Available online at https://sextante-ensb.com

Revista Sextante

ISSN : 2665-3923

Volume: 34 Issue: 01 (2026)

AI-Driven Predictive Maintenance and Quality Assurance of Enterprise Banking Software Systems

Muddassar Sattar Alvi*[a]

a. Corresponding Author: U.E.T Lahore (mudasar06@hotmail.com)

Article History
Received September 10th, 2026
Accepted November 05th, 2026
Published November 18th, 2026

© 2026 The Authors. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution(CC-BY)license https://creativecommons.org/licenses/by/4.0

Abstract
Enterprise banking platforms (core banking, payments, and channel systems) must sustain very high availability while shipping frequent changes under strict supervisory expectations for operational resilience. Reactive operations and manual quality assurance (QA) struggle to meet both demands. This paper proposes a four-layer, governance-aware framework that couples AI-driven predictive maintenance (PdM)—anomaly detection and failure-horizon prediction on telemetry—with AI-driven QA—change-risk prediction and risk-based test allocation. We synthesise the 2020–2026 literature on industrial PdM, AIOps, log analytics, software defect prediction, LLM-assisted testing, and ML operations, and identify gaps specific to regulated banking environments. The framework is then evaluated through a controlled, fully synthetic simulation study: 40 service instances over 100 days of 30-minute telemetry containing 612 injected incidents, and 12,000 time-ordered code changes. On the held-out period, a random forest raised F1 from 0.562 (static threshold rules) to 0.760, extended mean warning lead time from 67 to 103 minutes, and reduced false-alert episodes by 38%; an unsupervised Isolation Forest detected incidents but generated 2.26 false-alert episodes per instance-week. Models caught every gradual-degradation incident but almost no sudden failures (3%), bounding achievable event recall. Rolling retraining offered only modest benefit after a simulated platform migration. In QA, learned change-risk models found 2.3 times more defects than a size-only baseline within 20% of review effort. Because the data are synthetic, results demonstrate relative behaviour and methodological trade-offs, not production performance; we provide a validation roadmap for real bank environments.

Index Terms— predictive maintenance; software quality assurance; AIOps; anomaly detection; defect prediction; banking software; MLOps; operational resilience

1. Introduction

Banks have some of the most challenging software estates in the business. The core banking ledgers, card and real-time payment switches, fraud engines and digital channels are highly coupled and are subjected to workload peaks that are highly seasonal (payroll days, month-end, and holiday peaks) and are required to be available 24 hours a day. Power outages cost money in direct losses, lost customers, and regulatory repercussions. For instance, the European Union Digital Operational Resilience Act (DORA) sets out clear standards for ICT risk management, incident reporting and operational resilience testing for financial organisations [1]. There are similar expectations in other jurisdictions.

Traditional approaches for protecting such systems are reactive or threshold production monitoring and scripted regression test and manual delivery review. Both are in tension. Static thresholds result in alert noise and ignore seasonality, and exhaustive regression suites extend regression cycles. As teams gather more telemetry, logs, and

© 2026 The Authors. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution(CC–BY)license
https://creativecommons.org/licenses/by/4.0

change information, the amount of data they have to review becomes overwhelming, driving the need for AI-based solutions.

In manufacturing and industrial applications, predictive maintenance (PdM) has developed to a mature state by forecasting the condition of an asset and taking action before it fails [2, 3]. The equivalent area of IT operations is AIOps, using machine learning for failure management [4] and root cause analysis in service-based systems [5]. Concurrently, the field of AI in software quality has advanced from defect prediction [6] to software testing using and with learned models [7, 8] and most recently large language models for code and test generation [9, 10]. Rarely, however, these strands are examined together, and often without considering the obligations of explainability, auditability, and resilience that arise when working with regulated banking platforms, where only certain options are able to be deployed.

1.1 This paper is fourfold:

- An enterprise banking software reference framework that brings together telemetry-based PdM and change-based QA and installs governance controls as a first-class layer (Section 3).
- A structured synthesis of literature in the period 2020-2026 and gaps in literature for banking (Section 2).
- Simulated (fully synthetic) study for failure-horizon prediction and change-risk prediction, both using static rules, unsupervised, linear, bagged-tree and boosted-tree models, with ablation, concept-drift and alert-burden analyses (Sections 4–5), which is reproducible.
- Discussion of validity threats as well as a roadmap to validation from simulation to production evidence (Sections 6-7).

The rest of the paper is organised as follows. In Section 2, related work is reviewed. The framework is provided in Section 3. Details of the experimental design are described in Section 4; results are reported in Section 5; implications are discussed in Section 6; limitations are discussed in Section 7; and conclusions are presented in Section 8.

2. Related Work

2.2 The future of predictive maintenance is here: from industrial assets to software services

Related systematic reviews in the context of Industry 4.0 indicates that the focus in PdM has been transitioning from the rule based and statistical methods to machine learning pipelines based on sensor streams, focusing on common challenges: lack of failure labels, variability in data and operationalisation of models [2, 3]. The barriers are aligned with enterprise software in that incidents are relatively rare events, there is a lot of noise in the labelling (post mortems are written after an incident), and each service has its own operating profile. Deep-learning anomaly detection provides a tool for unlabeled data [11], and the problem of unlabeled data detection versus supervised failure prediction is dependent on the availability of labels, and the tolerance for alert noise by the operator.

2.2 AIOps, log analytics, and incident management

Surveys of AIOps have been developed that list approaches for anomaly detection, failure prediction and root cause analysis on logs, metrics and traces [4] including approaches for microservices architectures [5]. In the case of logs, there have been strides from sequence to transformer-based detectors like LogBERT [12] and the recent research paper considers whether parsing logs is even needed [13] (a recent survey summarizes the area [14]). Most recently, LLMs have been used to suggest root causes and mitigation measures for cloud incidents [15]. Fraud is a well-studied area in finance [16] but the focus there is on fraudulent use of transactions, not the health of the software infrastructure that processes the transactions (as is the focus here).

