International Journal of Artificial Intelligence, Data Science, and Machine Learning
Grace Horizon Publication | Volume 7, Issue 1, 141-148, 2026
ISSN: 3050-9262 | https://doi.org/10.63282/3050-9262.IJAIDSML-V7I1P126

Original Article

Scalable Adaptive Learning for Early Software Fault Prediction in Agile: From Reliability Gains to Sprint Planning Optimization

Mohit Bansal[1], Tanvi Arora[2], Rahul Chatterjee[3], Simran Kaur[4]
[1,4]Computer Science Department Shiv Nadar University Greater Noida, Uttar Pradesh, India.
[2]Artificial Intelligence Department Shiv Nadar University Greater Noida, Uttar Pradesh, India.
[3]Information Technology Shiv Nadar University Greater Noida, Uttar Pradesh, India.

Received On: 03/01/2026 ***Revised On: 06/02/2026*** ***Accepted On: 08/02/2026*** ***Published on: 10/02/2026***

Abstract *- Agile teams increasingly deliver software through rapid iterations, cloud-native deployment, and continuously evolving architectures. While this cadence accelerates value delivery, it also amplifies the operational cost of faults: late discovery increases rework, destabilizes sprint commitments, and elevates reliability risk. Traditional software defect prediction has produced strong results in offline, within-project settings, yet common limitations remain for modern Agile delivery: models degrade under concept drift, labels arrive late (often after release), and predictions are rarely integrated into sprint planning decisions in a decision-traceable manner. This manuscript proposes Scalable Adaptive Learning for Sprint Analytics (SALSA), a framework that couples adaptive fault prediction with sprint planning optimization. SALSA unifies drift-aware continual learning, transfer and federated strategies for low-data or privacy-constrained contexts, effort-aware evaluation aligned with Agile quality assurance capacity, and interpretable risk rationales suitable for governance in regulated environments. The framework extends beyond probability of fault to estimate reliability gain and to optimize sprint backlog selection under capacity constraints, balancing feature delivery with risk reduction. A worked sprint planning example demonstrates how predicted risk, effort, and expected reliability gain can be translated into a concrete backlog recommendation and quality budget allocation. The manuscript also outlines a replication-ready evaluation protocol emphasizing time-ordered validation, cost-effectiveness, and measurable planning outcomes. By connecting early fault prediction to sprint-level decision making, SALSA reframes defect prediction as an operational capability that improves both software reliability and sprint planning efficiency.*

Keywords *- Agile Software Development, Software Defect Prediction, Continual Learning, Concept Drift, Effort-Aware Metrics, Explainable AI, Federated Learning, Sprint Planning Optimization, Software Reliability, Devops Analytics.*

1. Introduction

Agile methods emphasize iterative delivery, frequent reprioritization, and short feedback loops. Practical sprint planning depends on limited capacity and uncertain information, especially when technical debt and latent defects compete with feature work. Agile planning guidance underscores the need for reliable estimation, transparent prioritization, and continuous refinement as information changes during execution [1]. However, fault-related work is often reactive: defects are detected late in the sprint or post-release, generating unplanned work, spillover, and reliability regressions.

Software defect prediction seeks to anticipate fault-prone artifacts so quality assurance effort can be focused earlier. Large-scale benchmarking studies show that many classifiers can be competitive when evaluated on standard metrics, but performance varies by dataset characteristics and experimental design [2]. Earlier work demonstrates that static code attributes can support useful predictors, indicating that repository-mined features can identify risk hot spots before runtime failures manifest [3].

Agile delivery introduces additional challenges. Cross-project generalization is difficult because feature distributions shift across domains and processes, and prediction success depends on data and process compatibility [4]. Moreover, Agile teams often require just-in-time risk signals near the moment of code change rather than coarse, release-level predictions. Just-in-time quality assurance research addresses this need by linking change-level signals to defect proneness in time-sensitive workflows [5].

Modern delivery pipelines intensify these problems through drift: codebases evolve rapidly, team practices shift, architecture refactoring changes coupling patterns, and platform migrations alter telemetry signals. Continual learning literature emphasizes that models trained on stationary batches can fail under nonstationary streams and can catastrophically forget past knowledge unless adaptation is managed explicitly [6]. In practice, reliability

improvements also depend on adjacent engineering practices such as automated testing strategies. Comparative analysis of testing frameworks in enterprise Java systems shows reliability sensitivity to testing approach and coverage mechanisms, reinforcing that predictive models should be connected to actionable interventions rather than treated as passive scoring tools [7]. Transfer learning has emerged as a key approach for low-data scenarios, aiming to leverage source projects when target data are limited, though distribution mismatch remains a principal barrier [8].

Recent work explicitly targets Agile fault prediction with scalable and adaptive machine learning, connecting early prediction to reliability and sprint planning efficiency [9]. Building on this direction, the present manuscript addresses an additional operational gap: prediction outputs are often not translated into sprint planning decisions under explicit capacity constraints, nor are they consistently governed through interpretable, auditable rationales.

