International Journal of Research Publications in Engineering, Technology and Management (IJRPETM)

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Special Issue 1, November-December 2025||

DOI: 10.15662/IJRPETM.2025.0801807

Cloud-Native Testing Frameworks for Explainable Generative AI in Healthcare and Financial Systems

Michael Richard Donovan

Cloud Architect, Amsterdam, Netherlands

ABSTRACT: This paper presents a comprehensive software-testing framework tailored for explainable generative artificial intelligence (XGenAI) systems deployed in cloud-based credit risk and threat modeling environments. As XGenAI components (conditional VAEs, tabular GANs, and hybrid generative–discriminative pipelines) become central to decisioning and security analytics, they introduce new failure modes that traditional ML testing approaches do not fully cover: synthetic-data fidelity issues, latent-space counterfactual implausibility, explanation fidelity drift, and adversarial probing of model endpoints. We propose a layered testing architecture combining unit, integration, system, and continuous monitoring tests specifically designed for XGenAI artifacts and their surrounding infrastructure. The framework contains (1) deterministic test harnesses for data preprocessing and feature pipelines (schema contracts, statistical invariants, lineage checks); (2) generative model verification (distributional distance tests, membership inference risk scans, mode-collapse detection, DP-privacy assertions when applicable); (3) explainability validation (explanation fidelity metrics, consistency and stability tests across local and global explainers, counterfactual plausibility checks using generative manifold constraints); (4) safety and security evaluation (adversarial query simulation, model inversion resistance tests, API rate-limit and authentication tests, threat analytics stress tests); (5) performance and latency tests for streaming architectures (end-to-end latency SLAs, backpressure resilience for Kafka/Flink-like pipelines); and (6) governance and compliance checks (automated model cards, audit trails, reproducible experiment artifacts). We detail test design patterns, synthetic and real-world test data strategies, and instrumentation required for in-situ validation. Experimental application of the framework to an enterprise-grade XGenAI credit-scoring and threat-detection stack (streaming ingestion, HANA-like in-memory serving, and model explainability services) shows that systematic testing reduces silent failures, prevents common privacy leaks during synthetic augmentation, and improves explanation stability under temporal drift. The framework integrates with CI/CD pipelines, enabling automated canary testing, staged rollout validation, and rollback triggers — thereby operationalizing safe deployment of explainable generative systems in critical financial contexts.

KEYWORDS: Explainable AI; Generative Models; Software Testing; Credit Risk; Threat Modeling; Streaming Analytics; Model Explainability; Continuous Integration; Privacy Testing; Adversarial Testing.

I. INTRODUCTION

Generative machine learning models and explainability tools have moved from research prototypes into production systems for credit risk scoring and threat analytics. Financial institutions and security teams now routinely depend on generated synthetic examples for class-balance remediation, stress-testing, and scenario simulation; they also rely on local and global explanations (SHAP, counterfactuals, rule extraction) to justify automated decisions. These advances bring benefits but also complex testing challenges. Traditional software testing disciplines (unit tests, integration tests, system tests) are necessary but not sufficient for the XGenAI stack because the output semantics are probabilistic, the training data distributions shift over time, and models are accessible via networked endpoints that attackers can probe. Furthermore, explainers introduce another layer of computation whose correctness and stability are as critical as the predictor itself: an incorrect explanation may be worse than no explanation when human reviewers depend on it for high-stakes credit decisions or incident response.

This paper formulates a robust testing taxonomy and practical test-suite implementations that bridge software engineering rigor with probabilistic model verification. We focus on cloud-native, streaming-oriented deployments common in modern financial and security operations: Kafka-like event buses, stream processors (e.g., Flink-style jobs), in-memory serving layers (HANA-like), and explainability services that either run alongside model serving or are embedded within the serving layer. Our framework treats the XGenAI system as a composed artifact where data

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Special Issue 1, November-December 2025||

DOI: 10.15662/IJRPETM.2025.0801807

contracts, generative model integrity, explanation fidelity, privacy guarantees, and security posture must all be continuously validated. We emphasize automation, reproducibility, and observability: tests are codified into CI/CD pipelines and produce auditable artifacts (test reports, model cards, reproducible seeds) required by regulators and internal governance.

