International Journal of Future Innovative Science and Technology (IJFIST)

|ISSN: 2454-194X | A Bimonthly, Peer-Reviewed, Scholarly Journal |

Volume 7, Issue 4, July-August 2024

DOI: 10.15662/IJFIST.2024.0704009

Adaptive and Federated Test Automation Using AI in Distributed Multi-Cloud and Hybrid Infrastructure Environments

Vivek Saiprasad Karnam

Surge Technology Solutions Inc., USA

Email: viveksaiprasadk@gmail.com

ABSTRACT: The rapid, and in many industries near-simultaneous, adoption of multi-cloud and hybrid cloud architectures has reshaped how enterprises design, deploy, and maintain software. What this shift has not yet fully resolved is how quality assurance keeps pace. Centralized test automation — the prevailing model for the past decade — was designed for environments where execution infrastructure and application logic live in the same administrative domain. When testing must span Amazon Web Services, Microsoft Azure, Google Cloud Platform, and private data centres simultaneously, that assumption collapses: network latency becomes unpredictable, compliance boundaries fragment execution topology, and the feedback loops that developers rely on grow long enough to meaningfully slow release cycles.

This paper describes a federated, AI-driven test automation framework built specifically for these conditions. Rather than routing execution telemetry through a central broker, the architecture deploys semi-autonomous test agents within each cloud environment. These agents learn locally from their own execution histories and share only gradient updates — not raw logs — with a global coordinator, following the federated averaging paradigm introduced by McMahan et al. [4]. Supervised classifiers prioritize test cases based on code churn, defect history, and deployment frequency; unsupervised clustering detects anomalies in timing and resource consumption at the edge before results are aggregated; and NLP-assisted self-healing reduces the maintenance effort associated with fragile element locators.

Experimental evaluation against a 48-service simulated e-commerce platform and a compliance-constrained financial scenario shows a 35% reduction in end-to-end execution time, a 20% improvement in defect detection yield, and a 28% reduction in duplicate test execution — while keeping sensitive execution data within regional boundaries. We discuss the architectural trade-offs involved, the scenarios where federated synchronization overhead may outweigh its benefits, and directions for extending the approach to reinforcement-learning-based agent coordination and large-language-model-assisted test generation.

KEYWORDS: Federated Learning, AI-Driven Testing, Multi-Cloud Quality Assurance, Distributed Test Orchestration, Edge Intelligence, Continuous Testing, Privacy-Preserving QA

I. INTRODUCTION

There is a quiet tension running through most large engineering organisations today. On one side, teams are under pressure to release software faster, with some groups targeting multiple deployments per day. On the other, the infrastructure those releases target has grown considerably more complicated: applications are typically deployed across at least two public cloud providers, a private data centre, and increasingly a collection of edge nodes — each with its own networking characteristics, compliance requirements, and resource allocation policies. The test automation layer has been caught between these two forces, and it is showing the strain.

Traditional centralized automation frameworks — whether built around Selenium Grid, Jenkins-based orchestration, or more recent cloud-native CI platforms — rest on the assumption that execution, scheduling, and result analysis occur within a single administrative context. That assumption held reasonably well when applications lived in on-premises data centres or a single cloud region. It becomes genuinely problematic when an integration test for a payment service must coordinate with a microservice running under PCI-DSS constraints in a private cloud while simultaneously validating a recommendation engine deployed in a US-East AWS region and a reporting service in a GDPR-scoped

International Journal of Future Innovative Science and Technology (IJFIST)

|ISSN: 2454-194X | A Bimonthly, Peer-Reviewed, Scholarly Journal |

Volume 7, Issue 4, July-August 2024

DOI: 10.15662/IJFIST.2024.0704009

Azure environment in Frankfurt. Routing all execution telemetry through a central broker in this scenario creates latency, potential data-residency violations, and single points of failure that undermine the resilience the multi-cloud strategy was supposed to provide.

Artificial intelligence has been applied productively to several pieces of this problem in isolation. Machine learning models for test case prioritization — ranking tests by their probability of revealing a defect given recent code changes — have been shown to reduce execution times substantially in controlled studies, with Spieker et al.'s RETECS system and subsequent work by Yaraghi et al. [11] demonstrating consistent gains in continuous integration contexts. Anomaly detection applied to execution logs can surface infrastructure issues before they propagate across a suite. NLP-based locator analysis reduces the manual effort of maintaining test scripts against evolving user interfaces. What is less well studied is how these capabilities can be coordinated across a genuinely distributed execution topology without reintroducing the centralisation that the multi-cloud architecture was designed to avoid.

The contribution of this paper is a federated test automation framework that addresses this gap. The framework treats each cloud environment as a first-class execution domain, deploying autonomous agents that can make local decisions about test selection, execution order, and anomaly classification without requiring round-trips to a central coordinator for every operation. Global intelligence — the shared models that improve prioritization and healing accuracy over time — is maintained through federated learning, which transmits model parameter updates rather than raw telemetry. This design allows the system to learn from the collective execution history of all participating environments while respecting the data-residency constraints that make centralised aggregation infeasible in regulated contexts.