This paper makes four contributions. First, it introduces SALSA, an end-to-end architecture for adaptive early fault prediction integrated with sprint planning optimization. Second, it specifies a drift-aware adaptive learning design combining continual learning with transfer and federated options. Third, it defines an effort- and planning-aligned evaluation and decision formulation that operationalizes predicted risk into reliability gain and sprint decisions. Fourth, it proposes decision transparency artifacts, enabling engineering governance compatible with high-stakes and regulated contexts.

2. Background and Related Work

2.1. Scalable Learning Signals in Modern Agile Systems

Agile fault prediction increasingly relies on heterogeneous signals beyond classic static metrics. Real-time and resource-constrained contexts motivate lightweight learning pipelines, where feature extraction and inference must remain efficient even under edge or near-edge constraints [10]. Many enterprise systems adopt microservices and cloud platforms, introducing service boundaries, distributed tracing, and operational telemetry as relevant predictors. Secure, compliant microservices architectures on platforms such as AWS and OpenShift highlight the operational complexity and the importance of reliability in regulated domains [11]. These characteristics motivate feature sets that incorporate deployment, runtime, and configuration signals alongside code and process metrics.

A central methodological issue is that defect prediction performance metrics must align with the resource budget of quality work. Effort-aware defect prediction emphasizes that artifacts differ in treatment cost, and that evaluation should measure cost-effectiveness rather than only classification accuracy [12]. Deep learning approaches have been applied to fault prediction, including CNN and RNN models that capture structural and sequential patterns, suggesting improved representation learning for complex software artifacts [13]. These methods are promising but can be brittle under drift and often require careful interpretability when deployed into planning workflows.

2.2. Explainability and Governance for Prediction-Guided Decisions

When predictions drive engineering actions, transparency becomes critical. High-stakes AI deployments in regulated environments require explainability and auditable decision pathways to establish trust and operational governance [14]. Local explanation methods such as LIME provide instance-level rationales that help stakeholders understand why a specific artifact is flagged as high risk [15]. In regulated operational pipelines that combine OCR, machine learning, and microservices, the need for traceability extends across the full processing chain and motivates governance patterns applicable to software delivery analytics as well [16]. Broader frameworks for explainable AI in high-stakes domains emphasize interpretability as a system-level requirement, not only a model property [17]. SHAP provides a theoretically grounded feature attribution approach that supports consistent local and global explanations across model classes, enabling comparative reasoning and decision review [18].

2.3. Comparative Defect Prediction, Privacy, and Operational Decision Making

Comparative studies of machine learning models for defect prediction demonstrate that no single learner dominates across datasets, motivating ensembles, meta-learning, and adaptive selection [19]. Privacy constraints also arise in multi-team and multi-organization contexts, where sharing raw defect data or code metrics may be restricted. Federated learning architectures for privacy-preserving detection tasks show how distributed training can occur without centralized data pooling, albeit with governance and protocol complexity [20].

Operational performance improvements in distributed systems also depend on platform-level optimizations such as predictive analytics and caching strategies, which can stabilize response times and reduce failure modes, reinforcing the importance of incorporating operational telemetry and intervention levers into prediction workflows [21]. In addition, AIOps-oriented root cause analysis and automated remediation for multi-system data integrity issues demonstrates how predictive and diagnostic analytics can reduce recovery time and improve operational reliability, offering an analogy for integrating fault prediction with actionable remediation in Agile settings [22]. Lightweight baselines such as decision trees and KNN remain relevant for interpretability and deployment simplicity, and comparative evaluation of these approaches provides useful anchors for understanding tradeoffs under constraints [23]. Federated learning at scale further motivates decentralized training designs suitable for privacy-preserving multi-team learning [24].

2.4. Decision Intelligence, Scalable Analytics Platforms, and Enterprise Modernization

Agile governance frameworks that combine decision intelligence and defect prediction highlight the value of architecture-centered lifecycle governance, linking predictive signals to planning and control processes [25]. Scalability considerations extend to compute-intensive simulation and analytics, where machine learning integrated with HPC pipelines illustrates design patterns for performance, reproducibility, and large-scale experimentation [26].

Enterprise analytics platforms increasingly leverage high-performance in-memory systems to support real-time decision workloads; research on SAP S/4HANA database layers illustrates how low-latency access patterns enable operational analytics [27]. Architecture modernization affects fault behavior: monolith-to-microservices transitions and gateway optimization can change failure propagation and observability patterns, requiring models to adapt to new architectural feature distributions [28]. Deep learning systems in other domains, such as emotion understanding pipelines, further demonstrate how multi-stage workflows combine representation learning, calibration, and human-facing interpretation, relevant for structuring explainable analytics in engineering processes [29].

