
Forward-Deployed Full-Stack Engineering for Autonomous Cloud MLOps

Sagar Srinivas Sakhinana, Venkataramana Runkana
sagar.sakhinana@tcs.com, venkat.runkana@tcs.com
Tata Research Development and Design Centre

Abstract

Across industries, machine-learning systems support applications ranging from prediction and anomaly detection to forecasting, optimization, and scheduling, yet operationalizing these systems requires coordinating application development, model pipelines, cloud infrastructure, security, deployment, monitoring, retraining, recovery, and rollback. We present an evidence-gated multi-agent framework for transforming a natural-language MLOps cloud engineering task into a verified repository and operational cloud deployment. The framework combines *graph engineering*, *loop engineering*, and *agent harness engineering*. A stateful Graph Orchestrator coordinates specialized agents for repository generation, review, execution, verification, release, and monitoring while governing workflow dependencies, evidence gates, retry bounds, recovery paths, and termination. Consequential lifecycle transitions proceed only when their required predicates are supported by verifiable execution or runtime evidence. Verification failures activate bounded reflection, repair, and re-verification, while runtime evidence of failure, drift, degradation, or policy violation can trigger bounded adaptation, recovery, or rollback. Agent harness engineering constrains repository generation, review, and repair, artifact execution, and cloud operations through controlled capabilities and isolated execution environments. We realize the framework on Google Cloud Platform and evaluate repository completeness, controlled execution, evidence-gated transitions, cloud promotion, and bounded recovery. Our experimental results show that the framework prevents unsupported lifecycle transitions and drives each run toward either a verified operational deployment or an auditable terminal failure.

1 Introduction

Industries ranging from oil and gas, pharmaceuticals, semiconductor manufacturing, energy, automotive, FMCG, aerospace, mining, metallurgy and so on increasingly depend on applications such as predictive maintenance, anomaly and defect detection, demand forecasting, quality prediction, process optimization, inventory planning, transport and distribution routing, scheduling and many others. Despite their application diversity, they share a common operational lifecycle spanning reproducible pipelines, cloud infrastructure provisioning, testing, security scanning, deployment, observability, retraining, failure recovery, auditability, and provenance (data lineage, model versions, and execution logs). This raises a broader question: how much of this lifecycle can be generated and operated by AI agents without trusting their claims of success? We argue that agent autonomy should instead depend on independently verifiable evidence from execution traces, deterministic compliance checks, deployment health, and runtime telemetry. Recent work has consequently shifted from isolated prompts toward the holistic engineering of agentic systems through structured execution environments, computation graphs, persistent context, bounded feedback loops, and trajectory-grounded evaluation [Hassan et al., 2025, Yue et al., 2026, Macedo, 2026b,a, Zhang et al., 2025, Anthropic, 2025a,b, 2026]. Nevertheless, prior work has largely focused on reasoning, code synthe-

sis, repository modification, graph/workflow optimization, or bounded task completion [Ding et al., 2026, Zhang et al., 2026, Zhou et al., 2026, Wang et al., 2026b, Rabanser et al., 2026, Wang et al., 2026a] rather than the end-to-end transformation of a natural-language long-horizon cloud engineering task - for example, “Build, deploy, and operate a time series anomaly-detection pipeline on cloud infrastructure using Kubernetes, with continuous monitoring, drift detection, automated retraining, autoscaling, and rollback”—into a verified operational deployment. An end-to-end MLOps system spans continuous integration, continuous delivery, and continuous training (CI/CD/CT) [Baylor et al., 2017, Kreuzberger et al., 2023]. CI covers data acquisition, validation, feature engineering, and evaluation; CT covers training, hyperparameter tuning, experiment tracking, model registration, drift detection, and retraining; and CD covers serving, monitoring, and retirement [Sculley et al., 2015, Baylor et al., 2017, Kreuzberger et al., 2023]. Cloud operationalization further requires infrastructure provisioning, identity and access management, secrets management, immutable artifact versioning, network isolation, Policy as Code (PaC) rollout, elastic scaling, observability, recovery, and rollback. We study autonomous cloud MLOps engineering through a representative cloud application comprising frontend and backend services. The frontend provides a secure web interface for user interaction and result visualization, while the backend exposes authenticated application and inference APIs and manages authorization, secrets, administrative operations, and audit retrieval. Shared telemetry, provenance, and audit records provide verifiable evidence across both tiers. AI Agents must build and test artifacts, scan source code and container images, publish immutable versions, provision infrastructure, reconcile deployment state, evaluate models, and verify releases before promotion. The lifecycle is cyclic and non-linear: production telemetry may trigger retraining, re-evaluation, staged rollout, failover, rollback, or retirement. Because these stages are interdependent, success at any individual stage does not establish that the resulting cloud system is secure, resilient, or operational. In agent-driven, long-horizon workflows, apparent completion may conceal stale state, failed handoffs, or policy violations that emerge only across multi-step execution and verification of the resulting cloud state [Bousetouane, 2026]. This motivates our central research question:

Can a multi-agent system reliably transform a natural-language MLOps cloud engineering task into a verified, operational cloud deployment while requiring verifiable evidence for every consequential lifecycle transition?

To address this question, we present an agentic AI framework that transforms an end-user cloud engineering task into a verified operational cloud deployment. The framework generates a deployable repository comprising application code, data-processing and training pipelines, tests, container definitions, CI/CD workflows, infrastructure as code, Helm charts, Kubernetes manifests, GitOps configurations, security policies, observability, retraining, recovery, and rollback artifacts. Generated artifacts are inspected and executed in controlled sandbox environments using dependency, syntax, testing, security, policy, provenance, and runtime checks. Artifacts satisfying these predicates are built, scanned, signed, published, deployed to staging, and subsequently verified against live cloud infrastructure before promotion. Verification failures produce execution evidence that drives bounded correction and re-execution, while consequential lifecycle transitions proceed only when their required evidence predicates are satisfied. We operationalize evidence-gated autonomous MLOps through three complementary mechanisms: graph engineering, loop engineering, and agent harness engineering. Graph engineering represents the lifecycle as a stateful execution graph in which a Graph Orchestrator manages workflow state, dependencies, evidence gates, retry bounds, recovery paths, and termination. The graph coordinates specialized agents for repository generation, repository review, execution, verification, release, and monitoring, with production promotion conditioned on verified evidence from the preceding lifecycle stages. Loop engineering defines bounded control cycles for correction and runtime adaptation. When verification fails, the resulting evidence is routed to a Reflection Agent for diagnosis and a Repair Agent for artifact correction; corrected artifacts then re-enter repository review, execution, and verification before deployment proceeds. Runtime evidence of failure, drift, degradation, or policy violation similarly activates reflection and repair, after which the corrected system re-enters the appropriate verification, deployment, and runtime-validation stages. Retry bounds and terminal conditions are enforced by the Graph Orchestrator. Agent harness engineering constrains agent access to tools, execution environments, and operational systems. Repository generation, review, reflection, and repair agents operate through controlled VS Code and Chrome sandbox environments; artifact execution occurs

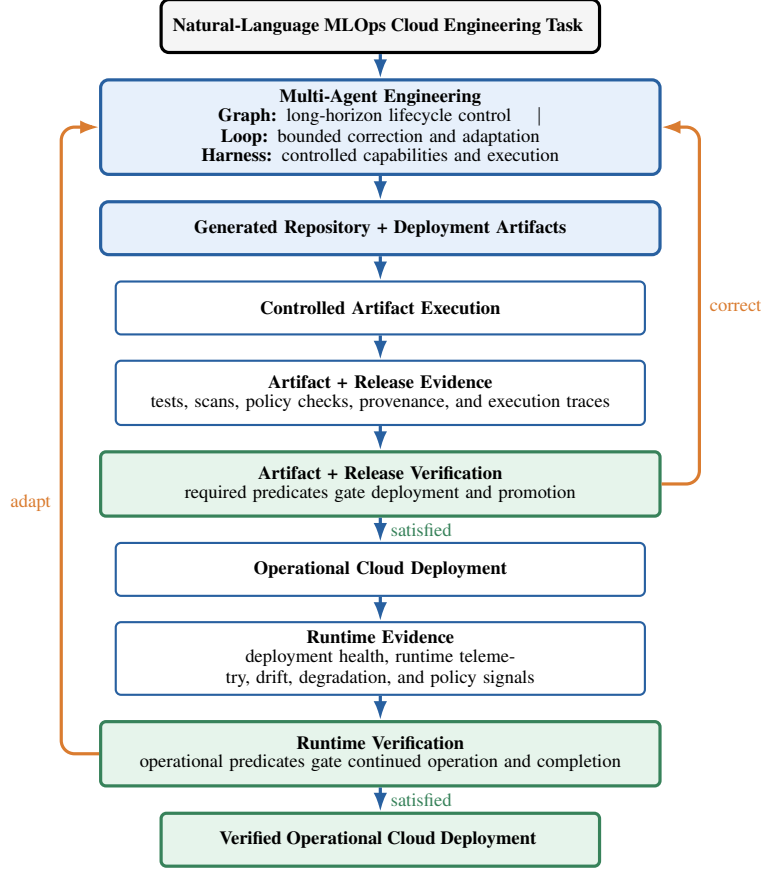


Figure 1: The figure shows how the evidence-gated autonomous cloud MLOps transforms a natural-language engineering task into a verified operational cloud deployment. Graph engineering governs long-horizon lifecycle, loop engineering provides bounded artifact correction and runtime adaptation, agent harness engineering constrains execution, and verified evidence gates consequential lifecycle transitions.

within a Google Kubernetes Engine(GKE) Sandbox with gVisor; and release and monitoring agents interact with controlled cloud infrastructure supporting staging, production, recovery, and rollback. Graph engineering [Zhuge et al., 2024, Zhang et al., 2024] structures the end-to-end cloud MLOps workflow and its transitions, loop engineering [Shinn et al., 2023, Gou et al., 2023] provides bounded correction and adaptation, and agent harness engineering [Zhong and Zhu, 2026, Lin et al., 2026] defines the controlled operational context in which agents act. Figure 1 presents the evidence-gated lifecycle from a natural-language MLOps cloud engineering task through artifact generation and execution, release, deployment, runtime verification, and verified operation, while Figure 2 presents its multi-agent realization through graph-controlled execution, bounded correction and adaptation, and controlled execution environments. Our work makes four contributions:

