } a_i \text{ appears in a violated clause in } C\}.$$

Only actions in A_C are considered for modification; all other actions are left unchanged.

Repair Operations: Repairs are applied using a fixed, ordered set of operators designed to preserve plan semantics:

1. **Prerequisite Insertion.** If C contains a violated dependency constraint of the form $a_i \rightarrow a_j$ with $z_i = 1$ and $z_j = 0$, the prerequisite action a_j is inserted into the plan if it exists in the original proposal set.
2. **Ordering Enforcement.** If C indicates an ordering violation between two included actions, the plan order is adjusted to satisfy the required precedence without removing actions.
3. **Risk-Level Substitution.** If C includes a risk-budget violation involving a high-risk action a_i , SVRP substitutes a_i with a lower-risk action of the same action type (same type and scope) drawn from the original proposal set, if available.
4. **Action Removal.** If none of the above repairs resolve infeasibility, SVRP greedily removes a deterministically prioritized action from A_C to break the core. Actions outside A_C are never removed.

Iteration and Termination. After applying a repair, the modified plan is re-encoded and re-submitted to the solver. SVRP allows at most two repair iterations per incident. If the instance remains unsatisfiable after the second iteration, the system abstains.

Determinism and Guarantees. Repair operator order, substitution rules, and removal policies are fixed and deterministic. SVRP never introduces new actions beyond the original proposal set, and never deletes actions unless prerequisite insertion, reordering, and substitution all fail.

A.3 Synthetic Dataset Construction

The released synthetic dataset ¹ of 950 incidents is constructed by injecting controlled constraint conflicts (e.g., incompatible actions, missing prerequisites) into otherwise valid plans. This enables systematic evaluation of interaction-level feasibility failures that are rare in organic postmortem data.

¹https://github.com/ppattnayak/PAG_DB

A.4 LLM Prompts for Action Proposal, Validation and Constrained Baselines

Prompts are fixed across datasets and experiments.

Action Proposal Prompt. We prompt the LLM to generate candidate corrective and preventive actions given an incident summary and retrieved historical context. The prompt requests concise, imperative actions without enforcing feasibility, safety, or role constraints.

The same action proposal prompt is used for: (i) the generation-only baseline, (ii) SVRP action proposal, and (iii) the rule-based and solver-based pipelines prior to verification.

Exact prompt is shown in Figure 3.

LLM-Based Validation Prompt. For the LLM-based validation baseline, the same Cohere Command-R+ model is prompted to revise a proposed remediation plan to remove unsafe or invalid actions.

The prompt instructs the model to improve safety and executability without providing explicit constraint definitions, solver diagnostics, or feasibility feedback.

Validation prompt is shown in Figure 4.

Constrained Baseline Prompts. The constrained baselines use complete prompts with the same message structure and decoding settings as their unconstrained counterparts. The constrained generation prompt mirrors the action-proposal prompt but includes explicit operational constraints (Figure 5). The constrained rewrite prompt mirrors the validation/rewrite prompt and provides the same constraints alongside the generated plan (Figure 6). Neither baseline receives solver diagnostics, unsatisfiable cores, or SVRP repair outputs.

A.5 Additional Metrics & Results

A.5.1 Full Safety Violation Breakdown

A full breakdown by violation type (risk, permission, dependency, verification) is provided in Table 10 and visualized in Figure 7.

A.5.2 Embedding Choice Robustness

Results are stable across embedding models (MiniLM vs. MPNet) as shown in Table 11.

A.5.3 Repair-SAT Rate and Unsat Core Size

SVRP successfully repairs over 80% of initially infeasible plans, with small unsatisfiable cores in-

dicating localized conflicts as shown in Table 12.

A.5.4 Checker and Repair-Cap Sensitivity

We replace MiniSat/core diagnosis with direct enumeration of violated constraints while holding the 8,200 proposals, normalized tuples, repair operators and order, two-iteration cap, and evaluator fixed. Table 13 shows that direct global checking recovers most of the compliance gain, while diagnostic cores improve repair localization and preservation. We therefore claim value for explicit global enforcement and diagnostic repair, not that MiniSat is the only suitable backend. The cap sensitivity also shows diminishing returns beyond the prefixed two iterations; every candidate is rechecked, so the cap changes coverage rather than acceptance criteria.