2.5. Automation, Observability, and Data Engineering for Adaptive Learning

Automated model selection and low-code machine learning pipelines can reduce adoption friction, particularly when retraining must occur frequently under drift [30]. Cloud-native monitoring and deployment optimization studies emphasize the role of observability tooling and Helm-based deployment strategies in maintaining reliability, highlighting telemetry sources that can enrich fault prediction models [31]. Federated AIOps frameworks for multi-cluster OpenShift show how distributed operational intelligence can be organized across clusters, aligning with multi-team Agile environments where training data and telemetry are naturally partitioned [32].

Machine learning-driven risk detection in CI/CD provides a direct connection between pipeline events and operational risk signals, supporting earlier intervention in delivery workflows [33]. Security and anomaly detection research for data in transit illustrates robust detection patterns that can inspire analogous anomaly signals in software delivery telemetry [34]. Predictive monitoring and error mitigation in change-data-capture pipelines further supports the argument that early-warning systems reduce downstream recovery cost in complex integrations [35].

Enterprise transition guidance for SAP Fiori in S/4HANA modernization highlights integration challenges and strategic implications, analogous to how modernization affects engineering process, telemetry, and defect distributions in Agile teams [36]. Enhancing OCR accuracy through neural networks shows how calibration and data-quality improvements translate into reduced error rates in high-stakes workflows, reinforcing the importance of continual tuning for performance stability [37]. CRM and deep learning systems for patient journey mapping highlight end-to-end data integration and predictive engagement pipelines, providing parallels for integrating multiple signals into coherent decision support [38]. Convergence optimization studies emphasize training stability, relevant for continual retraining regimes where frequent updates must remain stable and cost-effective [39].

Finally, performance comparison of defect prediction models reinforces the need for robust baseline evaluation and repeatable experimentation under Agile constraints [40]. Unified data engineering for smart mobility demonstrates real-time integration across heterogeneous streams, analogous to unifying SCM, CI/CD, and observability sources into a single feature plane for prediction [41]. Forecasting and decision support patterns in portfolio intelligence under uncertainty motivate analogous sprint portfolio selection under risk [42]. Platform comparisons such as Pivotal Cloud Foundry versus OpenShift emphasize that deployment substrates affect observability and performance, which can indirectly influence fault patterns and model features [43]. Graph neural networks for knowledge representation motivate encoding software architecture and dependency structures as graphs for more expressive risk modeling [44].

3. Problem Statement and Research Gap

3.1. Problem Statement

Consider an Agile project delivering work in discrete sprints indexed by t. Let A_t = {a_{t,1}, a_{t,2}, ..., a_{t,n_t}} denote candidate work items at sprint planning time. A work item may correspond to a user story, a change set, or a set of files and services impacted by planned work. Each item a_{t,i} has estimated effort e_{t,i} (e.g., story points) and impacts a set of artifacts M_{t,i}. The operational goal is to (i) predict early fault risk for impacted artifacts before or during sprint execution and (ii) optimize sprint selection and quality allocation under a capacity constraint E_t to maximize expected reliability improvement while limiting spillover and unplanned rework.

3.2. Research Gap

Despite extensive defect prediction research, several end-to-end gaps persist for modern Agile delivery. First, nonstationarity is pervasive; models can degrade rapidly as code, architecture, and delivery pipelines evolve, requiring drift-aware adaptation rather than fixed offline training [6]. Second, action alignment is often weak: models provide scores but do not translate those scores into sprint decisions or explicit quality budgets under capacity constraints [1]. Third, effort and cost are frequently implicit; quality actions differ in effort and opportunity cost, so usefulness must be evaluated via cost-effectiveness and fixed-effort perspectives [12]. Fourth, governance and interpretability are required when predictions influence planning; stakeholders need auditable rationales and stable explanations [14]. Fifth, multi-team environments introduce privacy and data-sharing constraints, motivating transfer and federated strategies that

preserve confidentiality while still learning from distributed experience [20].

4. Salsa Framework Overview

4.1. Architecture and Data Flow

SALSA comprises five layers: (1) signal ingestion from source control, work-item tracking, CI/CD pipelines, and runtime observability; (2) a time-indexed feature and label store supporting reproducible time splits and drift monitoring; (3) an adaptive model service implementing continual learning and optional transfer or federated learning; (4) an explanation and governance service that produces rationale artifacts for engineers and stakeholders; and (5) a sprint planning optimizer that translates predicted risk into backlog selection and quality allocation decisions.

4.2. Core Outputs and Interfaces

For each artifact m at sprint time t, SALSA outputs a calibrated probability p_t(m) of fault manifestation within a defined prediction window, an uncertainty estimate u_t(m) that supports active learning and targeted review, and an explanation x_t(m) that identifies the most influential signals for that prediction. For each work item a, artifact-level outputs are aggregated into an item-level risk score R_t(a), a predicted reliability gain under candidate interventions, and a recommended quality action plan. These outputs are delivered through dashboards, CI/CD gates, and planning-time reports designed to be reviewed during sprint ceremonies.