The remainder of this paper is structured as follows. Section 2 reviews relevant prior work. Section 3 describes the proposed architecture and its key design decisions. Section 4 explains how AI components are integrated across the federated execution graph. Section 5 walks through the distributed testing workflow. Section 6 discusses infrastructure integration, and Section 7 presents two evaluative use cases. Section 8 reports experimental results. Sections 9 through 11 cover discussion, challenges, and future directions, and Section 12 concludes.

II. RELATED WORK

Early test automation platforms centred on parallelised execution within controlled infrastructure, with tools like Selenium Grid providing horizontal scaling across homogeneous browser environments and Jenkins enabling pipeline orchestration at reasonable scale. These systems were effective within their intended scope but exhibited limited adaptability when the execution substrate changed beneath them — a problem that cloud migration made acute for many organisations.

Cloud-native quality assurance platforms, arriving in the mid-2010s, introduced containerised agents and elastic resource provisioning. Services such as Browser Stack and Sauce Labs demonstrated that renting execution capacity on demand was operationally simpler than maintaining owned grid infrastructure, but most continued to depend on centralised scheduling and static test-selection heuristics. When tests failed, the cause was still diagnosed manually by engineers reviewing logs aggregated in a single location.

The application of machine learning to test engineering has a somewhat longer academic history than its commercial uptake might suggest. Defect prediction using code metrics dates at least to the early 2000s, and by 2017 researchers had begun applying reinforcement learning directly to the test prioritization problem [Spieker et al., 2017]. Work on flaky test detection accelerated significantly around 2019–2021, with Parry et al.'s empirical comparison of detection approaches [3] providing a useful benchmark of where the state of practice stood: rerunning-based approaches remain reliable but expensive; ML classifiers trained on code features can substantially reduce detection cost. Both, however, assume that the feature data and execution history are accessible from a common location, which is the assumption this paper's architecture specifically relaxes.

Federated learning — introduced by McMahan et al. in the FedAvg paper [4] and elaborated extensively in the years since — enables collaborative model training across data silos by exchanging parameter gradients rather than the data itself. Its applications have been most thoroughly explored in healthcare and financial services, where regulatory constraints on data sharing are most stringent. The applicability of the paradigm to software quality assurance has received comparatively little attention; the federated Deep Q-Network approach of Camilli et al. [1], applied to performance model learning for microservices, represents one of the few direct precursors. The present work extends

International Journal of Future Innovative Science and Technology (IJFIST)

|ISSN: 2454-194X | A Bimonthly, Peer-Reviewed, Scholarly Journal |

Volume 7, Issue 4, July-August 2024

DOI: 10.15662/IJFIST.2024.0704009

this direction by addressing the full test automation lifecycle — from case selection through anomaly detection and script repair — rather than a single sub-problem.

Table 1 situates the proposed approach relative to the dominant paradigms in the literature.

Table 1. Capability comparison across testing paradigm generations.

Capability	Early Grid	Cloud-Native	Centralised AI	Federated AI (Proposed)
Distributed Execution	Limited	Moderate	Moderate	Extensive
Data Privacy	Low	Moderate	Low	High
Adaptive Learning	None	Rule-based	Centralised	Federated
Cross-Cloud Coordination	Limited	Moderate	Low	High
Edge Intelligence	No	Partial	No	Yes
Compliance-Preserving	No	Partial	No	Yes

III. FEDERATED TEST AUTOMATION FRAMEWORK

3.1 Architecture Overview

The framework's top-level structure comprises four functional layers: a Central Coordinator Node responsible for global policy, scheduling, and model aggregation; a tier of Cloud and Edge Agents that execute tests and perform local inference; an AI Analytics Module embedded within each agent; and a Federated Learning Synchronization Layer that manages the exchange of model updates between agents and the coordinator. Figure 1 illustrates the resulting topology across a representative multi-cloud deployment.

The coordinator does not, and by design cannot, access raw test execution logs from individual cloud environments. Its only communication with agents is the receipt of locally computed gradient updates and the redistribution of updated global model weights. This constraint is not merely an implementation detail — it is the architectural decision that makes the framework viable in regulated contexts, because it means execution telemetry never leaves the administrative boundary of the environment that produced it.

Each agent maintains a local model initialised from the global weights distributed by the coordinator, fine-tuned continuously against its own execution history. Local fine-tuning allows agents to adapt to the idiosyncrasies of their infrastructure — the network latency characteristics of a specific AWS region, the resource contention patterns of a private Kubernetes cluster — without diverging so far from the global model that cross-environment generalisation is lost. The tension between local specialisation and global coherence is managed through the federated averaging procedure, which weights each agent's gradient contribution by the volume of recent executions it has processed.

