

CORA: Config Risk Analyzer – Predicting Risk of Configuration Changes at Scale

Weiyan Sun ✉ 

Meta Platforms, Inc., Bellevue, WA, USA

Brian Ellis ✉ 

Meta Platforms, Inc., Bellevue, WA, USA

Madigan Kim ✉ 

Meta Platforms, Inc., New York, NY, USA

Gursharan Singh ✉ 

Meta Platforms, Inc., Menlo Park, CA, USA

Siri Uppalapati ✉ 

Meta Platforms, Inc., Sunnyvale, CA, USA

Audris Mockus ✉ 

Meta Platforms, Inc., New York, NY, USA

Jun Ge ✉ 

Meta Platforms, Inc., Menlo Park, CA, USA

Sahil Kumar ✉ 

Meta Platforms, Inc., Bellevue, WA, USA

Matt Steiner ✉ 

Meta Platforms, Inc., Menlo Park, CA, USA

Nachiappan Nagappan ✉ 

Meta Platforms, Inc., Bellevue, WA, USA

Abstract

Configuration changes – modifications to feature flags and service parameters – are important to operating services, but they can cause severe outages (SEVs) with substantial service disruption. While code diff risk prediction is well studied, configuration diff risk has received little attention. The code-oriented Diff Risk Score (DRS) model provides limited discriminatory power for config diffs because config risk stems not from code complexity but from the importance, blast radius, and operational readiness of the affected services. We describe the development of CORA (COnfig Risk Analyzer), a dedicated risk model for config diffs. A key insight is that the config-path-to-service mapping bridges from what changed to which services are affected, enabling risk assessment in terms of service criticality rather than code properties.

CORA evolved through three iterations. A logistic regression model with config-specific features achieved an 11% improvement in recall at 5% gating over the DRS model on config diffs. Used for a freeze period, it reduced config gating while outage impact decreased and config diff landing volume increased 91.4%. A unified LightGBM model trained on Meta-wide data achieved a 25.95% improvement in recall at 10% gating, with organization-level improvements ranging from 15% to 66%. An enhanced model with enriched service features and SEV-severity-aware training (having also explored Bayesian hyperparameter optimization) achieved a 22% improvement in recall at 15% gating, intercepting meaningful additional high-severity outage impact in backtesting. To our knowledge, CORA is the first system to apply predictive risk modeling specifically to configuration changes.

2012 ACM Subject Classification Software and its engineering → Software configuration management and version control systems; Software and its engineering → Risk management

Keywords and phrases Configuration changes, risk prediction, defect prediction, code freeze, software supply chain, gradient boosting, service reliability

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.86

Category Software Engineering in Practice Track Paper

Supplementary Material *Software*: <https://doi.org/10.5281/zenodo.21719068>

1 Introduction

Changes to software configuration are necessary to operate, tune, and optimize services, but these changes may introduce defects that lead to outages (SEVs). Researchers have studied the risk of source code changes extensively in the defect prediction literature [30, 22] and in industrial deployments of diff risk scoring [17, 1], but the risk of *configuration* changes



© Weiyan Sun, Audris Mockus, Brian Ellis, Jun Ge, Madigan Kim, Sahil Kumar, Gursharan Singh, Matt Steiner, Siri Uppalapati, and Nachiappan Nagappan;
licensed under Creative Commons License CC-BY 4.0

20th International Symposium on Empirical Software Engineering and Measurement (ESEM 2026).
Editors: Robert Feldt, Maria Paasivaara, Daniel Mendez, Stefan Wagner, and Marvin Muñoz Barón;
Article No. 86; pp. 86:1–86:22



Leibniz International Proceedings in Informatics
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

remains largely unexplored. Config changes (modifications to Sitevar, Gatekeeper, JustKnob, and Configurator artifacts [27]) constitute a meaningful fraction of all diffs and cause some of the more disruptive SEVs, including multiple SEV1s (the highest severity level, reserved for outages with large-scale user impact).

Mockus et al. [17] developed the Diff Risk Score (DRS) system to predict the risk of code diffs causing SEVs. DRS uses logistic regression models trained on historical data linking diffs to the SEVs they caused, with features drawn from code complexity, software supply chain centrality, and author expertise. The system gates diffs that exceed a risk threshold during code freeze periods, with thresholds tunable to different gating levels: 5% for weekends, 10% for yellow (medium-risk) zones, and 50% for red (high-risk) zones. Abreu et al. [1] extended DRS with LLM-based approaches that further improved recall at these gating levels.

However, these code-oriented features provide little discriminatory power for config diffs, whose risk stems not from logical errors in source code but from the importance, number, and interactions among the services affected by the configuration change. During the Freeze 1 code freeze, the DRS model missed several config-related SEV1s, prompting undifferentiated config gating that imposed a burden on development teams. As a concrete example, one Freeze 1 SEV1 was triggered by a small change to a feature-flag config path consumed by a high-criticality service: because the diff modified only one artifact and carried no code-complexity signal, the code-oriented DRS scored it as low risk and did not flag it, even though the affected service’s criticality and broad blast radius made the change inherently risky. The challenge in config risk prediction is mapping from *what changed* (the config path) to *who is affected* (the services) and assessing the criticality, blast radius, and operational readiness of those affected services.

In this paper, we report on a longitudinal industrial case study following the iterative action research methodology [25, 4] as applied in [17]: we develop and refine the model across three cycles, with each iteration motivated by findings from the previous one. The study is participatory in the action-research sense rather than a purely retrospective case study: a cross-functional team of researchers and practitioners (reliability engineers and config-safety experts) jointly diagnosed each problem, decided on and implemented the interventions, deployed them into production, and reflected on the outcomes to plan the next cycle (Table 1). The evaluation combines quasi-experimental backtesting (retrospective simulation on historical data with chronological splits) for model-vs-model comparisons (RQs 1, 3, 4) with an observational before/after comparison (RQ 2) and a partial ablation isolating the contributions of individual enhancements (RQ 4).

We describe **CORA (COnfig Risk Analyzer)**, a dedicated risk model for config diffs. The key insight behind CORA is that the config-path-to-service mapping provides a natural bridge from the diff to the service dependency graph, enabling the model to reason about risk in terms of service criticality, blast radius, and testing maturity rather than code complexity.

CORA has evolved through three iterations, progressively improving both model architecture and scope (here and throughout, Org_A through Org_E denote large, autonomous product organizations within Meta, anonymized for confidentiality):

- A logistic regression model with config-specific features, used for Org_A before Freeze 2, achieving an 11% improvement in recall at 5% gating over the code-oriented DRS model on config diffs;
- A unified gradient boosting tree (LightGBM) model trained on Meta-wide data, achieving a 25.95% improvement in recall at 10% gating; and
- An enhanced model with enriched service features and SEV-level-aware training (with Bayesian hyperparameter optimization also explored), achieving a 22% improvement in recall at 15% gating with additional outage impact intercepted in backtesting.

Contributions. We build on and extend the DRS work [17, 1] along four dimensions. First, we address the different risk profile of config changes, where risk derives from service-level properties rather than code logic, using the config-path-to-service mapping as the enabling structural insight. Second, we demonstrate that tree-based models outperform logistic regression for config diff risk, complementing the LLM-based improvements shown for code diffs in [1]. Third, we introduce SEV-severity-weighted training that differentiates the learning signal by outage severity. Fourth, we extend the evaluation framework beyond recall to include outage impact intercepted, directly quantifying the operational benefit of improved gating.

1.1 Research Questions

We address the following research questions, organized around the progressive development of CORA:

- RQ 1. *Config-Specific Risk Model.*** Can a dedicated config diff risk model outperform the code-oriented DRS model on config diffs?
- RQ 2. *Impact.*** How effective was CORA during the Freeze 2 risk management period?
- RQ 3. *Unified Tree-Based Model.*** Can a gradient boosting tree model trained on Meta-wide data outperform the logistic regression model for config diff risk prediction?
- RQ 4. *Enhanced Model.*** Can enriched service features, SEV-level-aware training, and hyperparameter optimization further improve the unified tree model?

The remainder of this paper is organized as follows. Section 2 provides background on code freeze practices and risk modeling. Section 3 describes the methodology, data collection, features, models, and evaluation approach. Section 4 presents the results for each research question. Section 5 discusses implications for developers. Section 6 addresses threats to validity. Section 7 concludes.