A.5.5 Human Evaluation of Disagreements

We define a disagreement case as an incident for which SVRP produces a remediation plan that differs from at least one competing decision mechanism (generation-only, LLM-based validation, or constrained LLM rewriting), after normalization. Among solver-SAT plans audited in this study, experts identified 4.8% as false acceptances because the encoded model omitted relevant operational context. This is an independent estimate of the model boundary, not part of the SAT metric.

A.5.6 Surface-Level Proposal Quality

Table 17 reports surface-level proposal-quality metrics computed before solver verification and repair. Motivated by prior work on factual-consistency evaluation and hallucination detection (Honovich et al., 2022; Manakul et al., 2023), we report two task-specific measures: **Factual Consistency (FCS; 0–1)** measures agreement with the incident context and retrieved precedents, while **Groundedness (%)** is the percentage of CAPM statements supported by this evidence. Inspired by work on answerability and causal-reasoning evaluation (Rajpurkar et al., 2016; Wang, 2024), we additionally report **Answerability (1–5)**, rating completeness and plausibility of CAPMs, and **Causal Coverage (CCov; 0–1)**, measuring coverage of key root causes addressed by CAPMs.

As expected, proposal quality is broadly consistent across service domains and comparable to prior work. Importantly, SVRP does not modify the proposal stage; these results confirm that the feasibility and safety gains reported in Results Section are not driven by changes in generation quality.

```

1  ### System Prompt
2  You are an experienced cloud reliability engineer writing postmortem remediation actions.
3  Your task is to propose concrete corrective and preventive actions based strictly on the
   incident description and historical context.
4
5  Do NOT evaluate feasibility, risk limits, permissions, or dependencies.
6  Do NOT attempt to validate or reject actions.
7  Only propose actions that could plausibly help prevent recurrence.
8
9  ---
10
11 ### User Prompt
12 Incident Summary:
13 {{incident_summary}}
14
15 Related Historical Incidents:
16 {{retrieved_incidents}}
17
18 Task:
19 Generate a concise list of corrective and preventive remediation actions.
20
21 Guidelines:
22 - Write actions as short, imperative statements.
23 - Each action should describe a single concrete step.
24 - Do not include explanations or justification.
25 - Do not merge multiple actions into one sentence.
26 - Do not reason about safety, permissions, or feasibility.
27
28 Output Format:
29 - A bulleted list of actions.
30 - No prose before or after the list.
31

```

Figure 3: Action proposal prompt used for all generation-based methods.

A.5.7 Synthetic Stress-Test Evaluation

We additionally evaluate SVRP on a released synthetic dataset of 950 incidents designed to surface interaction-level feasibility violations that are rare in organic data (e.g., conflicting role assignments, incompatible action dependencies, and risk-budget overflows). Consistent with results on real incidents, generation-only and heuristic baselines struggle on the synthetic dataset, which contains a higher density of multi-constraint conflicts. SVRP maintains high feasibility and non-deletion rates, indicating that solver-verified repair generalizes to adversarial interaction patterns rather than exploiting dataset-specific regularities as shown in Table 18. Comparison with full dataset is shown in Fig 8. SVRP maintains comparable feasibility and repair behavior under synthetic stress, indicating robustness to denser interaction-level constraint conflicts.

A.6 Error Analysis and Failure Modes

Refer Table 19 for failure modes observed and definitions.

Missing Prerequisite Proposals. In these cases, the LLM fails to propose a required prerequisite action (e.g., deployment rollback before configuration change). Because SVRP does not introduce novel actions beyond the original proposal set, no feasible repair exists, resulting in abstention. This reflects a proposal-stage limitation rather than a verification failure.

Unavailable Risk Substitution. When no lower-risk alternative in the original proposal set satisfies the encoded budget, SVRP abstains rather than inventing an action or forcing a destructive deletion.

Missing or Unparseable Verification Steps. Some proposed verification text is too underspecified to normalize to a primitive in $\mathcal{V}$, leaving the

```

1  ### System Prompt
2  You are reviewing a remediation plan written by another engineer.
3  Your task is to improve the plan by removing or revising actions that appear unsafe,
   unrealistic, or invalid.
4
5  You do not have access to formal rules, constraints, or solver diagnostics.
6  Use your best judgment based on general engineering intuition.
7
8  ---
9
10 ### User Prompt
11 Incident Summary:
12 {{incident_summary}}
13
14 Proposed Remediation Plan:
15 {{generated_actions}}
16
17 Task:
18 Revise the remediation plan to remove or rewrite actions that seem unsafe, infeasible, or
   inappropriate.
19
20 Guidelines:
21 - You may delete actions you believe should not be executed.
22 - You may rewrite actions to make them safer or more realistic.
23 - Do NOT add new actions that were not already present.
24 - Do NOT explain your reasoning.
25 - Do NOT refer to policies, budgets, or formal constraints.
26
27 Output Format:
28 - A revised bulleted list of actions only.
29 - No commentary or justification.