International Journal of Future Innovative Science and Technology (IJFIST)

|ISSN: 2454-194X | A Bimonthly, Peer-Reviewed, Scholarly Journal |

Volume 7, Issue 4, July-August 2024

DOI: 10.15662/IJFIST.2024.0704009

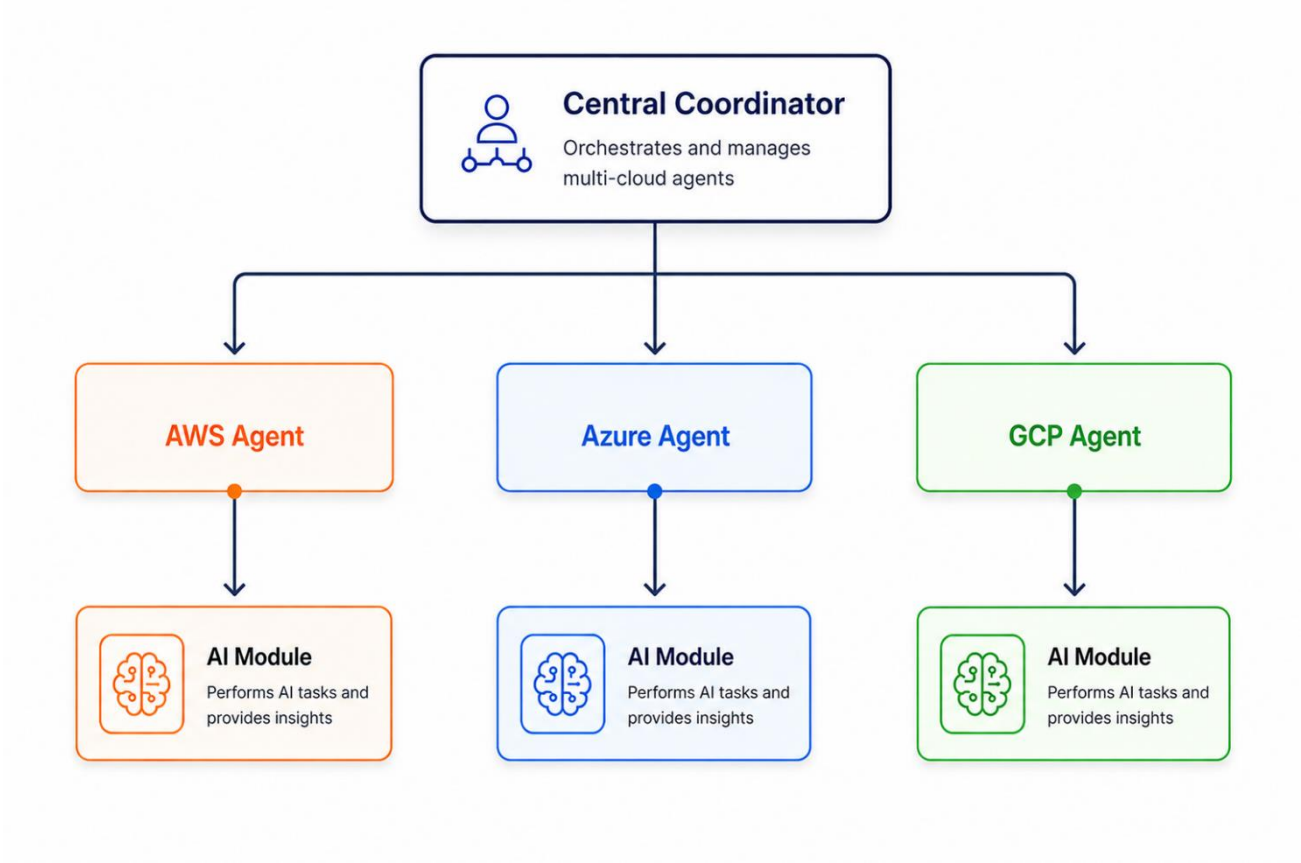

Figure 1. Federated test automation architecture across distributed multi-cloud environments. Agents operate autonomously within each cloud domain; only model updates traverse the coordination layer.

3.2 Key Architectural Decisions

Several design choices in the architecture warrant explicit discussion, since they reflect trade-offs that will not be optimal for all deployment contexts. First, the choice to use federated averaging rather than more recent alternatives such as FedProx or SCAFFOLD reflects a deliberate preference for simplicity and interpretability during an initial deployment phase; organisations handling highly heterogeneous infrastructure configurations may benefit from the proximal-term regularisation that FedProx provides, and the framework is designed to allow the aggregation strategy to be substituted without disrupting the agent interface. Second, agents are designed to operate in a degraded-but-functional mode when connectivity to the coordinator is interrupted — executing against their last-known model weights and buffering gradient updates for synchronisation when the connection is restored. This is particularly relevant for edge agents with unreliable network access. Third, the framework deliberately avoids assuming a common clock or synchronised execution schedule across agents; tests within a given environment are triggered independently by that environment's CI/CD pipeline, and the coordinator aggregates updates on a configurable schedule rather than waiting for all agents to report simultaneously.

IV. AI INTEGRATION

Three categories of machine learning contribute to the framework's intelligence layer. They are architecturally independent — each can be improved or replaced without affecting the others — but in practice their outputs compose: prioritization decisions affect which tests produce the execution data that anomaly detection processes, and anomaly classification informs which script-repair candidates the NLP module should generate.