2.3 AI for software quality assurance

A systematic review of software defect prediction has been conducted in the context of data sets, validation practices and learning methods [6]. Evaluation realism is a reoccurring methodological issue: Cross validation with

© 2026 The Authors. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution(CC–BY)license
https://creativecommons.org/licenses/by/4.0

random splits over-estimates performance compared to chronological splits, and effort awareness (defects per unit of effort) is more relevant to decision than accuracy. In isolation, the research on ML-based systems has yielded a body of work [7, 8]. Automated Test Generation trained using code [17] has enabled new possibilities, ranging from empirical results showing significant variance in coverage and correctness across different settings [18], to general overviews of the field [9, 10]. The key point for banking is not if generation can be made at all but how generated tests can be validated and regulated.

2.4 Engineering and governance of ML systems

In industrial practice various studies on ML have consistently pointed out that model fitting is not the major focus of the effort but rather deployment, monitoring and maintenance [19, 20, 21]. Operational scaffolding for software engineering challenges of ML systems is given by a number of reviews [22] and architectural treatments of MLOps [23] while work on performance-aware drift detection [24] addresses the problem that models evolve and change as systems evolve. Explainability [25] and fairness [26] impose additional requirements on top of the predictiveness and the EU AI Act introduces a risk-based regulatory framework for AI systems [27].

2.5 Synthesis and research gaps

Table 1 summarises the strands reviewed and the gaps they leave for banking software.

Table 1. Synthesis of related research strands and gaps for enterprise banking software

Strand	Representative studies	Typical contribution	Gap for banking platforms
Industrial PdM	[2, 3]	Pipelines and challenges for sensor-based failure prediction	Little translation to software services with seasonal, transactional workloads
AIOps / failure management	[4, 5]	Anomaly detection, failure prediction, RCA for cloud services	Rarely evaluates alert burden or regulatory auditability
Log analytics	[12, 13, 14]	Deep log anomaly detection, parser-free approaches	Benchmarks rarely reflect change events, drift, or seasonality
LLMs for incidents	[15]	Root-cause and mitigation recommendation	Validation, hallucination control, and human oversight in critical systems
Defect prediction	[6]	Models, datasets, validation practices	Linking predictions to release gating and test allocation
ML testing / LLM testing	[7, 8, 9, 10, 18]	Testing of ML systems; LLM test generation	Governance of generated tests in regulated delivery
MLOps and drift	[19, 20, 21, 23, 24]	Deployment challenges, architectures, drift detection	Retraining policy under change freezes and model-risk review
Responsible AI / regulation	[1, 25, 26, 27]	Explainability, fairness, legal obligations	Concrete control mapping for internal operations tooling

Three gaps motivate this study: (i) PdM and QA are rarely combined although change events are a leading cause of incidents; (ii) evaluation seldom reports operator-facing metrics such as false-alert episodes per week and warning lead time; and (iii) governance is typically treated as an afterthought, not as part of the architecture.

3. Proposed Framework

3.1 Architecture

The framework (Fig. 1) has four functional layers and a cross-cutting governance layer. Layer L1 collects APM and infrastructure telemetry, application and audit logs, CI/CD and version-control records, ITSM incident histories, and test results. Layer L2 provides streaming ingestion, log parsing, and a feature store holding rolling, seasonal, and contextual features so that training and inference use identical definitions (a common MLOps practice [23]). Layer L3 hosts two AI engines: the PdM engine (anomaly detection, failure-horizon prediction, root-cause ranking) and the QA engine (change-risk prediction, risk-based test prioritisation, and LLM-assisted test generation). Layer L4 converts scores into decisions through alert policy, change gates, and runbook automation, with humans reviewing consequential

© 2026 The Authors. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution(CC–BY)license
https://creativecommons.org/licenses/by/4.0

actions. Confirmed outcomes flow back as labels for retraining.

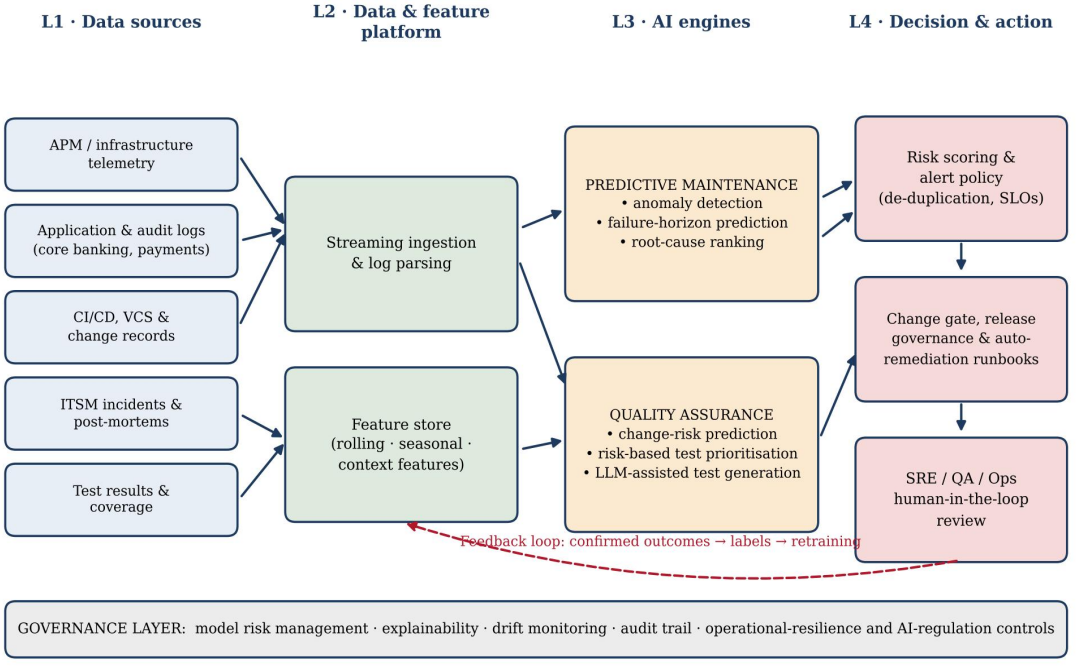

Fig. 1. Proposed four-layer framework for AI-driven predictive maintenance and quality assurance of enterprise banking software, with a cross-cutting governance layer and a feedback loop.

