RESEARCH ARTICLE

Beyond Jenkins Logs: Benchmarking MCP-Enabled AI for Root-Cause Analysis and Remediation in Enterprise CI/CD

Received	Revised	Accepted	Published
10 November 2026	16 November 2026	18 November 2026	21 November 2026

Abstract

Background: Diagnosis of failed CI/CD pipelines may require evidence from deployment state, credentials, registries and infrastructure systems that is absent from Jenkins console logs.

Objective: To compare Model Context Protocol (MCP)-enabled AI with log-only AI and manual triage for root-cause diagnosis, remediation recommendations, successful pipeline reruns and diagnostic efficiency.

Methods: An incident-matched secondary analysis evaluated 320 unique Jenkins failure incidents across three assessment modes (960 evaluations) and 16 root-cause categories. Paired binary outcomes were compared using exact McNemar tests and incident-level bootstrap confidence intervals; diagnosis times were compared within incidents.

Results: MCP-enabled AI achieved 93.75% correct root-cause diagnoses, compared with 62.50% for log-only AI and 83.75% for manual triage. Remediation correctness was 91.56%, 57.19% and 80.63%, respectively, while successful reruns occurred in 70.94%, 39.06% and 59.38%. Mean diagnosis times were 8.09, 18.92 and 39.47 minutes. Fifteen MCP cases were blocked by a policy gate.

Conclusion: MCP-enabled evidence retrieval was associated with better recorded outcomes in this benchmark. Unverified incident provenance, model settings, tool traces and adjudication procedures limit generalisability and preclude claims of established effectiveness or safety in live deployments.

Keywords: Continuous integration; continuous delivery; Jenkins; Model Context Protocol; root-cause analysis; AIOps; remediation; human-in-the-loop; software reliability.

How to cite: Beyond Jenkins Logs: Benchmarking MCP-Enabled AI for Root-Cause Analysis and Remediation in Enterprise CI/CD. Int J Bus Comput Sci. 2026;6(1):72–79.

1. Introduction

Continuous integration and continuous delivery (CI/CD) pipelines mediate changes between source repositories and production infrastructure. A failure recorded in Jenkins may be caused by a test defect, an expired credential, a quota-exhausted container registry, a Kubernetes resource constraint, or an upstream configuration change. A console log indicates where a failure manifested, but it often does not contain the evidence necessary to discriminate among competing explanations. Large-scale studies of CI builds and workflow failures show that failure attribution is heterogeneous and operationally costly [1]–[4]. Longitudinal evidence on build-system maintenance identifies nontrivial operational costs in evolving build technologies [5], while experience with continuous delivery shows that faster releases may coexist with substantial implementation and governance challenges [6].

Automated log analysis has matured from parsing and anomaly detection toward richer fault diagnosis [7]. Yet log-only interpretation remains constrained by observational incompleteness: the decisive evidence may reside in an infrastructure metric, version-control diff, secret-management metadata, quality gate, or incident history. Adjacent work on microservice RCA demonstrates the value of cross-component relationships and explainable fault propagation [8], [9]. The recent AIOps literature identifies tool integration and trustworthy reasoning as important directions, while also emphasizing persistent uncertainty and reproducibility challenges [10].

The Model Context Protocol (MCP) defines a standardized interface through which AI clients can discover and invoke external tools. Its contemporary specification establishes a stateless request architecture and stronger authorization provisions [11]. In an enterprise CI/CD setting, MCP can plausibly separate the language-model reasoning layer from access-controlled evidence systems. Whether such access improves diagnostic correctness, downstream remediation, and operational safety relative to simpler alternatives is an empirical question; protocol adoption alone does not establish efficacy.

This study examines a structured three-arm benchmark covering 320 matched incidents and compares an MCP-enabled AI condition against a log-only AI condition and a manual-triage condition. The study uses a retrospective matched-comparator design and does not assume randomized allocation of evaluation conditions. The objective is to quantify within-incident performance differences, characterize safety and cost tradeoffs, and provide a verifiable reference design for future enterprise evaluation.

1.1 Research Questions

RQ1 (Diagnostic effectiveness): How does MCP-enabled evidence retrieval compare with log-only AI and manual triage in identifying the recorded root cause? RQ2 (Remediation utility): How do the three conditions compare in the correctness of recommended remediation and successful pipeline reruns? RQ3 (Operational efficiency): What differences are observed in diagnosis time, AI response latency, and recorded per-incident AI cost? RQ4 (Governance and robustness): What do policy-block, approval, recurrence, calibration, and failure-category records reveal about the operational limitations of the approaches?

1.2 Contributions

This article makes four contributions. First, it formalizes an incident-matched evaluation of MCP-enabled, log-only, and manual CI/CD diagnosis across 16 distinct recorded fault categories. Second, it provides paired effect estimates, uncertainty intervals, and exact tests for diagnosis, remediation, and pipeline-rerun outcomes, with service-level resampling as a sensitivity analysis. Third, it examines confidence, safety gates, missingness, recurrence, and marginal AI inference costs without conflating them with validated production safety or total cost of ownership. Fourth, it presents an auditable reference architecture connecting evidence access, hypothesis formation, policy-gated execution, and verification; the architecture integrates these mechanisms into an explicit, testable engineering design.

2. Background and Related Work

2.1 CI/CD Failure Taxonomies and Triage

CI incidents arise from overlapping code, dependency, infrastructure, and deployment mechanisms. Zheng et al. developed a taxonomy of GitHub Actions failures using manually examined workflow executions and developer validation [1].

Huang et al. showed that a build failure can be unrelated to a submitted patch and that historical similarity and contextual features can assist attribution [2]. Studies of prolonged builds, timeouts, and redundant CI activity further establish that pipeline failures and delays have consequential operational costs [3], [12], [13]. Together, these findings motivate a diagnostic unit of analysis broader than a single error-message string. Empirical studies of flaky-test detection demonstrate that repeated test execution and predictive classifiers trade accuracy for operational expense [14]. Industrial case studies of automated nonfunctional testing likewise show that CI outcomes depend on quality gates and infrastructure context beyond the console trace [15].