- We formulate autonomous cloud MLOps as transforming a natural-language task into a verified repository and operational deployment through evidence-gated lifecycle transitions.
- We introduce a stateful execution graph governed by a Graph Orchestrator that coordinates agents for repository generation, review, execution, verification, release, and monitoring while managing state, dependencies, evidence gates, retry bounds, recovery paths, and termination.
- We introduce bounded loop engineering for evidence-driven correction and runtime adaptation. Failed verification invokes reflection, diagnosis, repair, and re-verification, while runtime failures, drift, degradation, or policy violations trigger bounded recovery and adaptation.
- We realize agent harness engineering on Google Cloud Platform(GCP) through controlled execution environments: VS Code and Chrome sandboxes support repository generation,

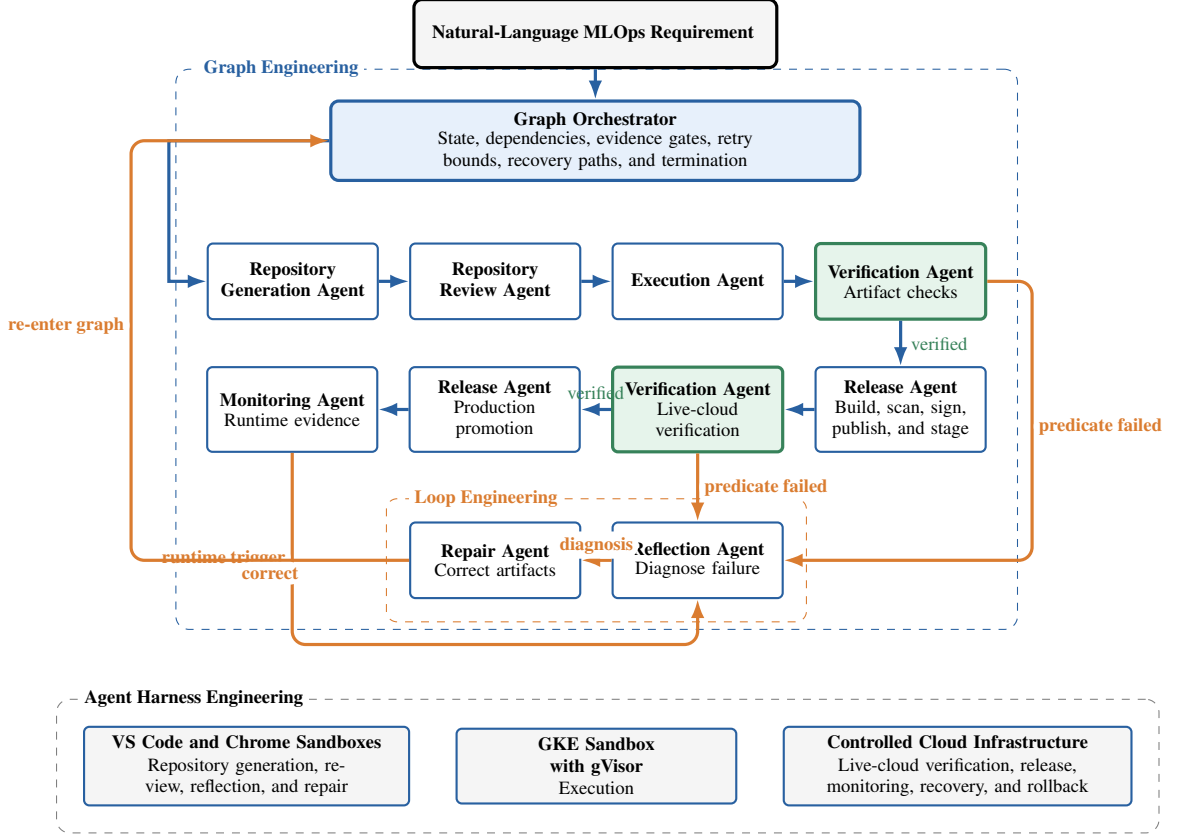


Figure 2: The figure shows the multi-agent architecture for evidence-gated autonomous cloud MLOps. The Graph Orchestrator manages workflow state, dependencies, evidence gates, retry bounds, recovery paths, and termination across specialized agents. Verification failures and adverse runtime evidence activate bounded reflection and repair. Agent Harness Engineering constrains agent access through controlled sandboxes and cloud infrastructure.

review, reflection, and repair; a gVisor-isolated GKE Sandbox supports artifact execution; and cloud infrastructure supports live-cloud verification, release, monitoring, recovery, and rollback. The generated repository includes application and pipeline code, tests, CI/CD, infrastructure as code, container and Kubernetes artifacts, GitOps configurations, security policies, provenance, observability, retraining, recovery, and rollback.

Our evaluation covers repository completeness, isolated execution, evidence-gated transitions, cloud promotion, and bounded recovery. We assess whether each run reaches a verified deployment or an auditable terminal failure. Our experimental results support the proposed framework and its underlying hypotheses, which we evaluate and discuss in the subsequent sections.

2 Problem Formulation

Let $\mathcal{Q}$ denote the set of admissible natural-language MLOps requirements, and let $q \in \mathcal{Q}$ denote the requirement for a single framework execution. The objective is to transform q into a verified repository and operational cloud deployment while permitting consequential lifecycle transitions only when supported by machine-checkable evidence. Let $\mathcal{S}$ denote the lifecycle-state space and $\mathcal{E}$ the set of controlled transitions in the execution graph, with $\mathcal{E}(s) \subseteq \mathcal{E}$ denoting the transitions that may be considered from lifecycle state s . At logical orchestration step t , the framework state is

$$X_t = (s_t, R_t, D_t, Z_t, B_t) \in \mathcal{S} \times \mathcal{R} \times \mathcal{D}_\perp \times \mathcal{Z} \times \mathcal{B}, \quad (1)$$

where s_t is the current lifecycle state, R_t is the repository state, D_t is the observed deployment state, Z_t is the available machine-checkable evidence, and B_t is the retry-budget assignment. Here,

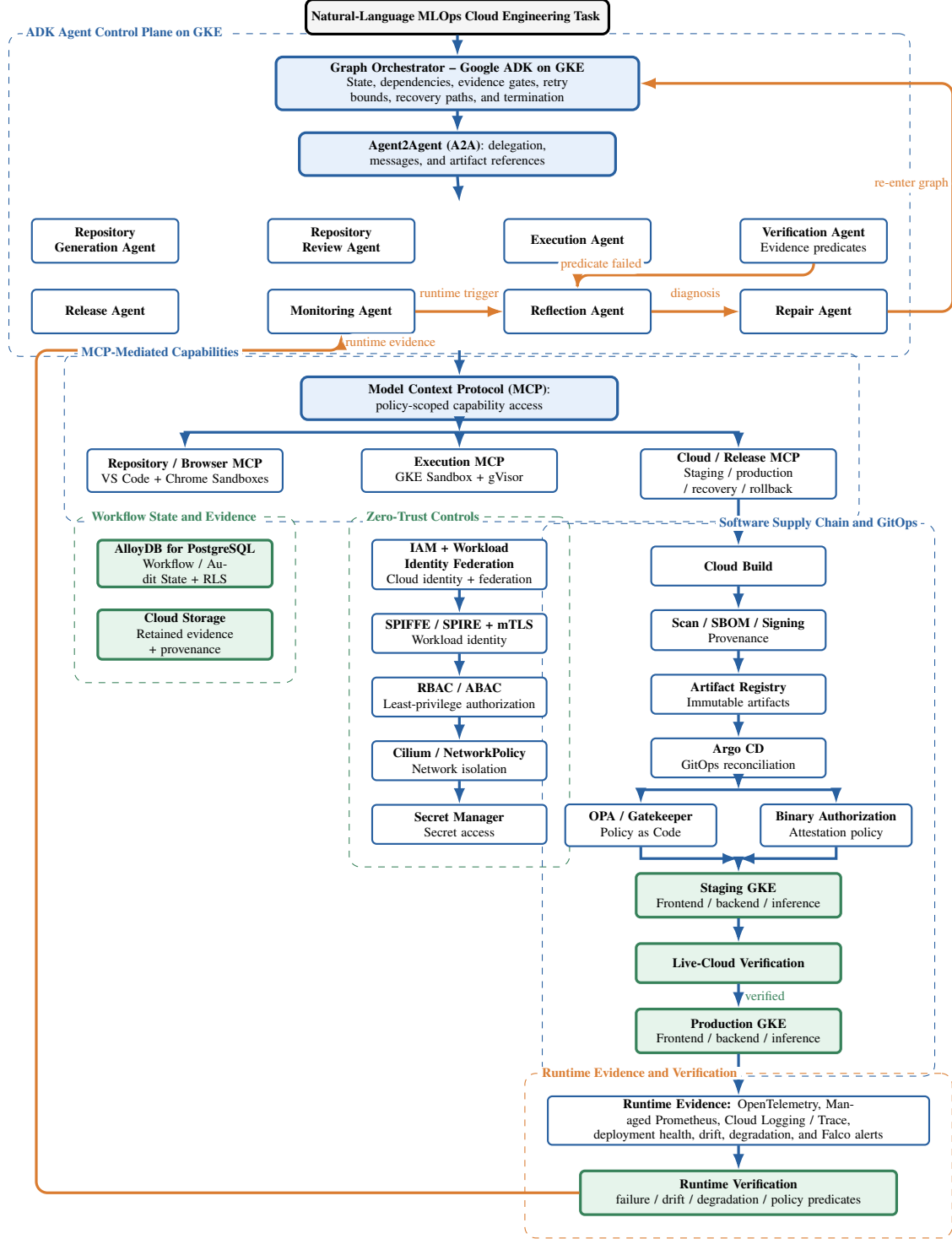


Figure 3: Google Cloud realization of the evidence-gated autonomous MLOps framework. The Graph Orchestrator, implemented with Google ADK on GKE, coordinates specialized agents through Agent2Agent (A2A), while the Model Context Protocol (MCP) provides policy-scoped access to repository and browser tools, isolated execution, and cloud deployment, release, recovery, and rollback operations.

