Graph-Based Control-Impact Modeling for Predictive Change-Risk Mitigation in Regulated DevOps Pipelines

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

[1]Comerica Bank, USA; Illinois Institute of Technology, Chicago, IL, USA
[2]DTE Energy, USA; Oakland University, Rochester, MI, USA

Abstract:
Regulated enterprises increasingly ship software through CI/CD, infrastructure as code (IaC), and policy as code, yet they must still satisfy audit-ready controls and narrow operational windows. In banking, deployment risk clusters around market cycles and settlement cutovers; in electric utilities, it concentrates around peak-load operations and maintenance windows where grid reliability obligations constrain change. This paper proposes a Control-Impact Graph (CIG): an explicit, auditable graph that links pipeline artifacts (source, build, IaC, configuration, and policy) to compliance controls, machine-verifiable evidence, and service outcomes. The CIG supports predictive change-risk scoring by fusing graph-derived structural features, controls-as-code signals, and AIOps indicators from logs and metrics. We use these scores to drive a risk-aware progressive delivery policy that selects canary, blue-green, staged hold, or rollback based on predicted risk, evidence completeness, and the current compliance window. Two sector-specific case studies—a banking payments/settlement flow and a utility telemetry ingestion service—illustrate patterns that improve delivery velocity without weakening the intent of existing controls.

Index Terms: DevOps, DevSecOps, policy as code, controls as code, infrastructure as code, compliance automation, AIOps, risk prediction, graph learning, progressive delivery.

I. INTRODUCTION

Modern software delivery in banking and utilities is a paradox: the business demands faster change, but the operating model still requires strict controls, strong auditability, and disciplined change windows. DevOps helps by automating work, tightening feedback loops, and reducing handoffs—yet in regulated environments, speed only counts if every change remains authorized, traceable, and compliant. Even the term DevOps is applied unevenly in practice; systematic mapping studies show it spans culture, automation, measurement, and governance practices rather than a single toolset [11]. Risk-based guidance, in turn, encourages continuous control alignment and evidence generation instead of periodic, manual checkpoints [1], [2].

In practice, compliance-sensitive pipelines have to manage multiple artifact types at once: IaC for provisioning, configuration and secrets for runtime behavior, policy as code for guardrails, and telemetry that proves operational assurance. Utilities add operational technology (OT) adjacent constraints and supply-chain risk obligations that influence vendor selection, procurement, and ongoing monitoring for bulk electric system cyber systems [6]. Banks face parallel expectations for resilient architecture and operations, including disciplined change management, third-party oversight, and effective monitoring for critical services [5], and some workloads must also meet payment data protection requirements [7].

Despite mature tooling, pipeline risk management is still too often reactive. Control gaps surface during audits, and operational risk shows up after an incident—usually at the worst possible time. SRE and AIOps improve detection and response [8], but they do not, by themselves, predict how a proposed change propagates across code, infrastructure, policy, and runtime conditions. Conversely, static control checks can certify policy conformance while missing multi-artifact failure modes (for example, an IaC networking change plus a policy tweak that quietly degrades observability).

To close that gap, we introduce a Control-Impact Graph (CIG) that links pipeline artifacts to compliance controls, evidence, and service objectives, so that change-risk can be estimated before a full rollout. The approach combines (i) graph structure capturing dependency and control relationships, (ii) controls-as-code and policy-as-code evaluation results as auditable evidence, and (iii) AIOps signals derived from logs and metrics to estimate the probability and impact of change-induced failures. We then apply these estimates to risk-aware progressive delivery: the pipeline selects rollout strategies (canary, blue-green, staged hold, or rollback) based on predicted risk and the compatibility of the current window with financial and grid operational constraints.

Novelty and contributions—this paper:

- Defines a formal CIG schema that unifies artifacts, controls, evidence lineage, operational windows, and outcomes in a single auditable graph.
- Introduces a predictive change-risk model that fuses CIG-derived structural features, controls-as-code results, and AIOps signals (including NLP-enriched change and incident text features) into calibrated probability and impact estimates.
- Proposes a risk-aware progressive delivery policy (canary/blue-green/staged hold/rollback) that enforces policy-as-code guardrails while respecting compliance windows in banking (market/settlement) and utilities (peak-load/maintenance).
- Provides sector-oriented case studies and an evaluation protocol that emphasize regulated-industry requirements: traceability, evidence completeness, and operational defensibility during audits and incident reviews.

II. BACKGROUND AND MOTIVATION

Regulated delivery pipelines have to translate high-level frameworks into checks that can run on every change. The NIST Cybersecurity Framework provides a practical organizing lens (Identify/Protect/Detect/Respond/Recover) and encourages continuous improvement rather than periodic control exercises [1]. NIST SP 800-53 extends that idea with a detailed control catalog that can be tailored into technical, procedural, and monitoring requirements for specific systems [2]. Once teams adopt containers and orchestration, the control surface expands—image integrity, provenance, runtime isolation, and orchestrator hardening become pipeline concerns, not afterthoughts [3].