2 Background and Related Work

Code freeze and software risk prediction. Code freezes (temporary restrictions on changes) protect systems during high-risk periods [28, 24, 2]. Organizations apply code freezes during periods of high traffic or reduced engineering availability, restricting individual diffs based on predicted risk rather than blanket prohibitions [17]. Freeze periods are organized into tiered zones with progressively stricter gating: *weekend freezes* gate the top 5% riskiest diffs, *yellow zones* (medium-risk periods) gate 10–15%, and *red zones* (highest-risk periods) gate 45–50%. A gated diff must receive explicit approval before landing. The gating thresholds are tunable, allowing the balancing of productivity against risk tolerance for each zone.

Statistical software risk models predict defects at the file level [18, 20, 8] or at the change level (just-in-time prediction) [19, 10, 30, 22]. Common predictors include code churn, cyclomatic complexity, authorship patterns, and prior defect history. A recurring challenge is operationalizing “defect”: many studies rely on noisy SZZ heuristics [29, 6]. In our setting, the incident response team manually determines each SEV’s trigger diff, providing high-quality labeled data [17].

The software supply chain framework [15, 16] captures risk arising from dependencies among code artifacts, developers, and organizations. Katz centrality [11] propagates importance through the dependency network; both DRS and CORA use it. For defect prediction, tree-based models such as LightGBM [12] show strong results [13, 30], and LLM-based approaches outperform logistic regression on code diffs [1].

Risk prediction: DRS. The DRS system is described in two prior publications. Mockus et al. [17] introduced risk models, progressing from expert-selected file paths to diff-level logistic regression, to selectively gate risky diffs during code freezes. The `DiffModel` uses features from defect history, interdependencies, coordination, expertise, and file complexity. At 27.3% gating, it captures 60.8% of SEVs. During a prior freeze period, only 16% of diffs were gated while landed diffs increased 42% and SEVs decreased 52%.

Abreu et al. [1] extended DRS with LLM models. A risk-aligned iDiffLlama-13B captures $1.40\times$, $1.52\times$, and $1.05\times$ as many SEVs as logistic regression at 5%, 10%, and 50% gating. CORA builds on this DRS infrastructure but addresses the different domain of config diffs, where code content features are unavailable and service-level properties drive risk.

Configuration change management. Despite the impact of configuration changes on systems, the research literature on config change risk prediction is thin. Configuration testing [21] focuses on verifying software behavior under different configurations, which is orthogonal to predicting the risk of a configuration *change*. Several industrial reports document config changes as a common source of outages, but none propose predictive models. To our knowledge, CORA is the first system that applies predictive risk modeling specifically to configuration changes.

3 Methodology and Data

In this section, we describe how we extend the DRS code diff risk modeling framework to config diffs. We first discuss the differences between config and code changes that motivated this work. We then describe the data collection, the features specific to config changes, the evolution of modeling approaches from logistic regression to gradient boosting trees, the feature selection process, the SEV-severity-aware training, the explainability methods, and the outcome measures we use to evaluate model performance.

3.1 Why Config Diffs Require a Dedicated Risk Model

The DRS model [17, 1] predicts the risk of code diffs causing SEVs (Section 2). While config diffs go through the same diff-based review and landing process as code diffs, their risk profile differs. Code diff risk derives from logical errors in source code, and its predictors are drawn from file complexity, authorship networks, and code interdependencies (see Tables 1 and 2 in [17]). Config changes, however, rarely contain logical errors in the traditional sense. Instead, the risk of a config change derives from the *importance*, *number* (blast radius), and *interactions* among affected services (Section 1).

This motivates a dedicated config risk model, CORA, that runs concurrently with the code-focused DRS model.

3.2 Discovering the Context

Following the participatory action research approach [25, 4] as applied in [17], we initiated the CORA effort after the Freeze 1 incident where the DRS model missed several config-related SEVs (e.g., S1 and S2, anonymized identifiers of two specific config-triggered incidents, not to be confused with severity levels). A cross-functional V-team (a virtual, cross-organizational working team) of researchers, reliability engineers, and config safety experts identified the config-path-to-service mapping as the main structural insight, enabling the model to reason about risk in terms of a service’s criticality, operational readiness, and maturity rather

■ **Table 1** Action research cycles in CORA development.

Diagnose	Plan & Act	Evaluate	Reflect
DRS misses config SEVs (Freeze 1) Logistic regression limits; org-specific models	Config features + GLM Unified GBT on Meta-wide data	RQ 1 backtest, RQ 2 deploy RQ 3 backtest	GLM cannot capture non-linear interactions Default GBT beats tuned GLM; feature leakage found
Default hyperparameters sub-optimal	Bayesian tuning, SEV-aware weights, enriched features	RQ 4 backtest, Freeze 3 deploy	Simultaneous precision + recall gains

than file-level churn. The team comprised two researchers who designed the models and evaluation, three ML engineers who implemented and deployed the models, and domain experts from each organization who provided feature candidates, validated risk interpretations, and resolved conflicts between model accuracy and deployment constraints (e.g., feature latency, explainability requirements). Table 1 maps each iteration to the canonical action research phases.

3.3 Data Collection

Consistent with the data collection procedure described in [17], we collect data from multiple sources and link them via diff identifiers and timestamps.

Config diff data. We obtain config diffs from Phabricator [23], including creation and landing times, affected config paths, author, reviewers, and test plans. We identify a config diff by the type of configuration artifact it modifies: Sitevar, Gatekeeper (GK), JustKnob (JK), or Configurator [27].

SEV data. As described in [17], a system tracks **every** SEV across the organization and records its trigger, which, in the case of config-related outages, is a config diff. The incident response team manually determines the trigger link from SEV to diff, providing labeled data that is rarely available in published research [19]. SEVs fall into four severity levels: SEV1 (highest user impact), SEV2, SEV3, and SEV4.

Service metadata. Config diffs, unlike code diffs, require service-level data. We collect service metadata from two primary source tables:

- **Service registry** [26]: provides service criticality tier, number of hosts, RAM allocation, service dependencies (used-by and uses relationships), and resource governance configurations;
- **Service maturity metrics:** provides integration testing [5] coverage and false positive rates, canary false positive rates, staged rollout coverage, capacity stress testing coverage, and deployment pipeline metadata. These correspond to the “Service testing,” “Service health,” and “Service metadata” feature categories in Table 3.

The config-path-to-service mapping links each config path modified by a diff to the set of services that consume that configuration. This is a many-to-many relationship: a single config path may affect multiple services, and a single service may consume multiple config paths.

■ **Table 2** Features for CORA inherited or adapted from the DRS DiffModel [17]. The notation follows [17]: d denotes a diff, f a file, d_f indicates diff d modifies file f , d_{sev} indicates diff d caused a SEV, a is the author, $cent(\cdot)$ is Katz centrality. All features are computed at the diff level; features that depend on files or services are aggregated (max/sum) over all files modified by or services affected by d .

Category	Measure	Description	Direction
Defects	$weighted_caused_sev(d)$	Weighted sum of prior SEV severity for files modified by the diff (Equation 1). Decays over time. Replaces binary had_sev .	Positive
Interdependencies	$centrality(d)$	Maximum Katz centrality of the config files in the diff, computed on the co-change, authorship, and call-graph network	Positive
	$num_of_files(d)$	Number of config files modified by the diff	Positive
Coordination	$log_diff_author_count(d)$	log of the number of distinct authors who have previously modified the config files in the diff	Positive
	$has_same_team_approver(d)$	Whether the diff has an approver from the same team group as the author	Negative
Expertise	$is_original_author(d)$	Whether the diff author is the original creator of the config file	Negative
Diff	$substantial_line_count_ratio(d)$	Ratio of substantial (non-whitespace, non-comment) lines changed to total file size	Positive
Codebase	$owning_pillars(d)$	Organizational pillar that owns the config path	Positive

Supply chain data. We retain the supply chain network data from the DRS model: co-change networks, authorship networks, and call-graph centrality, computed as described in [17]. These features remain relevant for config diffs because the config files reside in the same monorepo and the same population of engineers author and review them.

We define the following notation used throughout. Let d denote a config diff, f a config file/path modified by d , s a service affected by f , and d_{sev} an indicator that diff d caused a SEV. We write d_f to indicate that diff d modifies config path f , and $f \rightarrow s$ to indicate that config path f is consumed by service s . The set of services affected by a diff d is then $S(d) = \{s : \exists f, d_f \wedge f \rightarrow s\}$.