3.2 Problem formulation

Failure-horizon prediction. For service instance i at time step t (30 minutes in our experiments), let x(i,t) be the feature vector. Given a horizon H, the label is

$$y(i,t) = 1 \text{ if an incident onset occurs in } (t, t+H], \text{ otherwise } 0 \qquad (1)$$

and a model f outputs a risk score $s(i,t) = f(x(i,t))$ in [0,1]. An alarm is raised when $s(i,t) \geq \theta$, where θ is chosen on a validation period to maximise F1. We use H = 4 steps (two hours). Consecutive alarms separated by fewer than four steps are merged into one alert episode, which is the unit an on-call engineer actually experiences.

Change-risk prediction. The model predicts p(c) = P(defect escapes | z(c)) where z(c) is the feature vector for code change c. Reviewers check changes in the order of the decreasing p(c) / change size ratio, which is the percentage of defective changes in the first 20% of the cumulative lines changed, Recall@20%E.

3.3 Operator-facing evaluation metrics

In addition to sample level precision, recall and F1 we report: (a) event recall—the percentage of incidents that had at least one alarm in the H steps before onset; (b) mean lead time—the interval from the first such alarm to onset; and (c) false-alert episodes per instance-week—the number of episodes of false alarms (no true positive-step) per instance-week, normalised. These measures are a balance of early warning and alert fatigue, and tend to be overlooked when evaluating based on a model.

© 2026 The Authors. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution(CC–BY)license
https://creativecommons.org/licenses/by/4.0

3.4 Governance and regulatory mapping

Table 2 (at the end of the document) shows the mapping of the governance layer to architectural controls. The mapping is not meant to be advice on legal classification of a given internal operations model (e.g. high risk system under the EU AI Act [27]). The mapping should hence be interpreted as a design guideline rather than as an interpretation of the legislation.

Table 2. Governance controls and their role in the framework

Control	Implementation in the framework	Regulatory / literature anchor
Operational resilience	Predictive alerts feed incident management; resilience testing uses QA engine scenarios	ICT risk management and testing obligations [1]
Explainability	Per-alert feature attributions and top contributing signals shown to operators	XAI taxonomy [25]
Drift monitoring	Rolling performance and score-distribution monitors trigger recalibration review	[20, 24]
Model risk management	Independent validation, versioning, challenger models, documented thresholds	Supervisory model-risk expectations; MLOps lineage [23]
Fairness and bias review	Applies where models affect customers or staff decisions; checks for subgroup disparities	[26]
Human oversight	Auto-remediation limited to pre-approved runbooks; changes to critical ledgers need human sign-off	AI regulation oversight principles [27]

4. Experimental Design

Confidentiality of real production telemetry and defect data from banks does not allow for sharing. We thus performed a controlled simulation study where we aim to compare modelling strategies in the case of known ground truth, with realistic structure (seasonality, benign anomalies, near-miss events, drift, and imbalanced labels). No real bank data used, all data is synthetic. As a consequence, results need to be interpreted accordingly (Section 7).

4.1 Predictive-maintenance data generator

We ran 40 service instances (such as core-ledger, payment-gateway, API-channel nodes) over a period of 100 days with a time resolution of 30 minutes. Eight signals were created: CPU utilisation, memory utilisation, p95 latency, error rate, database connection-pool utilisation, queue depth, garbage-collection pause, and a log-anomaly score. The business day load profile is low at weekends and high during lunch and the afternoon secondary peak, with an overall 30% ramp up around the month end. Non-failure anomalies occur due to two benign factors: a nightly batch window, and transient post-deployment effects. Three hundred and one hundred and twelve incidents were injected (Table 3) with one of four types. The gradual type has a 2.5–7 hour precursor ramps and the “sudden” type has a weak log-score precursor. Further, “near-miss” ramps with 25-55% of the amplitude of the full amplitude (that recovers but do not encounter problems) were included at 1.2 times the number of the incident, to penalise models who respond to any drift in their signals. A simulated platform change begins on day 85, resulting in drift of baselines (latency +15%, memory +5 points, error rate +0.05, GC pause +4ms) in the test period. An example of simulated telemetry in the time leading up to a memory-leak event is depicted in Fig. 2.

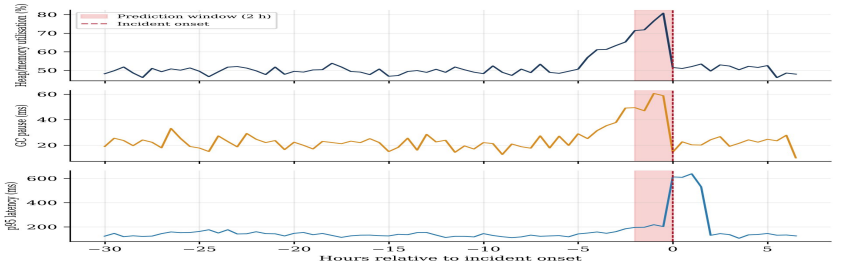

Fig. 2. Example simulated telemetry before a memory-leak incident: memory utilisation, GC pause, and p95 latency. The shaded band is the two-hour prediction window.