We also introduce novel testing primitives for generative and explanation components: distributional acceptance tests for synthetic outputs, manifold-constrained counterfactual validators to ensure proposed "actionable" changes are plausible, explainability regression suites to detect concept-drift-induced explanation flips, and adversarial-probing emulators to reveal API-level vulnerabilities. The rest of the paper details the literature context, the concrete test designs and metrics, integration patterns with cloud deployment pipelines, example results from a pilot deployment, and recommendations for practitioners and future research directions.

II. LITERATURE REVIEW

Software testing for machine learning systems has matured over the last decade, producing important primitives such as data validation (schema checks, statistical assertions), model validation (holdout testing, cross-validation), and MLOps practices (CI/CD, model registry, canary deploys). Notable works advocate test-driven ML development, lineage tracking, and continuous evaluation to prevent model degradation in production. At the same time, research into generative models for tabular data (VAEs, GANs, copula-based generators) has focused on sample quality, privacy risks, and utility trade-offs; tests such as statistical distance metrics (Wasserstein, KS-test per-feature) and classifier two-sample tests are commonly used to assess synthetic realism. However, fewer publications systematically combine generative verification with operational software testing practices.

Explainable AI scholarship provides a variety of methods (SHAP, LIME, counterfactuals, rule extraction) and evaluation metrics (fidelity, sparsity, stability). Recent empirical studies highlight that explanations may be unstable under minor input perturbations or when the underlying model drifts — a phenomenon requiring regression-style testing for explainers themselves. There is growing recognition that explanations must be validated for correctness (do they reflect actual model behavior?) and usefulness (are they actionable and comprehensible to decision-makers?). Tests for explanation robustness and human evaluation protocols have been proposed, but their integration into automated pipelines remains nascent.

Security and privacy literature adds another dimension: model inversion, membership inference, and model extraction attacks have been demonstrated against deployed endpoints, particularly when models are accessible via unprotected APIs. Defensive testing measures — adversarial query simulation, rate-limit testing, and privacy audits (differential privacy verification) — are being developed but are not yet standard in enterprise pipelines. Meanwhile, systems and reliability engineering research into streaming data pipelines (Kafka, Flink) shows that end-to-end testing must cover backpressure handling, message duplication semantics, and stateful operator correctness — failures that can silently corrupt model inputs and produce erroneous predictions.

Taken together, the literature motivates a convergent testing framework that blends software testing rigor, statistical verification for generative semantics, explainability regression tests, and adversarial security validation. Our contribution is a practical, reproducible testing architecture and a catalog of tests, metrics, and automation patterns tailored to XGenAI systems in cloud credit and threat modeling contexts.