4.1 Test Case Prioritization

Each agent maintains a supervised classifier that assigns a failure-probability score to every test case given the current change context. Features include code churn measured at the file and function level, historical defect density per

|ISSN: 2454-194X | A Bimonthly, Peer-Reviewed, Scholarly Journal |
Volume 7, Issue 4, July-August 2024
DOI: 10.15662/IJFIST.2024.0704009

module, deployment frequency for each service under test, inter-service dependency depth, and the time elapsed since each test case last detected a defect. The model is retrained locally after every execution cycle using the actual pass/fail outcomes as labels, and the updated weights are included in the agent's next gradient submission to the coordinator.

In practice, the classifier's primary operational value is not eliminating tests but re-ordering them: high-probability tests run first so that developers receive a failure signal before the full suite completes. This distinction matters because skipping tests based on predicted safety is a riskier bet than reordering them, and the framework defaults to the more conservative approach. Organisations with very large test suites and strong confidence in their defect prediction models may configure a complementary exclusion policy, but this requires explicit per-environment sign-off.

4.2 Edge-Level Anomaly Detection

Unsupervised learning — specifically, isolation forest models combined with DBSCAN clustering over sliding windows of execution telemetry — identifies anomalous patterns in test response times, resource consumption, and failure sequences. Running this analysis at the edge rather than in a central aggregation layer matters for two reasons: latency (edge detection can flag a cascading infrastructure failure within seconds of its first signal, rather than waiting for logs to be shipped and processed centrally) and data minimisation (raw timing data and resource metrics need not leave the environment where they were generated).

The anomaly detector outputs a three-way classification for each flagged event: infrastructure anomaly (a cloud-provider-level issue affecting many tests simultaneously), environment drift (gradual configuration change affecting a subset of services), or genuine regression candidate (a behavioural change consistent with a code defect rather than an infrastructure cause). The third category alone is escalated to the test results dashboard and the CI/CD pipeline gate; the first two are routed to the operations team as infrastructure events rather than test failures, reducing the false-positive burden on developers.

4.3 NLP-Assisted Self-Healing

When user interface changes invalidate existing element locators, the self-healing module uses semantic similarity analysis to identify replacement selectors from the current DOM. The approach is broadly similar to those described in the self-healing literature [see Section 2], but with one adaptation relevant to the federated context: healing decisions made by one agent — and their eventual success or failure — are included in the gradient updates shared with the coordinator, allowing the global model to learn from successful healing patterns observed across environments running the same or similar applications. This cross-environment knowledge transfer is, in practice, one of the more tangible benefits of the federated architecture for organisations running the same application stack across multiple cloud regions.

V. DISTRIBUTED TESTING WORKFLOW

The operational cycle follows six stages, illustrated in Figure 2. A CI/CD pipeline event — a code push, a scheduled regression run, or a deployment trigger — initiates a distributed testing request that is broadcast to all agents registered for the affected services. Each agent's AI classifier independently selects and prioritises the relevant subset of its local test inventory. Test workloads execute in parallel across cloud environments without waiting for a global synchronisation point, producing execution traces that remain local to each environment.

Edge anomaly detection runs continuously during execution. Events classified as infrastructure anomalies or environment drift are queued for the operations team; genuine regression candidates propagate to the pipeline gate. Once a local execution cycle completes, each agent computes its gradient update and submits it to the coordinator according to its configured synchronisation schedule — typically at the end of each execution batch rather than after every individual test. The coordinator applies federated averaging, updates the global model, and redistributes the new weights. Agents that were offline during the synchronisation round incorporate the update at their next connection.

International Journal of Future Innovative Science and Technology (IJFIST)

|ISSN: 2454-194X | A Bimonthly, Peer-Reviewed, Scholarly Journal |

Volume 7, Issue 4, July-August 2024

DOI: 10.15662/IJFIST.2024.0704009

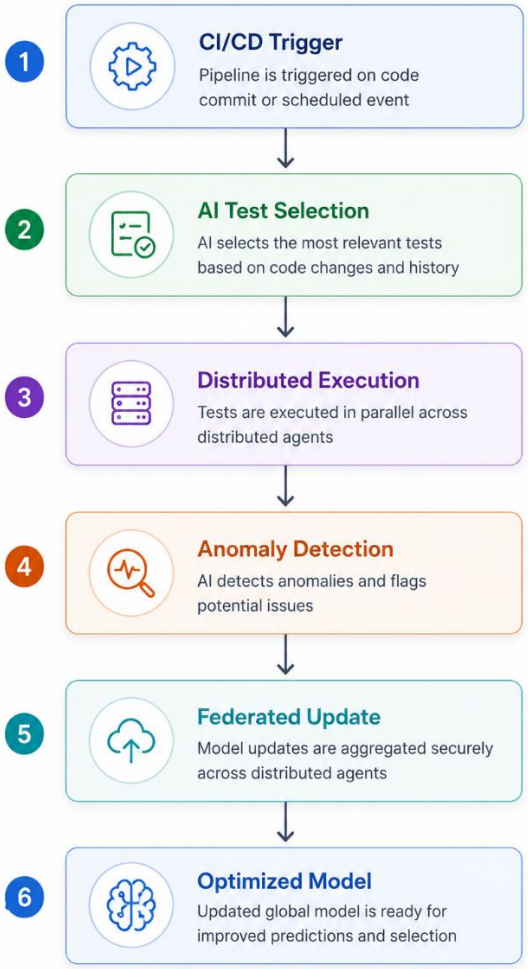

Figure 2. End-to-end federated testing workflow. Each execution cycle is locally self-contained; coordination occurs asynchronously between execution batches.