3.4 Model Features

At a high level, we follow the same feature selection procedure as in [17]: for each conceptual dimension, we select candidate measures from domain knowledge and prior literature, investigate correlations among them, and retain measures that are both predictive and easy to interpret. The main difference is that CORA introduces an entirely new dimension of features, *service-level properties*, that have no analogue in the code diff risk literature.

The features fall into six categories. Tables 2 and 3 show the full set of features used in the CORA models.

DRS-inherited features. Several features from the DRS DiffModel [17] remain relevant for config diffs. The most predictive is SEV history. Unlike the binary had_sev in the original DRS model, CORA uses a *weighted* and *decayed* measure of prior SEV involvement. The weight reflects SEV severity (SEV1=10, SEV2=3, SEV3=2, SEV4=1), and the temporal decay reduces the influence of older SEVs. This captures the intuition that a config path recently involved in a severe outage is much riskier than one involved in a minor incident years ago.

Supply chain centrality [11, 15] captures structural importance: the Katz centrality of a config file in the co-change/authorship/call-graph network reflects how central it is. Diffusion measures (number of files), coordination measures (number of prior authors, same-team approver), and expertise measures (original authorship) transfer directly from the DRS model.

Config-specific features: service criticality and blast radius. A distinctive characteristic of CORA is its explicit mapping from config paths to the services they govern. This mapping lets the model reason about risk in the context of what will be *affected* rather than what was

■ **Table 3** Config-specific features introduced in CORA. These features have no analogue in the code-oriented DRS model. d denotes a diff, f a config path, s a service, $S(d) = \{s : \exists f, d_f \wedge f \rightarrow s\}$ is the set of services affected by diff d . Max/sum aggregations are over $S(d)$.

Category	Measure	Description	Direction
Service criticality	$max_hosts(d)$	Maximum number of hosts serving any affected service $s \in S(d)$ in the past 24 hours	Positive
	$tw_median_ram(d)$	Median RAM allocated to the most resource-intensive affected service	Positive
	$is_tier1(d)$	Whether any affected service is classified as Tier 1 (highest priority)	Positive
Blast radius	$uses_service_count(d)$	Maximum number of upstream services consumed by any affected service $s \in S(d)$	Positive
	$used_by_service_count(d)$	Maximum number of downstream services that depend on any affected service $s \in S(d)$	Positive
Config properties	$has_materialized_JSON(d)$	Whether the config change has materialized JSON output (indicator of change type)	Positive
	$num_non_generated_files(d)$	Number of non-generated (human-authored) config files in the diff. Replaces raw num_of_files based on user feedback	Positive
Service testing	$servicelab_coverage(d)$	Maximum integration testing coverage percentage across affected services	Negative
	$servicelab_false_positive(d)$	Maximum integration testing false positive rate across affected services. High false positive rates indicate unreliable safety nets	Positive
Service health	$tumbleweed_coverage(d)$	Staged rollout coverage breadth across affected services	Negative
	$canary_false_positive(d)$	Canary test false positive rate for the most impacted service	Positive
Service metadata	$service_primary_conveyors(d)$	Number of primary pipelines [9] for affected services, indicating process complexity	Positive
	$tw_uses_entitlement(d)$	Whether any affected service uses resource governance	Positive
	$is_kfs_covered(d)$	Whether the affected service has monthly capacity stress testing coverage	Negative
	$is_ICSP_service(d)$	Whether the service is classified as an infrastructure service	Positive

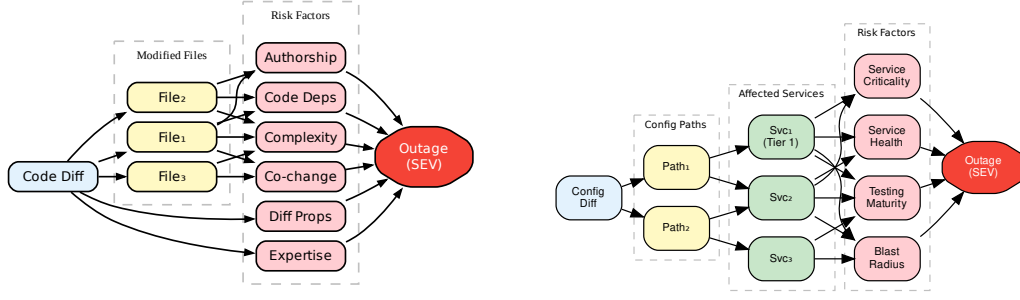
changed. When a config diff touches a path consumed by a high-criticality, widely depended-upon service, CORA can weigh that risk against the service’s operational readiness while reducing noise on changes to low-impact services.

Service criticality comprises: the number of hosts (max_hosts), indicating the scale at which the service operates; the allocated RAM (tw_median_ram), proxying computational importance; and the service tier classification (is_tier1), reflecting the assessed level of criticality.

The *service dependency graph* captures blast radius. The number of upstream services ($uses_service_count$) measures how many services the affected service consumes; the number of downstream services ($used_by_service_count$) measures how many other services depend on the affected service’s outputs. A config change to a service with many downstream dependents has higher blast radius.

Figure 1 illustrates the config diff supply chain. Unlike the code diff supply chain (Figure 1 in [17]), where risk flows through code interdependencies, the config supply chain routes risk through the config-path-to-service mapping and the service dependency graph.

Service testing, health, and config properties. Unlike code diffs where researchers measure test coverage at the file level, config diffs require service-level testing measures: integration testing coverage and false positive rate capture the reliability of safety nets, while staged



(a) Code diff: risk via dependencies, co-changes, authorship, expertise, diff properties

(b) Config diff: risk via config-path-to-service mapping, criticality, blast radius, testing

■ **Figure 1** Comparison of risk propagation for code diffs vs. config diffs. In (a), risk derives from file-level code properties and supply chain networks. In (b), risk derives from the services affected by the config change.

rollout coverage and canary false positive rate reflect broader service health (Table 3). Two config properties refined through user feedback, *has_materialized_JSON* (distinguishing production from non-production changes) and *num_non_generated_files* (excluding auto-generated artifacts), improve precision by reducing false positives.

3.5 Models

We describe the three model architectures we evaluate for CORA, corresponding to the evolutionary progression to the enhanced model.

Logistic Regression (CORA-GLM). The initial CORA model uses logistic regression, consistent with the DRS model [17, 1]. The same practical considerations motivate this choice: simplicity of fitting, low latency prediction (a single vector multiplication of fitted coefficients and predictors), and interpretability via coefficient signs and magnitudes. We $\log_{1p}$ -transform all continuous variables due to their skewed distributions. As in [17], we exclude measures that do not meaningfully contribute to explaining the variance (p-value < .1 or a difference in deviance under 2.6 for the likelihood ratio test).

The CORA-GLM model uses 33 features, including base features from Tables 2 and 3, interaction terms (e.g., SEV history $\times$ config type, materialized JSON $\times$ JustKnob type), log-transforms, and additional indicator variables for service properties (programming language, monitoring posture, automation state, capacity category), config artifact types, and review process quality (approver count, test plan length). Config paths known to be low-risk are identified via a *in_config_safe_paths* feature. Tables 2 and 3 list the base features from which the full set of 33 production features are derived; the remaining features are interaction terms, log-transforms, and indicator variables constructed from these base measures.

We initially train separate CORA-GLM models for Org_A and Org_E on organization-specific data to account for different config change patterns and SEV rates.

Gradient Boosting Trees (CORA-GBT). While logistic regression is simple and interpretable, it cannot capture non-linear relationships between features and risk. This limits config diff risk prediction, where interactions between features matter. For example, a config change to

a Tier 1 service may be low risk if that service has high integration testing coverage, but high risk if coverage is low. Logistic regression cannot model this interaction without explicit interaction terms, which are cumbersome to enumerate and evaluate.

We switch to Gradient Boosting Trees (GBT), specifically LightGBM [12], for the unified CORA model. GBTs naturally capture non-linear feature interactions, tolerate outliers in service metadata, and reduce the need for manual feature engineering and correlation management [7].

We train the CORA-GBT model on *Meta-wide* config diff data, unifying the previously separate Org_A and Org_E models. This increases the training data volume and diversity, allowing the model to learn patterns common across organizations while retaining the ability to differentiate organization-specific risk profiles via the *owning_pillars* feature. After backward feature elimination (Section 3.6), the final model uses 30 features. The final feature count is lower than the 33 features in CORA-GLM because GBTs learn interactions implicitly through tree splits, eliminating the need for the explicit interaction terms and indicator variables that accounted for roughly a third of the GLM feature set.

