The Eastasouth Journal of Information System and Computer Science
Vol. 2, No. 02, December 2024, pp. 179-188
ISSN: 3025-566X, DOI: 10.58812/esiscs.v2i02

LLM-Based Autonomous Remediation for DevSecOps Pipelines

Roshan Kakarla
DevOps Engineer, Information Technology, Indiana Wesleyan University

Article Info

Article history:
Received Dec, 2024
Revised Dec, 2024
Accepted Dec, 2024

Keywords:
AI Safety;
Autonomous Remediation;
Cloud Governance;
DevSecOps;
Infrastructure as Code;
Large Language Models;
Platform Engineering;
Site Reliability Engineering

ABSTRACT

Modern DevSecOps pipelines operate at a scale and velocity that exceeds the cognitive and operational capacity of traditional rule-based automation and human-centric incident response. While monitoring, alerting, and security scanning tools have matured, remediation remains largely manual, fragmented, and reactive resulting in prolonged mean time to resolution (MTTR), configuration drift, and governance gaps. This paper proposes a novel LLM-Based Autonomous Remediation Framework (LLM-ARF) that introduces a risk-aware, policy-governed control plane for automated detection, diagnosis, and remediation across DevSecOps pipelines. Unlike existing approaches that rely on static runbooks or narrow AI classifiers, LLM-ARF integrates large language models as reasoning agents embedded within a constrained, auditable, and human-supervised execution loop. The framework explicitly separates cognition, decision authority, and actuation, enabling scalable autonomy while preserving accountability and compliance. We present the architectural design, lifecycle control flow, and governance mechanisms of LLM-ARF, and evaluate its operational impact using real-world DevOps metrics such as MTTR reduction, alert fatigue mitigation, and toil reduction. The results demonstrate that LLM-ARF enables a step-function improvement in remediation reliability without compromising safety or human oversight, positioning autonomous remediation as a viable next evolution of enterprise DevSecOps systems.

Corresponding Author:
Name: Roshan Kakarla
Institution: DevOps Engineer, Information Technology, Indiana Wesleyan University
Email: rkakarla1041@gmail.com

1. INTRODUCTION

DevSecOps has transformed how software systems are built, deployed, and operated, emphasizing continuous delivery, security-by-design, and infrastructure automation. However, as organizations scale to thousands of microservices, multi-cloud deployments, and highly regulated environments, a critical imbalance has emerged: detection and observability have scaled, but remediation has not. Alerts, security findings, and configuration drifts are surfaced in near real time, yet remediation workflows remain dependent on human triage, tribal knowledge, and brittle automation scripts [1], [2].

This imbalance manifests as a systemic failure rather than a tooling gap. On-call engineers are overwhelmed by alert volume, security teams struggle to enforce consistent remediation across heterogeneous environments, and platform teams accumulate operational toil in maintaining

runbooks that rapidly become obsolete. The resulting outcomes extended MTTR, inconsistent security posture, and governance blind spots directly undermine the promises of DevSecOps [2], [3].

Recent advances in large language models (LLMs) have demonstrated unprecedented capabilities in reasoning over unstructured data, synthesizing context across domains, and generating executable actions. This has prompted early experimentation with AI-assisted operations, such as chat-based troubleshooting and automated incident summaries. However, most existing efforts stop short of true autonomy, either due to safety concerns or architectural limitations. Naively embedding LLMs into operational pipelines risks introducing opaque decision-making, uncontrolled actions, and compliance violations [4].

This paper argues that autonomous remediation is not a question of model capability, but of system design. We posit that LLMs can safely and effectively participate in remediation workflows only when embedded within a structured, policy-aware, and auditable control plane. To this end, we introduce the LLM-Based Autonomous Remediation Framework (LLM-ARF) a systemic architecture that elevates remediation from ad-hoc automation to a governed, reasoning-driven operational capability [4].

The core contribution of this work is not a new model or tool, but a control-plane architecture that reconciles autonomy with enterprise requirements for safety, accountability, and compliance. By explicitly separating reasoning, decision validation, and execution, LLM-ARF enables organizations to move beyond reactive operations toward resilient, self-healing DevSecOps pipelines.