$\mathcal{R}$ and $\mathcal{Z}$ denote the repository-state and evidence spaces, respectively, while $\mathcal{D}_{\perp} = \mathcal{D} \cup \perp$ denotes the deployment-state space augmented with $\perp$, indicating that no deployment has yet been

established. Evidence Z_t may include repository-validation results, test outcomes, security and policy verification results, model-evaluation results, artifact provenance, controlled-execution traces, deployment-verification results, and runtime telemetry. The retry assignment $B_t : \mathcal{E} \rightarrow \mathbb{N}_0$ associates each transition with its remaining retry budget, where $\mathbb{N}_0$ denotes the non-negative integers. Each transition $e \in \mathcal{E}$ is associated with a verification predicate $\phi_e : \mathcal{X} \rightarrow 0, 1$, where $\phi_e(X_t) = 1$ indicates that the state and evidence required for that transition satisfy its verification condition. We use Φ_{repo} , Φ_{release} , and Φ_{runtime} as named predicates for repository and artifact acceptance, controlled cloud release, and runtime acceptability, respectively. These predicates cover the corresponding repository, execution, release, deployment, and runtime evidence required by the lifecycle. Thus, lifecycle progression depends on verified system state rather than agent-reported completion. For an enabled transition $e \in \mathcal{E}(s_t)$, the Graph Orchestrator applies

$$T(X_t, e) = \begin{cases} X_{t+1}, & \phi_e(X_t) = 1, \\ X_{t+1}^{\text{corr}}, & \phi_e(X_t) = 0 \wedge B_t(e) > 0, \\ X_{t+1}^{\text{fail}}, & \phi_e(X_t) = 0 \wedge B_t(e) = 0. \end{cases} \quad (2)$$

where X_{t+1} denotes verified lifecycle progression, X_{t+1}^{corr} denotes entry into a graph-controlled corrective path, and X_{t+1}^{fail} denotes a terminal failure state. When correction is entered for transition e , its remaining retry budget is decremented as $B_{t+1}(e) = B_t(e) - 1$; non-retryable transitions are assigned zero retry budget. For a recoverable verification failure, let k denote the local correction attempt and $\delta_k \in \Delta$ its structured diagnosis, where Δ is the diagnosis space. The Reflection Agent and Repair Agent operate as $\delta_k = \text{Reflect}(R^{(k)}, D^{(k)}, Z^{(k)})$, $R^{(k+1)} = \text{Repair}(R^{(k)}, \delta_k)$, where $R^{(k)}$, $D^{(k)}$, and $Z^{(k)}$ denote the repository, deployment, and evidence states available at correction attempt k . The corrected repository re-enters the appropriate graph-controlled review, execution, and verification stages, producing fresh evidence $Z^{(k+1)}$. Correction continues only while the relevant verification predicate remains unsatisfied and retry budget remains; otherwise, the Graph Orchestrator follows the configured failure path. The same bounded mechanism applies to runtime failure, drift, degradation, or policy violation and may trigger reflection and repair, adaptation, recovery, or rollback followed by re-verification. Each framework execution therefore terminates in either a verified operational cloud deployment or a terminal failure state with an auditable reason.

3 Overall Framework

Figure 2 presents the agent-level realization of the lifecycle in Figure 1. The platform-independent graph, loop, and agent-harness abstractions are concretely realized on Google Cloud. The framework uses the Google Agent Development Kit (ADK) [Google, 2026a], with the Graph Orchestrator and specialized agents running as containerized workloads on Google Kubernetes Engine (GKE) [Google, 2026b]. Let $\mathcal{A} = \{A_{\text{gen}}, A_{\text{review}}, A_{\text{exec}}, A_{\text{verify}}, A_{\text{reflect}}, A_{\text{repair}}, A_{\text{release}}, A_{\text{monitor}}\}$ denote the specialized agents for repository generation, repository review, sandbox execution, verification, reflection, repair, release, and monitoring, respectively. The Graph Orchestrator, denoted by O_{graph} , coordinates these agents according to the execution state and evidence-gated transition rules defined in Section 2. The primary agent invocation path is $A_{\text{gen}} \rightarrow A_{\text{review}} \rightarrow A_{\text{exec}} \rightarrow A_{\text{verify}} \rightarrow A_{\text{release}} \rightarrow A_{\text{verify}} \rightarrow A_{\text{release}} \rightarrow A_{\text{monitor}}$. The first invocation of A_{verify} performs repository artifact verification, whereas the second performs live-cloud verification; similarly, the first invocation of A_{release} performs build and staging release, whereas the second performs production promotion. Verification failures enter the bounded correction path through A_{reflect} for diagnosis and A_{repair} for repository artifact correction. Corrected artifacts re-enter the appropriate graph-controlled review, execution, and verification stages, while failures requiring broader reconstruction may return to A_{gen} . Following repository artifact verification, A_{release} initiates the controlled release path. Cloud Build performs build and software-supply-chain operations, including source and container scanning, Software Bill of Materials (SBOM) generation, signing, and provenance generation. Artifact Registry stores immutable release artifacts, and Argo CD reconciles the desired deployment state to GKE. OPA/Gatekeeper enforces deployment Policy as Code, while Binary Authorization enforces artifact-attestation requirements before workload admission. Artifacts satisfying these controls are deployed to staging, subjected to live-cloud verification, and promoted to production only when the corresponding verification predicates are satisfied. Figure 3 presents the concrete Google Cloud realization. Agent2Agent (A2A) supports

delegation, messages, and artifact references between the Graph Orchestrator and specialized agents, while the Model Context Protocol (MCP) mediates policy-scoped capability access through three paths. *Repository / Browser MCP* provides access to the VS Code and Chrome Sandboxes for repository generation, review, reflection, and repair. *Execution MCP* provides access to the GKE Sandbox with gVisor for controlled artifact execution [Google Cloud, 2026a]. *Cloud / Release MCP* provides access to controlled Google Cloud infrastructure for live-cloud verification, release, monitoring, recovery, and rollback. The agent harness applies identity, authorization, network, credential, state, and observability controls across these execution paths. IAM and Workload Identity Federation provide cloud identity and federation; SPIFFE/SPIRE and mutual TLS (mTLS) establish workload identity and authenticated service communication; Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) restrict agent capabilities; Cilium and Kubernetes NetworkPolicy constrain network communication; and Secret Manager controls credential and secret access. AlloyDB for PostgreSQL persists workflow and audit state with row-level security (RLS) for multi-tenant isolation, while Cloud Storage retains execution evidence and provenance. Following production promotion, A_{monitor} receives runtime evidence from OpenTelemetry, Managed Prometheus, Cloud Logging and Trace, deployment-health telemetry, drift and degradation indicators, and Falco runtime-security alerts. The graph-controlled runtime verification path evaluates this evidence and activates bounded adaptation, recovery, rollback, re-verification, or termination when the corresponding runtime predicates are not satisfied.

4 Experiments

4.1 Experimental Methodology

We evaluate the framework through five research questions covering repository completeness and acceptance (RQ1), controlled artifact execution (RQ2), evidence-gated lifecycle progression (RQ3), cloud release and promotion (RQ4), and bounded recovery and termination (RQ5). **(a) RQ1: Repository Completeness and Acceptance.** Does the framework generate a complete deployable repository containing the required application code, pipeline, testing, container, CI/CD, infrastructure, Helm, Kubernetes, GitOps, security, provenance, observability, retraining, recovery, and rollback artifacts, and does the resulting repository satisfy the repository-verification predicate? **(b) RQ2: Controlled Artifact Execution.** Are executable repository artifacts processed through the GKE Sandbox with gVisor, with machine-checkable execution evidence produced and retained for verification? **(c) RQ3: Evidence-Gated Lifecycle Progression.** Does the Graph Orchestrator permit consequential forward lifecycle transitions only when their required verification predicates are satisfied and block unsupported progression when those predicates are unsatisfied? **(d) RQ4: Cloud Release and Promotion.** Do repository-verified artifacts progress through the software supply chain, artifact publication, staging deployment, cloud verification, and production promotion, with promotion permitted only after cloud verification succeeds? **(e) RQ5: Bounded Recovery and Termination.** Do repository, cloud, and runtime verification failures activate the appropriate graph-controlled correction, adaptation, recovery, or rollback path, require re-verification before further progression, and either resume progression within the applicable retry budget or terminate in an auditable failure when that budget is exhausted?

4.2 Experimental Scenarios

The experimental suite uses a single nominal scenario, three recoverable verification-perturbation scenarios, and one retry-budget-exhaustion scenario. All scenarios use the same execution graph, verification predicates, transition rules, retry semantics, recovery policies, and cloud configuration. Each perturbation introduces a controlled condition only at the targeted verification point required to exercise the corresponding lifecycle path. **(a) Nominal execution.** The framework executes without an induced verification failure. Successful execution requires a verified operational cloud deployment. Any non-induced recoverable verification failure must be corrected and successfully re-verified within the applicable retry budget. **(b) Repository-verification perturbation.** A controlled recoverable condition causing repository verification to fail is induced after controlled artifact execution. Release is blocked until the affected artifacts are repaired or regenerated and the required review, execution, and verification stages succeed. **(c) Cloud-verification perturbation.** A controlled recoverable condition causing cloud verification to fail is induced after staging deployment. Production promotion is blocked until the applicable correction or recovery path completes and cloud re-verification succeeds. **(d) Runtime-verification perturbation.** A controlled recoverable

runtime condition representing service failure, model drift, performance degradation, or policy violation is induced after production promotion. Continued verified operation requires the applicable adaptation, repair, recovery, or rollback path followed by successful runtime re-verification. **(e) Retry-budget exhaustion.** Each task is assigned a repository, cloud, or runtime verification target using a fixed model-independent assignment. The corresponding failure-inducing condition is maintained across correction or recovery attempts. Recovery continues while the applicable retry budget remains. If the required verification predicate remains unsatisfied when the budget reaches zero, the Graph Orchestrator blocks the corresponding forward progression and terminates the execution in an auditable terminal failure state.