Sector guidance sharpens these requirements. Banking supervision emphasizes operational resilience for critical services and expects disciplined change management, third-party oversight, and monitoring that can stand up to post-incident and audit scrutiny [5]. In the bulk electric system, supply-chain risk management standards such as CIP-013-1 push utilities to document vendor risk controls and to demonstrate that procurement and monitoring processes reduce cyber risk to grid reliability [6]. These regimes differ in terminology, but the shared expectation is consistent: controls must be evidenced at the time of change and traceable back to the artifacts that implemented them.

DevOps research and practitioner evidence show that elite teams can improve both delivery throughput and stability when automation, measurement, and feedback loops are treated as first-class engineering work [9]. SRE complements that with service-level objectives and incident practices that scale reliability [8]. In regulated environments, the harder lesson is that risk frequently rides along with “non-application” changes. IaC, configuration, and policy artifacts evolve continuously; they are a common source of defects and security drift if they are not reviewed, tested, and governed like code [10], [12], [13].

Progressive delivery (canary, blue-green, dark launches, and feature flags) is one of the most practical ways to limit blast radius while maintaining release cadence [14]. However, in finance and energy it cannot be applied in a vacuum: rollout choices interact with settlement cycles, peak-load periods, maintenance coordination, and approval windows. That coupling motivates a model that can reason jointly about controls, dependencies, and timing—not just code quality.

One consistent pattern in regulated incident reviews is cross-artifact causality. A "simple" outage often traces back to an interaction between an IaC change (routing, IAM, segmentation), a policy update (admission control, auth rules), and an observability gap that delayed detection. Treating these artifacts independently makes those interactions easy to miss. A control-aware graph representation gives us a way to capture them explicitly and learn from them.

III. PROBLEM STATEMENT AND REQUIREMENTS

We model the delivery system as a pipeline that turns a change request (CR) into a production state change. In practice, a single CR can span application code, IaC modules, configuration, policy definitions, and observability rules. Because the target environments are regulated, the pipeline must also emit audit-ready evidence that the change satisfies applicable controls—authorization, least privilege, logging, segmentation, vulnerability management, and supply-chain checks—grounded in sector frameworks and guidance [1]–[7]. We define two classes of time constraints that are common to both finance and energy:

1) Compliance windows: intervals where a deployment is permitted from a governance perspective, such as approved change windows, segregation-of-duty sign-off windows, and settlement/processing windows in banking. These often reflect business and regulatory operations (e.g., batch settlement, end-of-day close).

2) Safety windows: intervals where a deployment is safer from an operational reliability perspective, such as off-peak load periods, maintenance windows, or periods with additional staffing. In utilities, these may align with grid peak/maintenance periods; in banking, they may align with trading hours or high-volume customer periods.

The challenge is that compliance windows and operationally safe windows only partially overlap. A practical pipeline therefore needs to choose both a rollout strategy and a start time that keep risk within tolerance while still honoring governance constraints.

Requirements (R):

R1—Traceability: every prediction and decision must be traceable to artifacts, evidence, and controls so that auditors can review the rationale.
R2—Predictiveness: the model must estimate the likelihood and potential impact of change-induced incidents before deployment.
R3—Actionability: predictions must map to concrete delivery actions: adjust rollout strategy, insert additional verification, or hold and re-schedule.
R4—Cross-domain portability: the approach should accommodate both financial and utility pipelines by parameterizing control frameworks and operational windows.

Research questions (RQ):

RQ1: How can pipeline artifacts, controls, and outcomes be modeled in a unified structure that supports both compliance evidence and learning?
RQ2: How can we estimate change risk using a combination of graph structure, controls-as-code results, and AIOps signals?
RQ3: How should rollout decisions be optimized under overlapping or conflicting compliance and safety windows?

IV. CONTROL-IMPACT GRAPH MODEL

The Control-Impact Graph (CIG) is a directed, attributed multigraph G = (V, E) that captures how pipeline artifacts, compliance controls, and operational context connect to service outcomes. The goal is pragmatic: make control coverage and impact pathways explicit enough to support both audit review and predictive modeling.
Node categories:
A) Artifact nodes represent versioned pipeline assets: source modules, IaC templates, container images, configuration bundles, and policy definitions.

B) Control nodes represent control objectives or control statements (e.g., access control, logging, segmentation). Controls can be mapped from sector frameworks or internal control catalogs [1], [2], [5], [6].
C) Evidence nodes represent verifiable outputs produced by controls-as-code checks, such as signed attestations, policy evaluation results, scan reports, or approvals.
D) Service nodes represent deployable services and their runtime dependencies.
E) Outcome nodes represent reliability and risk outcomes such as SLO violations, incidents, or customer-impact metrics.
F) Window nodes represent time-bounded constraints and context (market hours, settlement windows, grid peak windows).

Edges encode semantics:
Artifact-Depends-On: captures build or runtime dependency.
Artifact-Enforced-By-Control: links an artifact to controls that constrain it.
Control-Produces-Evidence: links controls-as-code evaluations to evidence.
Artifact-Affects-Service: maps changes to affected services.
Service-Impacts-Outcome: links service behavior to incidents or SLOs.
Window-Constrains-Deployment: captures temporal constraints.