5. Adaptive Learning for Early Fault Prediction

5.1. Feature Construction

SALSA organizes features into five groups. Static code features capture complexity, size, coupling, and change-proneness proxies, consistent with evidence that static attributes enable effective predictors [3]. Change and process features capture churn, review activity, and developer interaction patterns aligned with just-in-time quality assurance practices [5]. Testing features incorporate coverage deltas, flaky-test signals, and framework-specific indicators; this connects predicted risk to actionable test strategy changes, reflecting reliability sensitivity to automated testing practices in enterprise systems [7]. Architecture and dependency features represent service interactions and call graphs and can be encoded through graph representations where available [44]. Operational features include deployment frequency, rollback events, latency and error trends, and cache-related indicators, motivated by cloud-native observability research and performance optimization patterns [31].

5.2. Model Family and Calibration

SALSA uses a hybrid ensemble to match heterogeneous signal types: gradient-boosted decision trees for tabular metrics, sequence models for time-ordered change histories, and optional graph components for dependency structure. The ensemble outputs a probability estimate that is calibrated using periodic reliability calibration. Calibration is essential because planning decisions depend on interpretable probabilities; miscalibration can cause teams to over-allocate quality work or ignore important risk signals. Calibration refresh is triggered by drift detection and by significant shifts in class balance across sprints.

5.3. Drift Detection and Continual Updates

Drift is monitored through (i) feature distribution change metrics, (ii) prediction residual trends, and (iii) uncertainty increases for recurrent artifact families. When drift is detected, SALSA increases retraining frequency and expands the window of recent data used for adaptation while applying continual learning constraints to reduce catastrophic forgetting [6]. To keep frequent retraining stable, SALSA adopts conservative optimizer schedules and convergence monitoring, which is particularly relevant in short retraining cycles [39].

5.4. Transfer and Federated Learning Modes

In low-data projects, SALSA supports transfer learning by initializing target models using representations learned from related projects, then adapting with target-specific data to reduce mismatch [8]. In privacy-constrained multi-team settings, SALSA supports federated learning, where model updates rather than raw data are aggregated. Governance and protocol design follow privacy-preserving patterns established for distributed detection tasks, emphasizing secure aggregation, client selection policies, and auditability [20]. Distributed orchestration and monitoring are aligned with federated AIOps design patterns to improve reliability of the training service across clusters [32].

5.5. Explainability Artifacts for Engineers

SALSA generates instance-level explanations using local surrogate models for on-demand interpretability and stable additive attributions for consistent global reasoning [15]. These artifact-level explanations are complemented by decision-level narratives used in sprint planning: a short justification summarizing the dominant signals, their directionality, and the recommended intervention. This approach aligns with the needs of auditable decision pathways in regulated environments [14].

6. Reliability Gain Modeling

6.1. Linking Predictions to Interventions

Predictions are most valuable when paired with interventions that reduce fault probability. Let q denote a quality action such as targeted test creation, additional review, refactoring of a hotspot, or deployment hardening. For each artifact m and action q, SALSA estimates an intervention effectiveness factor $\eta(q,m)$ in [0,1], representing the expected proportional reduction in fault probability if the action is executed. Estimating η can use historical outcomes, test coverage deltas, and post-incident learning. Reliability-oriented engineering studies emphasize that intervention choice matters; testing strategy changes, monitoring improvements, and deployment optimizations can materially affect reliability outcomes [7].

6.2. Expected Prevented-Fault Mass

Let pt(m) be the predicted fault probability for artifact m at time t. For a work item a that touches artifacts M(a), the

expected prevented-fault mass under action q is $\Delta(a,q) = \Sigma_{m \in M(a)} p_t(m) \cdot \eta(q,m)$. This term can be converted into expected reliability gain by mapping prevented-fault mass into changes in defect density, escaped defect rates, or service-level error budgets. When operational telemetry features are included, intervention effects can incorporate system-level factors such as cache behavior and latency error reduction, reflecting patterns observed in predictive analytics and caching optimization [21].

6.3. Cost and Uncertainty Considerations

Quality actions consume capacity and may introduce uncertainty. SALSA treats the cost C(q) as effort in story points or engineer-hours and includes uncertainty $u_t(m)$ in prioritization. For high-uncertainty high-risk artifacts, SALSA can trigger active learning: a limited budget of manual review to improve labels and reduce uncertainty for future sprints. This makes the system adaptive not only via retraining but also via targeted data acquisition.

7. Sprint Planning Optimization

7.1. Optimization Formulation

Let E_t be sprint capacity. For each candidate work item a, let V(a) be business value, e(a) be implementation effort, and R(a) be aggregated risk. SALSA chooses a subset S_t of items and a quality action q(a) for each chosen item to maximize risk-adjusted utility: maximize $\Sigma_{a \in S_t} (V(a) + \lambda \cdot \Delta\rho(a,q(a)) - \mu \cdot R\_residual(a,q(a)))$ subject to $\Sigma\, e(a) + \Sigma\, C(q(a)) \leq E_t$. Here λ tunes the emphasis on reliability gains and μ penalizes residual risk after intervention. The formulation turns quality risk into an explicit planning tradeoff consistent with Agile planning principles [1].