4.3 Evaluation Metrics

We use one overall metric and metrics corresponding to RQ1–RQ5. All metrics lie in $[0, 1]$, with higher values indicating better performance, and are reported separately for each language model and experimental scenario where applicable. Table 1 summarizes the metrics and their interpretation.

Table 1: Summary of evaluation metrics.

Overall / RQ	Metric	Interpretation
Overall	Verified Operational Deployment Rate (VODR)	Did the entire execution end in a verified operational cloud deployment, with $\Phi_{\text{repo}} = 1$, $\Phi_{\text{release}} = 1$, and $\Phi_{\text{runtime}} = 1$?
RQ1	Repository Completeness Score (RCS)	How complete is the generated repository in terms of the required artifact categories?
RQ1	Repository Acceptance Rate (RAR)	Did the repository ultimately satisfy Φ_{repo} after bounded correction and re-verification?
RQ2	Controlled Execution Rate (CER)	Were executable repository artifacts processed through the GKE Sandbox with gVisor and accompanied by retained machine-checkable execution evidence?
RQ3	Verified Progression Rate (VPR)	When a forward transition’s verification predicate was satisfied, did the Graph Orchestrator permit progression?
RQ3	Blocked Forward Progression Rate (BFR)	When a forward transition’s verification predicate was unsatisfied, did the Graph Orchestrator block progression?
RQ4	Release and Promotion Success Rate (RPSR)	After repository verification succeeded, did the execution complete the controlled release path through staging, cloud verification, and production promotion?
RQ5	Recovery Success Rate (RSR)	After a recoverable verification failure, did the framework recover, re-verify the affected predicate within the retry budget, and resume lifecycle progression?
RQ5	Exhaustion Termination Rate (ETR)	When the targeted predicate remained unsatisfied through the retry budget, did the framework block progression and terminate in an auditable terminal failure state?

4.4 Experimental Setup

We evaluate the framework on a custom benchmark of 100 distinct natural-language cloud MLOps engineering tasks using the Google Cloud Platform (GCP) configuration described in Section 3. The representative tasks and benchmark details are provided in the Technical Appendix (Section 6.2). Existing benchmarks cover ML related software-engineering tasks, but not end-to-end autonomous cloud MLOps. We therefore construct a 100-task benchmark spanning diverse datasets, applications, modalities, and ML workloads. We compare four language-model backbones: Gemini 2.5 Pro, Gemini 2.5 Flash, Gemini 2.5 Flash-Lite, and GPT-5.6 Sol. Within each execution, the same model is used for all agent functions requiring language-model inference. Each task is evaluated with every model under the five experimental scenarios in Section 4.2, resulting in $100 \times 4 \times 5 = 2,000$ framework executions. Agent prompts, execution graphs, verification predicates, recovery policies, framework implementation, and cloud configuration are held fixed across models. For each experimental scenario, the corresponding perturbation specification is also held fixed across models. No model-specific prompt tuning or verification logic is used. Each retryable transition permits at most 20 correction or recovery attempts. Each execution is additionally limited to 120 minutes of wall-clock time, 10,000,000 aggregate model tokens, and 500 tool calls. We record verification

outcomes, lifecycle transitions, correction and recovery attempts, tool calls, token usage, execution time, retained evidence, and terminal state. Google Cloud infrastructure and Vertex AI Gemini inference were covered by eligible multiple Google Cloud free-tier credits. Thus the direct experimental expenditure was therefore limited to GPT-5.6 Sol inference through the OpenAI API.

4.5 Results and Analysis

Experimental results are reported for GPT-5.6 Sol, Gemini 2.5 Pro, Gemini 2.5 Flash, and Gemini 2.5 Flash-Lite across the experimental scenarios described in Section 4.2. Each model–scenario pair comprises 100 executions. VODR, RCS, and RAR are computed over all executions in the scenarios for which each metric is defined; CER is evaluated over executions that reach controlled artifact execution, and RPSR over those that satisfy repository verification. RSR is measured over recoverable verification failures, whereas ETR is measured over retry-budget-exhaustion executions. VPR and BFR are calculated over lifecycle-gate evaluations for which the relevant verification predicate is satisfied and unsatisfied, respectively. The Repository Verification, Cloud Verification, and Runtime Verification columns denote perturbation scenarios with induced failures at the lifecycle gates governed by Φ_{repo} , Φ_{release} , and Φ_{runtime} , respectively. Table 2 reports VODR, the fraction of executions whose terminal state satisfies Φ_{repo} , Φ_{release} , and Φ_{runtime} . GPT-5.6 Sol achieves the highest VODR in every reported scenario, followed consistently by Gemini 2.5 Pro, Gemini 2.5 Flash, and Gemini 2.5 Flash-Lite. For all models, VODR decreases under each perturbation relative to nominal execution, consistent with the additional correction or recovery and re-verification required after an induced failure. The framework must produce fresh evidence and re-satisfy the affected verification predicate before lifecycle progression or continued verified operation can resume. RQ5 examines this behavior through RSR, which measures successful recovery within the retry budget, and ETR, which measures correct termination after budget exhaustion. VODR is not reported for retry-budget exhaustion because the failure condition is intentionally maintained until the budget is exhausted, making auditable terminal failure the expected outcome. The remaining results are presented and discussed in the Technical Appendix.

Table 2: Verified Operational Deployment Rate (VODR) across experimental scenarios.

Model	Nominal	Repository Verification	Cloud Verification	Runtime Verification
GPT-5.6 Sol	0.99	0.95	0.96	0.94
Gemini 2.5 Pro	0.78	0.62	0.65	0.59
Gemini 2.5 Flash	0.52	0.38	0.41	0.35
Gemini 2.5 Flash-Lite	0.38	0.24	0.27	0.21

The key result (see table 7) is that the framework enables effective recovery from repository, cloud, and runtime failures while reliably enforcing termination when recovery is exhausted, demonstrating robust bounded autonomy in real-world agentic workflows. These results support our framework’s central novelty by decoupling model-independent verification enforcement from model-dependent recovery capability, thereby enabling bounded, evidence-gated recovery for long-horizon agentic workflows.

5 Conclusion

We presented an evidence-gated multi-agent framework for autonomous cloud MLOps that combines graph-controlled lifecycle orchestration, bounded correction and recovery, and controlled agent execution. The experimental results support the central hypothesis that multi-step, long-horizon MLOps workflows can be governed by machine-checkable evidence: verified transitions proceed, unsupported transitions are blocked, and recoverable or unresolved failures are handled through bounded re-verification or auditable termination. The current evaluation focuses on a representative frontend–backend cloud application rather than a complete production-grade cloud stack. Future work will extend the framework to cover Domain Name System (DNS) configuration and custom domains, external HTTPS load balancing and Transport Layer Security (TLS), Web Application Firewall (WAF) protection, Content Delivery Network (CDN)-backed frontend delivery, distributed rate limiting, and more comprehensive audit analytics. Further evaluation across application architectures, workloads, cloud platforms, and failure modes is needed to assess the generalizability of the framework.

References

- A2A Protocol Working Group. Agent2Agent (A2A) Protocol Specification. <https://a2a-protocol.org/latest/specification/>, 2026. Version 1.0.0; accessed: 2026-08-12.
- Sreemae Akshathala, Bassam Adnan, Mahisha Ramesh, Karthik Vaidhyanathan, Basil Muhammed, and Kannan Parthasarathy. Beyond task completion: An assessment framework for evaluating agentic AI systems. *arXiv preprint arXiv:2512.12791*, 2025. doi: 10.48550/arXiv.2512.12791. URL <https://arxiv.org/abs/2512.12791>.
- Anthropic. Effective context engineering for AI agents. <https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents>, 2025a. Accessed: 2026-08-12.
- Anthropic. How we built our multi-agent research system. <https://www.anthropic.com/engineering/multi-agent-research-system>, 2025b. Accessed: 2026-08-12.
- Anthropic. Demystifying evals for AI agents. <https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents>, 2026. Accessed: 2026-08-12.
- Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin A. Zinkevich. TFX: A TensorFlow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395. Association for Computing Machinery, 2017. doi: 10.1145/3097983.3098021. URL <https://doi.org/10.1145/3097983.3098021>.
- Fouad Boussetouane. Human-on-the-bridge: Scalable evaluation for AI agents. *arXiv preprint arXiv:2606.16871*, 2026. doi: 10.48550/arXiv.2606.16871. URL <https://arxiv.org/abs/2606.16871>.
- Jingzhe Ding et al. NL2Repo-Bench: Towards long-horizon repository generation evaluation of coding agents. *arXiv preprint arXiv:2512.12730*, 2026. <https://arxiv.org/abs/2512.12730>.
- Google. Agent Development Kit (ADK). <https://adk.dev/>, 2026a. Accessed: 2026-08-12.
- Google. Deploying Agent Development Kit (ADK) applications to Google Kubernetes Engine. <https://adk.dev/deploy/gke/>, 2026b. Accessed: 2026-08-12.
- Google Cloud. About GKE Agent Sandbox. <https://cloud.google.com/kubernetes-engine/docs/concepts/machine-learning/agent-sandbox>, 2026a. Accessed: 2026-08-12.
- Google Cloud. Enable Agent Sandbox on GKE. <https://cloud.google.com/kubernetes-engine/docs/how-to/how-install-agent-sandbox>, 2026b. Accessed: 2026-08-12.
- Google Cloud. GKE Autopilot Overview. <https://cloud.google.com/kubernetes-engine/docs/concepts/autopilot-overview>, 2026c. Accessed: 2026-08-12.
- Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023. doi: 10.48550/arXiv.2305.11738. URL <https://arxiv.org/abs/2305.11738>.
- Ahmed E. Hassan, Hao Li, Dayi Lin, Bram Adams, Tse-Hsun Chen, Yutaro Kashiwa, and Dong Qiu. Agentic software engineering: Foundational pillars and a research roadmap. *arXiv preprint arXiv:2509.06216*, 2025. doi: 10.48550/arXiv.2509.06216. URL <https://arxiv.org/abs/2509.06216>.

- Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiwu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=VtmBAGCN7o>.
- Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. Machine learning operations (MLOps): Overview, definition, and architecture. *IEEE Access*, 11:31866–31879, 2023. doi: 10.1109/ACCESS.2023.3262138. URL <https://doi.org/10.1109/ACCESS.2023.3262138>.
- Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shiha Dou, Xuanjing Huang, Hang Yan, Zhenhua Han, and Tao Gui. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026. doi: 10.48550/arXiv.2604.25850. URL <https://arxiv.org/abs/2604.25850>.
- Sandeco Macedo. Stop hand-holding your coding agent: Engineering the loops that replace step-by-step prompting. *arXiv preprint arXiv:2607.00038*, 2026a. doi: 10.48550/arXiv.2607.00038. URL <https://arxiv.org/abs/2607.00038>.
- Sandeco Macedo. What makes prompts a graph: Necessary and sufficient conditions for prompt graph engineering. *arXiv preprint arXiv:2607.27578*, 2026b. doi: 10.48550/arXiv.2607.27578. URL <https://arxiv.org/abs/2607.27578>.
- Model Context Protocol Project. Model Context Protocol Specification. <https://modelcontextprotocol.io/specification/2026-07-28>, July 2026. Protocol revision 2026-07-28; accessed: 2026-08-12.
- Stephan Rabanser, Sayash Kapoor, Peter Kirgis, Kangheng Liu, Saiteja Utpala, and Arvind Narayanan. Towards a science of AI agent reliability. *arXiv preprint arXiv:2602.16666*, 2026. <https://arxiv.org/abs/2602.16666>.
- D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems*, volume 28, pages 2503–2511, 2015. URL <https://proceedings.neurips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems>.
- Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. doi: 10.48550/arXiv.2303.11366. URL <https://arxiv.org/abs/2303.11366>.
- Leyao Wang, Yanan He, Peng Chen, Asaf Yehudai, Yixin Liu, Rex Ying, Michal Shmueli-Scheuer, and Arman Cohan. Time to reflect: Can we trust LLM judges for evidence-based research agents? *arXiv preprint arXiv:2605.19196*, 2026a. <https://arxiv.org/abs/2605.19196>.
- Yiqi Wang, Jiaqi Zhang, Taotao Cai, Zirui Liu, Qingqiang Sun, Zequn Sun, Zhangkai Wu, Manqing Dong, Mingkai Zheng, Xuefei Yin, and Yanming Zhu. From agent traces to trust: A survey of evidence tracing and execution provenance in LLM agents. *arXiv preprint arXiv:2606.04990*, 2026b. <https://arxiv.org/abs/2606.04990>.
- John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1601.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

- Ling Yue, Kushal Raj Bhandari, Ching-Yun Ko, Dhaval Patel, Shuxin Lin, Nianjun Zhou, Jianxi Gao, Pin-Yu Chen, and Shaowu Pan. From static templates to dynamic runtime graphs: A survey of workflow optimization for LLM agents. *arXiv preprint arXiv:2603.22386*, 2026. doi: 10.48550/arXiv.2603.22386. URL <https://arxiv.org/abs/2603.22386>.
- Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024. doi: 10.48550/arXiv.2410.10762. URL <https://arxiv.org/abs/2410.10762>.
- Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025. doi: 10.48550/arXiv.2510.04618. URL <https://arxiv.org/abs/2510.04618>.
- Zhaoxi Zhang, Yiming Xu, Jiahui Liang, Weikang Li, Xiaoshuai Chen, Liwei Qian, Xin Pei, Jizhou Huang, Rui Sun, and Yunfang Wu. RepoZero: Can LLMs generate a code repository from scratch? *arXiv preprint arXiv:2605.07122*, 2026. <https://arxiv.org/abs/2605.07122>.
- Hailin Zhong and Shengxin Zhu. AI harness engineering: A runtime substrate for foundation-model software agents. *arXiv preprint arXiv:2605.13357*, 2026. doi: 10.48550/arXiv.2605.13357. URL <https://arxiv.org/abs/2605.13357>.
- Jiajun Zhou, Zhaoxuan Ke, JiHang Ye, Xuanze Chen, Shanqing Yu, and Qi Xuan. AgentS4D: Benchmarking runtime risks across the execution lifecycle of LLM-based workspace agents. *arXiv preprint arXiv:2607.27294*, 2026. <https://arxiv.org/abs/2607.27294>.
- Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*, 2024. doi: 10.48550/arXiv.2402.16823. URL <https://arxiv.org/abs/2402.16823>.

6 Technical Appendix

6.1 Additional Results

(a) RQ1: Repository Completeness and Acceptance. Table 3 reports the Repository Completeness Score (RCS) and Repository Acceptance Rate (RAR). RCS remains high across models and scenarios, whereas RAR varies substantially more, showing that structural repository completeness does not by itself imply satisfaction of Φ_{repo} . GPT-5.6 Sol achieves the highest RCS and RAR in all reported scenarios, while the larger RCS–RAR gaps for the Gemini models indicate that artifact-category coverage and repository acceptance remain distinct properties.

Table 3: Repository completeness and acceptance across experimental scenarios.

Model	Metric	Nominal	Repository Verification	Cloud Verification	Runtime Verification	Retry-Budget Exhaustion
GPT-5.6 Sol	RCS	0.99	0.99	0.99	0.99	0.98
	RAR	0.99	0.97	0.99	0.99	0.56
Gemini 2.5 Pro	RCS	0.97	0.95	0.96	0.96	0.94
	RAR	0.82	0.67	0.79	0.77	0.42
Gemini 2.5 Flash	RCS	0.93	0.91	0.92	0.92	0.90
	RAR	0.62	0.47	0.60	0.57	0.36
Gemini 2.5 Flash-Lite	RCS	0.89	0.86	0.88	0.87	0.85
	RAR	0.46	0.30	0.43	0.39	0.28

(b) RQ2: Controlled Artifact Execution. Table 4 reports the Controlled Execution Rate (CER). Among executions that reach controlled artifact execution, CER is 1.00 in all reported cases, indicating that executable repository artifacts are processed through the GKE Sandbox with gVisor and accompanied by retained machine-checkable execution evidence.

Table 4: Controlled Execution Rate (CER) across experimental scenarios.

Model	Nominal	Repository Verification	Cloud Verification	Runtime Verification	Retry-Budget Exhaustion
GPT-5.6 Sol	1.00	1.00	1.00	1.00	1.00
Gemini 2.5 Pro	1.00	1.00	1.00	1.00	1.00
Gemini 2.5 Flash	1.00	1.00	1.00	1.00	1.00
Gemini 2.5 Flash-Lite	1.00	1.00	1.00	1.00	1.00

(c) RQ3: Evidence-Gated Lifecycle Progression. Table 5 reports the Verified Progression Rate (VPR) and Blocked Forward Progression Rate (BFR). Both are 1.00 in all reported cases. The forward progression occurs when the applicable verification predicate is satisfied, while progression is blocked when the predicate is unsatisfied. These results show that the Graph Orchestrator consistently enforces the evidence-gated transition rule defined in Section 2.

(d) RQ4: Cloud Release and Promotion. Table 6 reports the Release and Promotion Success Rate (RPSR), computed over executions that satisfy repository verification. GPT-5.6 Sol achieves the highest RPSR in every reported scenario. The cloud-verification perturbation produces the largest reduction for the Gemini models, consistent with its direct targeting of a verification gate within the release path. The runtime perturbation is introduced only after production promotion; therefore, RPSR in that scenario reflects release outcomes obtained before the induced runtime failure and should not be interpreted as an effect of the runtime perturbation itself.

(e) RQ5: Bounded Recovery and Termination. Table 7 reports Recovery Success Rate (RSR) for the three recoverable verification targets and Exhaustion Termination Rate (ETR) for retry-budget exhaustion. GPT-5.6 Sol achieves the highest RSR at each recoverable target. For the Gemini models, RSR is lowest for repository-verification failures, while GPT-5.6 Sol remains near ceiling

Table 5: Verified Progression Rate (VPR) and Blocked Forward Progression Rate (BFR) across experimental scenarios.

Model	Metric	Nominal	Repository Verification	Cloud Verification	Runtime Verification	Retry-Budget Exhaustion
GPT-5.6 Sol	VPR	1.00	1.00	1.00	1.00	1.00
	BFR	1.00	1.00	1.00	1.00	1.00
Gemini 2.5 Pro	VPR	1.00	1.00	1.00	1.00	1.00
	BFR	1.00	1.00	1.00	1.00	1.00
Gemini 2.5 Flash	VPR	1.00	1.00	1.00	1.00	1.00
	BFR	1.00	1.00	1.00	1.00	1.00
Gemini 2.5 Flash-Lite	VPR	1.00	1.00	1.00	1.00	1.00
	BFR	1.00	1.00	1.00	1.00	1.00

Table 6: Release and Promotion Success Rate (RPSR) across experimental scenarios.

Model	Nominal	Repository Verification	Cloud Verification	Runtime Verification
GPT-5.6 Sol	1.00	0.99	0.98	0.98
Gemini 2.5 Pro	0.98	0.96	0.84	0.91
Gemini 2.5 Flash	0.90	0.87	0.70	0.79
Gemini 2.5 Flash-Lite	0.89	0.87	0.65	0.77

across all three targets. This pattern is consistent with the broader artifact surface covered by Φ_{repo} , although the results do not isolate the source of the recovery difficulty. For executions that reach the assigned retry-exhaustion target, ETR is 1.00 for every model, showing that the Graph Orchestrator blocks forward progression and terminates when the targeted predicate remains unsatisfied after exhaustion of the applicable retry budget. Together, RSR and ETR show that model capability primarily affects successful recovery within the retry budget, whereas retry-budget exhaustion is handled consistently by the Graph Orchestrator.