Enhanced GBT (CORA-GBT+). The enhanced model extends CORA-GBT with three methodological improvements, presented here in decreasing order of their measured contribution (Section 4.4): enriched service features, SEV-severity-aware training, and hyperparameter optimization.

Enriched service features. The enhanced model expands the feature set with three additional groups of signals, informed by deeper collaboration with config safety SMEs and operational experience:

- **Service metadata:** deployment pipelines [9] (*service_primary_conveyors*), resource governance (*tw_uses_entitlement*), capacity stress testing coverage (*is_kfs_covered*), and infrastructure service classification (*is_ICSP_service*).
- **Service healthiness:** staged rollout coverage and canary false positive rate, reflecting the breadth and reliability of safety nets beyond integration testing.
- **User-feedback-informed features:** replacing raw “number of files” with “number of non-generated files” (*num_non_generated_files*), discounting auto-generated artifacts that do not meaningfully increase risk.

SEV-level-aware training. We introduce tiered sample weights by SEV severity. Previously, training used a binary label (0/1) with weight = 1 for every sample, treating all SEVs as equally important. This dilutes the learning signal from more severe incidents. The enhanced model assigns:

$$w(d) = \begin{cases} 10 & \text{if } d \text{ caused a SEV1} \\ 3 & \text{if } d \text{ caused a SEV2} \\ 2 & \text{if } d \text{ caused a SEV3} \\ 1 & \text{if } d \text{ caused a SEV4 or no SEV} \end{cases} \quad (1)$$

This focuses the gradient boosting loss function on highest-impact events (SEV1s, which may cause large service disruption) while preserving calibration for lower-severity cases. The weights approximate average relative operational losses across severity levels, derived from historical incident data: SEV1s cause roughly an order of magnitude more disruption than SEV4s, SEV2s roughly three times, and SEV3s roughly twice. We treat these weights as fixed *a priori* rather than as tunable hyperparameters: they are set from these historical

operational-loss ratios, not selected to maximize validation accuracy, so we do not perform a hyperparameter sweep over them. The dominant lever is the up-weighting of SEV1s relative to the SEV3/SEV4 majority; what matters is the ordinal structure ($w_{\text{SEV1}} > w_{\text{SEV2}} > w_{\text{SEV3}} > w_{\text{SEV4}}$) rather than the exact magnitudes, and moderate perturbations (e.g., an SEV1 weight of 5 or 15) preserve the rank ordering of gradient contributions and thus the ranked risk list that gating operates on. Because gating decisions depend only on this ranking, we therefore do not expect the results to be sensitive to the precise weight magnitudes; a full factorial sweep over the weight vector remains future work. They are deliberately the same as those in the *weighted_caused_sev* feature (Table 2), though the two serve different purposes: the feature weights aggregate historical SEV severity for a config path, while the training weights up-weight the gradient contribution of high-severity samples during model fitting.

Hyperparameter optimization. The initial CORA-GBT model uses default LightGBM hyperparameters. For the enhanced model, we additionally explored Bayesian hyperparameter optimization over the space controlling the bias-variance tradeoff:

- *learning_rate* and *n_estimators* govern shrinkage and boosting depth;
- *num_leaves*, *max_depth*, and *min_data_in_leaf* set tree complexity;
- *feature_fraction* and *bagging_fraction* control stochastic sampling; and
- λ_{L1} and λ_{L2} affect regularization strength.

We use Bayesian optimization via Optuna [3] with a Tree-structured Parzen Estimator (TPE) sampler. Each trial samples a candidate configuration from the search space (using *suggest_int*, *suggest_float*, or *suggest_categorical* depending on parameter type) and evaluates it with LightGBM’s built-in stratified k -fold cross-validation (`lgb.cv`, $k=5$), scored by binary log-loss on the held-out folds. The TPE surrogate model learns from completed trials to prioritize promising regions of the search space rather than exploring exhaustively. We run 50 trials, sufficient for the nine-dimensional parameter space to converge while remaining tractable for the training data volume. Because the log-loss objective is not aligned with recall at a fixed gating percentage, this tuning did not improve recall at the operating gating levels in our reconstruction (Section 4.4); the deployed CORA-GBT+ therefore retains the default tree depth.

3.6 Feature Selection

For CORA-GLM, we select candidate features based on domain knowledge, examine correlations among features (we do not include features with $r > 0.7$ together), and exclude features that do not improve the likelihood ratio ($p < .1$, deviance difference < 2.6). For CORA-GBT, we expand the feature pool to 85 candidates. We use backward feature elimination, iteratively removing the least important feature by LightGBM’s split-based importance and re-evaluating on the validation set, converging at 30 features. We check all features for potential leakage through systematic offline/online consistency checks, and remove any feature whose value at prediction time could differ from its value at training time.

3.7 Explainability

Practical deployment requires that engineers understand *why* their diff was gated. For CORA-GLM, we infer feature importance through coefficient signs and magnitudes. For GBT models, we employ SHAP values [14] for per-prediction explanations and Partial Dependence Plots (PDP) for internal analysis of non-linear feature effects. We remap raw model scores to a 0–100 percentile scale for intuitive interpretation.

3.8 Evaluation Methodology

We evaluate all models using time-based data splits to prevent data leakage, consistent with [17, 1]. Table 4 shows the data splits for the two GBT model versions. Data retention policies limit the availability of historical feature values, so we cannot retroactively evaluate later models on earlier test sets, nor score earlier models on later data with features that did not yet exist. Each RQ therefore uses the test set and feature set available at the time of that iteration; cross-RQ comparisons of relative improvements reflect changes in model, features, training data, and test period simultaneously. Only the end-to-end bootstrap comparison (Table 11) evaluates both endpoints on the same test set. We consider absolute recall values sensitive and do not report them; all comparisons use relative improvements. To give a sense of scale: the baseline CORA-GLM captures fewer than one in four SEV-causing diffs at 10% gating and fewer than one in six at 5% gating, reflecting the difficulty of the prediction task at a 0.28% base rate. For CORA-GBT, we use a separate evaluation set for feature selection and early stopping. For CORA-GBT+, we fold the evaluation set into the training window to maximize recency, and run the hyperparameter search via k -fold cross-validation on the training set (Section 3.5) rather than a held-out evaluation split.

■ **Table 4** Time-based data splits for CORA model training and evaluation.

	Duration (CORA-GBT)	Duration (CORA-GBT+)	SEV rate
Train	18 months	33 months	0.283%
Eval	4 months	—	0.288%
Test	9 months	7 months	0.261%

We train the CORA-GBT model on 450,492 config diff samples with a SEV rate of 0.283%. The CORA-GBT+ training window extends further (Table 4), approximately doubling the training data relative to CORA-GBT. SEV-causing diffs account for fewer than 0.3% of all config diffs; the CORA-GBT models use *sample weighting* rather than resampling to address this imbalance, with severity-differentiated weights in the enhanced model (Equation 1). We evaluate models on both Org_A-specific and Meta-wide config diffs.

The primary outcome measure is the fraction of SEV-causing diffs captured at a given level of gating:

$$\text{CapturedSEVs}(g) = \frac{|\{d : d_{sev} \wedge r(d) \geq p_g\}|}{|\{d : d_{sev}\}|} \quad (2)$$

where $r(d)$ is the model’s predicted risk for diff d and p_g is the threshold such that a fraction g of all diffs exceed it. We evaluate at gating levels from 5% (weekend freeze) to 50% (red zone). At a SEV rate of 0.28%, gating 5% of diffs yields a precision of roughly 1% by construction; the gating percentage is a management decision that balances the number of engineers affected against the fraction of SEVs intercepted, not a model-tunable parameter. CORA also introduces an outage-impact measure that quantifies the operational severity of the SEV-causing diffs a model would gate:

$$\text{ImpactCaptured}(g) = \sum_{\{d : d_{sev} \wedge \text{impact}(d) > 0 \wedge r(d) \geq p_g\}} \text{impact}(d) \quad (3)$$

where $\text{impact}(d)$ is the operational loss attributed to the SEV caused by diff d (zero if the SEV caused no measurable loss). We use “captured” (and, in the results, “intercepted”) deliberately in place of “prevented”: in backtesting this measure sums the historical impact of SEV-causing diffs that the model *would have flagged* for review at gating level g . It does

not establish that gating those diffs would have averted the corresponding outages, since a gated diff may still land after approval, be resubmitted, or have its underlying issue surface through another change. It quantifies the severity-weighted risk surfaced by the model, not a proven reduction in outages.