7.2. Greedy and Solver-Based Strategies

Many teams prefer simple, explainable heuristics. SALSA provides a greedy strategy based on utility per unit cost and enforces a quality buffer for the top-risk tail. For organizations with mature operations research capability, SALSA can also export a mixed-integer formulation for solver-based optimization. Regardless of method, evaluation should be effort-aware because defect treatment cost varies; effort-aware metrics provide a more faithful view of planning benefit than raw classification statistics [12].

7.3. Interpretable Decision Logs

Every planning recommendation is accompanied by a decision log: top features driving risk, estimated effect of proposed interventions, and the constraint-based reason why certain items were included or excluded. This supports the auditability needed in governance-heavy environments and reduces resistance to model-guided planning [14].

8. Implementation Details and Mlops

8.1. Data Collection and Labeling

SALSA's signal ingestion pipeline collects source control metrics, issue tracker status transitions, CI/CD telemetry, and runtime observability events. Labeling is time-windowed: an artifact is labeled faulty if a defect linked to that artifact is introduced or detected within a defined horizon. This reduces leakage from future information and allows time-ordered validation. In environments where labels arrive late, SALSA maintains provisional labels and updates them as defects are confirmed, enabling continual refinement without corrupting evaluation splits.

8.2. Feature Store, Model Registry, and Deployment

A time-indexed feature store enables reproducible training and debugging. Model versions are tracked in a registry with metadata including training window, drift scores, calibration statistics, and explanation stability metrics. Deployment follows a canary pattern: new models score in shadow mode for a sprint window, then promote only if calibration and effort-aware metrics improve, aligning with reliability practices in cloud-native deployment optimization [31].

8.3. Security and Privacy Controls

In privacy-constrained settings, federated learning mode is used. Updates are encrypted and aggregated; clients are selected to balance representation and fairness. Anomaly detection applied to model update streams can detect poisoning or faulty clients, drawing inspiration from secure communication anomaly detection patterns [34]. In regulated environments, the governance layer stores explanation artifacts and decision logs to ensure traceability, analogous to governance requirements in regulated ML pipelines [16].

9. Evaluation Design

9.1. Datasets, Splits, and Baselines

Evaluation should use time-ordered splits to reflect deployment conditions: train on sprints 1..t, validate on t+1, and test on t+2, repeated across rolling windows. Cross-project experiments should explicitly report data, domain, and process differences that affect generalization [4]. Baselines include traditional models from benchmark studies [2], static-feature approaches [3], just-in-time predictors [5], deep learning variants [13], interpretable baselines such as decision trees and KNN [23], and recent comparative model sets [19].

9.2. Metrics and Outcomes

SALSA is evaluated on predictive, cost-effectiveness, and planning outcomes. Predictive metrics include AUC, F1, MCC, Brier score, and calibration error. Cost-effectiveness metrics include recall at fixed effort cutoffs and cost curves [12]. Planning outcomes include escaped defects per sprint, unplanned work ratio, spillover rate, and stability of sprint commitments. In CI/CD-integrated deployments, risk signals can also be evaluated against pipeline risk detection baselines that trigger earlier intervention based on build and deployment behaviors [33].

9.3. Drift and Adaptation Studies

A core evaluation dimension is robustness under drift. Experiments should compare performance decay of static models against SALSA's continual learning configuration. Drift detection quality can be assessed by correlating drift triggers with observed changes in defect density, architecture shifts, and platform migrations such as monolith-to-microservices transitions [28].

10. Case Study Walkthrough and Worked Example

10.1. Sprint Backlog Selection Example

Consider a sprint with capacity E_t = 20 story points and eight candidate items. Each item has an effort estimate, business value score, and aggregated risk score. Assume a standard quality action costs two points per selected item and yields reliability gain proportional to risk. SALSA ranks items by utility-per-cost and selects until capacity is exhausted while reserving a small quality buffer for emergent risk. This process explicitly trades business value against reliability gains and reduces late-sprint surprises.

10.2. Sensitivity to Reliability Weighting

Planning behavior depends on λ and μ. As λ increases, SALSA favors items with high reliability leverage even if business value is moderate; as μ increases, SALSA penalizes residual risk and may defer risky items that cannot be sufficiently mitigated within the sprint. Sensitivity analysis is essential for governance: teams can choose a reliability posture aligned with product criticality, compliance obligations, and error budgets. This mirrors how regulated domains require explicit and auditable decision thresholds [14].

10.3. Modernization and Telemetry-Rich Scenarios

In cloud-native modernization programs, observability signals and platform-level factors can materially affect fault patterns. Deployment substrate differences, monitoring configuration, and rollout strategies can change failure rates and detection latency. Comparative platform studies and monitoring optimization research illustrate that these operational choices influence reliability and should be represented in features and intervention modeling [43].