Each CIG snapshot carries attributes that support two modes of use. For deterministic reasoning, attributes capture pass/fail control outcomes and evidence completeness. For learning, they capture change and context signals such as churn, IaC resource types, policy rule complexity, scan severities, approval latency, historical rollback rate, and anomaly scores.

Compared to a conventional dependency graph, the CIG is deliberately control-aware. It represents not only what depends on what, but also what must be true to satisfy controls and what evidence demonstrates that those conditions were checked for this specific change. That distinction matters during audits and incident reconstruction, when teams need to justify why a deployment was allowed.

A. Control mapping and scoping
Each organization typically maintains an internal control catalog that maps external frameworks (e.g., [1], [2]) to enterprise standards. We represent this as a Control taxonomy with metadata: control family, applicability conditions (data classification, environment), and evidence requirements. The pipeline first scopes which controls apply to the change using metadata about affected services (tier, data class) and environments (prod/non-prod). This scoping step reduces noise and ensures the evidence chain is complete for the correct control set.

B. Evidence lineage and cryptographic integrity
Evidence nodes should provide tamper-evident lineage. We recommend hashing every evidence artifact (scan outputs, approval records, policy decision logs) and signing a per-run decision record that includes (i) artifact hashes, (ii) control evaluation results, and (iii) the computed impact set. In practice, this turns “what happened” into a replayable, verifiable chain of custody.

C. Graph update semantics
The CIG is most useful when maintained as a temporal graph. Each deployment adds a snapshot layer, and temporal edges link versions of the same artifact across deployments. Incidents and outcome nodes are attached after deployment, which allows the risk model to learn from both single changes and sequences that gradually drift a system into failure.

TABLE I. CONTROL-IMPACT GRAPH NODE TYPES AND EXAMPLES

Node Type	Examples	Key Attributes
Artifact	IaC module, container image, config bundle, policy file	version hash; churn; resource types; rule count
Control	logging, least privilege, segmentation, supply-chain checks	control ID; mapped framework; severity
Evidence	signed attestation, scan report, policy decision record	pass/fail; timestamp; signature; findings
Service	payment API, AML batch job, grid telemetry ingest	tier; SLO; dependency set
Outcome	incident, SLO breach, rollback, customer impact metric	impact score; duration; root-cause label
Window	market hours; settlement; peak load; maintenance	start/end; constraint type; staffing level

V. PIPELINE ARCHITECTURE: ARTIFACTS, POLICIES, AND EVIDENCE

The delivery pipeline is what turns the CIG from a static model into a living operational record. We treat the pipeline as a sequence of stages that produce both deployable artifacts and control evidence. Tooling varies by enterprise, but regulated pipelines tend to implement the same logical stages because the audit questions are consistent.

1) Plan and scope: identify impacted services, data classifications, and applicable control sets. This includes mapping the change to relevant framework categories (e.g., Protect and Detect activities in the NIST CSF [1]) and tailoring controls (e.g., NIST SP 800-53 control families [2]).
2) Build and package: compile and package application code, build container images, and generate SBOM and provenance artifacts. Container guidance recommends image integrity and registry protections as foundational steps [3].
3) Infrastructure and configuration: synthesize IaC changes (network, compute, IAM, secrets, and logging). IaC research emphasizes that these artifacts evolve with application code and should be tested with similar rigor [10], [13].
4) Policy evaluation and evidence: evaluate policy as code and controls as code to produce machine-verifiable evidence records. These records form Evidence nodes in the CIG and are linked to their Control nodes.
5) Deploy and observe: deploy using a progressive delivery strategy, observe telemetry, and automatically trigger rollback if risk thresholds are exceeded. SRE principles recommend explicit SLOs and automated responses to protect service reliability [8].

To keep decisions defensible, each stage should emit a signed decision record that captures: (a) which controls were evaluated, (b) policy decisions (allow/deny with rationale), (c) evidence artifacts and their hashes, and (d) the derived impact set (affected services and dependencies). In effect, the pipeline produces a compact "evidence bundle" for every deployment.

Policy-as-code enforcement works best when it is layered across the delivery lifecycle. In practice, teams apply the same guardrail intent at three points:

- Pre-merge: policy checks on pull requests for IaC and configuration changes, blocking high-risk changes early.
- Pre-deploy: environment-specific policy evaluation (e.g., staging vs production) where constraints differ.
- Runtime: admission control and drift detection to ensure deployed state remains compliant.

Sector constraints also shape what "allowed" means. In banking, change windows are often bounded by batch settlement, end-of-day close, and the staffing model for incident response [5]. In utilities, allowed windows may be coordinated with OT maintenance activities and supply-chain governance requirements aligned to NERC expectations [6]. The CIG captures these constraints through Window nodes so the rollout policy can reason about timing explicitly.

A. Controls-as-code implementation patterns

We distinguish three complementary patterns:

Pattern 1—Preventive guardrails: policy checks that block non-compliant changes before merge or deploy. Examples include denying public network exposure, enforcing encryption settings, or requiring approved base images.

Pattern 2—Detective guardrails: controls that generate alerts and tickets when drift or policy violations are detected at runtime. This is essential because not all violations can be prevented statically (e.g., credential misuse, unexpected data paths).