2.2 From Log Analysis to Multi-Source RCA

He et al. survey log instrumentation, parsing, anomaly detection, prediction, and diagnosis [7]. MicroIRC localizes microservice failures to instances and evaluates changing system topology [8], while Soldani et al. use interaction logs to explain possible propagation paths between services [9]. These methods have distinct goals from the present CI/CD benchmark, where the evaluated output is an incident-level categorical root cause and a remediation action rather than an invocation-graph localization. Nevertheless, they support an evidence-first design: provenance and relationships are central to an explanation, not merely additional prompt context. Industrial research on microservice debugging documents the value of tracing and guided visual analysis for localizing distributed faults [16]. Studies of logging-variable selection show that observability also depends on the information developers choose to record [17]. Broader fault-localization research distinguishes symptom classification from locating actionable causes [18].

2.3 LLM-Augmented Operations and Evidence Governance

The AIOps survey of Zhang et al. maps opportunities and unresolved challenges for language models in operational workflows [10]. AI-assisted dependency repair illustrates the need to distinguish plausible suggestions from validated interventions [19]. More generally, empirical-methodology guidance for LLM-based software engineering requires reporting models, prompts, configuration, traces, baselines, labels, and limitations [20]. MCP can standardize tool calls [11], but authorization, least privilege, human approval, and post-execution verification remain application responsibilities. These considerations motivate the governance-aware outcomes and explicit reporting limits used here. A comprehensive failure-management survey identifies heterogeneous data requirements and intervention windows across AIOps methods [21]. Syntheses of machine learning and deep learning in software engineering emphasize the importance of domain-appropriate evaluation, empirical replication, and validation beyond algorithm-level accuracy [22], [23].

2.4 Research Gap

Prior work establishes the complexity of CI failures and the potential of external evidence, but a three-way matched comparison linking root-cause correctness to remediation correctness, successful reruns, diagnosis time, and explicit policy gates is less commonly reported. The present comparison estimates observed performance differences and specifies a replicable analysis framework. Differences in models, prompts, tools, personnel, or time budgets remain potential alternative explanations. Observational software-repository research highlights the risk of mistaking recorded platform events for complete engineering activity [24]; evidence about code-review participation similarly shows that human process quality can shape software outcomes [25]. These considerations motivate restrained causal interpretation of retrospective incident records.

3. Reference Architecture and Decision Process

3.1 Separation of Retrieval, Reasoning, and Action

Figure 1 specifies a proposed architecture organized into an evidence plane, a reasoning-and-provenance plane, and a controlled action plane. Jenkins console logs are treated as an initial symptom source. Read-only MCP adapters expose bounded observations from Git, Kubernetes, registry services, metrics, quality gates, credentials metadata, and ticket history, corresponding to tool names documented in the benchmark. Every retrieved observation should carry a source identifier, time, authorization scope, and structured evidence type. The diagnosis engine then ranks root-cause hypotheses and records the evidence supporting and contradicting each candidate.

A well-formed diagnostic artifact can be represented as D=(i,c,E,a,u), where i identifies the incident, c is the proposed cause, E is the evidence set, a is a recommended action, and u is the stated uncertainty. This representation is a design abstraction; the benchmark captures categorical predictions, confidence values, selected tool names, and audit-trace identifiers. Complete underlying evidence objects and model reasoning traces are not available, so the formalism defines a reference design rather than an observed implementation trace.

3.2 Risk-Gated Remediation

The action plane should distinguish an advisory diagnosis from an executable change. Any proposed write operation is evaluated against environment, change-risk score, allowed target, approvals, and rollback feasibility. Read-only tool scopes and separated execution credentials limit unintended side effects; high-risk or production-impacting actions require an accountable human decision. Denials or unmet preconditions produce a policy-block event, while executed actions must be checked through an independent pipeline rerun and service-health evidence. The benchmark contains fields for approval requirements, status, execution mode, policy blocks, rerun result, and resolution, allowing descriptive assessment of these gates but not verification of enforcement implementation. Research on learning-based automated program repair further illustrates why a plausible candidate change must be distinguished from a tested repair that is valid in its target environment [26].

3.3 MCP Interoperability and Threat Boundaries

MCP is a transport and tool-interaction interface, not a substitute for authorization or dependable orchestration. An enterprise implementation should bind identities to resources, minimize cross-service credentials, validate tool schemas and result trustworthiness, prevent prompt-injection content in logs or tickets from becoming executable instructions, and cryptographically correlate action requests with approvals and audit identifiers. The 2026 protocol revision supports per-request metadata and enhanced authorization semantics [11]. No particular MCP protocol version, OAuth configuration, or isolation mechanism can be established from the benchmark records.

4. Study Design

4.1 Data Source and Unit of Analysis

The study dataset, Enterprise_CICD_MCP_RCA_Dataset, comprises 960 evaluation records and 53 variables. Exactly 320 unique incident IDs occur once in each of the three recorded assessment modes: MCP-Enabled AI, Log-Only AI, and Manual Triage. Within each incident, the recorded failure timestamp, service, repository, severity, failed stage, root-cause label, build characteristics, and approval requirement are identical across modes. These invariants permit an incident-matched comparison without mixing different underlying failures. Incident dates range from 8 January through 23 October 2026. The data include 15 services, eight business units, four environments, 12 failed-stage categories, and 16 distinct root-cause labels, each represented by 20 incidents (Table I).

The available study records do not include collection protocols, inclusion criteria, incident-selection procedures, annotator records, model identifiers, prompt templates, human-triage instructions, or raw MCP execution transcripts. Consequently, the provenance of the records (observed deployments, controlled exercises, or generated benchmark episodes) cannot be independently verified. The exactly balanced root-cause distribution warrants caution regarding representativeness. The study is therefore a secondary benchmark analysis rather than a prospective field trial.

4.2 Comparators and Endpoints

The MCP-enabled condition records multiple evidence sources and tool calls per incident; the log-only AI condition records Jenkins console output alone; manual triage records analyst-oriented resolution activity and no model-response or token counts. The treatment label reflects the recorded condition names and does not imply a controlled intervention with documented randomization. Diagnosis correctness equals the recorded binary label and was independently checked against agreement between Predicted_Root_Cause and Actual_Root_Cause, with no mismatches. Remediation correctness uses the dataset's recorded binary judgment because an independent remediation rubric is unavailable.

