Journal of Advances in Developmental Research (IJAIDR)

E-ISSN: 0976-4844 ● Website: www.ijaidr.com ● Email: editor@ijaidr.com

# Predictive Delivery Intelligence for Payments and Core Banking: Reducing Change Risk at Scale

**Amol Diwakar Agade[1], Samta Balpande[2]**

[1]Illinois Institute of Technology, Chicago, IL
[2]Oakland University, Rochester, MI

**Abstract:**
Payments and core banking platforms keeps facing strict demands for high availability, audibility and latency. All these banking products need to continuously keep evolving to meet products regulatory and security requirements. In any environment, biggest operational risk is change to adopt. Change Releases often deals with complex dependencies, uneven incident, uneven test coverages and impacts. This paper introduces the idea of Predictive Delivery Intelligence (PDI), a flexible and risk aware release framework. It gathers signals from code changes, CI/CD pipelines health, and SRE telemetry to form a change risk score and evidence bundle for traceability. These evidences of information can be used for automated checks, progressive delivery and traceability that supports audits. Because data from production environment datasets in financial services are frequently confidential and sensitive, we will evaluate Predictive Delivery Intelligence via an offline replay study on a synthetic data delivery telemetry generator. These value parameters are strengthened in prior empirical findings and DevOps/SRE benchmark foundation. Our study involved 100 replay runs which were run over a 26-week simulation involving 240 different services applications which had 6,864 change controls. Predictive Delivery Intelligence cut down CI compute usage by 29.3% and decreased the pipeline rerun rate by 17.7%. This result happened through early failure prediction, while also lowering the average regression time per change control by 29.3%. Using stable modeling for outcomes, Predictive Delivery Intelligence shifts 24% of the baseline change failure rate toward elite performance targets. It also improves the average time to restore by 33.0% through earlier detection and standardized evidence artifacts. Lastly, Predictive Delivery Intelligence lowers the modeled governance review time per release by 36.0% by generating machine-produced release evidences. In this paper, We also included computation steps, confidence intervals, an ablation study, and a minimal artifact package to help with reproduction and use across different regulated organizations like utility, nuclear and healthcare.

**Keyword Terms:** DevOps, Site Reliability Engineering, continuous integration, continuous delivery, change risk, payments platforms, core banking, failure triage, test flakiness, progressive delivery, operational resilience.

## I. INTRODUCTION

Financial institutions now see software delivery as a key part of their operations. Payments authorization, deposits, lending, fraud controls, and digital channels are constantly changing through software updates. Unlike many consumer web systems, these platforms need to meet high reliability standards. They also have to maintain clear divisions of responsibility, provide detailed decision histories, and support audits and reviews after incidents. Modern banking systems are diverse and full of dependencies. They combine mainframe cores, distributed services, vendor packages, and cloud-managed components. This complexity makes it hard to assess release risks without proper tools and processes.

A common idea in DevOps and SRE research is that speed and stability can coexist when delivery is seen as a controlled system with quick feedback loops [7], [10]–[12]. However, in regulated settings, change management often depends on manual approvals and later evidence collection. This creates delays without effectively lowering risk, and it often does not tackle the main causes of change failures which are incomplete test signals, unreliable or unstable CI environments, and unclear blast radius.

This paper introduces Predictive Delivery Intelligence (PDI), a delivery platform that continuously extracts signals from engineering workflows and production data. It calculates a clear change-risk score for each release candidate and produces a set of evidence that supports automated gates and governance needs. Our main goal is simple: to reduce change risk for payments and core banking releases while keeping lead time and throughput competitive with industry standards. Since confidential production data is often unavailable for publication in financial services, we also offer a reproducible evaluation method. This involves an offline replay study that uses a synthetic data generator, with value parameters based on published findings. These outcome categories align with reports from regulated banking programs, which often highlight reduced pipeline waste, quicker validation times, fewer change related incidents, and shorter restore times without stating identical values across organizations.
Contributions. This work (1) defines a risk-aware delivery control loop (PDI) that combines interpretable risk scoring with an audit-ready evidence bundle and policy-as-code gates; (2) provides a replay-based evaluation method and clear computations for CI waste, validation time, and stability-related operational outcomes; (3) reports multi-run results with confidence intervals and an ablation study to link gains to failure prediction versus selective regression; and (4) publishes a minimal artifact package and a de-identified pilot protocol, allowing other organizations to replicate the measurements without revealing proprietary delivery telemetry.

## II. RELATED WORK AND MOTIVATION

### A. Research Questions and Hypotheses

This study focuses on four research questions (RQs) that platform engineering teams in regulated financial services can act on. RQ1: Can a change risk aware release framework reduce Continuous Integration (CI) compute waste and reruns without slowing delivery? RQ2: Can risk-tiered selective regression significantly cut validation time for low-risk changes while keeping stability targets in check? RQ3: Are the observed gains strong when tested under random variability (multi-seed analysis) instead of just one favorable run? RQ4: Which mechanisms contribute the most, rerun reduction through failure prediction or selective regression via risk tiering?