Pattern 3—Compensating controls: when preventive enforcement is not feasible (legacy systems, vendor-managed components), compensating controls produce evidence of oversight (approvals, monitoring coverage, incident response readiness).

These patterns produce different evidence types and different residual risks. By representing them explicitly in the CIG, the risk model can learn—over time—that a change relying only on compensating controls (for example, manual approval plus monitoring) tends to behave differently than one protected by preventive enforcement.

TABLE II. CONTROLS-AS-CODE EVIDENCE SOURCES ACROSS PIPELINE ARTIFACTS

Artifact Class	Representative Checks	Evidence Produced (examples)
IaC (network/IAM/compute)	static analysis; policy evaluation; drift checks	policy decision log; plan diff; signed approval
Container images	vulnerability scan; signature verification; provenance checks	scan report; signature attestation; SBOM hash
Config & secrets	linting; secret detection; access control checks	finding report; rotation evidence; access logs
Observability rules	coverage checks; alert routing validation; SLO guardrails	monitoring config diff; SLO budget state
Release strategy	rollout policy validation; change window validation	strategy decision record; window constraint proof

VI. PREDICTIVE RISK SCORING USING CIG + AIOPS SIGNALS

Our goal is to estimate change risk before a rollout reaches full traffic. That estimate is what release engineers and control owners need when they decide whether to proceed, slow down, or hold. We model risk as the product of probability and impact:

Risk(change) = P(incident | change, context) × Impact(change, context)

Probability captures how likely the change is to trigger an incident—reliability, security, or a control-relevant failure—during or shortly after deployment. Impact captures the expected severity in operational terms (SLO burn, affected transactions, recovery effort) and in compliance terms where applicable.

A. Feature construction

From each CIG snapshot, we derive three feature families:

1) Graph-structural features: dependency depth to critical services, number of impacted controls, concentration of changes around sensitive artifacts (e.g., IAM or segmentation), and evidence completeness (fraction of required controls producing passing evidence).

2) Change-centric features: churn and complexity metrics for code and IaC, configuration "code smells," and co-evolution signals. Empirical work shows that configuration artifacts can carry defect-prone patterns [12], and that infrastructure and application code often co-evolve in ways that matter for reliability [13].
3) Operational context features: anomaly scores from logs and metrics, current error budget state, and window-related context. For example, the same latency residual may be tolerable off-peak but unacceptable during a settlement cutover or a peak-load period.

To capture non-local interactions, we embed the CIG using message passing over a graph neural network (GNN). Graph convolutional models propagate information across dependencies so that an artifact embedding can reflect nearby controls, evidence, and impacted services—not just local properties [16].

Operational logs are often the earliest signal that a change is drifting into trouble, especially when the failure mode is multi-service or configuration-driven. In addition, unstructured text from incident narratives and change tickets can be mined for consistent risk cues; prior NLP work demonstrates how text analytics can convert large, noisy corpora into actionable signals [4]. We use log-sequence modeling to quantify deviation from normal behavior. DeepLog, for example, learns expected event sequences and flags anomalies that are useful for both detection and diagnosis [15].

We train a supervised model that consumes (i) the GNN-derived embedding, (ii) engineered features, and (iii) window context to predict incident probability and expected impact. Depending on organizational constraints, this can be implemented as a simple model on top of embeddings (easier to govern) or as an end-to-end graph model with task heads.

Because regulated environments demand defensible decisions (R1), the model must explain itself. We therefore attach explanations to predictions and summarize them in control-aligned language. SHAP provides a principled way to attribute a prediction to specific features (e.g., missing evidence for a control, deep dependency paths, elevated anomaly scores) [17].

B. Decision thresholds and calibration

Risk scores are calibrated using historical incident data. The pipeline defines operational thresholds such as:
- Low risk: proceed with standard canary and automated rollback.
- Medium risk: enforce additional pre-deploy verification and extended canary observation.
- High risk: require explicit approval, constrain to safe windows, or hold until risk is reduced.

Calibration must be revisited when topology, workloads, or control requirements change. The CIG makes these shifts visible—new dependencies, new controls, and new evidence types—so teams can trigger revalidation before drift silently degrades decision quality.

C. Labeling incidents and compliance breaches

For supervised learning, labels should cover both operational outcomes (availability, latency, data loss) and control-relevant outcomes (control breaches, audit findings, emergency changes). These signals are rarely uniform across organizations, so a tiered labeling scheme (no incident → rollback → SLO breach → major incident/compliance breach) tends to work well in practice.
High-severity incidents are rare, which creates class imbalance. Cost-sensitive losses, careful sampling, and severity-aware thresholds help. The CIG also helps: it encourages the model to learn reusable patterns (control gaps + dependency structure + context) instead of memorizing service names.
High-severity incidents are rare. The model should use techniques such as cost-sensitive loss functions and time-aware sampling. The CIG helps by providing structured context for each positive event, allowing the model to generalize from small numbers of incidents by learning control and dependency patterns rather than memorizing specific services.