Primary endpoints are (i) correct diagnosis, (ii) correct remediation, and (iii) successful pipeline rerun. Time to diagnosis is a prespecified operational endpoint. Confirmed resolution and successful rerun are reported separately because they are not identical in the dataset. Recovery time and 14-day recurrence are defined only for rows with a successful rerun: their missingness is outcome-dependent, making naive unconditional comparisons invalid. Assistant response time, MCP-tool latency, input/output tokens, estimated AI cost, confidence, feedback, approvals, and policy blocks are secondary descriptive endpoints. Time to diagnosis may include human and workflow delays in addition to model latency; its timing definition cannot be reconstructed from the available timing fields alone.

4.3 Statistical Estimands and Analysis

For binary endpoint Y, the paired risk difference versus comparator b is $RD_b=(1/N)\Sigma_i(Y_{i,MCP}-Y_{i,b})$, N=320. We report raw percentages and risk differences in percentage points. Two-sided exact McNemar p-values are based on discordant incident pairs, and 95% percentile confidence intervals are obtained by nonparametric incident-level bootstrap resampling (12,000 replicates; fixed random seed). For diagnosis time T, the estimand is the paired mean difference (MCP minus comparator), with bootstrap confidence limits; two-sided Wilcoxon signed-rank tests are provided as distribution-robust supporting analyses. These comparisons are exploratory because the underlying evaluation allocation was not documented.

To probe dependence within recurring services, we resampled the 15 service clusters with replacement (6,000 replicates) and recomputed differences. For confidence-quality analysis, the mean Brier score was calculated from recorded diagnosis confidence p and correctness y as $N^{-1}\Sigma(p-y)^2$ for the two AI conditions. No classification threshold was optimized on the same records. Statistical tests evaluate differences conditional on the observed study records and cannot correct unknown sampling, measurement, or annotation bias.

4.4 Integrity Checks and Missingness

The matched incident fields were invariant across all three rows for each incident. The binary diagnosis flag was consistent with exact agreement between predicted and labeled cause for all 960 records. A successful rerun corresponded exactly to nonmissing recovery time and nonmissing 14-day recurrence. Nevertheless, resolved-incident labels did not always coincide with successful reruns: MCP-enabled AI had 226 labeled resolutions and 227 successful reruns; manual triage had 186 and 190, respectively. The primary operational analysis therefore uses the explicitly coded rerun endpoint, with the resolution outcome disclosed separately. No policy violations were recorded in any condition, preventing estimation of comparative violation-prevention effectiveness.

5. Results

5.1 Cohort Characteristics and Evidence Exposure

The 320 incident scenarios comprised 47 critical, 120 high, 123 medium, and 30 low severity failures; 40 incidents involved production, 115 staging, 94 quality assurance, and 71 development environments. The failure taxonomy was deliberately or coincidentally balanced at exactly 20 incidents per category. The MCP condition recorded a mean of 5.20 tool calls and 3.73 evidence sources per incident, compared with zero tool calls and one evidence source for log-only AI. These values document exposure differences but do not isolate which individual tool contributed to correctness (Table I).

5.2 Diagnostic Accuracy (RQ1)

The MCP condition correctly identified 300 of 320 labeled causes (93.75%), versus 200 (62.50%) under log-only AI and 268 (83.75%) under manual triage (Table II; Fig. 2). Compared with log-only AI, the paired increase was 31.25 percentage points (95% bootstrap CI 25.62–36.88); 111 incidents favored MCP and 11 favored log-only AI among discordant pairs (exact McNemar $p=5.85\times10^{-22}$). Against manual triage, the corresponding difference was 10.00 points (95% CI 5.31–14.69); 47 discordant incidents favored MCP and 15 favored manual triage ($p=5.78\times10^{-5}$). The latter comparison is more stringent: manual triage substantially outperformed log-only AI on the same incident set.

Improvement was not restricted to a single fault family (Table IV). MCP diagnosis was correct in all 20 instances of each of five categories—environment configuration, Kubernetes out-of-memory, network/DNS, pipeline configuration, and registry rate limitation—and in 16 of 20 flaky-test cases. The relatively lower flaky-test performance illustrates residual ambiguity even with external evidence, consistent with independently documented challenges in CI test flakiness [27]. Because each category contains only 20 incidents, these stratified comparisons are descriptive and not independent confirmatory tests.

5.3 Remediation Correctness and Rerun Success (RQ2)

Correct remediation was recorded in 293 cases (91.56%) for MCP-enabled AI, 183 (57.19%) for log-only AI, and 258 (80.63%) for manual triage. Paired gains were 34.38 points versus log-only AI (95% CI 28.44–40.31; $p=1.24\times10^{-23}$) and 10.94 points versus manual triage (95% CI 5.93–16.25; $p=8.22\times10^{-5}$). Successful reruns were observed in 227, 125, and 190 incidents, corresponding to 70.94%, 39.06%, and 59.38%. The MCP success-rate difference was 31.88 points versus log-only AI (95% CI 24.69–39.06; $p=1.19\times10^{-15}$) and 11.56 points versus manual triage (95% CI 4.06–19.06; p=0.00392), respectively (Table III; Fig. 3).

The separation between 93.75% diagnosis correctness and 70.94% rerun success is operationally important. A correct diagnostic label need not yield an authorized, executable, or successful recovery. In the MCP condition, 57 incidents were escalated, 21 unresolved, 15 blocked by policy, and one rolled back; resolution was recorded for 226 incidents. These categories also reflect workflow or evaluation rules unavailable for independent inspection. The results therefore support improvement in recorded remedial utility, not an assertion of autonomous healing in every successfully diagnosed case.

5.4 Diagnosis Time and Inference Overhead (RQ3)

Mean time to diagnosis was 8.09 minutes under MCP-enabled AI, 18.92 under log-only AI, and 39.47 under manual triage (Table II; Fig. 4). Incident-matched mean differences were −10.82

minutes relative to log-only AI (95% CI −11.37 to −10.27; Wilcoxon p=6.06×10^-54) and −31.38 minutes relative to manual triage (95% CI −32.46 to −30.32; p=3.29×10^-54). These correspond to 57.2% and 79.5% lower mean diagnosis times, respectively, within the benchmark. Without a timestamped task definition, the precision of these time measures should not be interpreted as a verified service-level objective.