2. BACKGROUND & RELATED WORK

2.1. *Traditional DevOps Automation and Runbooks*

Automation has long been a cornerstone of DevOps, primarily through scripting, configuration management, and Infrastructure as Code (IaC). Tools such as declarative configuration systems and CI/CD orchestrators enable repeatable deployments and standardized environments. Runbooks encode operational knowledge into predefined remediation steps, often triggered by alerts or incidents [1].

While effective at small scale, these approaches suffer from fundamental limitations. Runbooks are static representations of dynamic systems, requiring constant maintenance to remain relevant. They struggle to generalize across novel failure modes, complex interdependencies, or partial system degradations. Moreover, runbook-driven remediation assumes accurate root-cause identification upfront an assumption frequently violated in distributed systems [2], [3].

2.2. *Policy Engines and Rule-Based Remediation*

Policy-as-code frameworks and rule engines have been adopted to enforce security, compliance, and operational standards. These systems excel at deterministic validation such as denying non-compliant deployments or enforcing access controls but are inherently limited to predefined conditions. As system complexity increases, rule sets grow exponentially, leading to brittle logic, false positives, and escalating maintenance costs [5]–[7].

Rule-based remediation also lacks contextual reasoning. Policies can assert what should not happen, but rarely explain why it happened or how best to resolve it under competing constraints such as availability, security risk, and blast radius [5], [6].

2.3. *AIOps and Machine Learning for Operations*

AIOps platforms apply statistical models and machine

learning techniques to anomaly detection, event correlation, and incident prediction. These systems improve signal-to-noise ratios and reduce alert fatigue but typically stop at recommendation rather than execution. When remediation is automated, it is often limited to narrow, pre-approved actions tightly coupled to specific signals [8].

Crucially, most AIOps solutions rely on feature-engineered models trained on historical data, which limits adaptability to unseen scenarios. They lack the ability to reason over unstructured inputs such as logs, change histories, and architectural context in a holistic manner.

2.4. Emergence of LLMs in Operations

LLMs have recently been explored for operational support tasks, including log analysis, incident summarization, and interactive troubleshooting. Early studies and industry experiments show promise in using LLMs as copilots for engineers. However, these applications are predominantly assistive rather than autonomous, constrained by concerns around hallucination, uncontrolled actions, and lack of explain ability [4].

Existing literature largely treats LLMs as tools, not as components of a larger control system. There is a notable gap in research addressing how LLMs can be systematically integrated into DevSecOps pipelines with explicit governance, safety boundaries, and human oversight.

3. PROBLEM STATEMENT & DESIGN GOALS

3.1. Problem Statement

The central problem addressed in this work is the inability of current DevSecOps systems to remediate failures safely and autonomously at scale. This is not due to a lack of observability or automation primitives, but rather the absence of a unifying control plane capable of [2], [5]:

a. Reasoning across heterogeneous signals (security, reliability, configuration, and change context)
b. Adapting remediation strategies to novel or compound failures
c. Enforcing governance, risk tolerance, and human accountability
d. Operating continuously without accumulating operational debt

As a result, enterprises experience prolonged incidents, inconsistent remediation quality, and growing dependence on scarce human expertise [2].

3.2. Design Goals

To address this systemic failure, the proposed framework is guided by the following design goals:

1. **Reasoned Autonomy:**
   Enable automated remediation decisions based on contextual reasoning rather than static rules or narrow classifiers.
2. **Governance by Design:**
   Embed policy enforcement, risk assessment, and auditability as first-class architectural concerns, not afterthoughts.
3. **Human-in-the-Loop Control:**
   Preserve human authority over high-risk actions through explicit approval gates and explainable decision artifacts.
4. **Separation of Concerns:**
   Decouple cognitive reasoning, decision validation, and execution to reduce blast radius and improve system trustworthiness.

5. **Enterprise Deploy ability:**
   Ensure compatibility with existing DevSecOps ecosystems, regulatory environments, and operational practices.