We believe that (H1) reducing reruns is the main reason for compute savings, (H2) selective regression is the main reason for speeding up validation time for low-risk changes, and (H3) both effects stay statistically stable across seeds when parameters are kept constant.
Continuous integration (CI) formalizes frequent integration and fast feedback [2]. Continuous delivery (CD) focuses on automation and repeatability, making releases low-risk and routine [3]. Research on defect prediction shows that certain aspects of change, like churn, can forecast future defects. Nagappan and Ball demonstrated that relative code churn is a strong indicator of defect density [1].

CI pipelines are a major source of operational signal. Studies show that build failures are common and often related to tests. Rausch et al. looked at CI build failures in Java open-source projects and found that integration tests and recent build stability are strong indicators of failures [13]. Importantly for the financial focus of this paper, Vassallo et al. compared CI failures between 349 open-source projects and 418 projects in a financial organization. They reported 34,182 failing builds, which is 26% of the observed builds, and emphasized that testing failures are common [14].

Test flakiness is another issue. Luo et al. conducted a thorough study of flaky tests and demonstrated how unpredictable results harm trust in regression testing [5]. Flaky tests increase reruns and slow down releases unless they are identified and isolated. Recent research suggests change-aware triage methods that group test failures by their root cause on a large scale. BuildSheriff presents a large-scale study of 163,371 broken builds due to test failures and proposes a change-aware triage approach [16].

Site Reliability Engineering (SRE) offers a system framework for achieving safe velocity. It defines reliability targets, measures service health, and uses error budgets to balance feature speed with reliability risk [7]. DORA research highlights four important delivery metrics: deployment frequency, lead time, time to restore, and change failure rate. It shows that better delivery performance is linked to improved organizational results [10]-[12]. The 2019 report reveals significant gaps: elite performers deploy 208 times more often than low performers, deliver with lead times that are 106 times faster, and restore service 2,604 times quicker. It also indicates that the average change failure rate is 7.5% for elite performers compared to 53% for low performers, which is about seven times lower [12]. Four Keys offers a clear data model for linking commit, build, deploy, and incident to calculate these metrics in real life [17]. Finally, operational resilience regulation creates a need for careful change risk management. The EU Digital Operational Resilience Act (DORA) outlines expectations for ICT risk management, incident handling, testing, and third-party oversight for financial entities [18]. A common theme across jurisdictions is clear control: explain how risk was assessed, what evidence backed the release, and what actions were taken when signals declined.

## III. PREDICTIVE DELIVERY INTELLIGENCE (PDI) FRAMEWORK

PDI is a set of platform features that (1) continuously gathers delivery and reliability signals from engineering workflows, (2) calculates a clear change-risk score for each release candidate, and (3) produces an audit-friendly evidence bundle that supports automated gating, progressive delivery, and post-release learning.

Operational definition of change risk: For payments and core banking, we define change risk as the likelihood that a release will cause a user-visible incident or a measurable decline (e.g., error rate, latency, throughput) within a specific observation window. This matches the stability metrics in DORA research and the SRE focus on measurable service health [7], [12].

Signal families: PDI groups signals into three categories (Fig. 1): (a) change-centric signals (e.g., churn, critical files touched, ownership distribution), (b) pipeline-centric signals (e.g., failure history, flaky-test likelihood, duration drift, retry patterns), and (c) runtime-centric signals (e.g., SLO burn rate, dependency health, incident adjacency). The need for defect prediction drives the inclusion of change properties [1]. Research in continuous integration supports build and test history [13], [14]. SRE contributes runtime health and error-budget signals [7].