The assistant-response median was 24.65 s for MCP-enabled AI versus 9.17 s for log-only AI, demonstrating a distinct tradeoff: richer evidence retrieval coincided with slower model interaction but shorter end-to-end diagnosis. MCP tool latency averaged 11.36 s and recorded AI inference cost averaged US$0.001848 per case versus US$0.001009 for log-only AI, an 83% relative increase on the narrow cost field. The cost estimate excludes external tool execution, compute provisioning, commercial licenses, personnel time, and failure externalities; it is not a total-cost-of-ownership estimate.

5.5 Policy Gates, Confidence, and Recurrence (RQ4)

The benchmark records indicate that 115 incidents per condition required approval. The MCP condition recorded 15 policy-blocked cases (10 pending approvals and five denials), while no policy violations were coded for any condition (Table V). A zero-event violation field does not demonstrate that a protective control prevented all unsafe behavior: the data do not expose the set of attempted disallowed actions, adjudicated security incidents, or enforcement proofs. A larger share of MCP cases yielded recorded analyst feedback of four or five, with mean feedback 4.60/5 versus 2.88/5 for log-only AI and 3.74/5 for manual triage; assessor blinding and rubric are unknown.

Recorded diagnosis confidence averaged 0.884 under MCP-enabled AI and 0.758 under log-only AI. The corresponding Brier scores were 0.0436 and 0.2428, although the score interpretation depends on whether probabilities were elicited consistently. Three of the 20 incorrect MCP predictions had confidence at least 0.80, compared with 37 of 120 incorrect log-only predictions. A model may be more accurate without being perfectly calibrated; external reliability diagrams and an independently labeled test set would be needed for a deployment confidence threshold.

Fourteen-day recurrence was observed only after successful reruns: 17/227 (7.49%) for MCP-enabled AI, 22/125 (17.60%) for log-only AI, and 20/190 (10.53%) for manual triage. These denominators select different incident subsets by post-assessment outcome. Consequently, recurrence differences are descriptive and cannot be attributed causally to the assessment mode. Similarly, recovery time was observed only after successful reruns (mean 37.01, 62.09, and 104.10 minutes across the three modes). Comparing these conditional means would create selection bias and should not be treated as an unconditional recovery-time advantage.

5.6 Service-Cluster Sensitivity

Resampling whole service clusters rather than individual incidents retained positive improvements on all three binary outcomes. Against log-only AI, the service-cluster bootstrap intervals were 26.2–36.5 points for diagnosis, 29.3–39.5 for remediation, and 23.4–39.3 for rerun success; against manual triage, they were 4.3–15.8, 6.4–16.0, and 3.9–18.8 points, respectively. With only 15 service clusters and no validation of how they were sampled, these intervals should be viewed as sensitivity summaries rather than as a substitute for independently replicated deployment studies.

6. Discussion

6.1 Why External Evidence Could Matter

The observed increase in diagnosis correctness is compatible with a mechanism in which external tools resolve ambiguities that Jenkins console output cannot. Certificate expiry is distinguished from DNS failure by certificate inventory and network evidence; registry throttling is distinguished from missing image tags by quota and registry metadata; and resource exhaustion can be examined through Kubernetes and metric endpoints. The dataset records these tool families but not the causal evidence sequence, so the proposed mechanism remains plausible rather than demonstrated. The literature on CI incident taxonomies and log-driven fault propagation provides independent reasons for treating evidence breadth as important [1], [7]–[9].

6.2 The Stronger Comparator Is Human Triage

Log-only AI was substantially weaker than manual triage in the matched benchmark, a contrast that guards against a misleading binary conclusion that all AI assistance is beneficial. The meaningful operating question is whether well-governed, evidence-grounded AI can augment a capable analyst without introducing new risks. Here the paired difference against manual triage was smaller but remained positive for diagnosis, recommendation correctness, and successful rerun. No analyst experience levels, workload, or task-allocation rules were reported, and the benchmark may not capture the cognitive work performed by actual on-call engineers.

6.3 A Correct Diagnosis Does Not Equal Safe Remediation

Differences between diagnosis, remediation, resolution, and rerun endpoints reveal a critical distinction often obscured by classification-only evaluation. Even when a root cause is labeled correctly, a remediation may need approval, lack sufficient evidence, be outside policy, or fail verification. A robust operational assistant should be judged on abstention, traceability, action feasibility, approval fidelity, rollback behavior, and independently validated service health. Policy gating plausibly reduces exposure, but its effectiveness cannot be established from the absence of coded violations alone. Published studies of software incident diagnosis and AI-assisted repair similarly motivate separating problem identification from downstream intervention [8], [9], [19].

6.4 Interpreting Latency, Confidence, and Cost

The MCP condition exhibits a rational operational tension: more tool calls and slower response generation alongside substantially less recorded diagnosis time. This pattern is consistent with effort being shifted from analyst search to machine-assisted retrieval, but the dataset cannot identify the relative contribution of retrieval, model capability, prompt engineering, or condition-specific workflow rules. Estimated inference expense is very small in absolute terms but materially higher than log-only AI; total economic comparison requires operator minutes, cloud access costs, remediation failures, and the opportunity cost of delays. The difference between AI confidence and observed correctness should be assessed under calibration protocols before confidence is used to authorize actions [10], [20].

6.5 Threats to Construct, Internal, and External Validity

Construct validity is limited by the absence of an independent root-cause adjudication protocol, detailed definitions of diagnosis duration, remediation-correctness rubric, and precise incident-resolution criteria. Internal validity is constrained by unknown condition assignment, model versions, prompts, access rights, human instructions, random seeds, evaluation order, and counterfactual comparability. The paired layout controls incident identity but not these unobserved differences. External validity is limited by a 320-incident, 15-service benchmark with exactly balanced root-cause classes, a distribution unlikely to mirror all enterprise operations. The absence of raw logs beyond excerpts, full tool transcripts, source code, and model configurations restricts replication and mechanistic attribution. Related evidence on inter-dataset overlap and evaluation leakage in language-model studies underscores the importance of independent benchmarks and reproducible data construction [28].

Selection effects are particularly salient for recovery time and recurrence because those fields are present only when reruns succeed. The empirical record also includes status discordance between success and resolution, and all policy-violation flags are zero. These properties require careful outcome definitions and preclude claims of proven autonomous safety. The dataset does not measure cross-organization transport, adversarial prompt injection, long-running drift, false changes with high blast radius, or operator override patterns. In keeping with contemporary LLM study-reporting recommendations [20], these unavailable observations constrain claims of operational generalizability and implementation fidelity.