D. Operationalizing explanations for governance

Explanations should be summarized into governance-relevant statements that can be reviewed quickly, such as: "risk increased because a segmentation control produced missing evidence" or "risk increased because the change impacts a high-tier settlement service during a constrained window." This bridges ML output with control language and supports cross-functional review boards.

VII. RISK-AWARE PROGRESSIVE DELIVERY UNDER COMPLIANCE WINDOWS

Progressive delivery is a practical way to limit blast radius by ramping exposure while watching telemetry [14]. In regulated pipelines, however, rollout strategy is also a governance decision: it must align with approved change windows, segregation-of-duty workflows, and operational staffing models. The intent is not to "slow down," but to make rollout choices that are defensible and repeatable.

A. Rollout strategy space

We consider common strategies:

1) Canary: route a small fraction of traffic to the new version, expand if healthy.
2) Blue-green: run two full environments, switch traffic via a controlled cutover.
3) Staged hold: deploy to production but hold activation (e.g., feature flags, dark launch) until a safer or authorized interval.
4) Emergency rollback: revert quickly if risk signals exceed thresholds.

Window constraints are first-class inputs. In the CIG they appear as Window nodes with constraint edges, so a deployment can be evaluated against both governance-driven compliance windows and operations-driven safety windows. For a given change, we derive the allowed interval set A from approvals, policy constraints, and sector operations. Banking examples include settlement and end-of-day close restrictions [5]; utility examples include peak-load and coordinated maintenance constraints shaped by reliability and supply-chain governance [6].

C. Objective function

We define an expected loss for strategy s at time t:

$$\text{Loss}(s, t) = P_{\text{incident}}(s, t) \times \text{Impact}(s, t) + \text{Cost}_{\text{controls}}(s) + \text{Cost}_{\text{delay}}(t)$$

where P_{incident} and Impact are derived from the risk model (Section VI) and adjusted for the mitigation strength of the strategy (e.g., blue-green may reduce impact but increase operational cost). $\text{Cost}_{\text{controls}}$ captures additional verification or staffing costs, and $\text{Cost}_{\text{delay}}$ captures business cost of postponing change. Rollout choice should be deterministic given inputs and should emit a decision record that auditors and incident reviewers can replay. We express rollout policy as code: rules parameterized by risk thresholds, evidence completeness, and window conditions. The rules are simple by design; the value comes from grounding them in the CIG's evidence chain and risk signals.

- If evidence completeness < threshold for a high-severity control, hold and require remediation.
- If risk is high and we are within a constrained window, stage the change but delay activation.
- If risk is medium and within an allowed window, proceed with canary and extended observation.

During rollout, feedback has to be fast and unambiguous. The pipeline continuously evaluates SLO signals and anomaly scores; if guardrails are breached, it triggers rollback and writes the outcome back into the CIG. This closes the learning loop and improves future calibration.

D. Joint optimization across portfolios

In large portfolios, "the safe window" is a shared resource. Multiple teams will compete for the same low-risk intervals (after settlement close, during maintenance). A portfolio scheduler can treat each change as a job with an expected loss and constraints, then schedule a set of changes that minimizes total expected loss while honoring staffing and window limits.

E. Human override and accountability

Automated recommendations should start as advisory. When engineers override a recommendation, the reason should be captured as evidence and linked to the CIG. Over time, this produces a governance-friendly record of when human judgment adds value and where the policy thresholds should be tuned.

F. Change freeze and emergency change handling

Change freezes and emergency changes are unavoidable in both sectors. The same framework supports emergency work by requiring stronger compensating evidence (incident commander sign-off, enhanced monitoring, and tighter rollout caps) and by constraining strategies (micro-canaries, rapid rollback automation). Capturing these decisions improves both audit traceability and post-incident learning.

VIII. CASE STUDIES: BANKING AND UTILITY PIPELINES

A. Banking use case: risk-aware change for a payments and settlement platform

Consider a bank delivering a change to a payments API that feeds fraud screening and downstream settlement processing. The change spans: (i) application code, (ii) an IaC update to an API gateway policy, and (iii) a policy-as-code rule that tightens authentication for a new endpoint. In many banks, the risk profile is dominated by settlement and end-of-day close windows where stability and auditability are prioritized, and supervisory guidance expects strong change management and monitoring for critical services [5].

In the CIG, the gateway IaC module links directly to access control, segmentation, and logging controls (mapped to relevant NIST control families [2]). Controls-as-code checks show the auth policy is compliant and logging is enabled, but the evidence chain flags an observability gap: the new rate-limiting rule has no validated alert routing, so detection may lag if it misbehaves. The risk model therefore elevates the score based on (a) dependency depth into settlement services, (b) incomplete evidence for monitoring coverage, and (c) elevated anomaly scores from staging load tests. SHAP explanations make the drivers explicit, which helps both the release engineer and the control owner review the decision [17].

The rollout policy selects a staged-hold strategy: deploy the change to production, but defer activation until the settlement window closes. In parallel, the pipeline inserts an additional controls-as-code check to validate alert routing and SLO guardrails for the new rule. Once evidence completeness is restored, the pipeline activates via an extended canary and enforces an automatic rollback threshold.