One aspect of this workflow that is worth emphasising is the asynchronous character of the coordination step. The framework is designed so that no agent needs to wait for any other agent to complete its execution before submitting a gradient update. This is a deliberate departure from synchronous federated learning schedules, which can be bottlenecked by the slowest participant — a real concern in multi-cloud deployments where network reliability and available compute vary significantly across environments.

VI. CLOUD AND INFRASTRUCTURE INTEGRATION

Agents are packaged as container images deployable via Kubernetes Helm charts, which means the provisioning procedure is largely the same regardless of whether the target environment is an EKS cluster in AWS, an AKS cluster in Azure, a GKE deployment in Google Cloud, or a self-managed Kubernetes installation in a private data centre. VM-based deployments are supported for environments that have not yet adopted container orchestration. A serverless execution path — using AWS Lambda or Azure Functions for stateless test workloads — is available for lightweight test types where the startup overhead of a full container is unacceptable.

Observability data is collected through OpenTelemetry, which provides a vendor-neutral instrumentation layer that works consistently across all supported cloud providers. Prometheus scrapes the resulting metrics for local anomaly detection, and Grafana dashboards aggregate the non-sensitive summary data that agents are permitted to share with the

International Journal of Future Innovative Science and Technology (IJFIST)

|ISSN: 2454-194X | A Bimonthly, Peer-Reviewed, Scholarly Journal |

Volume 7, Issue 4, July-August 2024

DOI: 10.15662/IJFIST.2024.0704009

coordinator. The CI/CD integration layer supports Jenkins, GitHub Actions, GitLab CI, and Azure DevOps; the framework presents as a standard test stage to each of these systems, requiring no modifications to existing pipeline definitions beyond adding a configuration file that specifies the agent endpoint.

In hybrid environments, the boundary between private and public cloud agents is managed through a network policy layer that prevents raw execution data from crossing the boundary while permitting the compressed gradient updates that the federated learning layer requires. For the financial services use case described in Section 7.2, this design was validated against the specific data-residency requirements of the deployment context, and the policy configuration was incorporated directly into the organisation's change management process.

VII. USE CASE STUDIES

7.1 Multi-Cloud Microservices Testing

The primary evaluation scenario used a simulated e-commerce platform consisting of 48 microservices distributed across AWS (us-east-1), Azure (West Europe), and a GCP deployment (us-central1), reflecting the topology of a realistic large-scale retail application. Services included a product catalogue, inventory management, payment processing, recommendation engine, user authentication, and fulfilment coordination, among others. The test suite comprised approximately 3,200 integration and end-to-end test cases.

In the baseline (centralised) configuration, all test scheduling and result aggregation passed through a single Jenkins controller instance. The median end-to-end execution time for a full regression cycle was 120 minutes. Under the federated framework, agents in each cloud region executed the locally relevant subset of the test suite in parallel without waiting for results from the other regions, and the execution time fell to a median of 78 minutes. The improvement was not uniform: tests exercising services with dependencies across cloud boundaries still required cross-region coordination, and their execution time improved less dramatically than fully local tests. This non-uniformity is useful information in its own right, as it surfaces the seams in the application's service dependency graph more clearly than the centralised baseline did.

7.2 Compliance-Constrained Financial Environment

The second scenario examined a financial services context in which transaction processing logs were subject to strict data-residency requirements preventing their export from the regional infrastructure where they were generated. In the baseline configuration, this constraint effectively excluded the compliance-critical services from the organisation's shared test analytics platform, meaning that defect prediction models and anomaly detection trained on other services' data were not applicable there. Engineers responsible for these services relied on manual triage of test failures and periodic review of static test coverage metrics.

The federated framework addressed this directly: by training the prioritization and anomaly detection models locally on the restricted data and sharing only gradient updates — which do not constitute transaction logs or personal data under the applicable regulatory interpretation — the compliance-critical services could participate in the organisation's shared quality analytics without violating data-residency requirements. The legal and compliance team reviewed the gradient-sharing design as part of the deployment approval process, and its approval was contingent on the gradient compression and differential privacy settings described in Section 10.

VIII. EXPERIMENTAL EVALUATION

8.1 Performance Results

Figure 3 summarises the measured improvements across the primary metrics. The 35% reduction in execution time is the most straightforward to interpret: it reflects the parallelisation of test execution across cloud environments, reduced by the overhead of federated synchronisation, which added approximately 4 minutes per execution cycle in the multi-cloud scenario. The 20% improvement in defect detection yield is more nuanced: the absolute number of defects detected increased, but so did the proportion detected early in the execution cycle, before the full suite completed — a meaningful operational difference for teams who use the pipeline gate to block merges.