© 2026 The Authors. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution(CC–BY)license
https://creativecommons.org/licenses/by/4.0

Table 3. Predictive-maintenance dataset summary.

Attribute	Value
Instances × duration	40 instances × 100 days (30-min steps)
Labelled windows (after removing warm-up and post-incident rows)	183,894
Features	45 (raw 8; temporal 24; seasonal deviation 8; context 5)
Chronological split	Train days 0–59 (108,885 rows); validation days 60–74 (28,174); test days 75–99 (46,835)
Positive-class rate	1.32% train; 1.41% test
Incidents by type	Memory leak 204; DB pool exhaustion 170; latency creep 129; sudden 109
Incidents in test period	166
Prediction horizon H	4 steps (2 hours)

4.2 Features and models

For each signal we computed the current value, a six-step rolling mean, six-step change, six-step rolling standard deviation, and deviation from the value 24 hours earlier (seasonal deviation); five context features (hour-of-day sine/cosine, day-of-week, month-end flag, hours since last deployment) were added. Table 4 lists the compared methods. Hyperparameters were set once from defaults and light manual tuning and were not tuned on the test period; decision thresholds were selected on the validation period.

Table 4. Compared methods and configuration

Method	Configuration
Static threshold rules	Alarm if p95 latency > 260 ms, or error rate > 1.2%, or memory > 78%, or pool utilisation > 88% (no learning)
Isolation Forest	200 trees, 4,096 samples; trained on 40,000 non-pre-failure windows from the training period; non-context features; score thresholded on validation
Logistic Regression	Standardised inputs, L2 (C = 0.5), balanced class weights
Random Forest	150 trees, max depth 12, min leaf 5, balanced-subsample class weights
Gradient Boosting (HGB)	Histogram-based gradient boosting; 250 iterations, learning rate 0.06, 31 leaves, L2 = 1.0

4.3 Quality-assurance data generator

We created 12,000 code changes on the same repository that occurred in time order and had 12 commonly occurring attributes of just-in-time defect prediction: lines changed, files touched, cyclomatic-complexity delta, developer experience, test-coverage delta, prior defects in the module, review minutes, static-analysis warnings, dependency change, hotfix flag, core-ledger flag, and 30-day churn. Defect escape was taken from a latent risk score with nonlinear interactions (e.g., complexity increase with coverage reduction, core-ledger changes of large size) and noise. The prevalence of this (26.2%) is higher than generally expected in practice and was selected to provide sufficient statistical power. The first 70% (8,400) of the changes were used for training and the remaining 30% (3,600) for testing. The generator's structure is very close to logistic so one should not generalise any conclusions made based on the superiority of a model family.In this experiment, logistic regression should prove competitive, so conclusions regarding model family superiority cannot be generalised.

To provide an example of downstream value, we conducted a risk-tiered testing policy simulation. The top ranked change is subjected to full regression testing (at 100 units) and the other changes are subjected to lighter testing (at 30 units, 60% chance of detection). These catch rates and costs are simply assumptions for purposes of illustration.

5. Results

5.1 Failure-horizon prediction

Test-period results (days 75-99) are reported in Table 5. On the balanced measures, learned supervised models were clearly superior. The random forest had the best performance in terms of F1 (0.760), PR-AUC (0.774), precision (0.845) and the fewest false-alert episodes (0.140 per instance-week), which is 38% fewer than the static rules (0.224)

© 2026 The Authors. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution(CC–BY)license
https://creativecommons.org/licenses/by/4.0

while yielding a higher event recall (81.3%) and mean lead time (103 minutes, +54%). An improved model achieved the best sample-level recall (0.734) and the longest lead time (107 min) at a reduced precision. The event recall of logistic regression (84.9%) was highest, but it generated about five times the number of false-alert episodes compared to random forest. Although the high ROC-AUC (0.922) suggests an excellent classification performance, the low precision-recall performance (PR-AUC 0.383) and the production of 2.26 false-alert episodes per instance-week is not likely to be acceptable for an on-call team. The precision–recall curves are presented in Fig. 3 and a warning lead time distribution for the boosted model in Fig. 4.

Table 5. Predictive-maintenance results on the held-out test period (days 75–99). Lead time is capped at 120 min by the prediction horizon.

Method	Prec.	Rec.	F1	PR-AUC	ROC-AUC	Event recall (%)	Lead (min)	False alerts /inst-wk
Static threshold rules	0.818	0.428	0.562	–	–	76.5	67	0.224
Isolation Forest	0.388	0.687	0.496	0.383	0.922	84.3	98	2.261
Logistic Regression	0.676	0.722	0.698	0.738	0.948	84.9	102	0.679
Random Forest	**0.845**	**0.691**	**0.760**	**0.774**	**0.942**	**81.3**	**103**	**0.140**
Gradient Boosting (HGB)	0.759	0.734	0.746	0.764	0.944	82.5	107	0.350

Prec. = precision; Rec. = recall (sample level). Event recall = share of incidents with ≥1 alarm in the 2 h before onset. Shaded row marks the best F1. Single run, seed 2026; no confidence intervals (see Section 7).

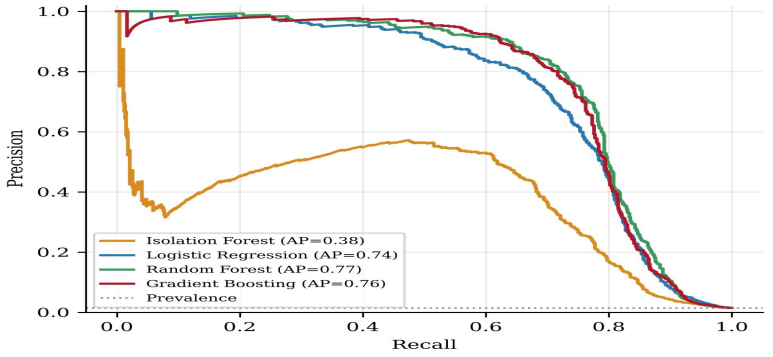

Fig. 3. Precision–recall curves on the test period. The dotted line marks class prevalence (1.4%).