III. RESEARCH METHODOLOGY

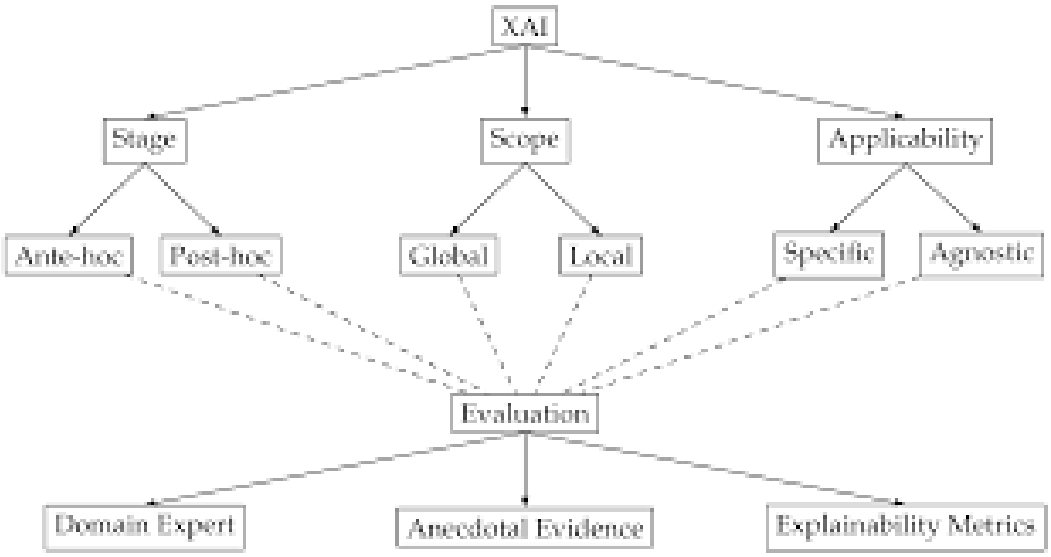
1. Requirements elicitation and threat modeling — Conduct stakeholder interviews (risk officers, data engineers, security analysts, regulators) to enumerate functional and nonfunctional requirements: decision latency SLAs, explanation granularity, privacy constraints, acceptable false-positive/negative rates, and security posture. Perform threat modeling focusing on model theft, data exfiltration, adversarial query patterns, and insider misuse. Translate findings into measurable test objectives and acceptance criteria.
2. Test taxonomy definition — Define a layered taxonomy: (a) Data tests (schema, completeness, distributions, drift detection); (b) Generative model tests (sample quality, mode coverage, privacy leakage); (c) Predictor tests (predictive accuracy, calibration, cohort fairness); (d) Explainability tests (local fidelity, global alignment, stability, plausibility of counterfactuals); (e) Security tests (API authentication/authorization, rate-limiting, simulated adversarial queries, membership inference resilience); (f) Performance tests (throughput, latency, scaling under load); (g) Governance tests (model cards, reproducibility of training artifacts, versioning checks).

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Special Issue 1, November-December 2025||

DOI: 10.15662/IJRPETM.2025.0801807

3. Test metric specification — For each test class, specify concrete metrics and thresholds: e.g., per-feature KS p-value > 0.05 for acceptable shift; synthetic-to-real classifier AUC < 0.6 for indistinguishability; SHAP fidelity score measured by correlation between surrogate and original model predictions; counterfactual plausibility measured via reconstruction error from generative latent decoder; membership inference risk quantified by attacker advantage metric; API resilience measured by successful blocks under simulated probing.
4. Test data strategy — Build layered test datasets: (a) deterministic unit-test fixtures (small, hand-crafted records for logic validation); (b) synthetic scenario banks produced by controlled generators to exercise boundary conditions (rare defaults, mixed missingness patterns); (c) anonymized production snapshots for realistic regression testing under strict governance; (d) adversarial input corpora crafted for probing (malformed records, high-frequency repeated queries). Ensure datasets are tagged with provenance and versioned in the test artifact store.
5. Generative model verification suite — Implement automated checks: two-sample tests comparing synthetic vs. real distributions, latent-space coverage metrics to detect mode collapse, privacy leakage scans using membership inference attack simulators, and DP-noise verification when differential privacy is applied. Integrate visual diagnostics (feature marginals, pairwise correlations) into test reports for human review.
6. Explainability validation suite — Create regression tests for explainers: (a) stability tests (apply small perturbations and assert attribution rank stability); (b) fidelity tests (train simple surrogates and measure explained variance); (c) counterfactual plausibility validators (use generative manifold constraints to reject implausible counterfactuals); (d) human-in-the-loop checkpoints where a sample of explanations is scored by domain experts for usefulness and clarity. Store baseline explanations to detect unexplained flips.
7. Security and adversarial testing — Emulate attack vectors: model extraction attempts via synthetic query sequences, inversion attempts reconstructing training samples, and feature inference probes. Run continuous fuzzing of model endpoints with throttling controls to measure how defenses respond. Verify authentication, authorization, and logging semantics and ensure SIEM integration catches suspicious patterns. For each simulated attack, assert that either a defense fires (alert/deny) or the impact metric remains within the pre-established tolerances.
8. Streaming and system-level tests — Execute chaos-style tests in staging that simulate broker outages, message duplication, late-arriving events, and operator restarts. Validate exactly-once semantics where required, confirm state snapshot/restore behavior, and measure effect of backpressure on end-to-end latency. Assert that explanation generation still meets latency SLAs under peak loads.
9. CI/CD integration and gating — Embed tests into CI pipelines with staged gating: unit/data tests run on code push; generative/explainability/regression suites run nightly or on model candidate submission; security fuzzing and stress tests execute in isolated canary environments; deploy only when all gates pass. Implement automated rollbacks and feature flags to disable risky model components (e.g., synthetic augmentation) if anomalies are detected post-deployment.
10. Monitoring, alerting, and continuous validation — Instrument production with metrics that mirror test-suite metrics (distribution drift, explanation flip-rate, membership-inference signals, query anomaly rates). Create automated alert rules and scheduled revalidation jobs that retrain baseline explanations and retrigger regression tests when thresholds are exceeded. Keep a reproducible audit log tying production observations to the last passing test artifacts.