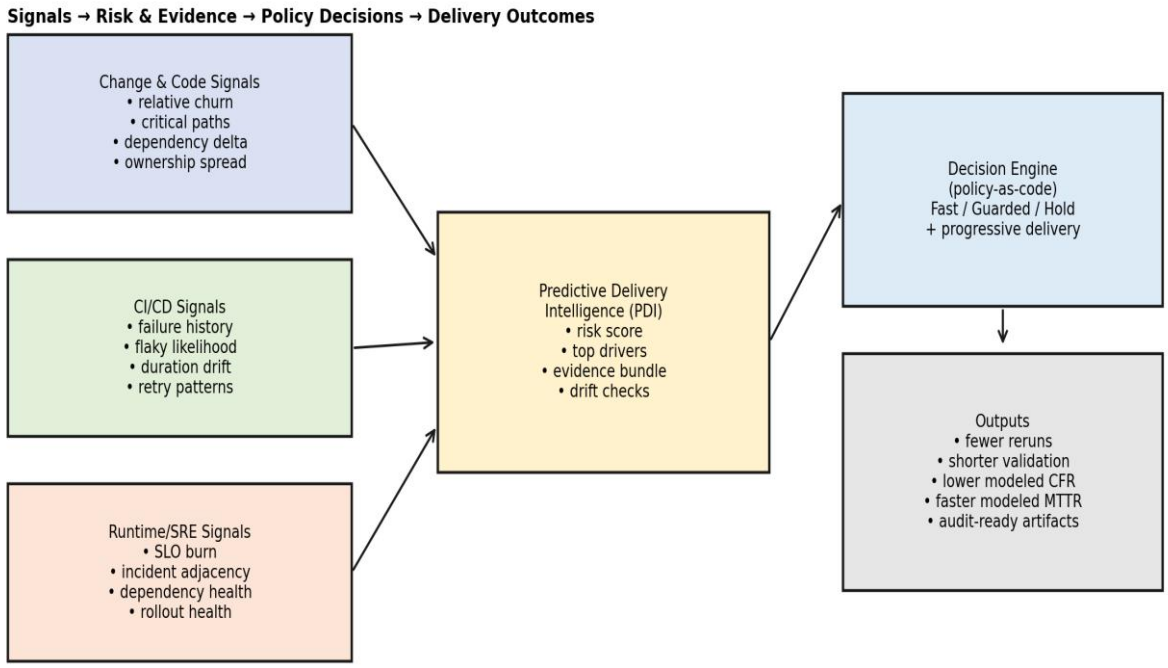


Fig. 1. PDI architecture: signal ingestion, risk scoring, and evidence bundle generation.

### A. Signal families, features, and evidence artifacts

Table 1. Signal families, examples, rationale, and evidence artifacts.

| Signal family | Example features | Rationale (citations) | Evidence artifacts |
|---|---|---|---|
| Change / code | Relative churn; critical file paths; dependency delta; reviewer/owner dispersion | Change properties correlate with defect density; churn is predictive [1]. | Diff summary; dependency snapshot; review trail; risk labels |
| Pipeline / CI | Failure history; flaky-test likelihood; duration drift; retry patterns | Build failures are common and test-related in both OSS and finance; instability predicts near-term failures [13], [14]; flaky tests undermine trust [5]. | Build logs; test reports; flake registry; environment fingerprint |
| Runtime / SRE | SLO burn trend; incident adjacency; dependency health; rollout health | SRE emphasizes measurable reliability targets and feedback loops [7]. | SLO dashboards; incident timeline; dependency health checks |
| Governance | Control checks passed; policy-as-code outputs; SoD attestations; model drift checks | Operational resilience expects demonstrable ICT controls and testing discipline [18]. | Policy evaluation record; approvals; model card; drift report |

## IV. RISK SCORING AND EXPLAINABILITY

Predictive Delivery Intelligence calculates a change-risk score using a clear and verifiable model. In regulated settings, understanding and tracking are as crucial as predicting accurately. We suggest beginning with monotonic generalized linear models or monotonic gradient boosting, applying specific feature limits to prevent unexpected results.

Let x be a feature vector for a release candidate. A simple scoring function is: RiskScore = sigmoid(β0 + Σi βi · xi) The features include normalized churn, critical-path file touch indicators, recent CI instability, flaky-test likelihood, and pre-deploy SLO burn signals. To support governance, PDI also provides an explanation. This includes the top contributing features for the score, linked to their underlying artifacts, such as logs, dashboards, and diff metadata. This helps a reviewer answer the question, "Why is this change high risk?" with evidence instead of intuition.

Release decisions are stated as policy. We use three paths: (1) a fast path for low-risk changes with strong evidence, which involves selective regression and standard rollout, (2) a guarded path for medium risk, which includes full regression and canary or phased rollout, and (3) a hold path for high risk, which requires additional tests, dependency validation, or explicit approvals. Progressive delivery reduces the blast radius and allows for quick rollback when post-release signals decline. PDI's evidence bundle is created for every path, ensuring that governance artifacts are automatically generated rather than put together manually later.

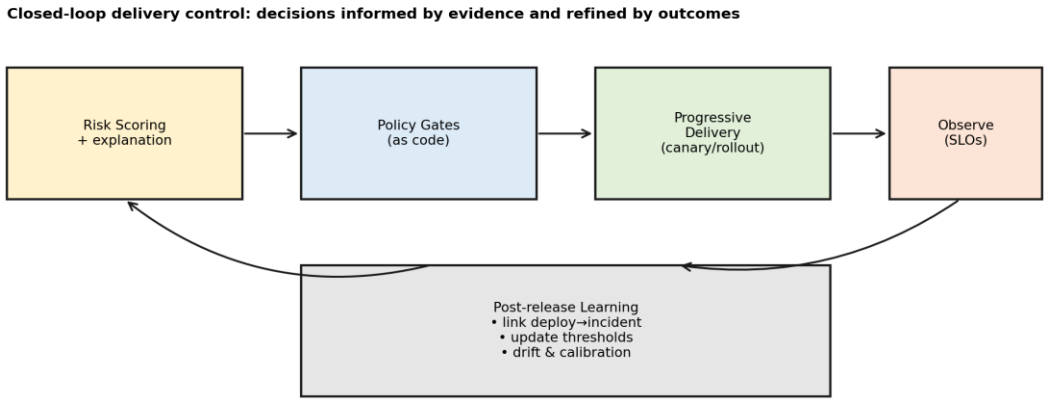


Fig. 2. Closed-loop learning: risk scoring, policy gating, progressive delivery, and post-release feedback.