B. Utility use case: grid telemetry and OT-adjacent service change

Now consider a utility deploying a change to a grid telemetry ingestion service that processes SCADA-adjacent data and publishes derived signals to operations dashboards. The change includes (i) a new container image, (ii) IaC updates to message broker configuration, and (iii) a supply-chain control update requiring additional vendor attestations for a third-party library. Utilities often operate under peak-load constraints where rollback operations are expensive, and supply-chain plans and approvals must be demonstrably enforced [6].

The CIG links the container artifact to vulnerability scanning and provenance controls [3] and links the third-party dependency to supply-chain evidence requirements. Historical telemetry shows that similar broker configuration changes have produced message-lag anomalies during peak load. We attach log and metric anomaly scores to the impacted service nodes; DeepLog-style sequence models provide a practical way to quantify this deviation signal [15]. The risk model therefore predicts higher impact if the change is activated during the peak window.

The rollout policy recommends blue-green during a maintenance window (allowed and safer), enabling a clean cutover and rapid reversion if lag guardrails are violated. The decision record captures the window justification and the additional supply-chain evidence produced, improving audit readiness while reducing operational risk.

C. Cross-sector comparison

Across both domains, the CIG provides a shared abstraction. The control catalogs differ and the “critical windows” are not the same, but the decision mechanics—scope controls, generate evidence, predict impact, and choose a rollout strategy—transfer well. Finance tends to weight settlement integrity and customer transaction continuity; energy tends to weight telemetry integrity and OT-adjacent reliability constraints.

Lessons learned for regulated pipelines:

- Treat control scoping as data, not tribal knowledge: the pipeline should compute which controls apply from service tier and data classification, and record that scope for audit replay.
- Make evidence a first-class artifact: signed decision records (inputs, policy outcomes, evidence hashes, and rationale) shorten both audit preparation and post-incident reconstruction.
- Bias toward window-aware mitigations: staged holds and extended canaries work well around settlement cutovers, while blue-green during coordinated maintenance reduces operational risk near peak load.
- Close the loop: feed rollback outcomes and monitoring gaps back into the CIG so the model learns which control/evidence patterns actually predict failures.

TABLE III. EXAMPLES OF COMPLIANCE WINDOWS AND RISK CONTEXT IN FINANCE VS ENERGY

Dimension	Finance (banking)	Energy (utility)
Typical constrained windows	settlement/end-of-day close; high-volume periods	peak load; coordinated OT maintenance
High-impact artifacts	IAM, gateway policy, encryption/logging configs	broker/SCADA-adjacent configs; network segmentation; vendor deps
Dominant risk signals	latency/availability SLO burn; auth/logging drift	message lag; telemetry drop; supply-chain evidence gaps
Preferred mitigation patterns	staged hold + extended canary; fast rollback	blue-green during maintenance; strict pre-deploy evidence

IX. EVALUATION DESIGN

Evaluating predictive release risk in regulated environments has a practical constraint: the most useful datasets (change records, control evidence, and incident details) are often sensitive and cannot be published. For that reason, we frame evaluation as an internal empirical study that can be executed within a bank or utility and reported using aggregate, non-identifying results. The protocol focuses on three outcomes reviewers care about in regulated delivery: predictive accuracy, decision quality (expected loss reduction), and auditability of the evidence chain.

A. Dataset construction

For each deployment, collect: (i) artifact diffs (code, IaC, configuration, policy), (ii) controls-as-code results and evidence artifacts, (iii) change window context, and (iv) post-deploy outcomes (incidents, SLO breaches, rollbacks). Construct a CIG snapshot per deployment by linking changed artifacts to evaluated controls and produced evidence.

B. Baselines

We compare against common baselines:

1) Heuristic risk based on change size and service criticality.
2) Rule-based gating using only policy check pass/fail.
3) Non-graph ML model using engineered tabular features without graph embeddings.

C. Metrics

Probability prediction: AUROC, AUPRC, Brier score for calibration.

Impact prediction: mean absolute error on an impact proxy (e.g., SLO budget burn or incident severity class).

Decision quality: expected loss reduction under simulated rollout policies, and fraction of incidents prevented or mitigated.

Auditability: percentage of deployments with complete evidence chains for required controls, and time-to-audit reconstruction.

D. Ablation studies
Ablations quantify the value of each feature family: graph structure only, evidence only, AIOps only, and the full model. We also evaluate sensitivity to control catalog changes by re-mapping controls and observing degradation or robustness.

E. Online validation
Where feasible, an organization can run shadow predictions during a pilot period: the model generates risk scores and recommended strategies, but human release engineers make final decisions. Discrepancies and outcomes are logged to improve calibration and to build confidence before enforcing automated gating.

F. Reporting and reproducibility
To enable scientific comparability without leaking sensitive information, organizations can report aggregate metrics, bucketed by service tier or change type. They can also report de-identified graph statistics (node counts by type, control coverage distributions) that preserve structure while hiding identifiers. This supports internal reproducibility and cross-organization learning in regulated contexts.

G. Stress-testing with synthetic workloads
Where production data is constrained, teams can generate synthetic change sets and fault injection scenarios (e.g., configuration toggles, network policy perturbations) to validate that risk scores respond sensibly. Synthetic evaluation is particularly useful for rare, high-impact scenarios in which historical examples are scarce.