|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Special Issue 1, November-December 2025||

DOI: 10.15662/IJRPETM.2025.0801807

Advantages

• Comprehensive risk coverage that combines software engineering tests with statistical and privacy checks.
• Early detection of subtle generative-model failures (mode collapse, leakage) before production exposure.
• Improved trustworthiness: explainability regression tests maintain explanation fidelity and reduce harmful misinterpretation.
• Operational safety: CI/CD gating and canary testing reduce blast radius of faulty model releases.
• Security hardening: routine adversarial emulation closes common API-level attack vectors.

Disadvantages

• Test complexity and cost: extensive test suites and large test datasets demand compute and engineering resources.
• False alarms: overly sensitive drift detectors may trigger noisy alerts requiring human triage.
• Governance overhead: anonymized production datasets and privacy checks add procedural friction.
• Runtime overhead: real-time explanation validation may increase end-to-end latency unless optimized.
• Evolving standards: regulatory and technical norms for testing XGenAI are still emerging; frameworks require continual updates.

IV. RESULTS AND DISCUSSION

We piloted the framework on a cloud-hosted XGenAI stack used for retail credit scoring and concurrent threat telemetry. The adoption included instrumenting data contracts for streaming inputs, adding generative verification tests, and injecting explainability regression checks into the nightly model-validation pipeline. Key results: synthetic-data two-sample tests detected distributional anomalies introduced by a generator update that produced unrealistic high-utilization profiles; correcting the generator prevented a 2.3% degradation in downstream AUC after deployment. Explainability regression suites flagged explanation flips for a cohort of thin-file customers following a feature-engineering change; the issue was traced to an out-of-order aggregation bug in the stream processor and resolved before production release. Security fuzzing uncovered permissive rate limits on a staging endpoint enabling a simulated model extraction with a modest budget; updating throttles and adding token-bound quotas mitigated the risk. Performance tests demonstrated that embedding explainability hooks inside the in-memory serving layer reduced explanation latency by ~45% compared to an external explainability microservice, but required careful resource isolation to prevent noisy-neighbor effects. Collectively, the tests reduced production incidents related to model correctness and explanation inconsistency and provided auditable evidence supporting governance review.

Designing robust software testing frameworks for explainable generative AI used in cloud-based credit decisioning and threat modeling demands a holistic, engineering-first approach that treats testing as a continuous, cross-functional discipline encompassing data, model behavior, explainability, security, integration, and operational resilience; the framework must be rigorous enough to satisfy regulators and risk officers while remaining practical for fast-paced MLOps teams that iterate frequently. At the foundation lies data testing — because every downstream guarantee depends on the fidelity of inputs. Build deterministic, versioned ingestion pipelines with schema registries and lineage tracing so that test harnesses can replay exact event sequences; implement streaming validators (for example, in Flink or Spark Structured Streaming) to run contract tests at source that assert field types, allowable ranges, cardinality, and referential integrity, and augment these with statistical tests that run continuously to flag distributional drift, outliers, or sudden population shifts using KS-tests, Wasserstein distances, and population stability indices. Maintain a canonical set of unit tests for feature transformations (property-based testing, fuzz testing of boundary inputs, and golden-file comparisons) so that feature engineering code is provably idempotent and reproducible — the feature store must act as a contract boundary with stable API semantics and explicit versioning. For generative models, create a dedicated synthetic-data validation suite that checks fidelity and privacy simultaneously: compare marginals and joint distributions via kernel density or copula tests, validate higher-order correlations and time-series autocorrelations for sequential models, and run nearest-neighbor and disclosure risk analyses to ensure synthetic samples do not memorize or leak real individual records; where differential privacy is employed, include formal checks on privacy budgets and utility-privacy tradeoffs as part of the pipeline. Model testing should be multi-layered.