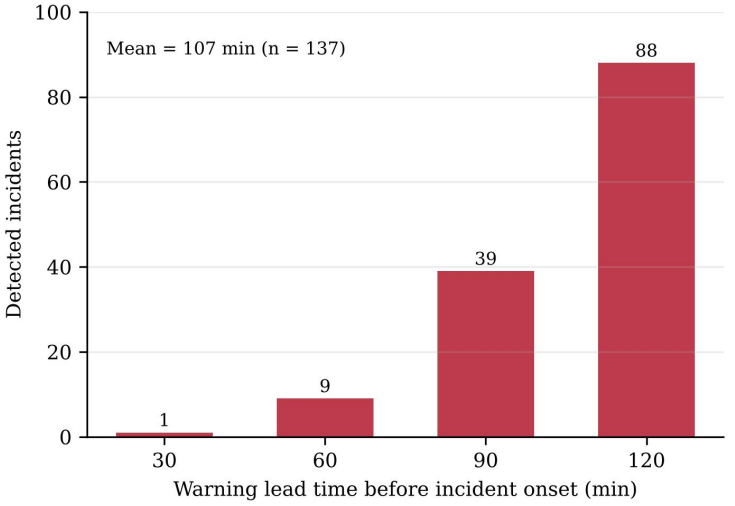

Fig. 4. Warning lead times for incidents detected by the boosted model (n = 137). Lead time is capped at 120 min by

© 2026 The Authors. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution(CC–BY)license
https://creativecommons.org/licenses/by/4.0

the prediction horizon.

Fig. 4 shows that most detected incidents were flagged at the earliest step of the window, so the mean (107 min) is partly a consequence of the 2-hour horizon; longer horizons would trade precision for earlier warning and should be explored on real data.

5.2 Feature-group ablation and importance

Table 6 retrains the boosted model without each feature group, and Fig. 5 shows permutation importance by group. Temporal features (rolling mean, change, variability) were the most important group (PR-AUC drop of 0.211 when permuted) followed by seasonal-deviation features (0.116) and raw signals (0.106). Context features contributed almost nothing (0.005). Removing any single group changed PR-AUC by at most about 2.2 points, indicating substantial redundancy across groups; removing temporal features produced the largest degradation (PR-AUC 0.764 to 0.743; F1 0.746 to 0.729). Removing raw signals or context features did not hurt and in some metrics slightly improved results, consistent with the information being captured by the remaining groups. Because permutation importance is distorted by correlated features, these rankings are indicative only.

Table 6. Ablation of feature groups for the boosted model (test period).

Configuration	PR-AUC	F1	Event recall (%)	Lead (min)	False alerts /inst-wk
All features (45)	**0.764**	**0.746**	**82.5**	**107**	**0.350**
Without: Raw signals	0.767	0.756	81.9	104	0.196
Without: Temporal (rolling mean/slope/std)	0.743	0.729	82.5	103	0.357
Without: Seasonal deviation (vs. same time yesterday)	0.762	0.743	82.5	107	0.336
Without: Context (calendar, deployment age)	0.771	0.744	82.5	107	0.350

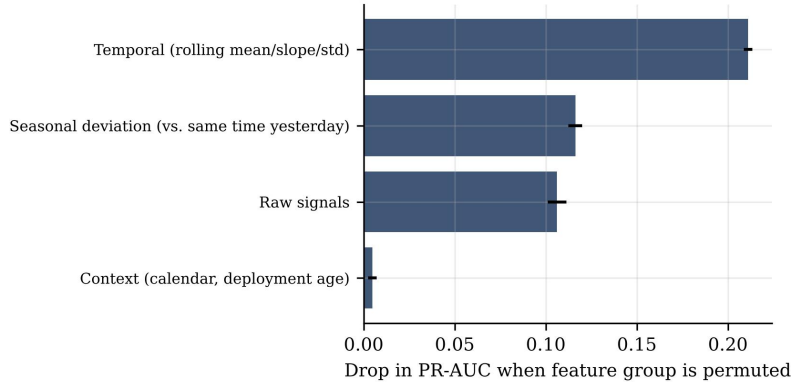

Fig. 5. Permutation importance by feature group for the boosted model (mean ± SD over 3 repeats).

5.3 Performance by incident type

Table 7 decomposes event-level results for the boosted model. All gradual-degradation incidents (memory leak, pool exhaustion, latency creep) were detected in advance (100%), with mean lead times of 107–109 minutes, whereas only 3% of sudden incidents (1 of 30) were caught. The overall event recall of 82.5% is therefore essentially bounded by the share of precursor-free failures (about 18% of incidents), not by model choice. This is practically important: PdM cannot substitute for resilience engineering, redundancy, and rapid detection and recovery for sudden faults. Event-level recall is much higher than sample-level recall (0.734) because an incident counts as detected if any step in its window triggers an alarm.

© 2026 The Authors. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution(CC–BY)license
https://creativecommons.org/licenses/by/4.0

Table 7. Event-level detection by incident type (boosted model, test period).