11. Discussion

11.1. Practical Adoption in Agile Rituals

SALSA is designed to fit Agile rituals. During backlog refinement, risk explanations help identify hidden complexity and test gaps; during sprint planning, the optimizer proposes a risk-adjusted sprint set; during execution, CI/CD gates flag high-risk changes for extra review. This approach encourages proactive quality rather than reactive defect triage. It also aligns with decision-intelligence-driven governance patterns where predictive analytics inform lifecycle control decisions [25].

11.2. Scalability, Compute, and Data Engineering Considerations

Scalability concerns arise in feature computation, retraining, and inference. Real-time integration of heterogeneous streams benefits from data engineering patterns that unify telemetry and event sources into consistent schemas [41]. Compute-intensive experimentation and simulation pipelines can support robust model selection and calibration at scale, leveraging HPC-inspired practices for reproducibility and throughput [26]. In-memory analytics platforms can reduce latency for feature retrieval and scoring in large enterprises [27].

11.3. Extensibility to Graph and Portfolio Reasoning

As system architectures become more interconnected, risk reasoning may benefit from graph representations of dependencies and knowledge. Graph neural network approaches motivate encoding service interactions and artifact relationships to propagate risk across dependency edges [44]. Portfolio optimization analogies in decision-support domains motivate treating sprint planning as a constrained portfolio problem under uncertainty, where risk and value are jointly optimized [42].

12. Threats to Validity

Key threats include label noise and delay (defects may be discovered after the sprint), confounding process changes (team turnover, refactoring campaigns, and tool migrations), and generalization limits across domains. Cross-project learning can fail without careful alignment of process and data assumptions [4]. Action-effect estimation can also be biased if intervention effectiveness is not calibrated using historical outcomes; rigorous post-sprint analysis is needed to update η estimates and prevent overconfident planning recommendations.

13. Conclusion and Future Work

This manuscript presented SALSA, a scalable adaptive learning framework that connects early software fault prediction to sprint planning optimization. By combining drift-aware continual learning, transfer and federated strategies, effort-aware evaluation, and interpretable explanations, SALSA operationalizes defect prediction as a planning and reliability capability rather than a standalone classifier. Future work includes empirical validation on multi-project time-ordered datasets and telemetry-rich enterprise environments, deeper graph-based modeling of dependency risk, and controlled studies measuring improvements in sprint stability and escaped defect reduction under different reliability-weight settings.

References

[1] M. Cohn, Agile Estimating and Planning. Upper Saddle River, NJ, USA: Prentice Hall, 2005.

[2] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, “Benchmarking classification models for software defect prediction: A proposed framework and novel findings,” IEEE Trans. Softw. Eng., vol. 34, no. 4, pp. 485–496, 2008, doi: 10.1109/TSE.2008.35.

[3] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” IEEE Trans. Softw. Eng., vol. 33, no. 1, pp. 2–13, 2007.

[4] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, “Cross-project defect prediction: A large scale experiment on data vs. domain vs. process,” in Proc. ESEC/FSE, 2009, pp. 91–100, doi: 10.1145/1595696.1595713.

[5] Y. Kamei et al., “A large-scale empirical study of just-in-time quality assurance,” IEEE Trans. Softw. Eng., vol. 39, no. 6, pp. 757–773, 2013, doi: 10.1109/TSE.2012.70.

[6] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, “Continual lifelong learning with neural

networks: A review," Neural Netw., vol. 113, pp. 54–71, 2019, doi: 10.1016/j.neunet.2019.01.012.

[7] S. R. Gudi, "Enhancing reliability in Java enterprise systems through comparative analysis of automated testing frameworks," Int. J. Emerging Trends Comput. Sci. Inf. Technol., vol. 4, no. 2, pp. 151–160, 2023, doi: 10.63282/3050-9246.IJETCSIT-V4I2P115.

[8] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in Proc. Int. Conf. Softw. Eng. (ICSE), 2013, pp. 382–391.

[9] S. K. Gunda, S. Yalamati, S. R. Gudi, I. Manga, and A. K. Aleti, "Scalable and adaptive machine learning models for early software fault prediction in agile development: Enhancing software reliability and sprint planning efficiency," Int. J. Appl. Math., vol. 38, no. 2s, 2025, doi: 10.12732/ijam.v38i2s.74.

[10] I. Manga, "Edge software engineering for lightweight AI: Real-time environmental data processing with embedded systems," J. Comput. Anal. Appl., vol. 34, no. 6, pp. 88–104, Jun. 2025.

[11] S. R. Gudi, "Design and evaluation of secure microservices architecture for HIPAA-compliant prescription processing on AWS and OpenShift," Int. J. Artif. Intell., Data Sci., Mach. Learn., vol. 5, no. 2, pp. 144–149, 2024, doi: 10.63282/3050-9262.IJAIDSML-V5I2P116.

[12] T. Mende and R. Koschke, "Effort-aware defect prediction models," in Proc. Euromicro Conf. Softw. Maintenance and Reengineering (CSMR), 2010, doi: 10.1109/CSMR.2010.18.