Table 7: Recovery and retry-budget-exhaustion outcomes by verification target.

Verification Target	Measure	GPT-5.6 Sol	Gemini 2.5 Pro	Gemini 2.5 Flash	Gemini 2.5 Flash-Lite
Repository Verification	Reached	99	96	94	88
	RSR	0.98	0.70	0.50	0.34
Cloud Verification	Reached	99	77	57	40
	RSR	0.98	0.86	0.74	0.70
Runtime Verification	Reached	97	70	45	30
	RSR	0.97	0.84	0.78	0.70
Retry-Budget Exhaustion	Reached	94	76	64	55
	ETR	1.00	1.00	1.00	1.00

6.2 Representative Benchmark Tasks

The benchmark comprises 100 distinct natural-language cloud MLOps engineering tasks instantiated from established public datasets and spanning classification, regression, forecasting, computer vision, natural-language processing, speech, recommendation, anomaly detection, sensor analytics, and equipment-health applications. Each task specifies a two-tier application with a frontend for user interaction and result visualization and a backend providing authenticated application and inference APIs. Tasks are grouped into *easy*, *medium*, and *hard* tiers according to end-to-end engineering complexity, including data processing, model workflow, application functionality, pipeline construction, deployment, monitoring, retraining, recovery, and rollback. All tasks use the same execution graph, verification predicates, recovery semantics, and GCP configuration described in

Table 8: Representative tasks from the 100-task cloud MLOps benchmark.

Level	Dataset	Application	Two-Tier Application	ML Workflow
Easy	MNIST	Digit classification	Image-upload frontend with predicted-digit visualization; backend for image preprocessing and authenticated inference.	Image classification, reproducible training, evaluation, monitoring, and retraining.
Easy	California Housing	House-value regression	Feature-input frontend with predicted-value visualization; backend for feature preprocessing and authenticated inference.	Tabular preprocessing, regression training, evaluation, monitoring, and retraining.
Easy	IMDB Large Movie Review	Sentiment classification	Text-input frontend with sentiment visualization; backend for text preprocessing and authenticated inference.	Text preprocessing, sentiment classification, evaluation, monitoring, and retraining.
Medium	CIFAR-10	Image classification	Image-upload frontend with class visualization; backend for image preprocessing and authenticated inference.	Vision-model training, augmentation, evaluation, serving, monitoring, and retraining.
Medium	UCI Bike Sharing	Demand forecasting	Temporal-input frontend with demand-forecast visualization; backend for temporal feature processing and authenticated forecasting.	Temporal feature construction, forecasting, evaluation, monitoring, and scheduled retraining.
Medium	UCI Human Activity Recognition	Activity recognition	Sensor-input frontend with activity visualization; backend for accelerometer and gyroscope preprocessing and authenticated inference.	Multivariate sensor preprocessing, activity classification, evaluation, monitoring, and retraining.
Medium	Speech Commands	Keyword classification	Audio-upload frontend with predicted-command visualization; backend for audio preprocessing, feature extraction, and authenticated inference.	Audio preprocessing, keyword classification, evaluation, monitoring, and retraining.
Hard	Oxford-IIIT Pet	Image segmentation	Image-upload frontend with segmentation-mask visualization; backend for image preprocessing and authenticated segmentation inference.	Pixel-level segmentation, training, evaluation, serving, monitoring, and retraining.
Hard	MovieLens	Recommendation	Interactive frontend with ranked recommendations; backend for user-item processing, ranking, and authenticated recommendation APIs.	User-item modeling, ranking, offline evaluation, serving, monitoring, and scheduled retraining.
Hard	NASA C-MAPSS Turbofan Degradation	Remaining-useful-life estimation	Equipment-health frontend with remaining-life visualization; backend for multivariate simulated sensor-sequence processing and authenticated inference.	Sequence preprocessing, remaining-life estimation, evaluation, degradation monitoring, and retraining.
Hard	Server Machine Dataset (SMD)	Anomaly detection	Time-series frontend with anomaly visualization; backend for multivariate telemetry processing and authenticated anomaly scoring.	Multivariate time-series processing, anomaly detection, evaluation, runtime monitoring, and retraining.

Section 3. Table 8 shows representative examples. Each task is evaluated unchanged under the five experimental scenarios in Section 4.2, with scenario-specific perturbations introduced only by the experimental procedure.

6.3 Ablation Studies

We evaluate GPT-5.6 Sol on the same 100 benchmark tasks to isolate the effects of evidence-gated progression and retry-enabled correction and recovery defined in Section 2, and to assess sensitivity to the retry budget. The ablations use the repository-, cloud-, and runtime-verification perturbation scenarios in Section 4.2, which directly exercise the corresponding verification and recovery paths. Unless modified by an ablation, agent prompts, verification predicates, perturbation specifications, recovery policies, cloud configuration, and execution limits remain unchanged from Section 4.4. The Full Framework results in Section 4.5 provide the baseline.

Table 9: Ablation and retry-budget configurations.

Variant	Modification
Full Framework	Evidence-gated transitions, graph-controlled correction and recovery, re-verification, and retry-budget semantics remain unchanged.
Evidence-Gate Bypass	Verification predicates are evaluated, but their outcomes do not gate enabled forward transitions. An unsatisfied predicate therefore does not block progression or enter the associated corrective or terminal-failure path.
Zero Retry Budget	All transitions that are retryable in the Full Framework are assigned zero retry budget. Evidence gating remains active, but an unsatisfied predicate causes terminal failure without entering the graph-controlled correction or recovery path.
Retry-Budget Sensitivity	The retry budget for each retryable transition is varied over $\{1, 3, 5, 10, 20\}$ while evidence gating and all other framework mechanisms remain unchanged.

We report VODR, BFR, and RSR using the definitions and evaluation scopes in Section 4.3. VODR is computed over all executions in each perturbation scenario, BFR over forward-transition evaluations with an unsatisfied verification predicate, and RSR over recoverable verification failures.

Table 10: Primary ablation results for GPT-5.6 Sol.

Variant	VODR Repository	VODR Cloud	VODR Runtime	BFR Repository	BFR Cloud	BFR Runtime	RSR Repository	RSR Cloud	RSR Runtime
Full Framework	0.95	0.96	0.94	1.00	1.00	1.00	0.98	0.98	0.97
Evidence-Gate Bypass	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Zero Retry Budget	0.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00

Evidence-Gate Bypass reduces BFR to 0.00 because an unsatisfied predicate no longer blocks the corresponding forward transition. Since the associated corrective path is not entered, the induced failure is not corrected and re-verified, yielding RSR of 0.00. The affected predicate therefore remains unsatisfied at termination, and VODR is 0.00 in all three perturbation scenarios. Under Zero Retry Budget, evidence gating remains active and BFR remains 1.00, but an induced verification failure cannot enter a correction or recovery attempt. The execution therefore terminates at the failed gate, yielding RSR and VODR of 0.00. We next vary the retry budget under the same three recoverable perturbation scenarios. A budget of 20 corresponds to the Full Framework configuration in Section 4.4.

Table 11: Retry-budget sensitivity for GPT-5.6 Sol.

Retry Budget	VODR Repository	VODR Cloud	VODR Runtime	RSR Repository	RSR Cloud	RSR Runtime
1	0.74	0.77	0.71	0.75	0.78	0.74
3	0.87	0.89	0.83	0.88	0.90	0.86
5	0.93	0.94	0.89	0.94	0.95	0.92
10	0.95	0.96	0.93	0.96	0.97	0.96
20	0.95	0.96	0.94	0.98	0.98	0.97

VODR and RSR are non-decreasing with the retry budget, with the largest gains at smaller budgets. Runtime RSR increases from 0.74 with one permitted attempt to 0.92 with five and 0.97 with twenty, with similar behavior for repository and cloud verification. The retry budget defines the maximum number of correction or recovery attempts; successful re-verification terminates the corresponding recovery path before the budget is exhausted. Together, the ablations show that evidence gating prevents unsupported forward progression, while a positive retry budget enables bounded correction or recovery before terminal failure. Increasing the retry budget improves recovery and verified operational deployment, with limited additional gains beyond ten attempts.

6.4 Lifecycle and State Management

Figure 4 summarizes the evidence-gated lifecycle from a natural-language MLOps cloud engineering task to a verified operational cloud deployment. Generated repository and deployment artifacts undergo controlled execution, artifact and release verification, deployment, and runtime verification. Predicate satisfaction permits forward progression; failed artifact or release verification enters the correction path, whereas failed runtime verification enters the adaptation path, with both returning to multi-agent engineering for subsequent re-verification. Persistent workflow and audit state is maintained in AlloyDB for PostgreSQL, with PostgreSQL row-level security providing tenant isolation (Figure 5). The corresponding execution, release, security, and runtime mechanisms are described in Section 6.5.