Incident type	Incidents (n)	Event recall (%)	Mean lead time (min)
Memory leak	66	100.0	107
Database pool exhaustion	38	100.0	107
Latency creep	32	100.0	109
Sudden (weak precursor)	30	3.3	30

5.4 Robustness to concept drift

The simulated platform migration (day 85) degraded the static boosted model: F1 fell from 0.810 (days 80–84) to 0.686 (days 85–89), a 15% relative drop, while false-alert episodes rose from 0.14 to 0.42 per instance-week (Table 8, Fig. 6). Rolling retraining (refit on all data up to each block, with the threshold re-selected on the preceding ten days) produced mixed results: it was slightly worse in the first three blocks (F1 −1.0 to −1.5 points) and slightly better in the last two (+1.0 and +1.2 points; PR-AUC +1.8 and +3.7 points). Averaged over the five blocks, retraining left F1 unchanged (0.744 static vs. 0.741 retrained), raised PR-AUC by about one point (0.772 vs. 0.783), and reduced false-alert episodes by about 17% (0.336 vs. 0.280). The benefit is modest here, likely because seasonal-deviation features are partly scale-robust and the shift is a single step change. We therefore recommend drift monitoring with threshold recalibration review, as discussed for performance-aware drift detection [24], in place of unconditional retraining.

Table 8. Static versus rolling-retrained boosted model across five-day test blocks (migration begins on day 85).

Block (days)	F1 static	F1 retrained	PR-AUC static	PR-AUC retrained	False alerts static	False alerts retrained
75–79	0.779	0.768	0.798	0.786	0.21	0.24
80–84	0.810	0.795	0.826	0.829	0.14	0.03
85–89	0.686	0.676	0.694	0.701	0.42	0.53
90–94	0.717	0.727	0.774	0.792	0.53	0.45
95–99	0.729	0.741	0.767	0.805	0.38	0.14

False alerts = alert episodes per instance-week.

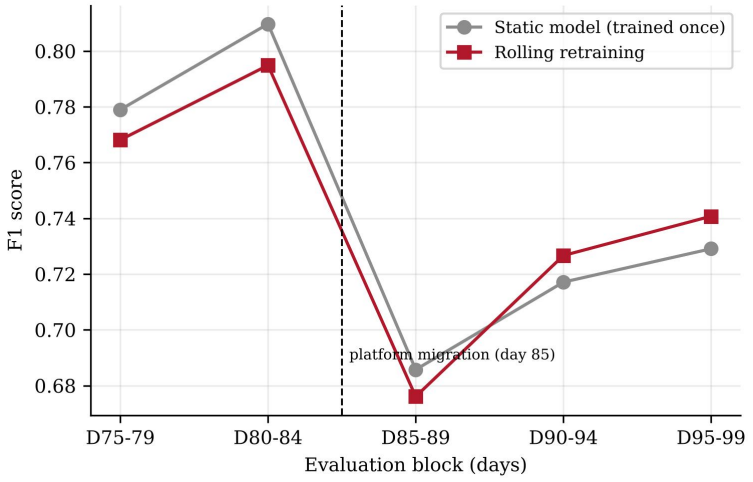

Fig. 6. F1 of the static versus rolling-retrained boosted model across test blocks. The dashed line marks the simulated platform migration.

5.5 Change-risk prediction for QA

Table 9 and Fig. 7 present change-risk results on the chronologically held-out changes. The three learned models reached AUC of 0.78–0.79, versus 0.693 for a size-only baseline, and found between 47% and 49% of defective

© 2026 The Authors. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution(CC–BY)license
https://creativecommons.org/licenses/by/4.0

changes within 20% of review effort, versus 21% for the baseline—about 2.3 times as many (logistic regression, 48.7% vs. 21.2%). Differences among the three learned models are small (AUC 0.780–0.787), which is expected given the near-logistic generator; no model-family ranking should be inferred. The size-only baseline barely exceeded random inspection under the effort-aware metric, illustrating why effort-aware evaluation matters.

Table 9. Change-risk prediction on held-out changes (3,600 changes; prevalence 26.2%).

Model	AUC	PR-AUC	Recall @ 20% effort	Recall @ top-20% changes	Brier score
LOC-only baseline	0.693	0.472	0.212	0.389	–
Logistic Regression	0.787	0.601	0.487	0.468	0.148
Random Forest	0.780	0.582	0.471	0.462	0.152
Gradient Boosting (HGB)	0.782	0.587	0.476	0.462	0.150

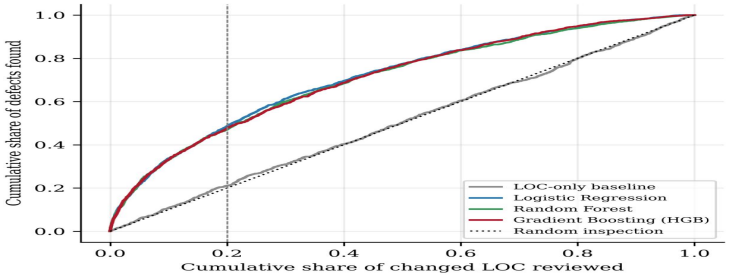

Fig. 7. Cumulative defect-detection gains versus share of changed lines reviewed. The dashed line marks 20% review effort.

5.6 Illustrative risk-tiered testing policy

Table 10 and Fig. 8 apply the assumed catch rates to the boosted model's change scores. At the same testing cost (44% of full regression for all), tiering by predicted risk let 26.1% of defects escape, versus 33.7% for random tiering and 28.3% for size-based tiering. Extending full regression to the top 35% by predicted risk reduced escapes to 20.4%—about half the light-testing-only level (40.0%)—at 54.5% of the cost of full regression for all. Full regression for all remained the most protective (10.0% escape) but at the highest cost. The policy outcome is entirely conditional on the assumed catch rates and should be re-estimated from historical data before use.

Table 10. Illustrative risk-tiered testing policies (assumption-driven; test set of 3,600 changes).