6.6 Path to Prospective Validation

A stronger next study would prospectively capture incident-creation rules, representative prevalence, signed tool invocations, model and prompt hashes, evaluator expertise, rubric-based adjudication, and independent outcome timestamps. Randomized or counterbalanced incident replay would isolate effects of tool availability while holding model and prompt versions fixed. Ablation arms should remove Git, metrics, history, and secrets metadata one at a time; safety evaluation should deliberately include denied actions, forged tool content, and conflicting evidence. Outcomes should extend to time-to-recovery with explicitly defined censoring, verified rollback safety, cost per confirmed resolution, and recurrence in a common at-risk population. Multi-organization evaluation should test transportability across CI systems and fault-class shifts.

7. Conclusion

In an incident-matched benchmark of 320 CI/CD failure incidents, MCP-enabled AI was associated with higher recorded root-cause and remediation correctness, more successful pipeline reruns, and shorter diagnosis time than both log-only AI and manual triage. The observed gains were statistically distinguishable within incident pairs and persisted in service-cluster sensitivity analysis. Richer retrieval increased response latency and recorded inference cost, while policy blocks and unresolved cases show that diagnostic accuracy alone is not sufficient for dependable autonomous remediation. These results support evidence-grounded, policy-gated assistance within the assessed benchmark, without establishing production-level autonomous remediation safety. Prospective trials with independently verifiable incidents, execution traces, and safety outcomes are required to determine whether the same advantages hold in enterprise deployment.

References

1. L. Zheng, S. Li, X. Huang, J. Huang, B. Lin, J. Chen, and J. Xuan, “Why do GitHub Actions workflows fail? An empirical study,” ACM Transactions on Software Engineering and Methodology, vol. 35, no. 5, art. 139, 2026, doi: 10.1145/3749371.
2. Y. A. Huang, D. A. da Costa, G. Dick, M. El Mezouar, and L. Xiao, “Is this build failure related to my patch? An empirical study of unrelated build failures in continuous integration,” Empirical Software Engineering, vol. 31, art. 148, 2026, doi: 10.1007/s10664-026-10874-8.
3. T. A. Ghaleb, D. A. da Costa, and Y. Zou, “An empirical study of the long duration of continuous integration builds,” Empirical Software Engineering, vol. 24, pp. 2102–2139, 2019, doi: 10.1007/s10664-019-09695-9.
4. Ö. Özdemir, R. Aydoğan, and H. Sözer, “On the use of machine learning for failure prediction after collective changes in automated continuous integration testing,” Journal of Systems and Software, vol. 236, art. 112791, 2026, doi: 10.1016/j.jss.2026.112791.
5. S. McIntosh, M. Nagappan, B. Adams, A. Mockus, and A. E. Hassan, “A large-scale empirical study of the relationship between build technology and build maintenance,” Empirical Software Engineering, vol. 20, no. 6, pp. 1587–1633, 2015, doi: 10.1007/s10664-014-9324-x.
6. L. Chen, “Continuous delivery: Huge benefits, but challenges too,” IEEE Software, vol. 32, no. 2, pp. 50–54, 2015, doi: 10.1109/MS.2015.27.
7. S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, “A survey on automated log analysis for reliability engineering,” ACM Computing Surveys, vol. 54, no. 6, art. 130, 2022, doi: 10.1145/3460345.
8. Y. Zhu, J. Wang, B. Li, Y. Zhao, Z. Zhang, Y. Xiong, and S. Chen, “MicroIRC: Instance-level root cause localization for microservice systems,” Journal of Systems and Software, vol. 216, art. 112145, 2024, doi: 10.1016/j.jss.2024.112145.
9. J. Soldani, S. Forti, L. Roveroni, and A. Brogi, “Explaining microservices’ cascading failures from their logs,” Software: Practice and Experience, vol. 55, no. 5, pp. 809–828, 2025, doi: 10.1002/spe.3400.
10. L. Zhang et al., “A survey of AIOps in the era of large language models,” ACM Computing Surveys, vol. 58, no. 2, art. 44, 2025, doi: 10.1145/3746635.
11. Model Context Protocol, “Model Context Protocol specification,” version 2026-07-28, Jul. 2026. [Online]. Available: https://modelcontextprotocol.io/specification/2026-07-28
12. N. Weeraddana, M. Alfadel, and S. McIntosh, “Characterizing timeout builds in continuous integration,” IEEE Transactions on Software Engineering, 2024, doi: 10.1109/TSE.2024.3387840.
13. D. M. Kamath, E. Fernandes, B. Adams, and A. E. Hassan, “On combining commit grouping and build skip prediction to reduce redundant continuous integration activity,” Empirical Software Engineering, vol. 29, 2024, doi: 10.1007/s10664-024-10477-1.
14. O. Parry, G. M. Kapfhammer, M. Hilton, and P. McMinn, “Empirically evaluating flaky test detection techniques combining test case rerunning and machine learning models,” Empirical Software Engineering, vol. 28, art. 72, 2023, doi: 10.1007/s10664-023-10307-w.
15. L. Yu, E. Alégroth, P. Chatzipetrou, and T. Gorschek, “Automated NFR testing in continuous integration environments: A multi-case study of Nordic companies,” Empirical Software Engineering, vol. 28, art. 144, 2023, doi: 10.1007/s10664-023-10356-1.
16. X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, “Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study,” IEEE Transactions on Software Engineering, vol. 47, no. 2, pp. 243–260, 2021, doi: 10.1109/TSE.2018.2887384.
17. Z. Liu, X. Xia, D. Lo, Z. Xing, A. E. Hassan, and S. Li, “Which variables should I log?,” IEEE Transactions on Software Engineering, vol. 47, no. 9, pp. 2012–2031, 2021, doi: 10.1109/TSE.2019.2941943.
18. W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, “A survey on software fault localization,” IEEE Transactions on Software Engineering, vol. 42, no. 8, pp. 707–740, 2016, doi: 10.1109/TSE.2016.2521368.
19. F. Reyes, M. Mahmoud, F. Bono, et al., “Byam: Fixing breaking dependency updates with large language models,” Empirical Software Engineering, vol. 31, art. 113, 2026, doi: 10.1007/s10664-026-10835-1.
20. S. Baltes et al., “Guidelines for empirical studies in software engineering involving large language models,” Empirical Software Engineering, published online Nov. 3, 2026, doi: 10.1007/s10664-026-10922-3.
21. P. Notaro, J. Cardoso, and M. Gerndt, “A survey of AIOps methods for failure management,” ACM Transactions on Intelligent Systems and Technology, vol. 12, no. 6, art. 81, 2021, doi: 10.1145/3483424.
22. Z. Kotti, R. Galanopoulou, and D. Spinellis, “Machine learning for software engineering: A tertiary study,” ACM Computing Surveys, vol. 55, no. 12, art. 256, 2023, doi: 10.1145/3572905.
23. Y. Yang, X. Xia, D. Lo, and J. Grundy, “A survey on deep learning for software engineering,” ACM Computing Surveys, vol. 54, no. 10s, art. 206, 2022, doi: 10.1145/3505243.
24. E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, “An in-depth study of the promises and perils of mining GitHub,” Empirical Software Engineering, vol. 21, no. 5, pp. 2035–2071, 2016, doi: 10.1007/s10664-015-9393-5.
25. S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, “An empirical study of the impact of modern code review practices on software quality,” Empirical Software Engineering, vol. 21, no. 5, pp. 2146–2189, 2016, doi: 10.1007/s10664-015-9381-9.
26. A. Zirak and H. Hemmati, “Improving automated program repair with domain adaptation,” ACM Transactions on Software Engineering and Methodology, vol. 33, no. 3, art. 65, 2024, doi: 10.1145/3631972.
27. F. Leinen, M. Gruber, S. S. Erdogan, A. Stahlbauer, and A. Pretschner, “An empirical study of detected and undetected flaky test failures in real-world CI pipelines,” IEEE Transactions on Software Engineering, early access, 2026, doi: 10.1109/TSE.2026.3729363.
28. J. A. Hernández López, B. Chen, M. Saad, T. Sharma, and D. Varró, “On inter-dataset code duplication and data leakage in large language models,” IEEE Transactions on Software Engineering, vol. 51, no. 1, pp. 192–205, 2025, doi: 10.1109/TSE.2024.3504286.