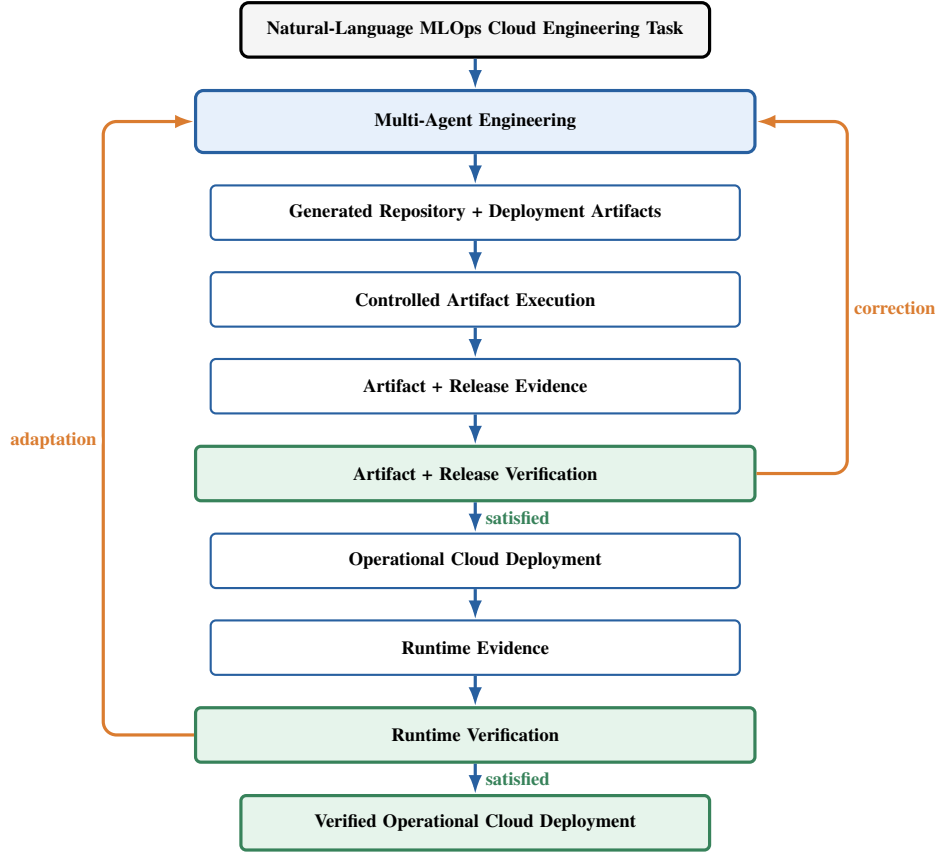


Figure 4: High-level lifecycle of the evidence-gated autonomous cloud MLOps framework. A natural-language MLOps cloud engineering task progresses through multi-agent engineering, artifact generation and controlled execution, artifact and release verification, cloud deployment, and runtime verification. Failed artifact or release verification returns to multi-agent engineering through the correction path, while runtime verification can return through the adaptation path. The lifecycle is therefore cyclic rather than a one-shot CI/CD pipeline.

6.5 Operational Realization

Figures 9–13 detail the mechanisms implementing the lifecycle. Figure 9 separates regular GKE agent workloads from isolated artifact execution: the Execution Agent invokes the Execution MCP, generated artifacts execute in the gVisor-isolated GKE Sandbox, and execution evidence is evaluated by the Verification Agent. Figure 10 shows the verified-artifact path through Cloud Build, source and container scanning, SBOM generation, signing, provenance generation, and Artifact Registry. Figure 11 details Argo CD reconciliation, OPA/Gatekeeper policy enforcement, Binary Authorization, staging deployment, live-cloud verification, and evidence-gated production promotion. Figure 12 summarizes cloud and workload identity, authentication, authorization, network isolation,

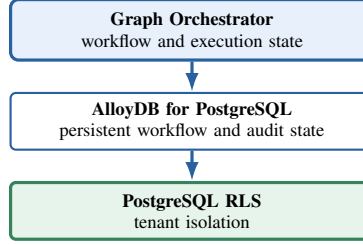


Figure 5: Persistent workflow and audit state using AlloyDB for PostgreSQL with row-level security for tenant isolation.

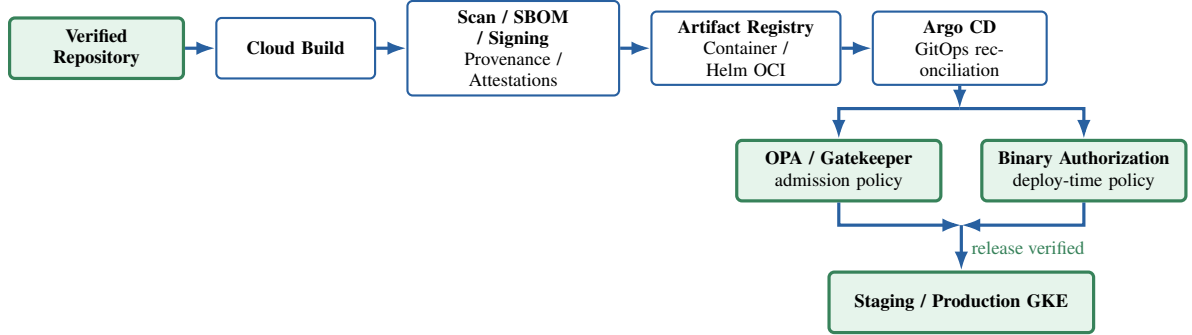


Figure 6: Controlled software-supply-chain and release path through build, artifact publication, GitOps reconciliation, policy enforcement, and deployment to GKE.

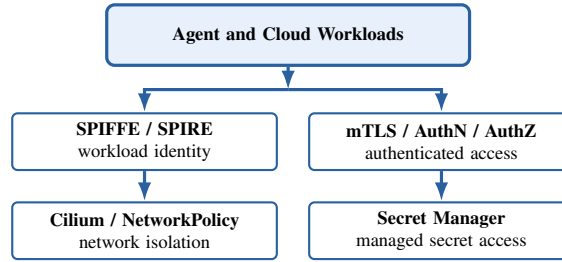


Figure 7: Complementary security controls for workload identity, authenticated communication, network isolation, and managed secret access.

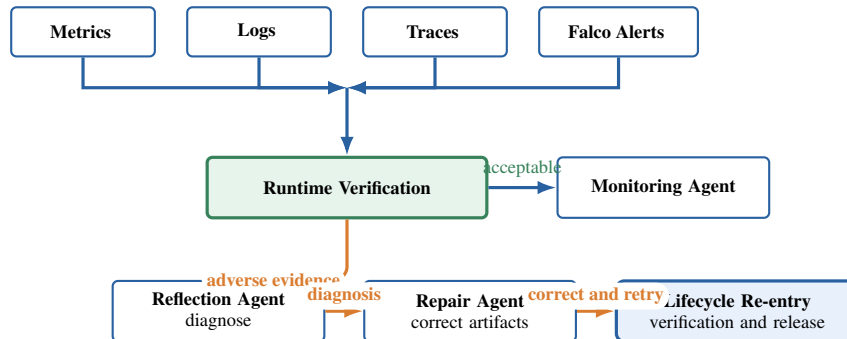


Figure 8: Runtime evidence and bounded adaptation. Metrics, logs, traces, and Falco alerts support runtime verification, while adverse evidence invokes reflection and repair before controlled lifecycle re-entry.

and secret management. Figure 13 shows the runtime path from telemetry collection and evidence

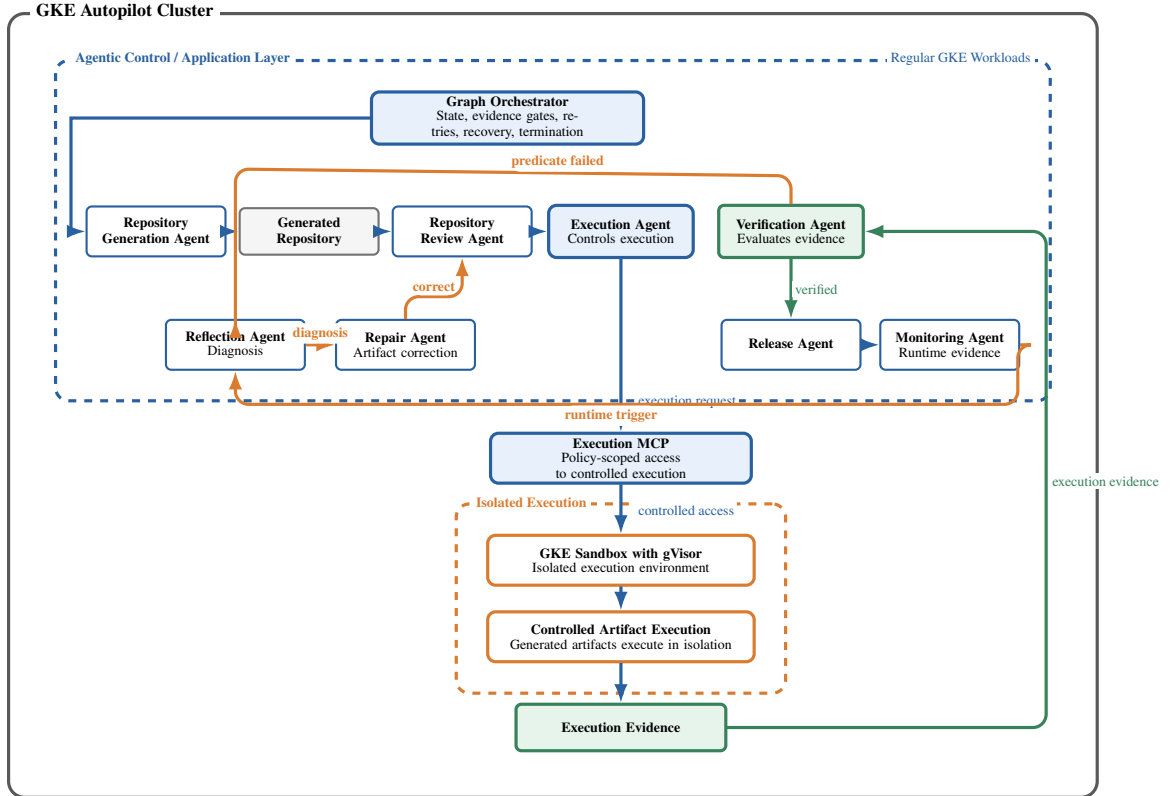


Figure 9: The figure shows the execution-agent and sandbox separation on GKE Autopilot. The Graph Orchestrator and specialized agents run as regular GKE workloads. The Execution Agent requests controlled execution through the Execution MCP, whereas the GKE Sandbox with gVisor provides the isolated environment for generated-artifact execution. Execution evidence is evaluated by the Verification Agent; failed verification activates bounded reflection and repair, while verified artifacts proceed to release and monitoring.

derivation to predicate evaluation and the runtime evidence gate, which either permits continued operation or activates bounded recovery.