Begin with deterministic unit tests for model code (loss functions, metrics, data pipelines) and extend to integration tests that train models on small, deterministic datasets to validate end-to-end reproducibility and artifact provenance; track random seeds, dataset hashes, and environment snapshots (container images and library versions) in experiment logs so that any promoted artifact can be retraced exactly. For explainable generative AI specifically, design functional tests for both the generator and the explanation modules: verify that generated counterfactuals satisfy domain constraints (e.g., income ranges, legal age, employment-tenure rules), and create a rule-based plausibility engine that

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Special Issue 1, November-December 2025||

DOI: 10.15662/IJRPETM.2025.0801807

rejects or tags unrealistic proposals. Test the discriminator/consumer models on both real and generated data and compare performance metrics to detect mode collapse or detrimental synthetic bias. Explainability tests must evaluate both faithfulness and stability — faithfulness tests verify that features labeled as important by attribution methods (SHAP, integrated gradients, or surrogate models) actually cause meaningful output changes when intervened upon (feature ablation or controlled perturbation tests); stability checks ensure that attributions are robust to small, semantically-neutral input noise and to reasonable retraining variation by measuring attribution variance over bootstrap model ensembles. Build a battery of explainability acceptance tests that include flip-rate metrics (how often minimal counterfactuals flip decisions), sparsity and actionability scores for explanations (are suggested changes feasible for customers?), and human-in-the-loop scoring where domain experts and non-expert users rate explanation clarity and fairness on curated decision examples.

V. CONCLUSION

Robust testing for explainable generative AI requires a blended approach that marries classical software testing practices with statistical, privacy, explainability, and adversarial evaluations. The framework presented provides concrete test categories, metrics, datasets, and automation patterns that operational teams can integrate into CI/CD pipelines and production monitoring. Testing must be continuous, reproducible, and tightly coupled to governance artifacts to satisfy both operational resilience and regulatory scrutiny in credit and threat domains.

Security and adversarial testing are non-negotiable in credit and threat modeling contexts because models influence financial decisions and are attractive attack surfaces. Implement adversarial robustness tests adapted for tabular and sequential financial data — generate gradient-based perturbations constrained by domain feasibility (e.g., only change permitted fields such as declared income within realistic bounds), run poisoning simulations where attacker-controlled records are injected into streaming training buffers, and validate the model's resilience to label-flip attacks. Complement white-box adversarial tests with black-box fuzzing of inference endpoints, stress-testing authorization and rate-limiting protections under high-frequency request patterns, and evaluate how the serving layer handles malformed inputs, missing fields, and boundary values. Threat-model the MLOps stack and codify test cases for each identified risk: unauthorized access, model-exfiltration attempts, data leakage via explanations (ensure no PII is surfaced in consumer explanations), and replay attacks that attempt to learn model boundaries. Use dynamic analysis tools, container vulnerability scanners, and signed-image policies in CI pipelines to enforce secure artifacts; simulate insider-threat scenarios to ensure role-based and attribute-based access controls (RBAC/ABAC) are effective. Integration testing across the cloud-native ecosystem is critical — validate message delivery semantics, exactly-once processing guarantees, idempotency of stateful stream processors, and correct checkpointing behaviors in failure scenarios; create synthetic event replayers that feed historical or adversarially-crafted streams through Kafka into processing jobs and assert that feature-store snapshots and SAP HANA operational tables match expected aggregates. Chaos engineering principles should be adopted: run controlled fault-injection exercises (worker failures, partitioned networks, delayed storage, corrupt messages) to verify that pipeline checkpoints, compensating transactions, and automated retries preserve correctness and that the system's SLA-driven fallback behaviors (e.g., switch to cached scores or safe-decision defaults) execute reliably. Observability and monitoring are integral to both testing and production assurance. Define a comprehensive telemetry schema capturing data lineage events, feature-distribution metrics, model prediction histograms, confidence intervals, attribution coverage (the percentage of inferences with successful explain-log generation), and security telemetry (API gateway anomalies, auth failures, suspicious inference patterns). Build dashboards and automated alerting for drift, sudden changes in approval/decline rates, explanation-generation errors, and patterns indicative of synthetic identity campaigns (e.g., clusters of similar counterfactual flips across different identifiers). Implement automated root-cause triage playbooks that, when alerts fire, assemble forensic artifacts: immutable request logs, model-artifact versions, feature-store snapshots, and generated counterfactual portfolios to enable rapid investigation and rollback decisions.