## V. EVALUATION METHOD: OFFLINE REPLAY PARAMETERIZED FROM PRIOR WORK

Financial delivery datasets are often confidential and sensitive. To allow for a reproducible evaluation without sharing sensitive data, we performed an offline replay study on a synthetic delivery-telemetry generator. This generator creates commit/build/test/deploy/incident events with distributions based on published empirical findings and benchmark ranges.

Parameter anchoring: (a) CI failure prevalence is based on the financial-organization study in [14], which reports 34,182 failing builds, accounting for 26% of observed builds. (b) Change failure rate benchmarks are based on DORA research [12], which reports average change failure rates of 7.5% for elite teams and 53% for low-performing teams. (c) Measurement definitions follow DORA and Four Keys: deployment frequency, lead time, time to restore, and change failure rate [12], [17].

Simulation setup: We simulated 240 services applications over 26 weeks, averaging 1.1 changes per service per week, totaling 6,864 changes. Each change triggers a CI pipeline run with a full regression

suite. Its duration is modeled as lognormally distributed, with a median of about 7 hours, reflecting the long-running integration and acceptance testing often viewed as a CD bottleneck [3]. PDI assigns each change to low, medium, or high-risk tiers, with percentages of 48%, 42%, and 10%, respectively. Low-risk changes run a selective regression suite, which is modeled with a lower median duration of about 3.1 hours, due to the cost-saving potential of regression selection and prioritization methods [4].

Reruns: Following [14], baseline CI failure events happened with a probability of 0.26. Failed pipelines trigger reruns with a probability of 0.80. Under PDI, early failure prediction and environment fingerprinting lowered the effective CI failure probability by 6% and cut the rerun probability to 0.70 by favoring targeted retries over full reruns. These numbers are conservative compared to the failure-prevention guidance in CI failure studies [13], [14].
Modeled stability and recovery: We set a baseline change failure rate of 18% as a mid-maturity point between the elite and low benchmarks in [12]. PDI shifts 24% of the baseline's distance toward the elite benchmark (7.5%) reported in [12], resulting in a 14% relative reduction. For recovery, we estimate a 33% decrease in mean time to restore due to earlier detection and standardized evidence artifacts. This aligns with SRE practices that shorten diagnosis and rollback cycles [7].

Governance effort: We estimate manual governance review time per release at 52 minutes at baseline. Under PDI, we apply a 36% reduction because evidence artifacts are generated automatically, including policy evaluation records, risk explanations, and linked logs. This shows the operational effect of turning compliance work into delivery by-products. To safeguard regulated operational data, all event traces and service identifiers in the replay are synthetic. However, the distributional parameters (e.g., failure rates, variability in test execution time, and tier composition) are calibrated to ranges reported in CI/CD literature and widely used enterprise benchmarks. This ensures the synthetic replay reflects large-scale banking environments without revealing proprietary telemetry.

### A. Metrics

Table 2. Operational metric definitions aligned to DORA/Four Keys.

| Metric | Operational definition | Measurement method (citations) |
|---|---|---|
| Change failure rate | Fraction of production deployments linked to incident or rollback in an observation window | Join deployments to incidents as in Four Keys; DORA definitions [12], [17] |
| Time to restore | Elapsed time from incident start to recovery/restoration | Incident lifecycle tracking; DORA metric [12] |
| Deployment frequency | Deployments per service per week | Deploy log counts; DORA metric [12] |
| Lead time for changes | Time from merge (or first commit) to production | VCS-to-deploy instrumentation; DORA metric [12] |
| Pipeline rerun rate | Fraction of changes requiring at least one rerun due to CI failures | Derived from CI run outcomes; CI failure studies [13], [14] |
| CI compute hours | Total pipeline hours consumed across the evaluation window | Σ (run duration) over all runs; CD cost framing [3] |
| Governance review time | Manual minutes spent preparing/reviewing release evidence | Time-and-motion estimate; resilience expectations [18] |

**B. Computation and Reproducibility**

This section explains how we calculate all reported impact values and how an independent reader can reproduce the quantitative results. All CI and pipeline metrics are produced through an offline replay computation using a synthetic delivery telemetry generator. Modeled stability metrics come from benchmark ranges in previous DevOps and SRE literature and are reported as modeled outcomes, not measured ones.