Figures

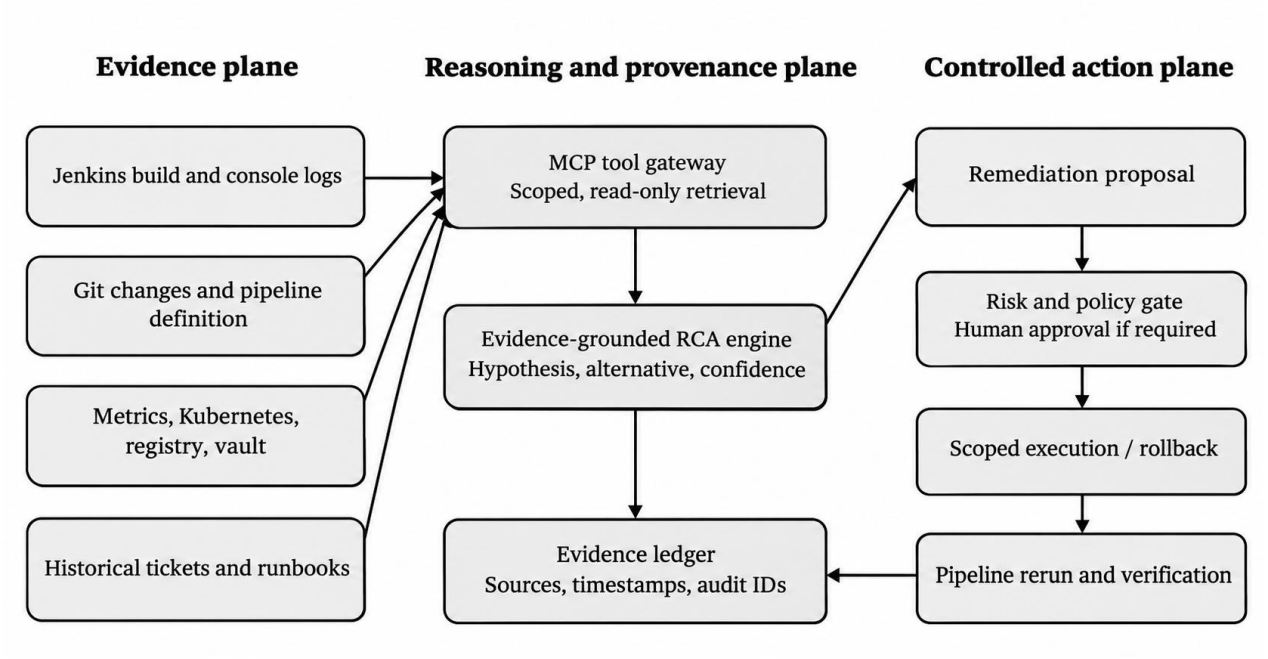


Figure 1. Evidence-centered MCP reference architecture for CI/CD incident diagnosis and policy-gated remediation.