Statistical comparison methodology. When comparing two models (e.g., CORA-GLM vs. DRS, or CORA-GBT vs. CORA-GLM), we use the following procedures:

Recall comparison at fixed gating. For each gating level g , we compute `CapturedSEVs` for both models on the same test set and report the relative improvement $\Delta = (\text{recall}_{\text{new}} - \text{recall}_{\text{base}}) / \text{recall}_{\text{base}}$. Because the test sets are identical across models and the outcome (SEV) is binary, the comparison is paired: both models score the same diffs, and improvements reflect genuine reranking of SEV-causing diffs rather than sampling variation.

Multiple gating levels. We evaluate at 5%, 10%, 15%, 20%, 45%, and 50% gating. Because these are nested thresholds on the same ranked list, no multiple-comparison correction is applied. Organization-level breakdowns (Table 8) are descriptive stratified analyses showing heterogeneity in improvement magnitude, not independent hypothesis tests.

Year-over-year production comparison. The RQ 2 comparison (Freeze 1 vs. Freeze 2) is a natural experiment with a difference-in-differences (DiD) structure: the treatment is the deployment of CORA, and the outcome is both the gating rate and the outage rate. However, because the treatment applies to the entire population simultaneously (no concurrent control group), we cannot isolate the CORA effect from other concurrent changes (e.g., improved engineering practices, different config change mix). We therefore report the YoY comparison as descriptive evidence of production impact rather than a causal estimate.

Confirmatory bootstrap analysis. Following the iterative action research design, RQs 1, 3, and 4 each informed the next iteration; applying statistical testing at each intermediate step would conflate exploratory model development with confirmatory evaluation. We therefore reserve statistical testing for the *cumulative* improvement of the final model (CORA-GBT+) over the initial deployed baseline (CORA-GLM). We compute 95% bootstrap confidence intervals by resampling all test-set diffs with replacement ($B=10,000$ iterations), computing the recall difference Δ_{recall} for each bootstrap sample, and reporting the 2.5th and 97.5th percentiles. A confidence interval that excludes zero confirms that the improvement is reliable.

Ablation analysis. For the enhanced model (CORA-GBT+), we isolate the contribution of three of the four enhancements (enriched service features, SEV-level-aware training, and hyperparameter optimization) by adding them incrementally to a reconstructed CORA-GBT baseline on a common training window and measuring the marginal change in `CapturedSEVs` on the held-out test set; the extended training window is held fixed and not separately ablated. Full tables and the deployed end-to-end comparison appear in the supplementary material.

3.9 Operational Use

When a config diff is submitted for landing, CORA computes a risk score in real time; if the score exceeds the threshold for the current freeze period, the diff is gated. The risk score, top risk factors (via SHAP for GBT models), and the current gating zone appear in the

■ **Table 5** CORA-GLM vs. DRS on Org_A (Freeze 1 backtest). ■ **Table 6** Freeze 1 vs. Freeze 2 (Org_A config diffs). Changes may reflect CORA, concurrent improvements, or config mix.

Gating	Impr.
5%	+11%
20%	+8.3%
50%	+5.7%
60%	–

Metric	Freeze 1	Freeze 2	Change
Gating (overall)	27.1%	21.7%	–5.4pp
Gating (high-risk)	100%	85%	–15pp
Diffs landed	14,277	27,322	+91.4%
Diffs/developer	4.7	5.9	+25.5%
Bypass rate	83.8%	51.6%	–32.2pp
Outage impact	high	reduced	↓
SEVs delayed	–	4	–

Phabricator UI [1]. CORA operates concurrently with the DRS model: we classify diffs as “config” or “code” based on the artifact type and apply the appropriate model. Diffs modifying both receive scores from both models, with the maximum determining the gating decision.

3.10 Implementation

All models are implemented in Python. Logistic regression (CORA-GLM) uses `statsmodels` GLM with a binomial family and logit link. Gradient boosting trees (CORA-GBT, CORA-GBT+) use LightGBM [12]; the initial CORA-GBT uses default hyperparameters (which already outperform tuned logistic regression), while CORA-GBT+ uses Bayesian optimization via Optuna [3] with a TPE sampler and `lgb.cv` for candidate evaluation. Explainability uses the SHAP library [14] and scikit-learn’s Partial Dependence Plots; scores are remapped to a 0–100 percentile scale using `QuantileTransformer`.

4 Results

4.1 RQ 1. Config-Specific Risk Model

Can a dedicated config diff risk model outperform the code-oriented DRS model on config diffs?

To evaluate the benefit of config-specific features, we compare the CORA-GLM model (Section 3.5) against the DRS logistic regression model [17] applied to config diffs. Both models are logistic regressions; the only difference is that CORA-GLM includes config-specific features (service criticality, blast radius, and service testing) in addition to the DRS supply chain features. We evaluate on Org_A config diffs using backtesting on the Freeze 1 period.

Table 5 shows the percentage of SEVs captured (**CapturedSEVs**) at various gating levels. CORA-GLM achieves an 11% improvement in recall at 5% gating and a 5.7% improvement at 50% gating relative to the code-oriented DRS model applied to config diffs. The improvement is larger at lower gating levels, where the model must be more discriminating. This is precisely the regime that matters for weekend and yellow-zone freezes, where gating overhead is tightly constrained.

At 60% gating, CORA-GLM captures 82% of all SEVs, 95% of SEVs with measurable impact, all 4 SEV1-causing diffs, and all 5 high-impact SEV-causing diffs. These are precisely the config diffs that the code-oriented DRS model missed, leading to the escalation of config gating to 100% during the Freeze 1 red zone. Because the RQ 1 test set contains a small number of SEV-causing diffs (fewer than 100), the recall improvements should be interpreted as exploratory; the confirmatory bootstrap analysis in Table 11 provides the rigorous end-to-end comparison.

The more discriminative config-specific features are `max_hosts`, `is_tier1`, and `used_by_service_count`, all absent from the code-oriented DRS model.

RQ 1: CORA-GLM achieves 11% improvement in recall at 5% gating over DRS. At 60% gating, it captures 82% of all SEVs and blocks all high-impact outage-causing diffs. Config-specific features (service criticality, blast radius, testing) are the main discriminators.

4.2 RQ 2. Impact

How effective was CORA during the Freeze 2 risk management period?

We deployed the CORA-GLM model for Org_A config diffs in time for the Freeze 2 risk management window, comparing it to the Freeze 1 window when config gating relied on the code-oriented DRS model.

Table 6 summarizes the key metrics. Gating decreased while outage impact also decreased, despite a 91.4% increase in landing volume, and gating delayed four SEVs. The volume increase reflects two factors: (1) a major increase in the developer population landing config diffs in Org_A, and (2) the bypass rate on gated diffs dropped from 83.8% to 51.6%, consistent with increased trust in gating decisions, though alternative explanations (policy changes, different developer population) cannot be ruled out. The per-developer landing rate increased 25.5% after normalizing for headcount. As an observational before/after comparison within an active research program, CORA deployment is one plausible explanation for these improvements, but concurrent changes in engineering practices, config change mix, and headcount may also contribute.

Coverage gap. An important operational finding: 44% of config *changes* in Org_A (measured by volume of changes, *not* by share of config-related SEVs) do not land through a reviewable diff. These non-diff changes include Gatekeeper rollouts performed through the UI, server-management updates, and other mechanisms that bypass the Phabricator diff-review process. Because CORA scores diffs at review time, these changes fall outside its scope: they produce no diff artifact and offer no review checkpoint at which a risk score could gate them. Extending risk prediction to non-diff config changes – reusing the same service-level features, which do not depend on a diff – is a promising direction for future work.

RQ 2: With CORA deployed, config gating decreased from 27.1% to 21.7% and outage impact decreased, while config diff landing volume increased 91.4%. The bypass rate dropped from 83.8% to 51.6%, consistent with increased trust in gating decisions. These are *observational* year-over-year changes: CORA’s deployment coincides with concurrent changes in engineering practices, config mix, and headcount, so the effect cannot be attributed to CORA alone.