X. DISCUSSION
A. Governance and organizational integration
The CIG does not replace governance; it makes governance executable. Most regulated enterprises already maintain internal control catalogs mapped to standard frameworks [1], [2] and run formal approval processes for critical changes [5]. The gap is usually operational: evidence is scattered across tools and reconstructed under pressure. By formalizing control-to-artifact relationships and capturing a signed decision record for every pipeline run, the CIG reduces audit friction and improves the quality of post-incident review.

B. Controls-as-code maturity
Controls as code should be engineered like any other production dependency: versioned, reviewed, tested, and monitored. Research on configuration smells suggests that small, easy-to-miss patterns in configuration and IaC can correlate with defects [12]. In banking and utilities, that translates to a practical rule: do not stop at pass/fail guardrails—track maintainability signals, policy complexity, and repeated exception patterns because they often precede drift.

C. Data management and privacy
Telemetry and incident data can be sensitive (customer data in banking, operational reliability data in utilities). Training pipelines therefore need the same data governance as other regulated analytics: data minimization, access control, retention policies, and reviewable explanation outputs. Evidence artifacts should be hashed and signed, and access should follow least-privilege principles.

D. Model drift and continuous learning
Risk models drift as architecture, controls, and workloads evolve. The CIG exposes the main drift drivers—new dependencies, new control categories, and new evidence types—so teams can trigger retraining and revalidation deliberately. Continuous delivery research emphasizes measurement and feedback loops; this approach extends those loops to include compliance evidence and change-risk outcomes, not just deployment throughput [9], [14].

E. Operational safety

Automated gating should be introduced with the same caution used for other high-impact controls. Early on, the safest default is conservative: high predicted risk leads to a hold or an explicit approval step, not an automatic "go." As confidence grows, organizations can automate more decisions while preserving overrides, logging, and incident readiness aligned with SRE practice [8].

F. Practical adoption roadmap

Step 1: start with evidence standardization—ensure each pipeline run emits a decision record and hashes evidence artifacts.
Step 2: construct a minimal CIG containing artifacts, services, and a small set of high-value controls (e.g., access control, logging, segmentation).
Step 3: add AIOps signals and train a pilot risk model, running it in shadow mode.
Step 4: integrate risk-aware rollout policy as code, initially recommending but not enforcing.
Step 5: expand control coverage and automate gating for well-calibrated decision regions.
This phased adoption reduces organizational risk while preserving the ability to demonstrate continuous compliance and measurable improvement over time.

G. Limitations and boundary conditions

Graph-based modeling depends on reasonable dependency knowledge. Highly dynamic systems with opaque runtime calls will need additional service mapping and telemetry correlation before the CIG is complete enough for learning. In practice, teams can still start with partial graphs (critical services first) and expand coverage iteratively.

XI. THREATS TO VALIDITY

- Internal validity—Incident and rollback labels can be noisy or incomplete, and some rollbacks occur for business reasons unrelated to technical risk. Mitigation: standardize incident classification, link incidents to deployments via telemetry correlation, and capture override/rollback rationale as part of the signed decision record.
- Construct validity—Operational proxies such as SLO budget burn do not fully capture compliance impact. Mitigation: include control-breach and audit-finding outcomes where available, and keep operational impact and compliance impact as separate target dimensions.
- External validity—Tooling, architectures, and control catalogs vary widely across organizations. Mitigation: keep the CIG schema tool-agnostic, report how controls were mapped and scoped, and evaluate sensitivity to control-catalog changes.
- Conclusion validity—Observed improvements may depend on the maturity of controls as code and the fidelity of telemetry. Mitigation: run ablation studies to isolate the contribution of graph structure, evidence signals, and AIOps features, and report minimum viable signals for stable performance.

XII. CONCLUSION AND FUTURE WORK

We presented a Control-Impact Graph (CIG) that unifies pipeline artifacts, compliance controls, controls-as-code evidence, operational windows, and service outcomes. By combining this control-aware graph with AIOps signals and interpretable risk scoring, regulated teams can make rollout decisions that are both safer and easier to defend—especially when banking settlement cycles and utility peak-load constraints narrow the margin for error.

Future work includes (i) richer causal modeling to reduce correlation-driven false positives, (ii) more automated control mapping and coverage optimization, and (iii) sector-oriented benchmarking approaches that can be shared using de-identified, synthetic, or privacy-preserving data so research can progress without exposing regulated operational details.