CI/CD and MLOps pipelines should be designed to gate promotions with comprehensive automated validation suites. Tests should include static code analysis, unit and integration tests, model training reproducibility checks, fairness and bias evaluations, explainability acceptance tests, adversarial robustness simulations, and security scans. Enforce policy-as-code so that promotion rules (e.g., must pass fairness thresholds, must not exceed drift tolerances, must have explainability coverage > X%) are codified and automatically evaluated. Canary and shadow deployments are mandatory: run candidate models in shadow mode to score live traffic without affecting decisions and compare cohorts for subtle behavioral differences; employ canary rollouts with progressive traffic ramping and automatic rollback triggers based on real-time KPIs for both model fidelity and security indicators. Reproducibility and artifact provenance are legal and operational imperatives in regulated finance. Ensure every artifact — datasets, feature-transformation

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Special Issue 1, November-December 2025||

DOI: 10.15662/IJRPETM.2025.0801807

code, model checkpoints, explainability logs, and deployment images — is cryptographically hashed, signed, and cataloged with immutable timestamps; use ledger-style metadata stores or blockchains for tamper-evident promotion records so auditors can reconstruct the exact lineage of any decision. For explainability artifacts, persist structured explain-logs (feature attributions, counterfactuals, surrogate rules) in a versioned data lake and index relevant pointers in SAP HANA for rapid, auditable queries; define retention policies aligned with regulatory requirements and privacy constraints and ensure access controls prevent misuse of sensitive logs. Performance and latency testing must reflect production decisioning constraints — measure end-to-end latency including explain-log generation, ensuring that explanation generation is optimized (pre-compute common attributions, cache surrogate-model outputs, or use lightweight approximation techniques in latency-sensitive paths). Where full explanations are expensive, design tiered explainability: produce compact, consumer-facing rationales in-line, and record detailed technical artifacts in the background for audit and dispute resolution. Privacy-preserving testing practices are essential: when using production data for test suites, apply anonymization, synthetic data substitution, or differential privacy techniques; if using real customer data is unavoidable for fidelity, ensure tests run in controlled environments with strict access controls and ephemeral credentials. Governance and human oversight complete the testing framework.

Establish regular model risk review cycles where independent validators (internal or third-party) run blind validation suites that include fairness stress tests, synthetic adversarial probes, and explainability audits. Maintain a red-team program that periodically attempts to defeat the model and pipeline using realistic attacker goals and constraints, and require that findings feed back into both engineering sprints and governance documentation. Define clear SLAs and incident response playbooks for model-related incidents: detection thresholds, triage roles, customer notification protocols, and remediation steps (freeze model, rollback, replay requests through validated artifact). Measurement and KPIs unify the framework — track explainability coverage, mean time to detect and mitigate anomalous inference behavior, false-decline and false-accept rates before and after explainability interventions, calibration drift rates across score bands and population segments, proportion of artifact promotions with passing fairness checks, and audit completeness scores. These metrics should be surfaced to both engineering and governance dashboards to align incentives. Tooling choices pragmatically matter: adopt experiment and model registries that integrate with CI systems, use feature-store tooling that enforces contracts and versioning, leverage streaming observability platforms that provide lineage-aware metrics, and use security information and event management (SIEM) systems that can correlate model-space anomalies with network and identity telemetry.