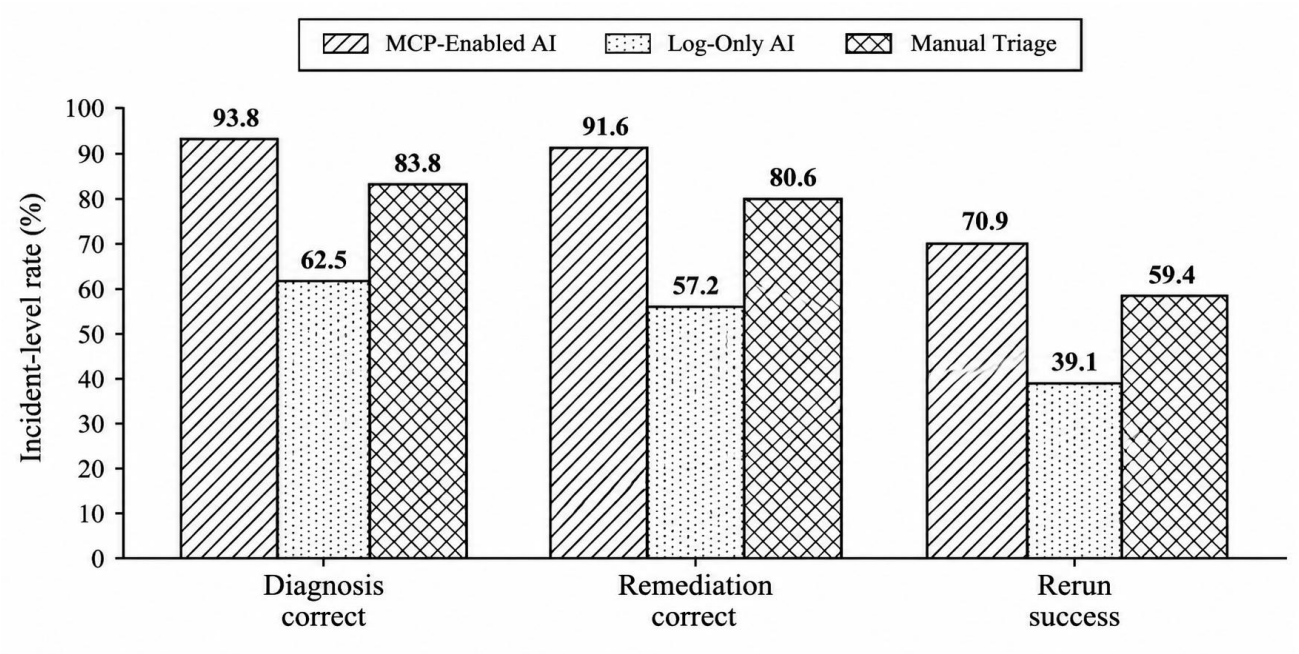


Figure 2. Recorded rates of correct diagnosis, correct remediation, and successful rerun under each assessment mode (320 matched incidents per condition).

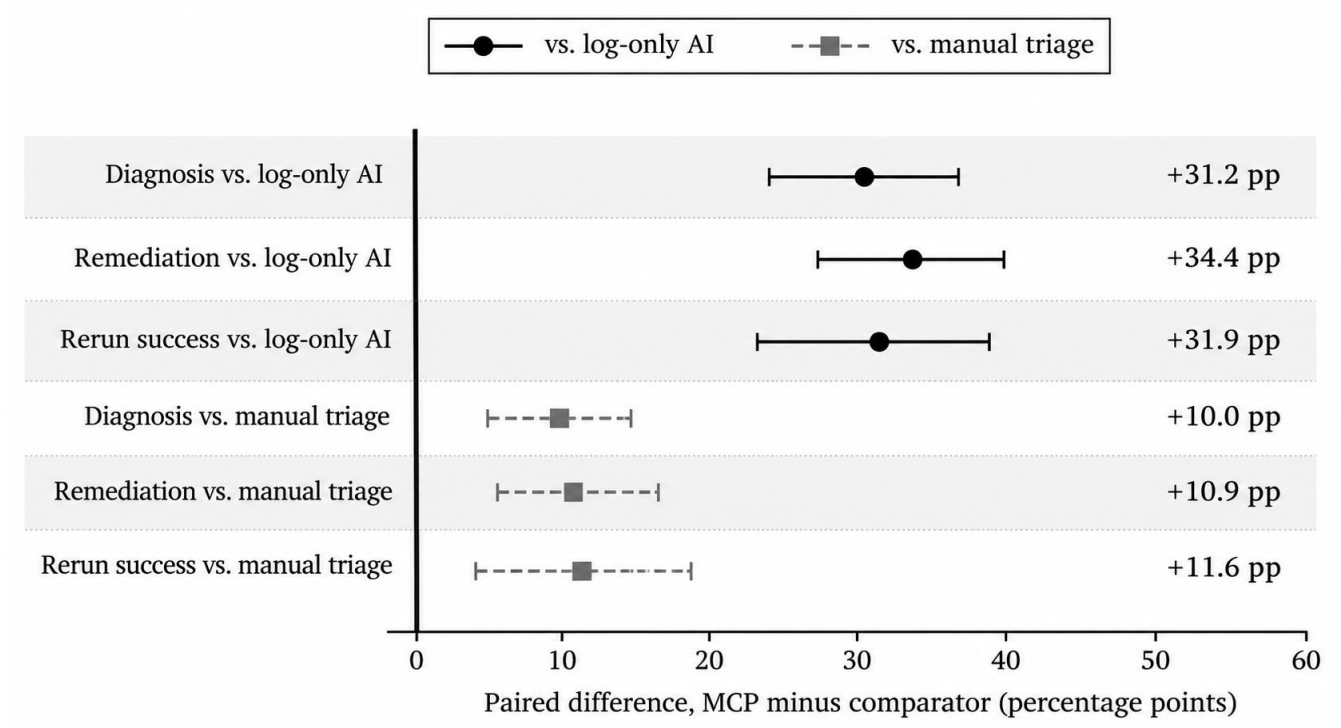


Figure 3. MCP-versus-comparator incident-matched risk differences for three binary endpoints. Lines show 95% percentile bootstrap intervals; all values are percentage-point differences.

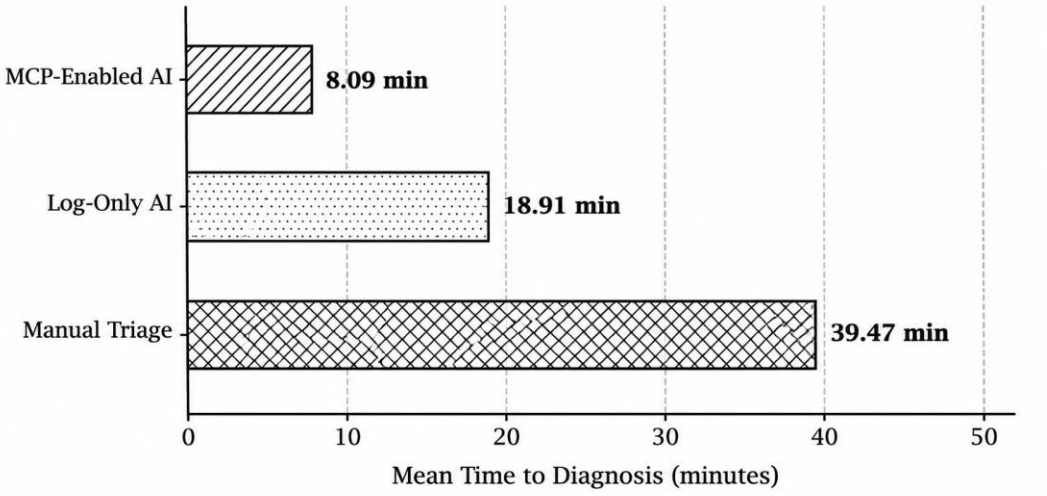


Figure 4. Observed mean time to diagnosis in minutes for the three assessment modes. Mean comparisons are distinct from AI response-generation latency.