[13] S. K. Gunda, "A deep dive into software fault prediction: Evaluating CNN and RNN models," in Proc. Int. Conf. Electronic Systems and Intelligent Computing (ICESIC), Chennai, India, 2024, pp. 224–228, doi: 10.1109/ICESIC61777.2024.10846549.

[14] S. S. G. Bandari, S. D. Sivva, and R. R. Thalakanti, "Regulatory grade fraud detection using explainable artificial intelligence with auditable decision pathways and empirical validation on banking data," Int. J. Artif. Intell., Data Sci., Mach. Learn., vol. 5, no. 3, pp. 139–147, 2024, doi: 10.63282/3050-9262.IJAIDSML-V5I3P115.

[15] S. K. Gunda, "Enhancing Software Fault Prediction with Machine Learning: A Comparative Study on the PC1 Dataset," 2024 Global Conference on Communications and Information Technologies (GCCIT), BANGALORE, India, 2024, pp. 1-4, https://doi.org/10.1109/GCCIT63234.2024.10862351.

[16] S. R. Gudi, "AI-driven fax-to-digital prescription automation: A cloud-native framework using OCR, machine learning, and microservices for pharmacy operations," Int. J. Emerging Res. Eng. Technol., vol. 5, no. 1, pp. 111–116, 2024, doi: 10.63282/3050-922X.IJERET-V5I1P113.

[17] I. Manga, "Towards explainable AI: A framework for interpretable deep learning in high-stakes domains," in Proc. Int. Conf. Soft Computing for Security Applications (ICSCSA), Salem, India, 2025, pp. 1354–1360, doi: 10.1109/ICSCSA66339.2025.11170778.

[18] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 4768–4777.

[19] S. K. Gunda, "Analyzing machine learning techniques for software defect prediction: A comprehensive performance comparison," in Proc. Asian Conf. on Intelligent Technologies (ACOIT), Kolar, India, 2024, pp. 1–5, doi: 10.1109/ACOIT62457.2024.10939610.

[20] R. R. Thalakanti, S. S. Goud Bandari, and S. D. Sivva, "Federated learning for privacy preserving fraud detection across financial institutions: Architecture protocols and operational governance," Int. J. Emerging Res. Eng. Technol., vol. 5, no. 2, pp. 108–114, 2024, doi: 10.63282/3050-922X.IJERET-V5I2P111.

[21] S. R. Gudi, "Leveraging predictive analytics and Redis-backed caching to optimize specialty medication fulfillment and pharmacy inventory management," Int. J. AI, BigData, Computational and Management Studies, vol. 5, no. 3, pp. 155–160, 2024, doi: 10.63282/3050-9416.IJAIBDCMS-V5I3P116.

[22] Reddy Mittamidi VK., "AI/ML powered intelligent root cause analysis and automated remediation for multi system data integrity issues," IJAIBDCMS, vol. 6, no. 4, pp. 133–141, Nov. 2025. Available: https://ijaibdcms.org/index.php/ijaibdcms/article/view/338

[23] S. K. Gunda, "Machine learning approaches for software fault diagnosis: Evaluating decision tree and KNN models," in Proc. Global Conf. on Communications and Information Technologies (GCCIT), Bangalore, India, 2024, pp. 1–5, doi: 10.1109/GCCIT63234.2024.10861953.

[24] I. Manga, "Federated learning at scale: A privacy-preserving framework for decentralized AI training," in Proc. Int. Conf. Soft Computing for Security Applications (ICSCSA), Salem, India, 2025, pp. 110–115, doi: 10.1109/ICSCSA66339.2025.11170780.

[25] S. D. Sivva, R. R. Thalakanti, S. S. G. Bandari, and S. D. R. Yettapu, "AI-driven decision intelligence for agile software lifecycle governance: An architecture-centered framework integrating machine learning defect prediction and automated testing," IJETCSIT, vol. 4, no. 4, pp. 167–172, Dec. 2023. Available: https://ijetcsit.org/index.php/ijetcsit/article/view/554

[26] S. K. Gunda, "Accelerating scientific discovery with machine learning and HPC-based simulations," in Integrating Machine Learning Into HPC-Based Simulations and Analytics, B. Ben Youssef and M. Ben Ismail, Eds. IGI Global Scientific Publishing, 2025, pp. 229–252, doi: 10.4018/978-1-6684-3795-7.ch009.

[27] T. Raikar, "High-performance in-memory computing: A research study on SAP S/4 HANA database layer," American Journal of Technology, vol. 4, no. 2, pp. 93–113, 2025, doi: 10.58425/ajt.v4i2.449.

[28] S. R. Gudi, "Deconstructing monoliths: A fault-aware transition to microservices with gateway optimization using Spring Cloud," in Proc. Int. Conf. on Electronics and Sustainable Communication Systems (ICESC),

Coimbatore, India, 2025, pp. 815–820, doi: 10.1109/ICESC65114.2025.11212326.