4.3 RQ 3. Unified Tree-Based Model

Can a gradient boosting tree model trained on Meta-wide data outperform the logistic regression model for config diff risk prediction?

We evaluate the CORA-GBT model (Section 3.5) against CORA-GLM. Tables 7 and 8 show the results.

The unified tree-based model achieves a 25.95% relative improvement (+6.3pp) in %SEV caught at 10% gating for Meta-wide config diffs and a 32.30% relative improvement (+4.2pp) at 5% gating for Org_A config diffs, corresponding to additional outage impact intercepted.

Organization-level analysis. Training on Meta-wide data offers the advantage of learning patterns that transfer across organizations. The improvements range from 15% for Org_A (which already has a well-tuned organization-specific model) to 66% for Org_D (which benefits most from the shared training data and non-linear modeling). Organizations with smaller local datasets benefit more from Meta-wide training.

■ **Table 7** Recall improvement: CORA-GBT vs. CORA-GLM.

Gating	Improvement	Scope
5%	+32.30%	Org _A
10%	+25.95%	Meta-wide

■ **Table 8** Organization-level improvement at 10% gating.

Organization	Improvement
Org _A	+15%
Org _B	+22%
Org _C	+38%
Org _D	+66%
Meta-wide	+25.95%

■ **Table 9** Recall improvement: CORA-GBT+ vs. CORA-GBT at gating levels.

Gating Level	Recall Improvement	Outage Impact Intercepted	Freeze Scenario
5%	+25%	–	Weekend
15%	+22%	substantial	Low/med risk
45%	+10.6%	moderate	High risk

The more impactful features by SHAP value are *weighted_caused_sev* (prior SEV history), *centrality* (supply chain importance), *log_diff_author_count* (coordination complexity), and *substantial_line_count_ratio* (change magnitude). Config-specific service features (*max_hosts*, *is_tier1*, *uses_service_count*) complement the supply chain features. Partial Dependence Plots reveal non-linear relationships that logistic regression cannot capture: for example, risk escalates sharply for services with numerous hosts, and the interaction between service tier and test coverage is automatically captured by tree splits.

Methodological insights from the transition. *Empirical findings:* (1) GBTs with default hyperparameters already outperform tuned logistic regression, suggesting that non-linear feature interactions are a first-order effect in config risk; (2) systematic offline/online feature consistency checks identified discrepancies between training-time and prediction-time feature values, and resolving them improved production reliability; (3) a data-vs-code classification of config changes did not improve model performance. These findings informed the design of CORA-GBT+: the weighted, time-decayed SEV history measure replaced the binary *had_sev* flag, and GBT’s implicit interaction modeling eliminated explicit interaction terms.

RQ 3: The unified CORA-GBT model achieves a 25.95% improvement in recall at 10% gating Meta-wide and 32.30% at 5% gating for Org_A. Organization-level improvements range from 15% (Org_A) to 66% (Org_D), demonstrating that tree-based models trained on Meta-wide data outperform organization-specific logistic regression.

4.4 RQ 4. Enhanced Model

Can enriched service features, SEV-level-aware training, and hyperparameter optimization further improve the unified tree model?

We evaluate the CORA-GBT+ model (Section 3.5) against the CORA-GBT baseline. The evaluation uses a test set shifted forward from the CORA-GBT test period to reflect recent SEV patterns.

Overall performance. Table 9 shows the recall improvement at gating levels corresponding to the three freeze scenarios.

The enhanced model achieves a 22% improvement in %SEV caught at 15% gating (for low/medium risk periods) with substantial incremental outage impact intercepted, and a 10.6% improvement at 45% gating (for high-risk periods) with moderate incremental outage impact intercepted. At 5% gating, the improvement is 25%, demonstrating that the enhancements are more effective at the lower gating levels where discriminatory precision matters most.

Precision improvement. Unlike many model upgrades that improve recall at the expense of precision, the enhanced model improves both: Meta-wide precision increases by over 20% at all gating levels. This is operationally valuable because higher precision means fewer engineers are gated unnecessarily, reducing friction while intercepting more SEVs.

Org_A-specific results. For Org_A config diffs, the enhanced model achieves a 25% recall improvement at 5% gating and a 30.77% improvement in high-impact SEV recall. Org_A config SEVs have direct and quantifiable operational impact, so the impact-weighted recall measures the model’s ability to flag the more severe incidents for review.

Ablation analysis. A partial per-component ablation – isolating three of the four enhancements on a common training window and the shared test set – indicates that the *enriched config/service features* are the most consistent contributor to **CapturedSEVs**, in line with CORA’s central premise that config risk derives from service-level properties. SEV-level-aware training (Equation 1) has only a small effect on *overall* recall; its intended benefit is concentrated in *high-severity* (SEV1/SEV2) recall, which aggregate **CapturedSEVs** does not capture. Hyperparameter optimization, tuned to cross-validated log-loss, did *not* improve recall at the operating gating levels in our reconstruction – the log-loss objective is not aligned with recall at a fixed gating percentage – and the deployed CORA-GBT+ accordingly uses the default tree depth. The extended training window reduces temporal drift but is not separately ablated. Because the reconstructed base already trains on the full recent window, the marginal effects are small: much of the end-to-end CORA-GBT+ vs CORA-GBT gain reflects retraining on more recent data together with the enriched features. The full per-component tables (marginal **CapturedSEVs** at every gating level, Meta-wide and for Org_A) and the deployed end-to-end comparison are provided as supplementary material.

RQ 4: CORA-GBT+ achieves 22% improvement in recall at 15% gating and 10.6% at 45% gating. A per-component ablation attributes the gain primarily to enriched config/service features together with retraining on recent data; SEV-level-aware training targets high-severity recall, while hyperparameter tuning did not improve recall at the operating gating levels.

4.5 Summary of Results

Table 10 summarizes the progressive improvements achieved across the four CORA model versions.

Test₁ and Test₂ denote the non-overlapping test periods from Table 4; Test₂ is shifted forward to reflect recent SEV patterns. Because each RQ uses the test set available at that iteration (Section 3.8), per-RQ improvements are not directly comparable; only the bootstrap comparison in Table 11 evaluates both endpoints on the same test set.

The trajectory shows consistent improvement along three axes: model sophistication (logistic regression to enhanced GBT), scope (Org_A-only to Meta-wide), and outage impact intercepted (from flagging all high-impact SEVs in backtesting to incremental impact intercepted per cycle). We deployed CORA-GBT+ for the Freeze 3 (2025 Q4) risk-management

■ **Table 10** Evolution of CORA model performance.

Version	Model	Scope	Recall Improvement	Period
RQ 1	Logistic Regression	Org _A	+11% at 5% gating vs. DRS	Freeze 1
RQ 2	(Retrospective)	Org _A	reduced outage impact YoY	Freeze 2
RQ 3	LightGBM	Meta-wide	+25.95% at 10% gating	Test ₁
RQ 4	Enhanced LightGBM	Meta-wide	+22% at 15% gating	Test ₂
	(Deployed)	Org _A	+22% config SEV recall	Freeze 3

■ **Table 11** Bootstrap 95% CIs for recall difference: CORA-GBT+ vs. CORA-GLM.

Gating	Δ Recall	Rel. Impr.	95% CI	Scope
5%	+3.1pp	+19.7%	[+0.1pp, +6.3pp]	Meta-wide
10%	+4.6pp	+17.3%	[+1.1pp, +7.9pp]	Meta-wide
15%	+4.0pp	+11.0%	[+0.4pp, +7.5pp]	Meta-wide
45%	+8.9pp	+13.2%	[+5.2pp, +12.6pp]	Meta-wide
5%	+5.2pp	+38.3%	[+0.7pp, +9.3pp]	Org _A
15%	+9.6pp	+30.7%	[+4.6pp, +14.7pp]	Org _A
45%	+12.1pp	+18.6%	[+6.8pp, +17.4pp]	Org _A

window; consistent with the backtested RQ 4 result, it improved config SEV recall by roughly 22%. We deliberately do *not* report a production outage-impact year-over-year figure for Freeze 3: config diff volume grew substantially year-over-year and the residual outage loss was dominated by diffs that bypassed gating, so a raw before/after comparison cannot be cleanly attributed to CORA. We use **ImpactCaptured** (Equation 3) internally; specific values are organization-sensitive and we report only relative changes.

Confirmatory statistical analysis. As described in Section 3.8, we report bootstrap confidence intervals for the *cumulative* improvement of the final model (CORA-GBT+) over the initial deployed baseline (CORA-GLM).