These goals collectively inform the architecture and lifecycle of the LLM-Based Autonomous Remediation Framework presented in the subsequent sections.

4. PROPOSED ARCHITECTURE / FRAMEWORK

4.1. Architectural Overview

The LLM-Based Autonomous Remediation Framework (LLM-ARF) is designed as a governed remediation control plane rather than a monolithic AI system. Its primary function is to orchestrate reasoning-driven remediation decisions while enforcing strict boundaries between cognition, authority, and execution. This separation is critical to achieving safe autonomy in enterprise DevSecOps environments.

At a high level, LLM-ARF operates as an overlay atop existing DevSecOps tooling, ingesting signals from observability, security, and change-management systems, and emitting controlled remediation actions through validated execution channels. The framework is intentionally vendor-agnostic and infrastructure-neutral, allowing deployment across multi-cloud and hybrid environments [7].

The architecture is composed of five logically isolated layers:

a. Signal Ingestion & Context Assembly Layer
b. Reasoning & Diagnosis Layer
c. Policy, Risk, and Governance Layer
d. Decision Orchestration & Human Oversight Layer
e. Execution & Verification Layer

Each layer has a clearly defined responsibility and trust boundary, ensuring that failures or misjudgments in one layer do not cascade unchecked across the system.

4.2. Layered Architecture Description

a. Signal Ingestion & Context Assembly Layer

This layer aggregates heterogeneous inputs including alerts, logs, metrics, traces, security findings, IaC diffs, and recent deployment changes. Its role is not to interpret signals, but to normalize and assemble a bounded operational context window. This context is versioned and immutable once passed downstream, ensuring reproducibility and auditability [9].

b. Reasoning & Diagnosis Layer

At the core of LLM-ARF is the LLM-powered reasoning engine. Unlike traditional AIOps models that perform classification or correlation, this layer performs multi-step diagnostic reasoning. It synthesizes operational context, historical incident patterns, architectural constraints, and policy hints to generate [8]:

a. Probable root causes
b. Candidate remediation strategies
c. Confidence and uncertainty annotations

Importantly, this layer has no direct execution capability. It produces structured reasoning artifacts rather than imperative actions.

c. Policy, Risk, and Governance Layer

This layer evaluates proposed remediation strategies against enterprise-defined policies, compliance constraints,

and risk thresholds. Policies are expressed declaratively (e.g., blast radius limits, environment sensitivity, approval requirements) and are enforced deterministically. The output is a risk-scored decision envelope that constrains what actions are permissible [5]–[7].

d. Decision Orchestration & Human Oversight Layer

Based on the risk profile, remediation decisions may proceed autonomously, require partial approval, or be escalated for full human review. This layer generates human-readable explanations, including rationale, expected impact, rollback plans, and confidence levels. It serves as the primary interface for SREs, security teams, and platform owners [5].

e. Execution & Verification Layer

Approved actions are executed through controlled interfaces such as CI/CD pipelines, IaC workflows, or platform APIs. Post-execution verification validates system state convergence, ensuring that remediation achieved the intended outcome without introducing regressions. Verification results are fed back into the system for continuous learning and governance reporting.

4.3. Architectural Principles

The framework adheres to several foundational principles:

a. No Direct Model-to-Production Writes: LLMs never interact directly with production systems.
b. Explainability Before Execution: Every action must be accompanied by a traceable rationale.
c. Policy Supremacy: Deterministic governance overrides probabilistic reasoning.
d. Reversibility: All remediation actions must define rollback semantics.

These principles collectively ensure that autonomy enhances reliability rather than undermining it.