[29] G. V. Krishna, B. D. Reddy, and T. Vrindaa, "EmoVision: An intelligent deep learning framework for emotion understanding and mental wellness assistance in human computer interaction," IJAIDSML, vol. 6, no. 4, pp. 14–20, Oct. 2025. Available: https://ijaidsml.org/index.php/ijaidsml/article/view/295

[30] I. Manga, "AutoML for all: Democratizing machine learning model building with minimal code interfaces," in Proc. Int. Conf. on Sustainable Computing and Data Communication Systems (ICSCDS), Erode, India, 2025, pp. 347–352, doi: 10.1109/ICSCDS65426.2025.11167529.

[31] S. R. Gudi, "Monitoring and deployment optimization in cloud-native systems: A comparative study using OpenShift and Helm," in Proc. 4th Int. Conf. on Innovative Mechanisms for Industry Applications (ICIMIA), Tirupur, India, 2025, pp. 792–797, doi: 10.1109/ICIMIA67127.2025.11200594.

[32] S. K. R. Vanama, "AI report - Federated AIOps for multi-cluster OpenShift," IJAIBDCMS, vol. 6, no. 2, pp. 96–108, May 2025. Available: https://ijaibdcms.org/index.php/ijaibdcms/article/view/336

[33] R. R. Thalakanti and S. S. Goud Bandari, "Intelligent continuous integration and delivery for banking systems using machine learning driven risk detection with real world deployment evaluation," IJAIBDCMS, vol. 5, no. 4, pp. 168–175, 2024, doi: 10.63282/3050-9416.IJAIBDCMS-V5I4P118.

[34] S. R. Gudi, "Ensuring secure and compliant fax communication: Anomaly detection and encryption strategies for data in transit," in Proc. 4th Int. Conf. on Innovative Mechanisms for Industry Applications (ICIMIA), Tirupur, India, 2025, pp. 786–791, doi: 10.1109/ICIMIA67127.2025.11200537.

[35] V. K. Reddy Mittamidi, "Leveraging AI and ML for predictive monitoring and error mitigation in change data capture pipelines," IJETCSIT, vol. 6, no. 3, pp. 104–111, Aug. 2025. Available: https://ijetcsit.org/index.php/ijetcsit/article/view/515

[36] T. Raikar and V. Apelagunta, "Implementing SAP Fiori in S/4HANA transitions: Key guidelines, challenges, strategic implications, AI integration recommendations," Journal of Engineering Research and Sciences, vol. 4, no. 11, pp. 1–9, 2025, doi: 10.55708/JS0411001.

[37] S. R. Gudi, "Enhancing optical character recognition (OCR) accuracy in healthcare prescription processing using artificial neural networks," European Journal of Artificial Intelligence and Machine Learning, vol. 4, no. 6, 2025, doi: 10.24018/ejai.2025.4.6.79.

[38] A. K. Kishore Varma Alluri, "Using Salesforce CRM and deep learning (CNN) techniques to improve patient journey mapping and engagement in small and medium healthcare organizations," IJAIDSML, vol. 6, no. 4, pp. 101–109, Nov. 2025. Available: https://ijaidsml.org/index.php/ijaidsml/article/view/330

[39] R. R. Thalakanti, "Enhancing convergence in fully connected neural networks via optimized backpropagation," in Proc. 2nd Int. Conf. on Computing and Data Science (ICCDS), Chennai, India, 2025, pp. 1–6, doi: 10.1109/ICCDS64403.2025.11209625.

[40] S. K. Gunda, "Comparative analysis of machine learning models for software defect prediction," in Proc. Int. Conf. on Power, Energy, Control and Transmission Systems (ICPECTS), Chennai, India, 2024, pp. 1–6, doi: 10.1109/ICPECTS62210.2024.10780167.

[41] I. Manga, "Unified data engineering for smart mobility: Real-time integration of traffic, public transport, and environmental data," in Proc. Int. Conf. Soft Computing for Security Applications (ICSCSA), Salem, India, 2025, pp. 1348–1353, doi: 10.1109/ICSCSA66339.2025.11170800.

[42] A. K. Kishore Varma Alluri, "Salesforce CRM framework for real time DeFi portfolio intelligence and customer engagement forecasting in Web3 based decentralized finance ecosystems using ML techniques," IJAIBDCMS, vol. 6, no. 4, pp. 99–107, Nov. 2025. Available: https://ijaibdcms.org/index.php/ijaibdcms/article/view/319

[43] S. R. Gudi, "A comparative analysis of Pivotal Cloud Foundry and OpenShift cloud platforms," The American Journal of Applied Sciences, vol. 7, no. 07, pp. 20–29, 2025, doi: 10.37547/tajas/Volume07Issue07-03.

[44] I. Manga, "Scalable graph neural networks for global knowledge representation and reasoning," in Proc. Int. Conf. on Inventive Systems and Control (ICISC), Coimbatore, India, 2025, pp. 1399–1404, doi: 10.1109/ICISC65841.2025.11188341.