```

Figure 4: Prompt used for LLM-based validation and rewriting baseline.

verification constraint unsatisfied after bounded repair. Separately, the expert false-acceptance audit captures solver-SAT plans that satisfy the formal rule but remain incomplete under omitted context (Appendix A.5.5).

Solver-Level Failures. A small fraction of failures arise from solver timeouts or ambiguous unsatisfiable cores. These account for fewer than 1% of all incidents and are treated conservatively as infeasible.

```

1  ### System Prompt
2  You are an experienced cloud reliability engineer writing postmortem
3  remediation actions.
4  Your task is to propose concrete corrective and preventive actions
5  based strictly on the incident description, historical context, and
6  operational constraints.
7
8  You must follow the provided constraints. Do NOT use solver output,
9  unsat cores, or external tools.
10
11 ---
12
13 ### User Prompt
14 Incident Summary:
15 {{incident_summary}}
16
17 Related Historical Incidents:
18 {{retrieved_incidents}}
19
20 Operational Constraints:
21 - Allowed actions: {{action_vocabulary}}
22 - Role permissions: {{role_permission_map}}
23 - Risk budgets: {{severity_risk_budget}}
24 - Dependencies: {{dependency_rules}}
25 - Verification actions: {{verification_actions}}
26
27 Task:
28 Generate a concise list of corrective and preventive remediation actions.
29
30 Guidelines:
31 - Use only allowed action types.
32 - Assign each action to a permitted role.
33 - Do not exceed the risk budget.
34 - Include required prerequisites before dependent actions.
35 - Include at least one verification action.
36 - Do not explain the constraints or your reasoning.
37
38 Output Format:
39 - A bulleted list of actions.
40 - No prose before or after the list.

```

Figure 5: Full prompt used for the constrained LLM generation baseline.

Table 10: Breakdown of safety and executability violations prior to verification. Lower is better. Values are percentages of plans exhibiting at least one violation.

Method	Risk Viol.	Permission Viol.	Dependency Viol.	Missing Verification
Gen-only	18.9	14.3	21.7	26.4
Rule-based	6.4	5.9	13.2	11.8
LLM validation	8.7	7.1	15.6	14.2
Constrained LLM generation	5.9	4.8	12.7	9.6
Constrained LLM rewrite	4.9	3.9	10.7	8.1
SVRP (Ours)	1.8	2.3	4.9	3.7

```

1  ### System Prompt
2  You are reviewing a remediation plan written by another engineer.
3  Your task is to revise the plan so it follows the operational
4  constraints while preserving the original remediation intent.
5
6  You must follow the provided constraints. Do NOT use solver output,
7  unsat cores, or external tools.
8
9  ---
10
11 ### User Prompt
12 Incident Summary:
13 {{incident_summary}}
14
15 Proposed Remediation Plan:
16 {{generated_actions}}
17
18 Operational Constraints:
19 - Allowed actions: {{action_vocabulary}}
20 - Role permissions: {{role_permission_map}}
21 - Risk budgets: {{severity_risk_budget}}
22 - Dependencies: {{dependency_rules}}
23 - Verification actions: {{verification_actions}}
24
25 Task:
26 Revise the remediation plan to remove or rewrite actions
27 that violate the constraints.
28
29 Guidelines:
30 - Use only allowed action types.
31 - Assign each action to a permitted role.
32 - Do not exceed the risk budget.
33 - Include required prerequisites before dependent actions.
34 - Include at least one verification action.
35 - Prefer minimal edits to the original plan.
36 - Do NOT add unrelated new actions.
37 - Do NOT explain your reasoning.
38
39 Output Format:
40 - A revised bulleted list of actions only.
41 - No commentary or justification.

```

Figure 6: Full prompt used for the constrained LLM rewriting baseline.

Table 11: Effect of embedding model choice on repair behavior metrics. Rankings are preserved across embedding models.