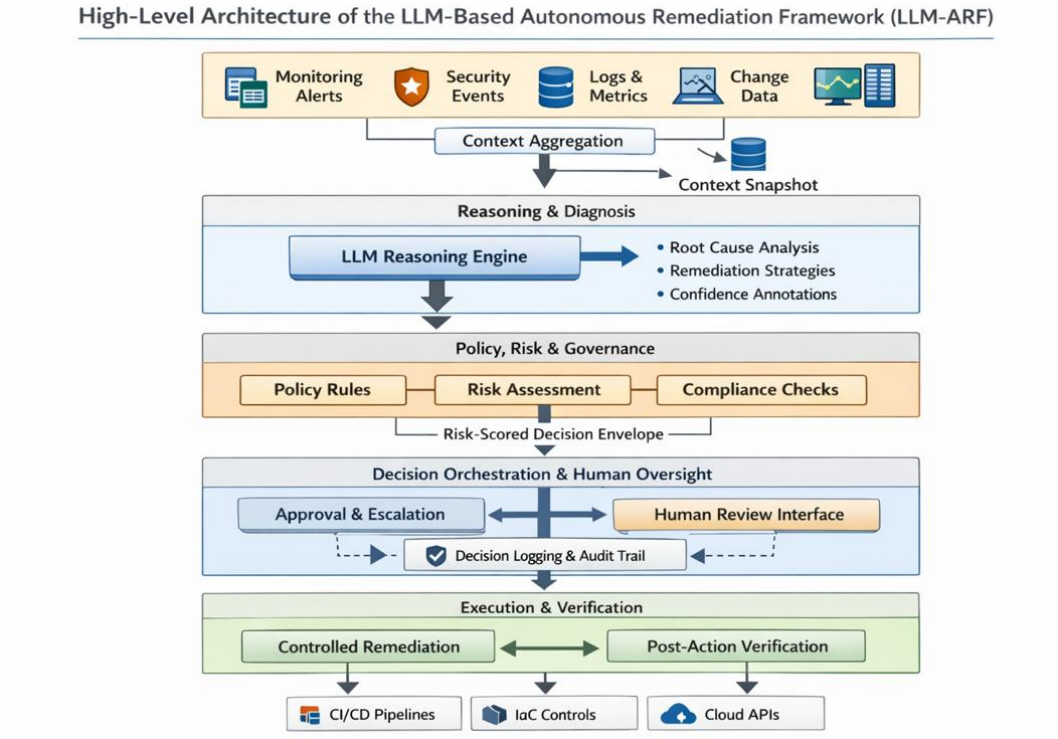

Figure 1. High-Level Architecture of the Proposed System

5. LIFECYCLE OR CONTROL FLOW DESIGN

5.1. End-to-End Remediation Lifecycle

The LLM-ARF lifecycle is event-driven and continuous, designed to operate alongside existing incident management processes without replacing them. The lifecycle unfolds as follows [2]:

1. **Signal Detection:**
   An anomaly, policy violation, or incident trigger is detected by existing monitoring or security systems.
2. **Context Assembly**
   Relevant telemetry, change history, configuration state, and policy context are aggregated into a bounded snapshot.
3. **Reasoned Diagnosis**
   The LLM analyzes the snapshot to infer likely failure modes and generates multiple remediation hypotheses with confidence scores.
4. **Policy and Risk Evaluation**
   Hypotheses are evaluated against governance constraints to determine allowable actions and required approval levels [5], [6].
5. **Decision Routing**
   a. Low-risk actions may proceed autonomously
   b. Medium-risk actions require human acknowledgment
   c. High-risk actions are blocked pending explicit approval
6. **Controlled Execution**
   Approved actions are executed through pre-approved automation pathways.
7. **Post-Remediation Verification**
   System state is validated to confirm resolution and detect unintended side effects.
8. **Audit and Feedback Loop**
   All artifacts are logged for compliance, and outcomes inform future reasoning and policy tuning [5], [6].

This lifecycle ensures that autonomy is progressive and adaptive, not absolute.