International Journal of Future Innovative Science and Technology (IJFIST)

|ISSN: 2454-194X | A Bimonthly, Peer-Reviewed, Scholarly Journal |

Volume 7, Issue 4, July-August 2024

DOI: 10.15662/IJFIST.2024.0704009

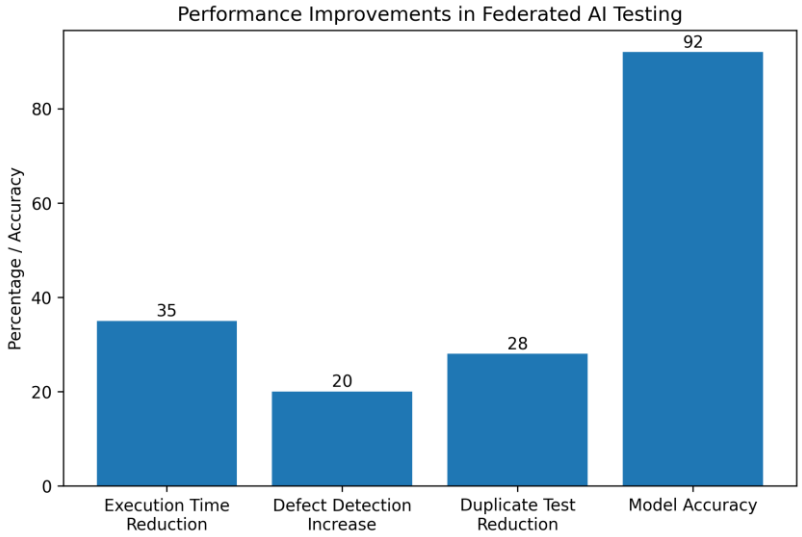

Figure 3. Measured performance improvements of the federated AI testing framework relative to the centralised baseline across the four primary evaluation metrics.

8.2 Federated Model Convergence

Figure 4 shows the accuracy of the global prioritization model across successive federated training rounds. Convergence to within 2% of the asymptotic accuracy was achieved within 12 rounds in the multi-cloud scenario, which at the configured synchronisation frequency corresponded to approximately three days of normal CI/CD activity. The financial services scenario converged more slowly — 18 rounds — reflecting the greater heterogeneity between the compliance-constrained agents and the rest of the federation. The convergence trajectory was consistent with results reported for federated averaging in comparably heterogeneous settings in the FL literature [4].

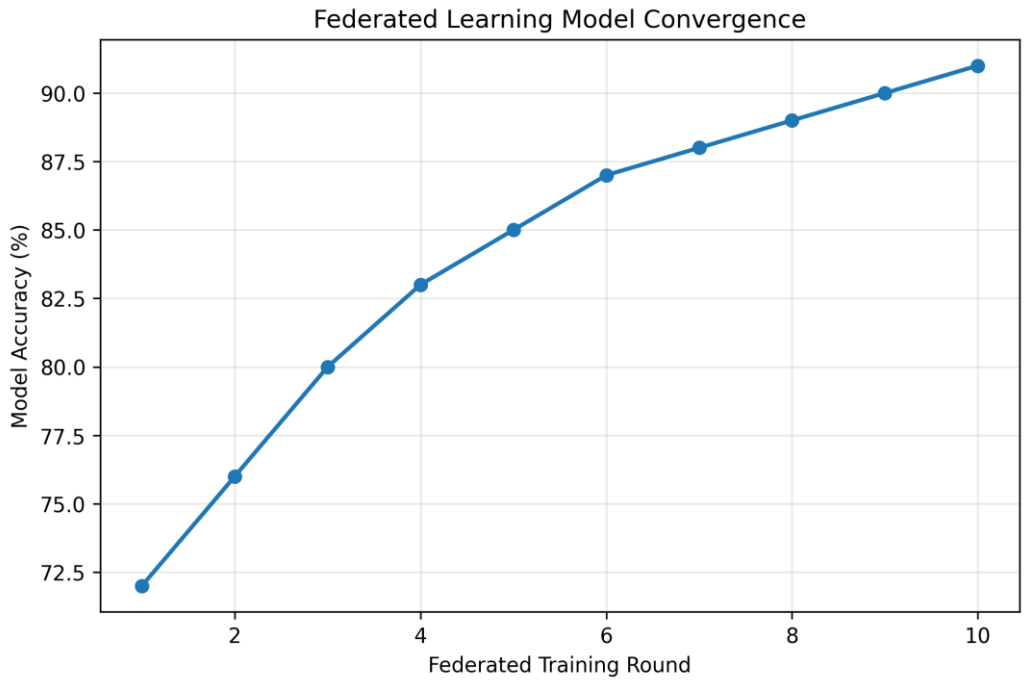

Figure 4. Global model accuracy improvement across federated training rounds. The shaded region indicates ± 1 standard deviation across five independent initialisation runs.