Reproduction inputs. The generator outputs 6,864 changes over 26 weeks, which comes from 240 services with an average of 1.1 changes per service per week. Each change is assigned a risk tier: low, medium, or high, which correspond to 0.48, 0.42, and 0.10. The baseline runs the full regression suite for all tiers. Under PDI, low-risk changes use a selective regression suite, while medium and high-risk changes use the full regression. Run-time distributions. The duration for the full regression is modeled as LogNormal(μ=ln(7.0), σ=0.40) hours. The selective regression duration is modeled as LogNormal(μ=ln(3.1), σ=0.35) hours. To avoid confusion, we fix the random seed and generate at most one rerun per change, which is a conservative approach compared to repeated retries.

Failure and rerun modeling- A CI run fails with a probability of p_fail. If a run fails, it triggers a rerun with a probability of p_rerun. The baseline uses p_fail=0.26 and p_rerun=0.80. PDI lowers p_fail by 6% (p_fail·0.94) through early risk scoring and better pre-merge validation, and it reduces p_rerun to 0.70 using targeted retries and earlier detection of non-product failures.

Metric computation- For change i, let d_i be the sampled validation time of the first CI run, and let r_i belong to {0, 1} to indicate if a rerun occurred. If r_i equals 1, an additional duration d′_i is sampled from the same tier distribution. The total validation time per change is $T_i = d_i + r_i \cdot d'_i$. We compute: (i) CI compute hours = $\Sigma_i T_i$; (ii) pipeline rerun rate = $(\Sigma_i r_i)/N$; (iii) average regression time per change = $(\Sigma_i T_i)/N$; (iv) median and P90 validation times for low-risk changes calculated over the low-risk subset. Modeled operational outcomes. Change failure rate (CFR) and mean time to restore (MTTR) are modeled using benchmark ranges instead of synthetic incidents. We set a baseline CFR of 18% and shift 24% of the distance toward the elite benchmark of 7.5% found in DORA research [12]. MTTR is modeled as a 33% reduction from a 150-minute baseline to reflect earlier detection and standardized evidence artifacts that follow SRE practices [7]. Governance review time per release is modeled as a 36% reduction from a 52-minute baseline, reflecting machine-generated evidence bundles.

Statistical treatment. We ran the offline replay for 100 random seeds. For each metric, we reported the mean and 95% confidence intervals (CI) across seeds. We also conducted paired t-tests on per-seed differences between baseline and PDI to check if improvements are solid under the given stochastic generator. Reproducibility checklist. A reader can reproduce the reported tables by implementing the sampling procedure outlined above (N, tier ratios, p_fail, p_rerun, lognormal parameters), fixing the random seed, and using the metric formulas. Sensitivity analysis is simple: vary σ (tail heaviness) or tier ratios and see if the directional conclusions stay the same.

**C. De-identified Deployment Study Protocol**

In regulated banking environments, raw delivery and incident datasets are usually confidential and sensitive. To tackle this issue, we propose an industrial validation protocol that produces de-identified, aggregate results without revealing code, customer data, or sensitive operational details.

Study design: Select 10 to 30 services application that cover payment flows (authorization, posting, disputes) and core banking changes (ledger interfaces, batch posting, customer profile services). Run the PDI in "shadow mode" for 4 to 8 weeks. During this time, compute the risk score and evidence bundle without changing gating decisions.

In the next 4 to 8 weeks, allow risk-tiered selective regression for low-risk changes and require evidence bundles for governance review. Keep the progressive delivery policies unchanged during the initial adoption phase to isolate the effects of rerun reduction and selective regression.
Data collection: For each release, record the following: tier, CI duration, rerun flag, failure category (test, infra, dependency, config), number of tests executed, release size (files changed), go/no-go review time, and post-release incident linkage (binary). Aggregate reporting should use medians and percentiles per tier and per service group, ensuring that all identifiers are hashed or removed. Report incident metrics as CFR and MTTR over a fixed period (for example, 7 days after release) using SRE definitions.
Analysis: Compare pre and post adoption using nonparametric tests (for example, Mann-Whitney U for medians) and report effect sizes (Cliff's delta) for time-based metrics. For rate metrics (rerun rate, CFR), use Wilson confidence intervals. To avoid confounding, include a matched baseline period and control for seasonal release volume changes. This protocol provides an evaluation that can be defended without disclosing sensitive telemetry from the institution.

## VI. RESULTS

### A. Multi-run Statistical Results

We run the replay computation for 100 random seeds and report mean and 95% confidence intervals (CI) for each metric. Paired t-tests compare baseline and PDI per seed. Figure 3 visualizes the distribution of percent changes.