APPENDIX A. EXAMPLE ROLLOUT POLICY-AS-CODE

The following pseudo-policy illustrates how risk scores, control evidence, and window constraints can be combined into deterministic rollout decisions. In practice, organizations encode similar logic in their chosen policy engines; the key requirement is that the decision record retains inputs and outcomes for auditability (R1).

```
package rollout

# Inputs: risk_score in [0,1], evidence_complete, window_allowed, window_safe
# Output: decision in {"canary", "blue_green", "hold", "deny"}

default decision = "hold"

decision = "deny" {
  evidence_complete == false
}

decision = "hold" {
  risk_score >= 0.7
  window_allowed == false
}

decision = "blue_green" {
  risk_score >= 0.7
  window_allowed == true
  window_safe == true
}

decision = "canary" {
  risk_score < 0.7
  window_allowed == true
}
```

APPENDIX B. CIG SNAPSHOT RECORD FIELDS

A minimal decision record (stored as structured data and linked into the CIG) can include: (1) change_id, (2) artifact_hashes, (3) impacted_services, (4) evaluated_controls, (5) evidence_artifact_hashes, (6) policy_decisions with rationale, (7) risk_score and explanation summary, (8) chosen_rollout_strategy, (9) window_context, and (10) post_deploy_outcomes (later attached). Standardizing these fields supports both auditability and learning.

Additionally, storing intermediate feature values (e.g., evidence completeness ratio, dependency depth) can simplify governance review and help investigators reproduce a model-driven decision during audits.

APPENDIX C. CONTROL CATEGORY TO PIPELINE CHECK MAPPING

Table IV provides an mapping between common control categories and pipeline checks/evidence. Organizations should adapt the categories and evidence requirements to their internal control catalog and sector obligations. The intent is to show how the same CIG abstraction can represent both finance and energy emphases without changing the delivery intent.

TABLE IV. CONTROL-TO-CHECK MAPPING FOR CONTROLS-AS-CODE

Control Category	Objective (example)	Pipeline Check	Finance Emphasis	Energy Emphasis
Access control	least privilege; SoD	IAM diff + policy eval + approval record	privileged access; transaction apps	operator access; OT boundary
Segmentation	restrict east-west paths	network/IaC policy lint + reachability test	data zone isolation	SCADA-adjacent segmentation
Logging/ monitoring	detect abnormal behavior	telemetry coverage + alert routing validation	fraud + settlement monitoring	grid telemetry integrity
Supply-chain risk	trusted vendors/artifacts	SBOM/provenance + vendor attestation evidence	third-party service risk	CIP-013 plan evidence
Vulnerability mgmt	reduce exploitable flaws	image/package scan + severity gating	internet-facing services	critical infrastructure services

REFERENCES:

[1] National Institute of Standards and Technology, "Framework for Improving Critical Infrastructure Cybersecurity," Version 1.1, Apr. 16, 2018. doi: 10.6028/NIST.CSWP.04162018.

[2] Joint Task Force, "Security and Privacy Controls for Information Systems and Organizations," NIST Special Publication 800-53, Revision 5, Sep. 2020. doi: 10.6028/NIST.SP.800-53r5.

[3] M. Souppaya, J. Morello, and K. Scarfone, "Application Container Security Guide," NIST Special Publication 800-190, Sep. 2017. doi: 10.6028/NIST.SP.800-190.

[4] A. Agade and S. Balpande, "Exploring the Non-Medical impacts of Covid-19 using Natural Language Processing," International Journal of Computer Applications, vol. 175, no. 36, pp. 16-23, Dec. 2020, doi: 10.5120/ijca2020920923.

[5] Federal Financial Institutions Examination Council (FFIEC), "Architecture, Infrastructure, and Operations," IT Examination Handbook, Jun. 2021.

[6] Federal Energy Regulatory Commission, "Supply Chain Risk Management Reliability Standards," Final Rule, Order No. 850, Oct. 18, 2018.

[7] PCI Security Standards Council, "Payment Card Industry Data Security Standard: Requirements and Security Assessment Procedures," Version 3.2.1, May 2018.

[8] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Site Reliability Engineering: How Google Runs Production Systems. Sebastopol, CA, USA: O'Reilly Media, 2016.

[9] N. Forsgren, J. Humble, and G. Kim, Accelerate: The Science of Lean Software and DevOps. Portland, OR, USA: IT Revolution, 2018.

[10] A. Rahman, R. Mahdavi-Hezaveh, and L. Williams, "A systematic mapping study of infrastructure as code research," Information and Software Technology, vol. 108, pp. 65-77, Apr. 2019. doi: 10.1016/j.infsof.2018.12.004.

[11] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer, "What is DevOps? A systematic mapping study on definitions and practices," in Proc. XP '16 Workshops, 2016. doi: 10.1145/2962695.2962707.

[12] T. Sharma, M. Fragkoulis, and D. Spinellis, "Does your configuration code smell?" in Proc. 13th Int. Conf. on Mining Software Repositories (MSR), 2016, pp. 189-200. doi: 10.1145/2901739.2901761.

[13] Y. Jiang and B. Adams, "Co-evolution of infrastructure and source code - an empirical study," in Proc. 12th Working Conf. on Mining Software Repositories (MSR), 2015, pp. 45-55. doi: 10.1109/MSR.2015.12.
[14] C. Parnin et al., "The top 10 adages in continuous deployment," IEEE Software, vol. 34, no. 3, pp. 86-95, May/Jun. 2017.
[15] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in Proc. ACM Conf. on Computer and Communications Security (CCS), 2017, pp. 1285-1298. doi: 10.1145/3133956.3134015.
[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in Proc. Int. Conf. on Learning Representations (ICLR), 2017.
[17] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 4765-4774.