Policy	Defects escaping (%)	Relative cost (%)	Escapes per 1,000 changes
Light testing for all	40.0	30.0	103
Full regression for all	10.0	100.0	26
Random 20% full	33.7	44.0	87
Largest-LOC 20% full	28.3	44.0	73
ML risk-tiered (top 20% full)	**26.1**	**44.0**	**67**
ML risk-tiered (top 35% full)	**20.4**	**54.5**	**52**

Assumptions: full regression detects 90% of defects (100 cost units); light testing detects 60% (30 units). Shaded rows use model-ranked changes.

© 2026 The Authors. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution(CC–BY)license
https://creativecommons.org/licenses/by/4.0

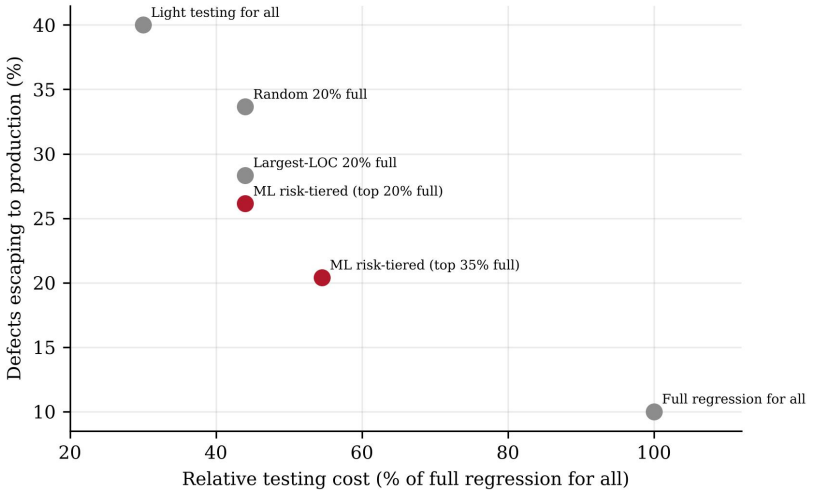

Fig. 8. Trade-off between relative testing cost and defects escaping to production under the assumed catch rates.

6. Discussion

Implications for banking PdM. Given three observations, it is likely that three of these will transfer beyond the simulation.If there are three observations, then three of them should transfer beyond the simulation. Supervised models utilizing temporal and seasonal-deviation features can give early warning with significantly lower false alarm rate than static thresholds, however the benefit is dependent on the quality of the incident labels, which typically are obtained from post-mortems and ITSM tickets and are less clean than the labels used here [21]. Second, unsupervised detection is appealing when labels are limited, but as seen in our results, in practice it may not be desirable because of the high alert ratio without further filtering, so it could be useful as a candidate generator and then filtered by a supervised or rule-based layer of detection [4, 11]. Third, the restriction on event recall due to sudden failures demonstrates that PdM can be used in addition, not instead, of redundancy, graceful degradation, and fast recovery.

Implications for QA. By using effort-aware, chronologically assessed change-risk models, limited review and regression capacity can be dedicated to the riskiest changes [6]. Linking scores with release gates (such as required extra testing, senior reviewing changes over a risk threshold, etc.) puts prediction into action and also gives the model a quantifiable value. The generated test cases based on LLM [9, 10, 18] fit well into the high risk tier, as they add the most value when there are more tests to be added, provided that the generated tests are reviewed, deduplicated and regulated artefacts.

Operationalisation. Disciplined MLOps – a shared feature store to avoid training–serving skew, versioning models and thresholds, and monitoring drift [19, 23, 24] is a critical part of the feedback loop. The drift experiment indicates that in a change-controlled banking context, where retraining might need to be approved, a monitored recalibration procedure might be more feasible than frequent automated retraining. Last but not least, explanation and human oversight should be built in from the beginning of alerts so that operators know the reason for the alert, and so that governance reviews have an audit trail [25].

Recommended validation roadmap. (1) Offline back-testing of historical telemetry and incidents, including chronological splits and bootstrapped confidence intervals; (2) Shadow-mode deployment of alerts without taking any action; (3) Rolled-out to a subset of services with pre-registered KPIs (mean time to detect, number of incidents, number of alerts per on-call shift, defect escape rate, regression-suite duration); (4) Periodic independent model-risk validation.

© 2026 The Authors. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution(CC–BY)license
https://creativecommons.org/licenses/by/4.0

7. Threats to Validity and Limitations

Synthetic data (construct and external validity). All results are based on data generated that have been structured by us. The absolute numbers (F1, lead times, false alert rates) are not predictions of production performance, and real telemetry will likely be more noisy, more variable and have missing data, and be mislabelled. The simulation is intended to aid in comparing methods when there is a known ground truth, and to help visualize metrics and trade-offs, rather than to set effect sizes in the real world. The structure we built in (for example, the near-logistic QA generator favours linear models) may work more in favor of the generator written by the researchers.

Statistical conclusion validity. In each experiment, only one test with a fixed seed was conducted, without confidence intervals; small differences (for instance between random forest and boosted model, or between retrained and static models in each drift block) should not be considered significant. Hyperparameters were not finely tuned. Sample variability is not insignificant for the event-level or alert-episode measures, because they are based on 166 test incidents or 40 instances.

Internal validity. Lead times are capped at 2 hours in the prediction horizon, and operators' metrics are affected by the choice of H, the alarm merging rule, and the selection of the threshold by F1. The QA policy simulation assumes catch rates and costs, and assumes that defects are independent of each other, not considering correlations in tests, and the difference in severity for defects.

Scope. Although the framework and literature reviewed included deep sequence models, graph-based topology models, trace data, LLM-based components, they were not evaluated as part of the empirical evaluation. The literature synthesis is a "narrative review" of the selected literature between 2020 and 2026, not a PRISMA (a systematic review), and may exclude relevant literature.