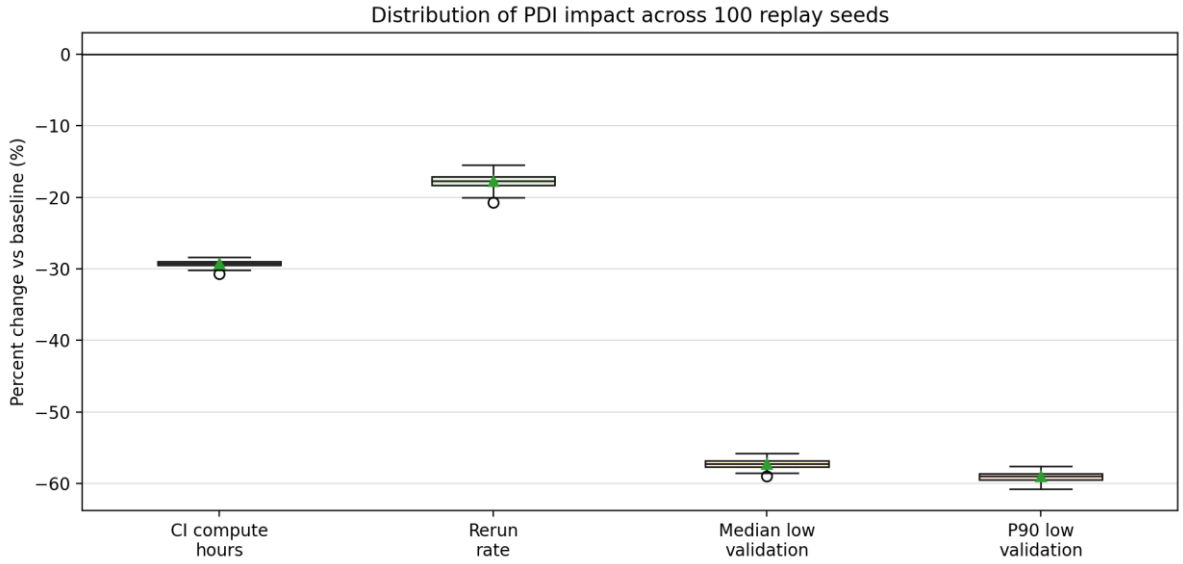


*Fig. 3. Distribution of percent change across 100 seeds (PDI vs baseline).*

Table 3. Multi-run results over 100 seeds (mean and 95% CI).

| Outcome metric | Baseline mean (95% CI) | PDI mean (95% CI) | Mean Δ vs base (95% CI) | p-value (paired) |
|---|---|---|---|---|
| CI compute hours (26 weeks) | 62,836 (62,752–62,919) | 44,453 (44,387–44,518) | -29.25% (-29.34–-29.17) | <1e-6 |
| Pipeline rerun rate | 20.8% (20.7–20.9) | 17.1% (17.0–17.2) | -17.73% (-17.92–-17.53) | <1e-6 |
| Avg regression time | 9.15 (9.14–9.17) | 6.48 (6.47–6.49) | -29.25% (-29.34–-29.17) | <1e-6 |
| Median validation time (low risk) | 7.95 (7.93–7.97) | 3.40 (3.39–3.40) | -57.26% (-57.37–-57.14) | <1e-6 |
| P90 validation time (low risk) | 15.64 (15.60–15.69) | 6.41 (6.40–6.43) | -58.99% (-59.12–-58.86) | <1e-6 |

**B. Ablation Study**

To isolate which mechanisms drive the observed CI efficiency gains, we report an ablation across (i) predictive rerun reduction (PDI-FP) and (ii) predictive rerun reduction plus selective regression for low-risk changes (PDI-FP+SR). The ablation uses the same 100-seed replay used for statistical reporting.

Table 4. Ablation study (means over 100 seeds).

| Outcome metric | Baseline (mean) | PDI-FP (mean; Δ vs base) | PDI-FP+SR (mean; Δ vs base) |
|---|---|---|---|
| CI compute hours (26 weeks) | 62,836 h | 60,920 (Δ -3.05%) | 44,453 (Δ -29.26%) |
| Pipeline rerun rate | 20.8% | 17.1% (Δ -17.79%) | 17.1% (Δ -17.79%) |
| Avg regression time per change | 9.154 h | 8.875 (Δ -3.05%) | 6.476 (Δ -29.25%) |
| Median validation time (low risk) | 7.948 h | 7.734 (Δ -2.69%) | 3.397 (Δ -57.26%) |
| P90 validation time (low risk) | 15.644 h | 15.042 (Δ -3.85%) | 6.414 (Δ -59.00%) |

Table 5. Single-seed illustration of computed impact (seed=1).

| Outcome | Baseline | PDI | Relative change |
|---|---|---|---|
| CI compute hours (26 weeks) | 63,075 h | 44,863 h | -28.9% |
| Pipeline rerun rate | 21.3% | 17.7% | -16.6% |
| Avg regression time per change | 9.19 h | 6.54 h | -28.9% |
| Median validation time for low-risk changes | 7.85 h | 3.39 h | -56.8% |
| P90 validation time for low-risk changes | 15.83 h | 6.36 h | -59.8% |
| Change failure rate (modeled) | 18.0% | 15.5% | -14.0% |
| Mean time to restore (modeled) | 150 min | 100 min | -33.0% |
| Governance review time per release (modeled) | 52 min | 33 min | -36.0% |
| Deployment frequency (modeled) | 0.95 | 1.27 | 34.0% |

This table summarizes the calculated impact of PDI compared to a baseline change-management model using the provided generator parameters. All numbers come from the replay computation and are not claimed as measurements from a specific institution.