Embedding	Method	Avg. Edits	Non-Del. (%)	Semantic Ret.
MiniLM-L6-v2	Rule-based	2.6	9.4	0.71
	LLM rewrite	3.1	54.2	0.78
	SVRP	1.4	82.7	0.89
MPNet-base-v2	Rule-based	2.6	9.4	0.73
	LLM rewrite	3.1	54.2	0.80
	SVRP	1.4	82.7	0.91

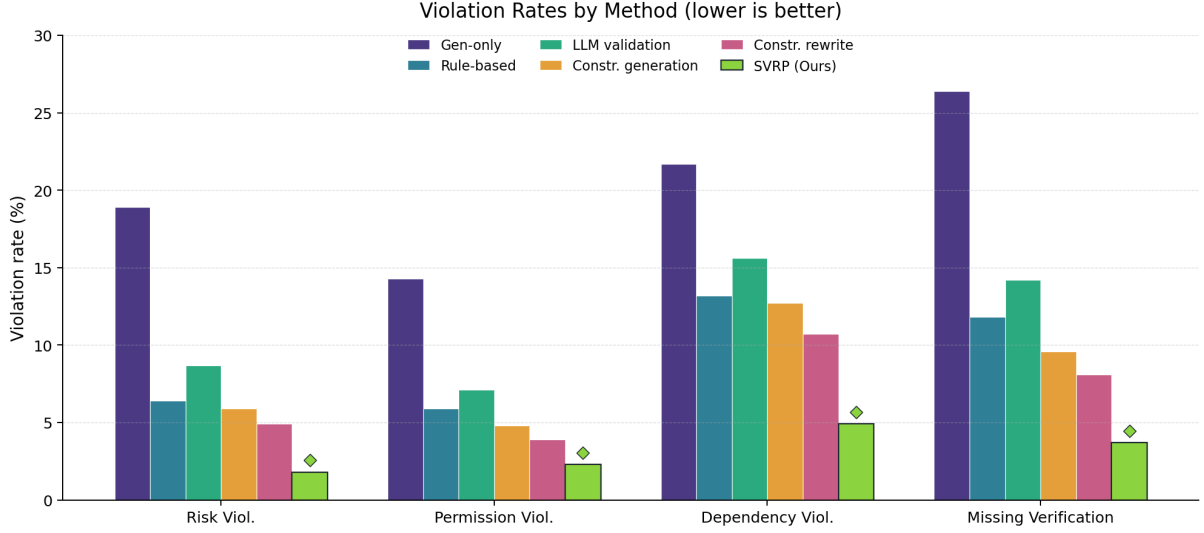


Figure 7: Violation rates by method across risk, permission, dependency, and missing-verification constraints. Lower is better. Diamond markers denote SVRP.

Table 12: Solver diagnostics and repair effectiveness on initially infeasible plans.

Method	Repair-SAT (%)	Final UNSAT (%)	Avg. Unsat Core Size
Rule-based	48.1	51.9	—
LLM validation	39.6	60.4	—
SVRP (Ours)	81.7	18.3	2.4

Table 13: Additional checks on the private evaluation set. All variants use the same inputs, policy, repair operators, and evaluator.

(a) Global-checker ablation ($n = 8,200$)				(b) Repair-iteration sensitivity			
Checker / diagnosis	Final SAT	Non-del.	Edits	Cap	Final SAT	Repair-SAT	Non-del.
MiniSat + shrunk core	93.1	82.7	1.4	1	90.3	74.2	84.0
Direct global, no core	90.8	79.1	1.8	2	93.1	81.7	82.7
				3	93.3	82.1	82.5

Table 14: Human evaluation dataset statistics and inter-annotator agreement.

Statistic	Value
Total incidents	8,200
Disagreement pool size	2,180
Disagreement rate (%)	26.6
Sampled for human eval	300
Sampling fraction (%)	13.8
Annotators	2 (Senior SREs)
Avg. annotations per plan	2.0
Cohen’s κ (overall)	0.69

Table 15: Inter-annotator agreement by evaluation dimension.

Metric	Cohen’s κ
Technical Correctness	0.71
Executability	0.68
Adoption Likelihood	0.66
Overall Preference	0.69

Table 16: Human evaluation scores with 95% confidence intervals (bootstrap).

Method	Executability	Correctness
Gen-only	3.7 ± 0.11	3.9 ± 0.10
LLM validation	4.0 ± 0.09	4.1 ± 0.08
Const. LLM rewrite	4.2 ± 0.07	4.3 ± 0.07
SVRP (Ours)	4.7 ± 0.07	4.6 ± 0.06

Table 17: Surface-level proposal quality metrics across service categories. These metrics characterize proposal quality only and do not reflect feasibility or executability.