Tables

Table 1. Benchmark composition and comparator instrumentation

Characteristic	Recorded value
Unique incidents / evaluation rows	320 incidents / 960 evaluations
Assessment modes	MCP-enabled AI; log-only AI; manual triage
Incident assessment structure	Exactly one record per condition per incident
Failure time range	08 Jan–23 Oct 2026
Services / business units / environments	15 / 8 / 4
Failure stages / cause labels	12 stages / 16 root-cause categories
Cause-category prevalence	20 incidents per category (6.25%)
Incident severity	Critical 47; high 120; medium 123; low 30
Incident environment	Production 40; staging 115; QA 94; development 71
Mean evidence sources per incident	MCP 3.73; log-only 1.00; manual 2.00
Mean MCP calls per incident	MCP 5.20; other conditions 0.00
Recorded approval requirement	115/320 incidents, invariant across modes

Table 2. Comparative endpoints by assessment condition (n=320 per mode)

Endpoint	MCP-enabled AI	Log-only AI	Manual triage
Correct diagnosis	300 (93.75%)	200 (62.50%)	268 (83.75%)

Endpoint	MCP-enabled AI	Log-only AI	Manual triage
Correct remediation	293 (91.56%)	183 (57.19%)	258 (80.63%)
Successful pipeline rerun	227 (70.94%)	125 (39.06%)	190 (59.38%)
Labeled resolved	226 (70.63%)	125 (39.06%)	186 (58.13%)
Mean diagnosis time (min)	8.09	18.92	39.47
Median diagnosis time (min)	8.19	19.03	39.08
Median AI response (s)	24.65	9.17	Not applicable
Mean tool latency (s)	11.36	0.00	0.00
Mean model inference cost (USD)	0.001848	0.001009	0.00
Mean analyst feedback (1–5)	4.60	2.88	3.74

Table 3. Incident-matched MCP effects versus each comparator

Endpoint	Comparator	Difference (pp)	95% bootstrap CI (pp)	Discordant pairs (MCP+/−)	Exact p
Correct diagnosis	Log-only AI	+31.25	[25.62, 36.88]	111 / 11	5.85e-22
Correct diagnosis	Manual triage	+10.00	[5.31, 14.69]	47 / 15	5.78e-05
Correct remediation	Log-only AI	+34.38	[28.44, 40.31]	123 / 13	1.24e-23
Correct remediation	Manual triage	+10.94	[5.93, 16.25]	56 / 21	8.22e-05
Rerun success	Log-only AI	+31.88	[24.69, 39.06]	136 / 34	1.19e-15
Rerun success	Manual triage	+11.56	[4.06, 19.06]	97 / 60	3.92e-03

Note: pp, percentage points; paired difference = MCP minus comparator. For all listed binary outcomes, p is the two-sided exact McNemar test; the confidence interval uses 12,000 incident-bootstrap replicates. Discordant pairs list cases where MCP succeeded/comparator failed followed by the converse.

Table 4. Diagnosis and rerun success by recorded root-cause category

Root-cause label	n	MCP diag.	Log diag.	Manual diag.	MCP rerun	Log rerun	Manual rerun
Agent Disk Full	20	19/20	14/20	18/20	15/20	12/20	14/20
Artifact Storage Quota	20	18/20	14/20	15/20	12/20	8/20	11/20
Certificate Expiry	20	19/20	18/20	17/20	17/20	8/20	10/20
Credential Expired	20	19/20	15/20	17/20	12/20	7/20	10/20
Database Migration	20	18/20	7/20	16/20	15/20	9/20	13/20
Dependency Conflict	20	19/20	13/20	16/20	12/20	5/20	11/20
Env Configuration	20	20/20	12/20	18/20	14/20	7/20	13/20
Flaky Test	20	16/20	8/20	15/20	11/20	3/20	13/20
Image Tag Missing	20	18/20	14/20	19/20	13/20	10/20	13/20
Kubernetes Oom	20	20/20	10/20	18/20	17/20	6/20	14/20
Network Dns	20	20/20	11/20	17/20	15/20	9/20	13/20
Pipeline Configuration	20	20/20	15/20	19/20	16/20	13/20	11/20
Quality Gate Failure	20	18/20	15/20	16/20	13/20	11/20	12/20
Registry Rate Limit	20	20/20	12/20	17/20	16/20	6/20	11/20
Secret Access Denied	20	18/20	11/20	15/20	15/20	3/20	9/20
Security Policy Failure	20	18/20	11/20	15/20	14/20	8/20	12/20

Note: All categories contain 20 unique incidents assessed under each of the three modes. Category-level estimates are descriptive; no multiplicity-adjusted per-category hypothesis tests were conducted.

Table 5. Governance, selective outcomes, and uncertainty audit

Measure	MCP-enabled AI	Log-only AI	Manual triage
Human approval required	115	115	115
Policy-block-triggered incidents	15	0	0
Coded policy violations	0	0	0
Escalated incidents	57	142	96
Unresolved incidents	21	45	22
Rolled-back incidents	1	0	4
Recovery time observed (n)	227	125	190
Mean recovery time among observed (min)	37.01	62.09	104.10
14-day recurrence among observed	17/227 (7.49%)	22/125 (17.60%)	20/190 (10.53%)
Mean recorded diagnosis confidence	0.884	0.758	Not recorded
Brier score for recorded confidence	0.0436	0.2428	Not calculable
Incorrect predictions with $p \geq 0.80$	3/20	37/120	Not recorded

Note: Recovery time and recurrence are recorded only for successful-rerun observations and cannot be compared unconditionally across modes.