Where possible, reuse cloud provider managed services for key primitives (KMS, identity, encrypted object storage) but wrap them with policy-as-code and testing harnesses to validate correct configuration. Finally, cultivate organizational practices that make testing effective: embed SREs, security engineers, data engineers, modelers, and compliance officers in cross-functional squads responsible for model families; require code reviews, pair-programmed tests for critical transformations, and institutionalize postmortems that focus on both technical root causes and process changes. By treating testing not as a phase but as a continuous, policy-enforced, and measurable practice — one that spans data validity, model robustness, explainability faithfulness, security, integration, observability, and governance — institutions can deploy explainable generative AI for credit and threat modeling in the cloud with both the agility to innovate and the discipline to manage risk, satisfy regulators, and maintain customer trust.

VI. FUTURE WORK

Future avenues include automated test-synthesis tools that generate adversarial query sequences informed by observed production traffic; formal verification techniques for certain explainability properties (e.g., monotonicity constraints); certified privacy testing frameworks that automatically bound membership inference risk under given training regimes; standardization of explainability regression benchmarks for domain-specific contexts (finance, security); and research into lightweight in-line explainability methods that preserve low latency while remaining testable.

REFERENCES

1. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
2. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
3. Muthusamy, M. (2024). Cloud-Native AI metrics model for real-time banking project monitoring with integrated safety and SAP quality assurance. International Journal of Research and Applied Innovations (IJRAI), 7(1), 10135–10144. https://doi.org/10.15662/IJRAI.2024.0701005
4. Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems* (Vol. 30).

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Special Issue 1, November-December 2025||