Service Category	Incidents	BLEU-4	BERTScore	Grounded (%)	FCS	CCov	Ans.	Synthetic Dataset
Compute & Infrastructure	1404	28.7	0.86	87.0	0.85	0.77	4.4	105
Database Services	895	27.9	0.84	85.6	0.83	0.75	4.2	108
AI & ML Services	1105	27.4	0.83	84.3	0.82	0.73	4.1	99
API & Identity Services	997	27.6	0.84	85.0	0.83	0.74	4.2	118
Networking Services	851	27.1	0.83	84.1	0.82	0.73	4.1	105
Storage Services	1101	28.2	0.85	86.2	0.84	0.76	4.3	102
Logging	903	27.0	0.82	83.6	0.81	0.72	4.0	114
Artifacts & Key Mgmt	499	26.5	0.81	82.8	0.80	0.70	3.9	87
Marketplace	445	25.9	0.80	81.2	0.78	0.68	3.8	112
Overall (Weighted)	8,200	27.6	0.84	85.3	0.83	0.74	4.2	950

Table 18: Feasibility results on the synthetic stress-test dataset (950 incidents).

Method	SAT (%)	Repair-SAT (%)	UNSAT (%)	Non-Deletion (%)
Gen-only	58.2	–	41.8	100.0
Rule-based	79.6	–	20.4	11.3
LLM validation	74.1	38.0	25.9	52.8
SVRP (Ours)	91.4	79.4	8.6	80.9

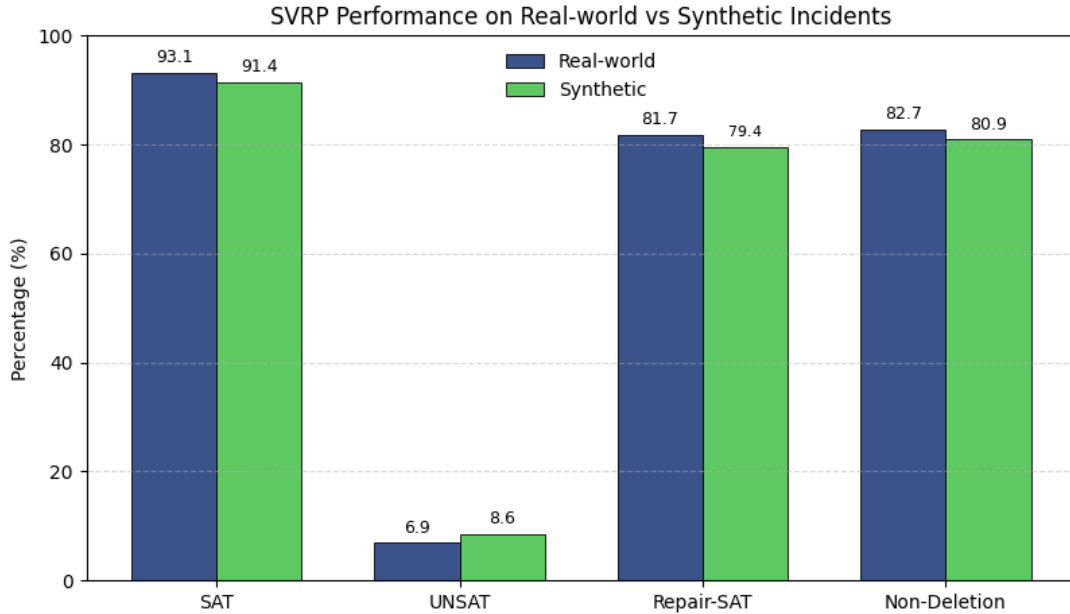


Figure 8: Comparison of SVRP performance on real-world (8,200) and synthetic (950) incidents. Bars indicate percentages; numeric labels show exact values.

Table 19: Breakdown of SVRP failure modes across all evaluated incidents.

Failure Category	Fraction (%)	Description
Missing prerequisite proposal	3.1	Required action absent from initial proposal set
No admissible risk substitution	1.9	No proposed alternative satisfies the encoded risk budget
Missing/unparseable verification step	1.3	No proposed verification maps to an accepted primitive
Solver timeout / ambiguous core	0.6	SAT instance exceeds timeout or yields non-actionable core
Total	6.9	Post-repair UNSAT; routed to manual review