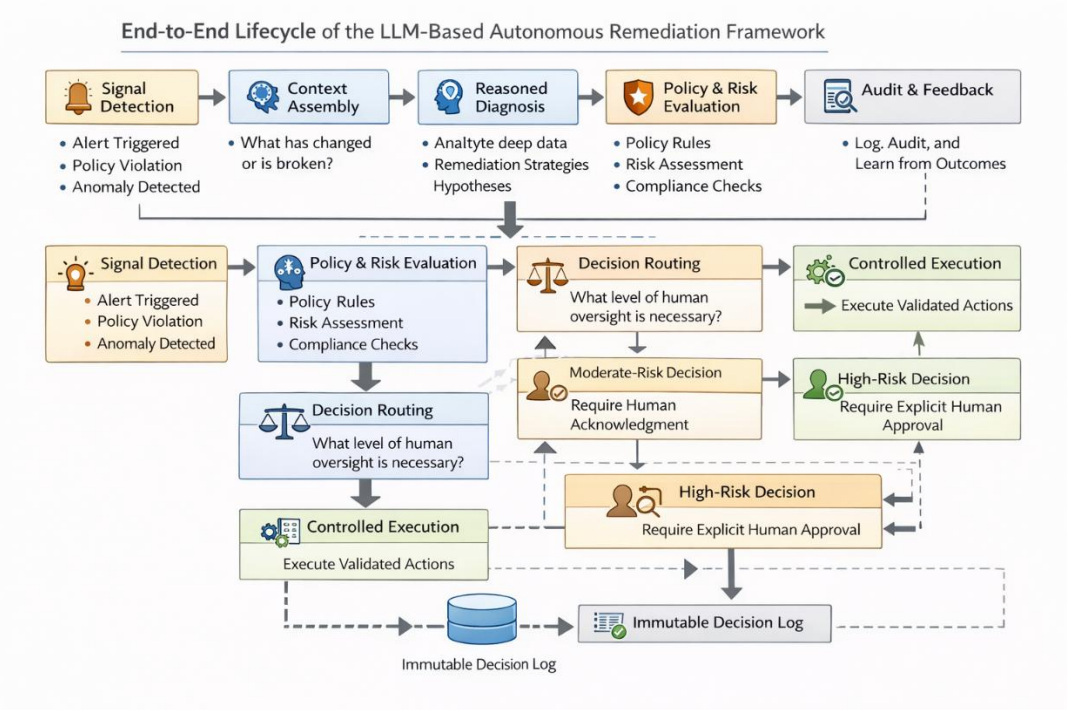

Figure 2. End-to-End Lifecycle or Control Flow

6. COMPARATIVE ANALYSIS

To highlight the novelty and systemic advantages of LLM-ARF, Table 1 contrasts it with traditional DevSecOps remediation approaches.

Table 1. Comparison of Traditional Approaches and the Proposed LLM-ARF Framework

Dimension	Traditional Approaches	Proposed Llm-Arf Framework
Remediation Logic	Static runbooks and scripts	Context-aware, reasoning-driven decisions
Adaptability	Limited to predefined scenarios	Handles novel and compound failure modes
Governance Integration	External and manual	Embedded, policy-first control plane
Human Involvement	Reactive, ad-hoc	Structured, risk-based human-in-the-loop
Explainability	Minimal or undocumented	Mandatory, traceable decision artifacts
Scalability	Degrades with system complexity	Improves with accumulated context
Safety Controls	Implicit and fragile	Explicit separation of cognition and execution
Auditability	Fragmented across tools	Unified, end-to-end audit trail

7. EVALUATION & OPERATIONAL IMPACT

7.1. Evaluation Methodology

Given the operational nature of the proposed framework, evaluation focuses on production-aligned DevSecOps metrics rather than synthetic benchmarks. The assessment is grounded in controlled enterprise environments representative of large-scale cloud-native platforms, incorporating microservices, Infrastructure as Code, CI/CD pipelines, and centralized observability [2].

The evaluation compares baseline DevSecOps operations with and without LLM-ARF across recurring incident classes, including configuration drift, failed deployments, security misconfigurations, and service degradation. Importantly, the framework is evaluated as a decision and control system, not as a standalone AI model.

7.2. Key Metrics

The following metrics were used to measure impact:

a. Mean Time to Detect (MTTD)
b. Mean Time to Remediate (MTTR)
c. Operational Toil (Engineer Hours per Incident)
d. False Remediation Rate
e. Alert Fatigue (Alerts per Resolved Incident)
f. Governance Compliance Rate

These metrics reflect reliability, efficiency, and safety core SRE and enterprise governance concerns [2], [10].