Table 11 shows the 95% bootstrap confidence intervals (Section 3.8, $B=10,000$) for the recall difference between CORA-GBT+ and CORA-GLM on the test set (331,933 Meta-wide config diffs, 831 SEVs). All intervals exclude zero, confirming that the improvements are reliable. Since CORA-GLM already outperforms the code-oriented DRS model on config diffs (RQ 1), and CORA-GLM is a strict superset of DRS features, these confidence intervals are conservative estimates of the improvement over DRS.

5 Discussion

CORA follows the participatory action research methodology [25, 4] as applied in [17]: each iteration identifies a problem, develops a solution, evaluates it in production, and uses the findings to motivate the next cycle. The Freeze 1 incident (missed SEV1s) motivated CORA-GLM. The Freeze 2 deployment validated config-specific features and revealed limitations of logistic regression (inability to capture non-linear interactions). The transition to CORA-GBT addressed those limitations and produced new insights: default GBT hyperparameters already outperformed tuned logistic regression, indicating that non-linear feature interactions are a first-order effect in config risk; offline/online feature consistency checks uncovered discrepancies that improved production reliability; and data-vs-code config classification proved uninformative. The CORA-GBT+ iteration used these findings to add enriched service features and SEV-severity-aware training (and to explore Bayesian hyperparameter optimization), yielding simultaneous precision and recall improvements driven mainly by the enriched features and retraining on more recent data.

The per-component ablation in RQ 4 (supplementary material) isolates three of the four enhancements on a common training window and test set. It shows that enriched config/service features are the most consistent contributor, that SEV-severity weighting mainly shifts high-severity recall rather than aggregate recall, and that log-loss-tuned hyperparameter optimization did not improve recall at the operating gating levels. Because the enhancements were developed iteratively rather than in a controlled factorial experiment – and because the reconstructed base already trains on the full recent window – these marginal estimates are approximate and do not sum to the end-to-end gain, much of which reflects recent-data retraining. A fully factorial design (including the training window, held fixed here) could quantify each component’s independent contribution, but doing so in a production system with time-varying data and evolving engineering practices remains a challenge for future work.

User feedback drives feature engineering throughout: *num_non_generated_files* and *has_materialized_JSON* emerged from engineer feedback on false positives, and the removal of interaction terms in the GBT models reduced the feature space without loss. Training on Meta-wide data rather than organization-specific subsets benefits organizations with smaller local datasets (RQ 3), consistent with transfer learning where shared patterns compensate for sparse local signal.

Implications for practitioners. Our experience suggests several recommendations: (1) use config-to-service mapping as the main enabler for config risk prediction; (2) differentiate config and code risk models, as code-oriented models both over-gate safe config changes and under-gate risky ones; (3) measure safety net reliability (false positive rates), not just coverage; (4) weight training by SEV severity to focus on the more severe incidents; and (5) iterate frequently with feedback.

6 Threats to Validity

Generalizability. Drawing general conclusions from empirical studies in software engineering is difficult because any process depends on a potentially large number of relevant context variables [17]. Some aspects of our study provide limited internal diversity, but we are careful not to overstate external validity. First, although the study is conducted within a *single company* (Meta), it spans multiple large, autonomous product organizations (Org_A – Org_D), each with different config change patterns and SEV rates. This is not equivalent to evaluation across independent companies: all of these organizations share Meta’s infrastructure, tooling, and engineering culture, so external validity beyond Meta is not established. Second, we *hypothesize* that the central insight – that config risk derives from service-level properties (criticality, blast radius, operational readiness) rather than code-level properties – generalizes to other service-oriented architectures, because any organization that manages services through configuration faces the same mapping problem from what changed to which services are affected. This “reusable-blueprint” argument is a hypothesis grounded in the structure of the problem, not a finding validated outside Meta. Third, the specific features CORA uses (service criticality tiers, host counts, service dependencies, test coverage, canary infrastructure) are common across service-oriented architectures, but their predictive value in other organizations remains to be tested.

Construct Validity. Following [17, 1], we use SEV-causing diffs as the outcome variable for risk prediction. This operationalization has an advantage over many defect prediction studies: the incident response team manually determines the trigger diff for each SEV, avoiding the

noisy SZZ heuristics [29, 6] that many published studies use. However, this outcome captures only diffs that caused SEVs severe enough to be tracked; config changes that degraded performance without triggering a SEV are not captured as positive examples, potentially biasing the model toward catastrophic failures. For config changes specifically, gradual degradations (e.g., a config change that increases latency by 10% without triggering an alert) may be common but invisible to the model. The service-level features (criticality, blast radius) would plausibly flag such changes, but the model has no training signal to learn from them.

As noted in [17], the observational nature of the data creates challenges for interpreting individual feature effects due to latent confounding. For example, services with higher test coverage may appear *more* risky in the data, not because testing increases risk, but because high-risk services are more likely to have invested in testing. We mitigate this through feature selection favoring interpretable directional effects and SHAP-based per-prediction explanations, but the underlying confounding persists.

The evaluation metric (**CapturedSEVs** at a given gating level) directly measures what the code freeze system optimizes for. However, as noted in [1], other metrics such as the reduction in SEV count, developer productivity impact, and long-term behavioral changes are also important but harder to measure rigorously.

Internal Validity. The numerical results for Research Questions 1, 3, and 4 rely on backtesting: we simulate how the model would have performed had it been deployed during the evaluation period. Actual performance may differ, for example due to behavioral changes induced by gating (engineers modifying their diffs in response to risk scores). The partial ablation in RQ 4 isolates three of the four enhancements (enriched features, SEV-level-aware training, and hyperparameter optimization) on a common training window; the extended training window is held fixed and not separately ablated. Because the enhancements were developed iteratively within an action research cycle rather than in a controlled factorial experiment – and because the reconstructed baseline already trains on the full recent window – these marginal attributions are approximate, are subject to sampling noise at the observed SEV counts, and do not sum to the end-to-end gain. The confirmatory bootstrap analysis (Table 11) tests only the cumulative end-to-end improvement.

A second concern is temporal drift: risk patterns may change over time as the codebase, services, and engineering practices evolve. We mitigate this by using time-based data splits (Section 3.8) so that the model is always evaluated on data from a later period than its training data, and by extending the training window in the enhanced model (CORA-GBT+) to reduce the gap between training and deployment. Beyond temporal drift in feature distributions, *concept drift* may also occur: the nature of what constitutes a risky config change may shift qualitatively as the organization adopts new config mechanisms (the 44% of non-diff config changes in Section 4.2 illustrate this evolving landscape).

A third concern is the feedback loop created by deployment: when the model gates high-risk diffs, it prevents the SEVs that would have validated its predictions, making future evaluation harder. The bypass rate drop (83.8% to 51.6%) is itself evidence of behavioral change. Engineers may also modify their diffs in response to risk scores, altering the distribution of future training data in ways that are difficult to account for.

Finally, while SEV1 and SEV2 trigger attribution is likely high-quality (major incidents receive thorough investigation), SEV3 and SEV4 trigger attribution may be noisier. Since SEV3/SEV4 constitute the vast majority of positive labels, label noise at lower severity levels could affect model training, though the SEV-level-aware weights (Equation 1) partially mitigate this by down-weighting lower-severity samples.

Conclusion Validity. The primary threat to conclusion validity is the low base rate of SEV-causing diffs (<0.3% of all config diffs). While the total number of diffs in each test set is large (>100K), the number of positive samples (diffs that caused SEVs) is small: 831 in the Meta-wide test set and 446 in the Org_A subset. As described in Section 3.8, we treat the intermediate comparisons (RQs 1, 3, 4) as exploratory and reserve confirmatory testing for the cumulative improvement of CORA-GBT+ over CORA-GLM. Bootstrap 95% confidence intervals (Table 11) confirm that the end-to-end improvement is reliable at all gating levels, with all intervals excluding zero. Nonetheless, individual high-impact SEVs can disproportionately affect the results, and the improvements should be interpreted in conjunction with the outage-impact-captured measure (Equation 3), which weights SEVs by their operational severity.