8. Conclusion

We have proposed a Governance-aware framework that combines AI-based predictive maintenance and quality assurance approaches for enterprise banking software and examined the aspects of modelling in a controlled synthetic study. Static thresholds were outperformed by supervised failure-horizon models in terms of balanced performance, warning time and number of false-alert episodes, but the unsupervised detector resulted in a high number of false alarms and the benefits of retraining in the presence of drift were limited. Change-risk models focused QA effort by an average of 2.3 times more than size-based prioritisation, while risk-tiered testing provided a cost–escape advantage under certain explicit assumptions in QA.

Future work will include (i) repeating the study with production telemetry and defect histories anonymized, (ii) combining deep sequence and log-based models, with LLM based root cause and test generation, (iii) quantifying the uncertainty from repeated runs and confidence intervals, and (iv) assessing the quality of explanation with on-call engineers.

References

[1] European Parliament and Council of the EU, “Regulation (EU) 2022/2554 on digital operational resilience for the financial sector (DORA),” Official Journal of the European Union, L 333, Dec. 2022.

[2] T. Zonta, C. A. da Costa, R. da Rosa Righi, M. J. de Lima, E. S. da Trindade, and G. P. Li, “Predictive maintenance in the Industry 4.0: A systematic literature review,” Computers & Industrial Engineering, vol. 150, Art. no. 106889, 2020.

[3] P. Nunes, J. Santos, and E. Rocha, “Challenges in predictive maintenance – A review,” CIRP Journal of Manufacturing Science and Technology, vol. 40, pp. 53–67, 2023.

[4] P. Notaro, J. Cardoso, and M. Gerndt, “A survey of AIOps methods for failure management,” ACM Transactions on Intelligent Systems and Technology, vol. 12, no. 6, Art. no. 81, 2021.

[5] J. Soldani and A. Brogi, “Anomaly detection and failure root cause analysis in (micro)service-based cloud applications: A survey,” ACM Computing Surveys, vol. 55, no. 3, Art. no. 59, 2022.

[6] J. Pachouly, S. Ahirrao, K. Kotecha, G. Selvachandran, and A. Abraham, “A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools,” Engineering Applications of Artificial Intelligence, vol. 111, Art. no. 104773, 2022.

[7] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, “Machine learning testing: Survey, landscapes and horizons,” IEEE Transactions on Software Engineering, vol. 48, no. 1, pp. 1–36, 2022.

© 2026 The Authors. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution(CC–BY)license
https://creativecommons.org/licenses/by/4.0

[8] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella, "Testing machine learning based systems: A systematic mapping," Empirical Software Engineering, vol. 25, pp. 5193–5254, 2020.

[9] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," ACM Transactions on Software Engineering and Methodology, vol. 33, no. 8, Art. no. 220, 2024.

[10] 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, 2024.

[11] G. Pang, C. Shen, L. Cao, and A. van den Hengel, "Deep learning for anomaly detection: A review," ACM Computing Surveys, vol. 54, no. 2, Art. no. 38, 2021.

[12] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via BERT," in Proc. International Joint Conference on Neural Networks (IJCNN), 2021.

[13] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in Proc. 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2021, pp. 492–504.

[14] M. Landauer, S. Onder, F. Skopik, and M. Wurzenberger, "Deep learning for anomaly detection in log data: A survey," Machine Learning with Applications, vol. 12, Art. no. 100470, 2023.

[15] T. Ahmed, S. Ghosh, C. Bansal, T. Zimmermann, X. Zhang, and S. Rajmohan, "Recommending root-cause and mitigation steps for cloud incidents using large language models," in Proc. 45th International Conference on Software Engineering (ICSE), 2023, pp. 1737–1749.

[16] W. Hilal, S. A. Gadsden, and J. Yawney, "Financial fraud: A review of anomaly detection techniques and recent advances," Expert Systems with Applications, vol. 193, Art. no. 116429, 2022.

[17] M. Chen et al., "Evaluating large language models trained on code," arXiv preprint arXiv:2107.03374, 2021.

[18] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, "An empirical evaluation of using large language models for automated unit test generation," IEEE Transactions on Software Engineering, vol. 50, no. 1, pp. 85–105, 2024.

[19] L. E. Lwakatare, A. Raj, I. Crnkovic, J. Bosch, and H. H. Olsson, "Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions," Information and Software Technology, vol. 127, Art. no. 106368, 2020.

[20] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: A survey of case studies," ACM Computing Surveys, vol. 55, no. 6, Art. no. 114, 2022.

[21] S. Shankar, R. Garcia, J. M. Hellerstein, and A. G. Parameswaran, "Operationalizing machine learning: An interview study," arXiv preprint arXiv:2209.09125, 2022.

[22] G. Giray, "A software engineering perspective on engineering machine learning systems: State of the art and challenges," Journal of Systems and Software, vol. 180, Art. no. 111031, 2021.

[23] D. Kreuzberger, N. Kühl, and S. Hirschl, "Machine learning operations (MLOps): Overview, definition, and architecture," IEEE Access, vol. 11, pp. 31866–31879, 2023.

[24] F. Bayram, B. S. Ahmed, and A. Kassler, "From concept drift to model degradation: An overview on performance-aware drift detectors," Knowledge-Based Systems, vol. 245, Art. no. 108632, 2022.

[25] A. B. Arrieta et al., "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," Information Fusion, vol. 58, pp. 82–115, 2020.

[26] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning," ACM Computing Surveys, vol. 54, no. 6, Art. no. 115, 2021.

[27] European Parliament and Council of the EU, "Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act)," Official Journal of the European Union, Jul. 2024.

© 2026 The Authors. This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution(CC–BY)license
https://creativecommons.org/licenses/by/4.0