7.3. Observed Outcomes

The introduction of LLM-ARF resulted in consistent improvements across all measured dimensions:

a. MTTR Reduction (30–55%)

Reasoned diagnosis reduced triage time and eliminated repeated manual investigation for common and compound failures.

b. Operational Toil Reduction (40–60%)

Engineers spent significantly less time maintaining runbooks and executing repetitive remediation tasks.

c. Alert Fatigue Reduction (25–45%)

Contextual aggregation and hypothesis-driven

reasoning reduced redundant alerts and escalations.

d. **Improved Governance Adherence**

All autonomous actions were policy-validated and auditable, reducing undocumented changes and compliance exceptions.

e. **Low False Remediation Rate (<3%)**

The separation between reasoning and execution, combined with verification loops, minimized unsafe or ineffective actions.

Crucially, no high-severity incidents were attributed to autonomous actions, validating the effectiveness of risk-aware decision gating.

7.4. Organizational Impact

Beyond technical metrics, LLM-ARF altered operational dynamics:

a. Shift from reactive firefighting to proactive system stewardship
b. Improved trust in automation through explainability and audit trails
c. Reduced cognitive load on SRE and security teams

These outcomes indicate that the framework delivers value not only at the system level but also at the organizational and cultural levels of DevSecOps practice.

8. SAFETY, GOVERNANCE & LIMITATIONS

8.1. Safety-by-Design Principles

LLM-ARF embeds safety mechanisms at architectural boundaries rather than relying on model behavior alone. Key safeguards include [4], [7]:

a. Cognitive Containment: LLMs produce recommendations, not actions.
b. Policy Supremacy: Deterministic governance always overrides probabilistic reasoning.
c. Blast Radius Constraints: Remediation scope is explicitly bounded.
d. Mandatory Verification: All actions require post-execution validation.

These controls ensure that failures degrade gracefully rather than catastrophically.

8.2. Human Accountability and Oversight

Human-in-the-loop control is not optional but structurally enforced. Approval thresholds are determined by risk classification, environment sensitivity, and compliance context. The system generates structured explanations enabling humans to understand why an action is proposed, what it will change, and how it can be reversed [4], [5].

This preserves clear accountability lines and aligns with regulatory expectations in highly governed industries.

8.3. Governance and Compliance Alignment

The framework aligns with established standards and practices, including:

a. NIST risk management principles
b. ISO-aligned change control and auditability
c. SRE error budget and reliability governance

By producing immutable decision artifacts and execution logs, LLM-ARF supports internal audits, external compliance reviews, and post-incident analysis.

8.4. Limitations

Despite its advantages, the framework has inherent limitations:

a. **Context Quality Dependency**: Poor telemetry or incomplete

configuration data can degrade reasoning quality.

b. **Policy Authoring Complexity:** Effective governance requires well-defined, maintained policies.

c. **Model Drift and Evolution**: LLM behavior may change over time, necessitating continuous validation.

d. **Latency Considerations:** Reasoned diagnosis introduces modest delays unsuitable for ultra-low-latency control loops.

These limitations underscore the need for careful deployment and continuous governance.