Researcher and Organizational Bias. The authors are practitioner-researchers who designed, built, deployed, and evaluated CORA within their own organization. This dual role is intrinsic to action research but introduces potential bias: feature selection, interpretation of results, and the choice of evaluation metrics and gating levels were made by the same team that developed the system and reflect the organization’s operational priorities. Reporting relative rather than absolute figures (for confidentiality) also limits external scrutiny of the magnitudes. We mitigate these risks in three ways: the outcome measures and evaluation protocol are inherited from prior DRS work [17, 1] rather than devised to favor CORA; the SEV-to-trigger-diff labels are determined by an independent incident-response team, not by the authors; and confirmatory conclusions rest on bootstrap testing on a held-out, later-in-time test set (Table 11). An independent replication in a different organizational context would nonetheless provide stronger evidence.

7 Concluding Remarks

Configuration changes are a common source of outages, yet the defect prediction literature focuses almost exclusively on source code. We describe CORA, the first dedicated risk model for configuration diffs, and demonstrate its effectiveness through three iterations of participatory action research. A logistic regression model with config-specific features achieves an 11% improvement at 5% gating over the code-oriented DRS model and captures all diffs causing high-impact outages (RQ 1). During Freeze 2, config gating decreased while outage impact fell (RQ 2). A unified LightGBM model achieves 25.95% improvement at 10% gating Meta-wide, revealing that non-linear feature interactions are a first-order effect in config risk prediction (RQ 3). An enhanced model adding enriched service features and SEV-severity-aware training achieves 22% improvement at 15% gating with simultaneous precision gains, and was deployed for the Freeze 3 (2025 Q4) window, where it improved config SEV recall by roughly 22% (RQ 4).

The key insight is that config diff risk derives from service importance, blast radius, and operational readiness rather than code complexity. The config-path-to-service mapping provides the structural bridge enabling this reasoning, and applies to any organization managing configuration changes in service-oriented architectures.

References

- 1 Rui Abreu, Vijayaraghavan Murali, Peter C. Rigby, Chandra Maddila, Weiyang Sun, Jun Ge, Kaavya Chinniah, Audris Mockus, Megh Mehta, and Nachiappan Nagappan. Moving faster and reducing risk: Using LLMs in release deployment. In *Proceedings of the 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2025. doi:10.1109/icse-seip66354.2025.00045.

- 2 Bram Adams and Shane McIntosh. Modern release engineering in a nutshell – why researchers should care. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 5, pages 78–90, 2016. doi:10.1109/SANER.2016.108.
- 3 Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2623–2631, 2019. doi:10.1145/3292500.3330701.
- 4 Fran Baum, Colin MacDougall, and Danielle Smith. Participatory action research. *Journal of Epidemiology and Community Health*, 60(10):854–857, 2006. doi:10.1136/jech.2004.028662.
- 5 Kaiwen Chow, Yanbing Wang, Qingsen Wang, and Andromeda Hailu. ServiceLab: Preventing tiny performance regressions at hyperscale through pre-production testing. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 185–202. USENIX Association, 2024. URL: <https://www.usenix.org/conference/osdi24/presentation/chow>.
- 6 Daniel Alencar Da Costa, Shane McIntosh, Weiyi Shang, Uirá Kulesza, Roberta Coelho, and Ahmed E. Hassan. A framework for evaluating the results of the SZZ approach for identifying bug-introducing changes. *IEEE Transactions on Software Engineering*, 43(7):641–657, 2017. doi:10.1109/TSE.2016.2616306.
- 7 Carsten F. Dormann, Jane Elith, Sven Bacher, Carsten Buchmann, Gudrun Carl, Gabriel Carré, Jaime R. García Marquéz, Bernd Gruber, Bruno Lafourcade, Pedro J. Leitão, et al. Collinearity: A review of methods to deal with it and a simulation study evaluating their performance. *Ecography*, 36(1):27–46, 2013. doi:10.1111/j.1600-0587.2012.07348.x.
- 8 Todd L. Graves, Alan F. Karr, J. S. Marron, and Harvey Siy. Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering*, 26(7):653–661, 2000. doi:10.1109/32.859533.
- 9 Mislav Grubić, Josip Balić, Ilias Leontiadis, Soteris Demetriou, Dat Duong, and Kaushik Veeraraghavan. Conveyor: One-tool-fits-all continuous software deployment at Meta. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 539–555. USENIX Association, 2023. URL: <https://www.usenix.org/conference/osdi23/presentation/grubic>.
- 10 Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E. Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering*, 39(6):757–773, 2013. doi:10.1109/TSE.2012.70.
- 11 Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953. doi:10.1007/BF02289026.
- 12 Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30 (NeurIPS)*, pages 3146–3154, 2017. URL: <https://proceedings.neurips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html>.
- 13 Hossein Keshavarz and Meiyappan Nagappan. ApacheJIT: A large dataset for just-in-time defect prediction. In *Proceedings of the 19th International Conference on Mining Software Repositories (MSR)*, pages 191–195, 2022. doi:10.1145/3524842.3527996.
- 14 Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30 (NeurIPS)*, pages 4765–4774, 2017. URL: <https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html>.
- 15 Audris Mockus. Insights from open source supply chains. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, page 3, 2019. Industry Keynote. doi:10.1145/3338906.3342813.

- 16 Audris Mockus. Securing LLM-based software supply chains. In *38th IEEE/ACM International Conference on Automated Software Engineering (ASE), Workshop on LLMs in Software Engineering*, 2023. Invited talk.
- 17 Audris Mockus, Rui Abreu, Peter C. Rigby, David Amsallem, Parveen Bansal, Kaavya Chinniah, Brian Ellis, Peng Fan, Jun Ge, Bingjie He, Kelly Hirano, Sahil Kumar, Ajay Lingapuram, Andrew Loe, Megh Mehta, Venus Montes, Maher Saba, Gursharan Singh, Matt Steiner, Weiyan Sun, Siri Uppalapati, and Nachiappan Nagappan. Leveraging risk models to improve productivity for effective code un-freeze at scale. *ACM Transactions on Software Engineering and Methodology*, 2025. doi:10.1145/3722216.
- 18 Audris Mockus, Randy L. Hackbarth, and John Palframan. Risky files: An approach to focus quality improvement effort. In *Proceedings of the 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 691–694, 2013. doi:10.1145/2491411.2494572.
- 19 Audris Mockus and David M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, 2000. doi:10.1002/bltj.2229.
- 20 Nachiappan Nagappan and Thomas Ball. Use of relative code churn measures to predict system defect density. In *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pages 284–292, 2005. doi:10.1109/ICSE.2005.1553571.
- 21 Changhai Nie and Hareton Leung. A survey of combinatorial testing. *ACM Computing Surveys*, 43(2):11:1–11:29, 2011. doi:10.1145/1883612.1883618.
- 22 Jalaj Pachouly, Swati Ahirrao, Ketan Kotecha, Ganeshsree Selvachandran, and Ajith Abraham. A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools. *Engineering Applications of Artificial Intelligence*, 111:104773, 2022. doi:10.1016/j.engappai.2022.104773.
- 23 Phacility. Phabricator. <https://www.phacility.com/phabricator/>. Open-source code review platform, originally developed at Facebook.
- 24 Md Tajmilur Rahman and Peter C. Rigby. Release stabilization on Linux and Chrome. *IEEE Software*, 32(2):81–88, 2015. doi:10.1109/MS.2015.31.
- 25 Ernest T. Stringer and Alfredo Ortiz Aragón. *Action Research*. SAGE Publications, 5th edition, 2020.
- 26 Chunqiang Tang. Meta’s hyperscale infrastructure: Overview and insights. *Communications of the ACM*, 68(2):52–63, 2025. doi:10.1145/3701296.
- 27 Chunqiang Tang, Thawan Kooburat, Pradeep Venkatachalam, Akshay Chander, Zhe Wen, Aravind Narayanan, Patrick Dowell, and Robert Karl. Holistic configuration management at Facebook. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP)*, pages 328–343. ACM, 2015. doi:10.1145/2815400.2815401.
- 28 Linus Torvalds. Linux code freeze. *Linux Journal*, 1994(1es):4–es, 1994.
- 29 Chadd Williams and Jaime Spacco. SZZ revisited: Verifying when changes induce fixes. In *Proceedings of the 2008 Workshop on Defects in Large Software Systems (DLSD)*, pages 32–36, 2008. doi:10.1145/1390817.1390826.
- 30 Yunhua Zhao, Kostadin Damevski, and Hui Chen. A systematic survey of just-in-time software defect prediction. *ACM Computing Surveys*, 55(10):1–35, 2023. doi:10.1145/3567550.