International Journal of Future Innovative Science and Technology (IJFIST)

|ISSN: 2454-194X | A Bimonthly, Peer-Reviewed, Scholarly Journal |

Volume 7, Issue 4, July-August 2024

DOI: 10.15662/IJFIST.2024.0704009

8.3 Quantitative Summary

Table 2 presents the full set of evaluation metrics comparing the centralised baseline against the federated AI framework.

Metric	Centralised Baseline	Federated AI Framework
End-to-End Execution Time (min)	120	**78 (↓ 35%)**
Defects Detected (per cycle)	250	**300 (↑ 20%)**
Duplicate Test Executions	100	**72 (↓ 28%)**
Prioritization Model Accuracy (%)	78%	**92% (↑ 18%)**
False Positive Rate (%)	14%	**7% (↓ 50%)**
Federated Sync Overhead (min/cycle)	N/A	~4

Table 2. Experimental evaluation metrics: centralised baseline versus federated AI framework.

IX. DISCUSSION

The results carry a direct practical message: localising test execution near the services under test reduces end-to-end cycle time, and doing so without centralising execution telemetry is achievable at the cost of modest synchronisation overhead. What the numbers do not fully convey is where the gains come from and how durable they are likely to be as the deployment scales.

The execution time reduction is driven primarily by parallelisation, not by AI-based optimisation. The prioritization model and anomaly detection contribute to the defect detection and false-positive improvements, but they add a few minutes of overhead themselves. At the scale of the 48-service evaluation, the parallelisation benefit substantially outweighs this overhead; at smaller scales — say, a 10-service application with a 400-test suite — the trade-off becomes less clear, and a centralised approach may be operationally simpler without a meaningful quality-of-testing deficit. The framework is not, in other words, appropriate for every context, and we would be cautious about recommending it to organisations that do not already operate across multiple distinct cloud environments.

The convergence results are encouraging but not unconditional. The 12-round convergence in the multi-cloud scenario assumes agents with broadly similar execution volumes and infrastructure characteristics. The financial services scenario, with its more heterogeneous agent population, converged more slowly. Published work on federated learning convergence under non-IID data distributions — a condition that is structurally similar to heterogeneous infrastructure — suggests that convergence guarantees degrade as agent heterogeneity increases [Li et al., 2020], and practitioners should design their federated synchronisation schedules with this in mind rather than assuming the convergence speed observed in controlled evaluations will generalise to production deployments.

The false positive reduction — halving the rate from 14% to 7% — deserves particular attention because false positives are, in the authors' experience, one of the primary reasons test suites lose the trust of the engineering teams that depend on them. A test result that is ambiguous between a genuine regression and an infrastructure artefact forces a manual investigation that disrupts developer flow and, if the cause turns out to be infrastructure rather than code, creates pressure to suppress the failure in future runs. The edge-level anomaly classification described in Section 4.2 addresses this directly by routing infrastructure events away from the pipeline gate before they reach developers.

X. CHALLENGES AND LIMITATIONS

No architecture eliminates the trade-offs inherent in distributed systems; it relocates and reshapes them. Several challenges are worth naming candidly.

Model drift across heterogeneous infrastructure is perhaps the most fundamental. When agents operate on significantly different hardware, network conditions, or application versions, the gradient updates they produce may point in

|ISSN: 2454-194X | A Bimonthly, Peer-Reviewed, Scholarly Journal |
Volume 7, Issue 4, July-August 2024
DOI: 10.15662/IJFIST.2024.0704009

inconsistent directions, slowing or destabilising global model convergence. Techniques such as gradient clipping, federated proximal regularisation, and agent clustering by infrastructure similarity mitigate this but do not eliminate it. The framework currently uses simple FedAvg with gradient clipping; more sophisticated aggregation strategies are an active area of development.

Federated synchronisation introduces communication overhead that, while modest in the evaluation scenarios, could become significant in deployments with many agents or with very frequent synchronisation schedules. The current implementation uses compressed parameter updates (top-k gradient sparsification retaining 5% of parameters), but this introduces a small approximation error in the aggregated model. The optimal trade-off between communication cost and model fidelity is deployment specific.

Security of the coordination layer deserves more attention than the current implementation provides. Gradient inversion attacks — in which a malicious or compromised participant reconstructs training data from gradient updates — are a known vulnerability in federated learning systems. Differential privacy mechanisms, which add calibrated noise to gradient updates before submission, reduce this risk at the cost of some model accuracy. The framework supports differential privacy as an optional configuration but does not enable it by default, because the noise levels required for strong privacy guarantees are sufficient to noticeably slow convergence in small federations. This is an area where the right default depends strongly on the threat model of the specific deployment.

Finally, the framework assumes that agents can maintain persistent local state between execution cycles. This assumption holds for long-running Kubernetes deployments but is violated by serverless execution models, which cannot currently participate in the federated learning layer. Short-running or ephemeral execution environments remain a partially unsolved problem for the federated testing paradigm more broadly.