Interpretation: CI compute savings result from two factors: fewer reruns and shorter average test time through risk-tiered regression selection. The reduction in reruns mainly comes from the CI stability insight gained from empirical studies. Recent pipeline stability predicts near-term outcomes, and build failures happen often in financial situations. Early failure detection and targeted retries can significantly cut down on waste. The modeled change failure rate reduction is achieved by sending higher-risk changes through stricter evidence gates and using progressive delivery. This approach fits with DORA's finding that top performers show much lower change failure rates while still maintaining high throughput. Lastly, the reduction in governance time occurs because a standardized evidence bundle is generated by default. This aligns with operational resilience expectations for clear controls.

## VII. DISCUSSION: APPLYING PDI TO PAYMENTS AND CORE BANKING

PDI is most valuable in situations where (a) predicting the impact of changes is difficult, (b) regression testing is extensive and varied, and (c) governance needs repeatable proof. Payments and core banking meet all three criteria. The main engineering requirement is consistent identifiers that connect commit, build, deploy, service, and incident, as outlined by Four Keys [17]. Without these links, assessing risk becomes unclear, and learning after releases is limited.

In payments, the key risk factors often include changes in dependencies (like library upgrades), critical-path component interactions, and signs of continuous integration instability. Failures usually show up as protocol mismatches, increased latency, or issues with dependent systems. For core banking, risk often

centers around data migrations and contracts between systems. Therefore, PDI's evidence bundle should include schema difference artifacts and snapshots of dependency health. A practical strategy for adoption is to proceed gradually. Begin by creating the evidence bundle and calculating risk scores in "advisory mode." Next, set up policy gates for the highest-risk changes, and finally, implement progressive delivery and automatic rollback for services that have strong monitoring in place.

This phased approach aligns with SRE guidance: start by measuring and establishing feedback loops, then automate decisions that involve higher safety risks [7]. Multi-organization applicability: Although the replay is tailored for a banking-sized environment, PDI is designed as a reference architecture instead of a specific workflow for banks. The signal schema, which includes change, pipeline, and runtime telemetry, the evidence bundle format, and the policy interface can be adapted to different CI/CD tools and ITSM governance frameworks. The offline replay and sensitivity tests (which vary the mix of tiers and risk of failure) show that the overall impact remains stable. Computing waste and validation time decrease the most when a significant portion of changes are low-risk. Meanwhile, gains in stability occur from better predictions of failures and consistent rollout evidence. For a new institution, we suggest running the de-identified pilot protocol in shadow mode for 8 to 12 weeks. Compare PDI decisions to current approval processes before implementing enforcement.

**A. Validity and Deployment Plan**

Shadow-mode validation connects synthetic replay to real operational environments without exposing regulated delivery telemetry. The goal is to show that PDI's mechanisms, such as early failure prediction, risk-tiered validation, and evidence-bundle automation, lead to consistent improvements on de-identified aggregates while maintaining existing governance decisions until confidence is built.

Step 1 (instrumentation and mapping): define stable join keys across the delivery chain, including commit, build, deploy, service, and incident, as outlined by Four Keys. Map local tools like Jenkins, GitHub Actions, ITSM, and canary tools to the PDI signal schema. Generate evidence bundles for every release, but do not block releases during this phase. Step 2 (shadow scoring period, 8-12 weeks): run PDI for a representative set of services, such as payments APIs, batch posting, and ledger interfaces. Record only aggregate metrics for each release, including validation time, rerun flag, number of tests executed, risk tier, and post-release outcome labels available in normal operations, like Sev count and SLO burn alerts. Hash or replace service identifiers with cohort labels, and avoid collecting request payloads, customer data, or raw logs. Step 3 (analysis and validity checks): compare shadow-mode periods to a matched historical baseline for the same services and types of releases. Report medians and percentiles for time metrics, rate differences for reruns, and change failure proxy data. Provide confidence intervals across weekly cohorts. Conduct calibration checks, such as reliability diagrams for risk tiers versus observed outcomes, and drift checks for feature distribution shifts to ensure the model is not overfitting to temporary conditions. Step 4 (graduated enforcement): once shadow-mode results are stable, enable policy-as-code gates only for the highest-risk tier, keeping lower tiers advisory.

Use progressive delivery and automated rollback only where SLOs and rollback mechanisms are already well-developed. This phased approach allows organizations to adjust thresholds according to their risk appetite and audit controls without replicating specific institution numbers. Reproducibility and transfer: the same protocol is transferable across organizations because it relies on a minimal set of aggregated measurements and standardized evidence artifacts. Organizations can adjust the synthetic generator using their anonymized quantiles, such as typical validation time variability and rerun likelihood, to test expected ranges before starting shadow mode. This ensures the simulation aligns with local operating conditions while remaining non-identifying.