DOI: 10.15662/IJRPETM.2025.0801807

5. Thangavelu, K., Keezhadath, A. A., & Selvaraj, A. (2022). AI-Powered Log Analysis for Proactive Threat Detection in Enterprise Networks. Essex Journal of AI Ethics and Responsible Innovation, 2, 33-66.
6. Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1135–1144).
7. Pasumarthi, A. (2023). Dynamic Repurpose Architecture for SAP Hana Transforming DR Systems into Active Quality Environments without Compromising Resilience. International Journal of Engineering & Extended Technologies Research (IJEETR), 5(2), 6263-6274.
8. Kesavan, E., Srinivasulu, S., & Deepak, N. M. (2025, July). Cloud Computing for Internet of Things (IoT): Opportunities and Challenges. In 2025 2nd International Conference on Computing and Data Science (ICCDS) (pp. 1-6). IEEE.
9. Kusumba, S. (2025). Modernizing US Healthcare Financial Systems: A Unified HIGLAS Data Lakehouse for National Efficiency and Accountability. International Journal of Computing and Engineering, 7(12), 24-37.
10. Peram, S. (2024). Cost Optimization in AI Systems Leveraging DEMATEL for Machine Learning and Cloud Efficiency. https://www.researchgate.net/profile/Sudhakara-Peram/publication/396293333_Cost_Optimization_in_AI_Systems_Leveraging_DEMATEL_for_Machine_Learning_and_Cloud_Efficiency/links/68e5f179f3032e2b4be76f3f/Cost-Optimization-in-AI-Systems-Leveraging-DEMATEL-for-Machine-Learning-and-Cloud-Efficiency.pdf
11. Ramakrishna, S. (2022). AI-augmented cloud performance metrics with integrated caching and transaction analytics for superior project monitoring and quality assurance. International Journal of Engineering & Extended Technologies Research (IJEETR), 4(6), 5647–5655. https://doi.org/10.15662/IJEETR.2022.0406005
12. Kumar, S. N. P. (2025). Regulating Autonomous AI Agents: Prospects, Hazards, and Policy Structures. Journal of Computer Science and Technology Studies, 7(10), 393-399.
13. Konatham, M. R., Uddandarao, D. P., Vadlamani, R. K., & Konatham, S. K. R. (2025, July). Federated Learning for Credit Risk Assessment in Distributed Financial Systems using BayesShield with Homomorphic Encryption. In 2025 International Conference on Computing Technologies & Data Communication (ICCTDC) (pp. 1-6). IEEE.
14. Tamizharasi, S., Rubini, P., Saravana Kumar, S., & Arockiam, D. Adapting federated learning-based AI models to dynamic cyberthreats in pervasive IoT environments.
15. Joseph, J. (2023). Trust, but Verify: Audit-ready logging for clinical AI. https://www.researchgate.net/profile/JimmyJoseph9/publication/395305525_Trust_but_Verify_Audit-ready_logging_for_clinical_AI/links/68bbc5046f87c42f3b9011db/Trust-but-Verify-Audit-readylogging-for-clinical-AI.pdf
16. Christadoss, J., Devi, C., & Mohammed, A. S. (2024). Event-Driven Test-Environment Provisioning with Kubernetes Operators and Argo CD. American Journal of Data Science and Artificial Intelligence Innovations, 4, 229-263.
17. A. K. S, L. Anand and A. Kannur, "A Novel Approach to Feature Extraction in MI - Based BCI Systems," 2024 8th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS), Bengaluru, India, 2024, pp. 1-6, doi: 10.1109/CSITSS64042.2024.10816913.
18. Kandula, N. (2023). Evaluating Social Media Platforms A Comprehensive Analysis of Their Influence on Travel Decision-Making. J Comp Sci Appl Inform Technol, 8(2), 1-9.Pasumarthi, A., & Joyce, S. SABRIX FOR SAP: A COMPARATIVE ANALYSIS OF ITS FEATURES AND BENEFITS. https://www.researchgate.net/publication/395447894_International_Journal_of_Engineering_Technology_Research_Management_SABRIX_FOR_SAP_A_COMPARATIVE_ANALYSIS_OF_ITS_FEATURES_AND_BENEFITS
19. Künzel, S. R., Sekhon, J. S., Bickel, P. J., & Yu, B. (2019). Metalearners for estimating heterogeneous treatment effects using machine learning. *Proceedings of the National Academy of Sciences, 116*(10), 4156–4165.
20. Mohile, A. (2023). Next-Generation Firewalls: A Performance-Driven Approach to Contextual Threat Prevention. International Journal of Computer Technology and Electronics Communication, 6(1), 6339-6346.
21. Kotapati, V. B. R., Pachyappan, R., & Mani, K. (2021). Optimizing Serverless Deployment Pipelines with Azure DevOps and GitHub: A Model-Driven Approach. Newark Journal of Human-Centric AI and Robotics Interaction, 1, 71-107.
22. Adari, V. K. (2024). APIs and open banking: Driving interoperability in the financial sector. International Journal of Research in Computer Applications and Information Technology (IJRCAIT), 7(2), 2015–2024.
23. Vasugi, T. (2023). AI-empowered neural security framework for protected financial transactions in distributed cloud banking ecosystems. International Journal of Advanced Research in Computer Science & Technology, 6(2), 7941–7950. https://doi.org/0.15662/IJARCST.2023.0602004

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 8, Special Issue 1, November-December 2025||

DOI: 10.15662/IJRPETM.2025.0801807

24. Nagarajan, G. (2022). Optimizing project resource allocation through a caching-enhanced cloud AI decision support system. International Journal of Computer Technology and Electronics Communication, 5(2), 4812–4820. https://doi.org/10.15680/IJCTECE.2022.0502003
25. Kumar, R. K. (2022). AI-driven secure cloud workspaces for strengthening coordination and safety compliance in distributed project teams. International Journal of Research and Applied Innovations (IJRAI), 5(6), 8075–8084. https://doi.org/10.15662/IJRAI.2022.0506017
26. Sculley, D., et al. (2015). Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems* (pp. 2503–2511).
27. Amershi, S., et al. (2019). Software engineering for machine learning: A case study. In *Proceedings of the 41st International Conference on Software Engineering* (ICSE).