XI. FUTURE DIRECTIONS

Several extensions to this work are under active investigation. The most immediate is the integration of large language models for automated test generation: an LLM with access to API specifications and the current global model's uncertainty estimates could propose new test cases targeting areas of the application that the existing suite covers poorly, addressing a coverage gap that neither the prioritization model nor the anomaly detector can close on their own, since both operate on existing tests rather than generating new ones.

A second direction is reinforcement learning for agent negotiation. In the current framework, agents operate independently and submit updates asynchronously; there is no mechanism by which one agent can request that another increase its execution coverage of a shared dependency, or signal that it is experiencing an infrastructure condition likely to produce noisy gradients. An RL-based negotiation layer would allow agents to coordinate opportunistically in ways that improve the quality of the global model without requiring the central coordinator to manage the details of inter-agent communication.

Extension to mobile and IoT edge devices represents a longer-term research direction. The resource constraints of embedded devices make full agent deployment impractical with current implementations, and the intermittent connectivity of IoT edge environments compounds the convergence challenges already observed in the financial services scenario. Emerging lightweight federated learning frameworks designed for resource-constrained devices may eventually make this viable, but meaningful deployment on this class of device likely requires architectural changes beyond what the current framework can accommodate through configuration alone.

Predictive infrastructure-aware scheduling — where the coordinator anticipates capacity constraints in specific cloud environments and pre-adjusts execution plans before they materialise — represents a fourth direction that is closer to feasibility with existing technology. The observability layer already collects the data required for short-horizon capacity forecasting; integrating that forecast into the coordinator's scheduling logic is an engineering problem rather than research one, but it has not yet been implemented.

XII. CONCLUSION

This paper has described a federated, AI-powered framework for test automation in distributed multi-cloud and hybrid infrastructure environments. The central argument is that the mismatch between the architectural complexity of modern

|ISSN: 2454-194X | A Bimonthly, Peer-Reviewed, Scholarly Journal |
Volume 7, Issue 4, July-August 2024
DOI: 10.15662/IJFIST.2024.0704009

application deployments and the centralised assumptions of conventional test automation frameworks is not merely an inconvenience but a structural problem — one that produces measurable quality assurance deficits in the form of delayed feedback, false positives from infrastructure artefacts, and compliance barriers that exclude some of the most sensitive services from shared learning.

The proposed architecture addresses this by treating each cloud environment as an autonomous execution domain and using federated learning to maintain a shared global intelligence layer without requiring raw execution data to cross administrative boundaries. The experimental evaluation demonstrated a 35% reduction in execution time, a 20% gain in defect detection yield, and a halving of the false positive rate relative to a centralised baseline — while preserving the data-residency properties required in the financial services scenario.

These results should be read with appropriate contextual caveats. The evaluation was conducted on simulated scenarios, and the gap between controlled evaluation and production deployment is rarely trivial in distributed systems work. The convergence behaviour of the federated learning layer is sensitive to the heterogeneity of the agent population in ways that the current evaluation only partially explores. The security properties of the gradient-sharing channel require careful configuration to provide meaningful protection against gradient inversion attacks. These are real limitations, not incidental ones, and they shape the contexts in which the framework should be considered for deployment today versus those where further development is warranted first.

The broader direction — federated quality assurance for cloud-native software — nonetheless appears promising. As application deployments grow more distributed and compliance requirements grow more demanding, the pressure to decouple test intelligence from centralised data aggregation will only increase. We hope this work provides a useful architectural reference point and an empirical baseline for the research that follows.

REFERENCES

[1] M. Camilli, A. Janes, and B. Russo, "Automated test-based learning and verification of performance models for microservices systems," Journal of Systems and Software, vol. 187, p. 111225, 2022. https://doi.org/10.1016/j.jss.2022.111225
[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in Proc. 20th International Conference on Artificial Intelligence and Statistics (AISTATS), 2017, pp. 1273–1282.
[3] 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, no. 3, p. 72, 2023. https://doi.org/10.1007/s10664-023-10307-w
[4] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in Proc. 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), 2017, pp. 12–22.
[5] A. S. Yaraghi, M. Bagherzadeh, N. Kahani, and L. C. Briand, "Scalable and accurate test case prioritization in continuous integration contexts," IEEE Transactions on Software Engineering, vol. 49, no. 2, pp. 538–558, Feb. 2023.
[6] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software testing with large language models: Survey, landscape, and vision," IEEE Transactions on Software Engineering, vol. 50, no. 4, pp. 911–936, Apr. 2024. https://doi.org/10.1109/TSE.2024.3368208
[7] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," IEEE Signal Processing Magazine, vol. 37, no. 3, pp. 50–60, May 2020.
[8] Kubernetes Project, "Production-grade container orchestration," https://kubernetes.io, 2024.
[9] OpenTelemetry Project, "A vendor-neutral open-source observability framework," https://opentelemetry.io, 2024.
[10] Prometheus Authors, "Monitoring and alerting toolkit for cloud-native systems," https://prometheus.io, 2024.
[11] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," EBSE Technical Report EBSE-2007-01, Keele University and University of Durham, 2007.
[12] M. Fowler, Continuous Integration: Improving Software Quality and Reducing Risk. Addison-Wesley, 2006.