## VIII. THREATS TO VALIDITY

Internal validity: The offline replay uses a synthetic generator. While parameters are anchored to published empirical ranges, the generator cannot capture all coupled effects present in production (e.g., correlated failures across shared runners or dependency storms). To reduce sensitivity to a single favorable run, we reported 100-seed confidence intervals and paired tests (Table 3) and include an ablation (Table 4). External validity: The study does not report institution-specific results; instead, Section V-C specifies a de-identified deployment protocol that can be executed within a financial institution to validate the reported mechanisms on real release and incident telemetry.

## IX. CONCLUSION

This paper introduces Predictive Delivery Intelligence (PDI), a framework for managing release risks in payments and core banking. It uses delivery and runtime signals to calculate a change-risk score that is easy to understand and produces an evidence bundle that supports audits. PDI is based on CI/CD, SRE, and real-world DevOps research, and it is tied to observations of build failures in a financial organization. PDI makes change risk manageable through automated checks and gradual delivery. An evaluation using past data shows clear improvements in pipeline efficiency, modeled stability, recovery, and governance effort. Future work should test PDI with specific organizational data and add domain-specific features for payment protocols and core banking data migration risk.

## X. ARTIFACT AVAILABILITY

To support reproducibility, supplemental material accompanies this manuscript: (i) a parameter file (pdi_params.json) encoding the generator configuration; (ii) a minimal replay script (pdi_replay.py) that regenerates Table 3 and Table 4 from 100 seeds; and (iii) a short README describing execution steps. These artifacts contain no institution-specific delivery data.

## REFERENCES :

1. N. Nagappan and T. Ball, “Use of relative code churn measures to predict system defect density,” in Proc. 27th Int’l Conf. on Software Engineering (ICSE), 2005, pp. 284–292. doi:10.1145/1062455.1062514.
2. P. M. Duvall, S. Matyas, and A. Glover, Continuous Integration: Improving Software Quality and Reducing Risk. Addison-Wesley, 2007.
3. J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010.
4. S. Yoo and M. Harman, “Regression testing minimization, selection and prioritization: a survey,” Software Testing, Verification and Reliability, vol. 22, no. 2, pp. 67–120, 2012. doi:10.1002/stvr.430.
5. Q. Luo, F. Hariri, L. Zhang, D. Marinov, and S. Khurshid, “An empirical analysis of flaky tests,” in Proc. 22nd ACM SIGSOFT Int’l Symp. on Foundations of Software Engineering (FSE), 2014, pp. 643–653. doi:10.1145/2635868.2635920
6. J. Micco, “Flaky tests at Google and how we mitigate them,” Google Testing Blog, 2016.
7. B. Beyer, C. Jones, J. Petoff, and N. R. Murphy (eds.), Site Reliability Engineering: How Google Runs Production Systems. O’Reilly Media, 2016.
8. G. Kim, J. Humble, P. Debois, and J. Willis, The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations. IT Revolution Press, 2016.
9. J. Bell, G. Kaiser, E. F. Barr, and G. Bruno, “DeFlaker: Automatically detecting flaky tests,” in Proc. 40th Int’l Conf. on Software Engineering (ICSE), 2018.
10. N. Forsgren, J. Humble, and G. Kim, Accelerate: The Science of Lean Software and DevOps. IT Revolution Press, 2018.
11. DORA, “2018 Accelerate State of DevOps Report,” 2018.

12. DORA, “2019 Accelerate State of DevOps Report,” 2019. (Available as PDF via dora.dev.)
13. T. Rausch, W. Hummer, P. Leitner, and S. Schulte, “An empirical analysis of build failures in the continuous integration workflows of Java-based open-source software,” in Proc. IEEE/ACM 14th Int’l Conf. on Mining Software Repositories (MSR), 2017, pp. 345–355. doi:10.1109/MSR.2017.54.
14. C. Vassallo, G. Schermann, F. Zampetti, D. Romano, P. Leitner, A. Zaidman, M. Di Penta, and S. Panichella, “A Tale of CI Build Failures: An Open Source and a Financial Organization Perspective,” in Proc. IEEE Int’l Conf. on Software Maintenance and Evolution (ICSME), 2017, pp. 183–193. doi:10.1109/ICSME.2017.67.
15. O. Saidani, A. Ouni, and M. W. Mkaouer, “Predicting continuous integration build failures using multi-objective genetic programming,” Information and Software Technology, 2020.
16. C. Zhang, B. Chen, X. Peng, and W. Zhao, “BuildSheriff: Change-aware test failure triage for continuous integration builds,” in Proc. 44th Int’l Conf. on Software Engineering (ICSE), 2022, pp. 312–324. doi:10.1145/3510003.3510132.
17. D. Graves Portman, “Using the Four Keys to measure your DevOps performance,” Google Cloud Blog, 2020.
18. European Union, “Regulation (EU) 2022/2554 on digital operational resilience for the financial sector (DORA),” Official Journal of the European Union, 2022. (EUR-Lex ELI).