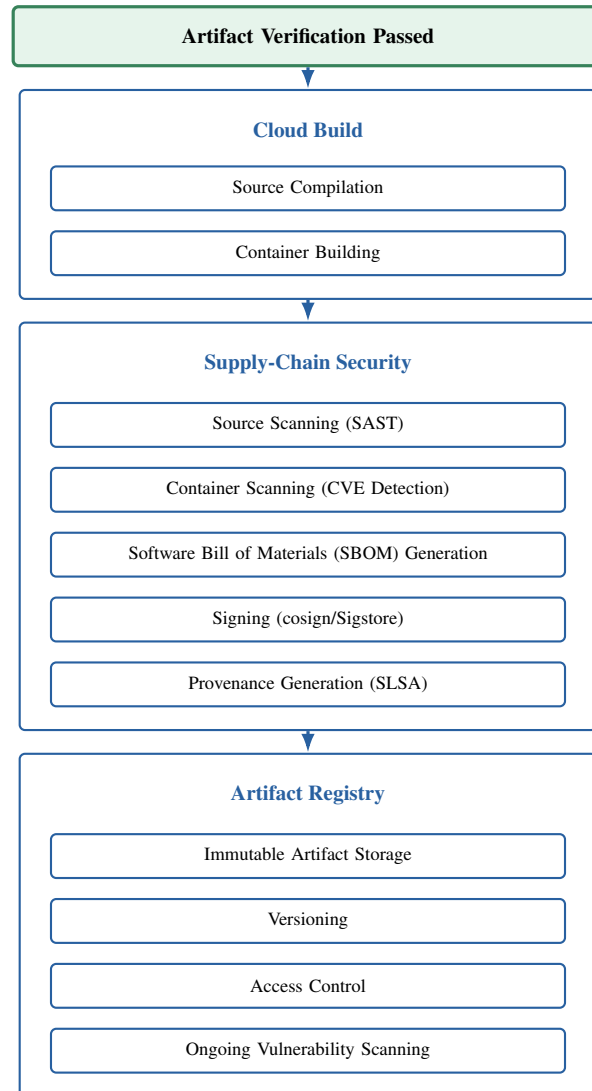


Figure 10: The figure depicts build and Artifact Pipeline.

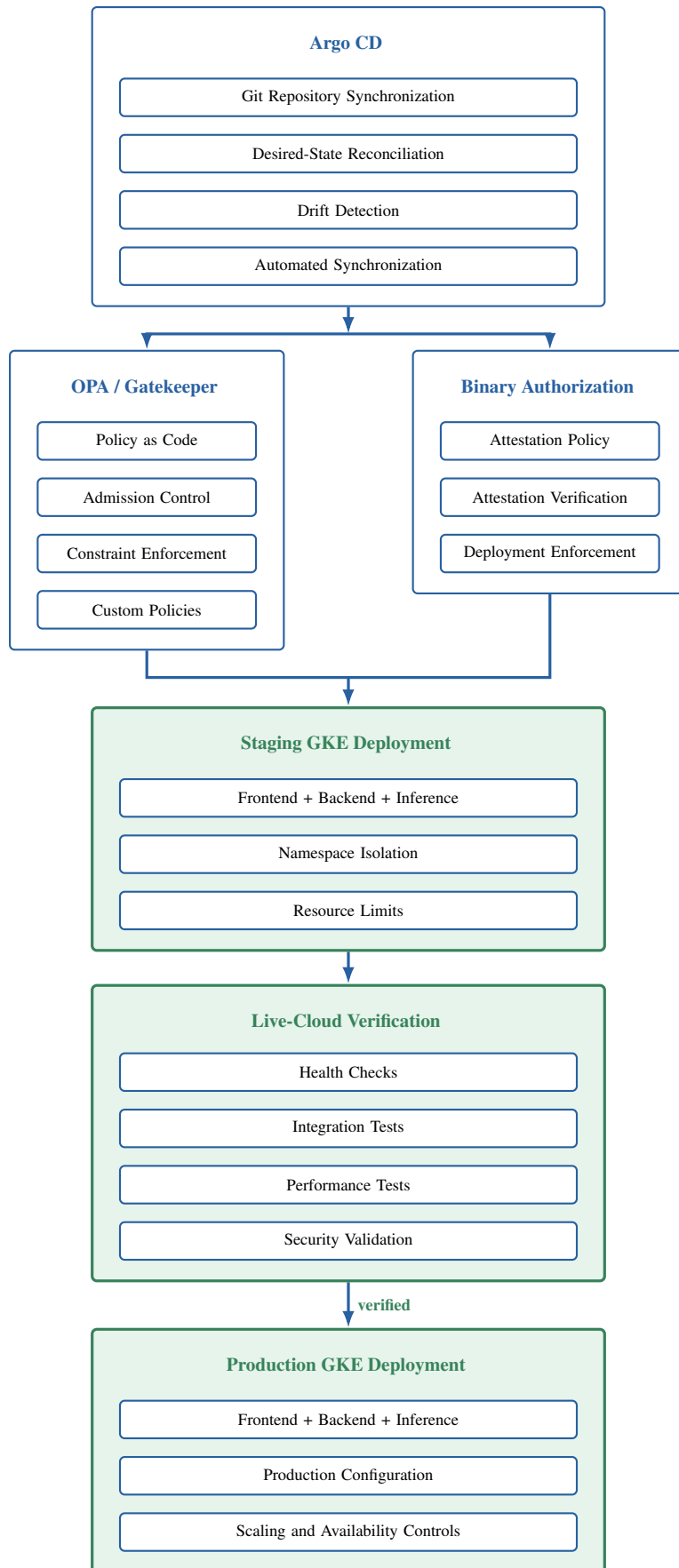


Figure 11: GitOps reconciliation, policy enforcement, and evidence-gated promotion from staging to production.

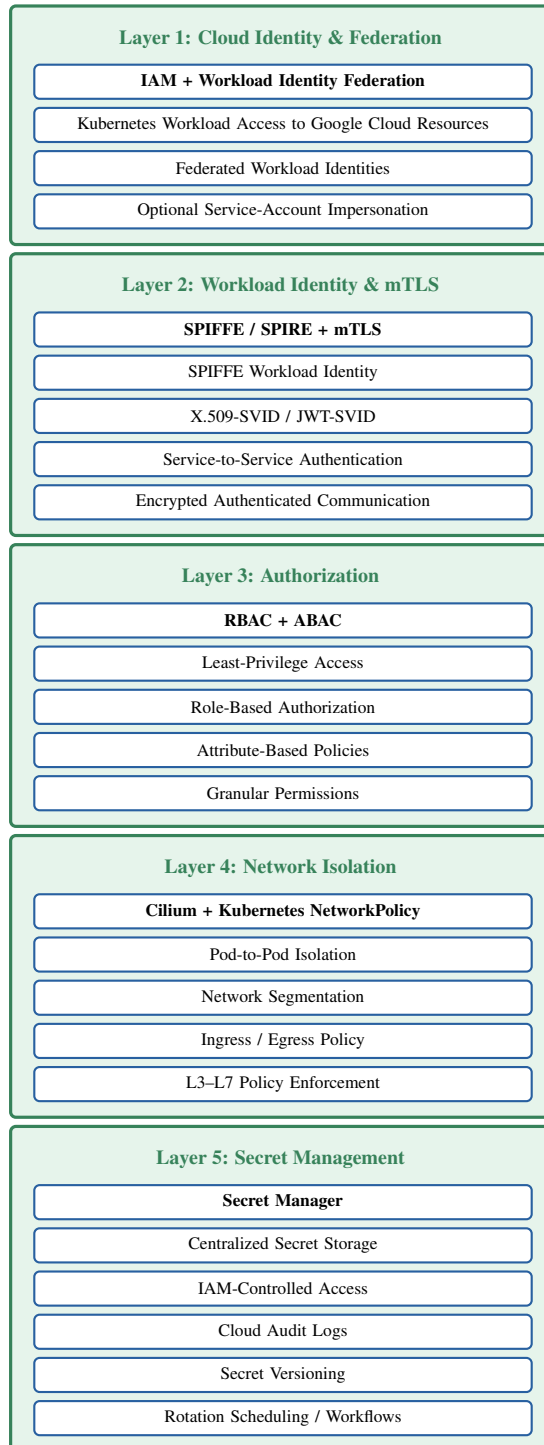


Figure 12: Multi-layer zero-trust security controls spanning cloud identity, workload identity, authorization, network isolation, and secret management.

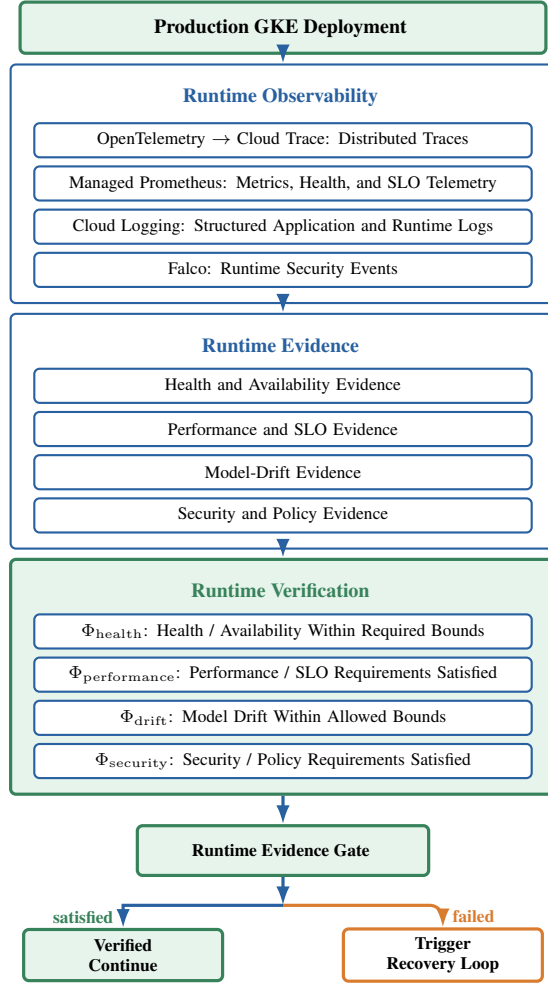


Figure 13: Runtime observability, evidence derivation, and evidence-gated verification for continued operation or bounded recovery.