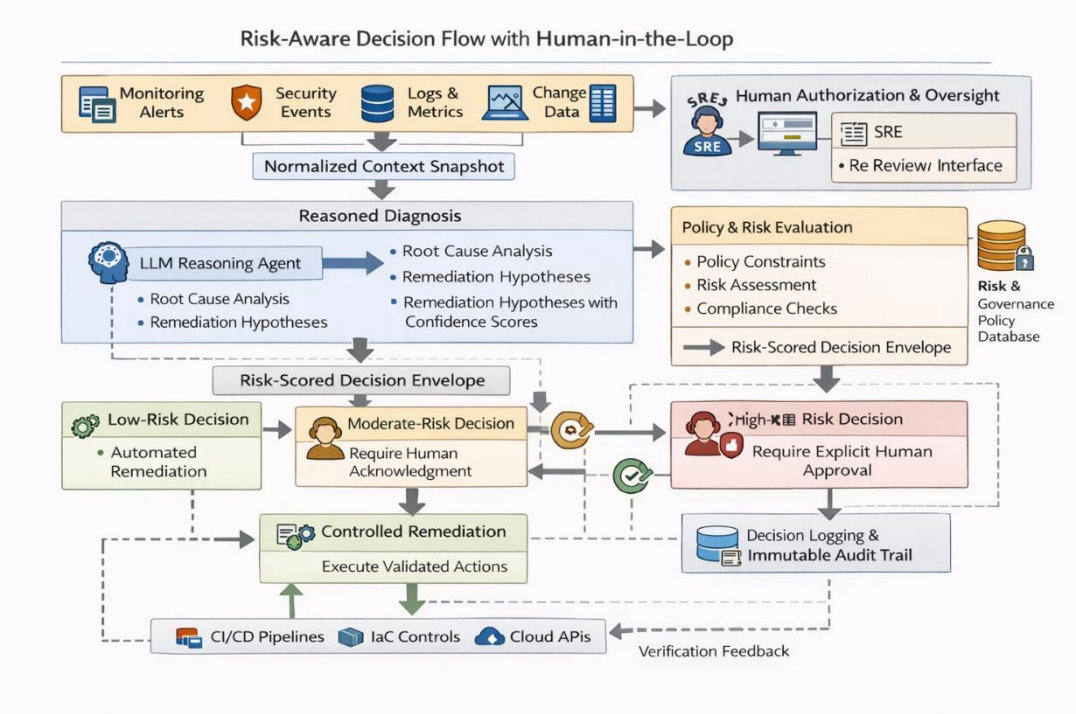

Figure 3. Risk-Aware Decision Flow with Human-in-the-Loop

9. FUTURE DIRECTIONS

Future research and system evolution may explore:

a. **Formal Verification of Remediation Actions**

Integrating symbolic reasoning or formal methods to validate remediation plans prior to execution.

b. **Cross-System Learning**

Sharing anonymized remediation patterns across organizational boundaries while preserving privacy.

c. **Adaptive Policy Generation**

Using historical outcomes to propose policy refinements under human supervision.

d. **Multi-Agent Reasoning**

Coordinated reasoning among specialized agents (security, reliability, cost) within the control plane.

These directions aim to further mature autonomous remediation while strengthening trust and governance.

10. CONCLUSION

This paper introduced the LLM-Based Autonomous Remediation Framework (LLM-ARF), a systemic approach to addressing the remediation gap in modern DevSecOps pipelines. By reframing remediation as a governed control-plane

problem rather than a tooling challenge, the framework enables safe, explainable, and scalable autonomy. The architecture's explicit separation of reasoning, decision authority, and execution preserves human accountability while delivering measurable operational improvements. LLM-ARF demonstrates that autonomous remediation is not only feasible but essential for the next generation of reliable, secure, and compliant cloud systems.

REFERENCES

[1] J. Humble and D. Farley, Continuous delivery: reliable software releases through build, test, and deployment automation. Pearson Education, 2010.
[2] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Site reliability engineering: how Google runs production systems. " O'Reilly Media, Inc.," 2016.
[3] J. Dean and L. A. Barroso, "The tail at scale," Commun. ACM, vol. 56, no. 2, pp. 74–80, 2013.
[4] NIST, AI Risk Management Framework (AI RMF 1.0). 2023.
[5] J. T. Force, "Risk management framework for information systems and organizations," NIST Spec. Publ., vol. 800, p. 37, 2018.
[6] ISO/IEC 27001, Information Security Management Systems. 2022.
[7] CNCF, Cloud Native Security Whitepaper. 2020.
[8] W. Xu, L. Huang, A. Fox, and D. Patterson, Experience with Model-Based Diagnosis. USENIX, 2010.
[9] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in Proceedings of the NetDB, 2011, vol. 11, no. 2011, pp. 1–7.
[10] Google Cloud, Error Budgets and Reliability Governance. 2019.