

Adaptive Release Intelligence For Enterprise Mobile Applications: An Ai-Driven Framework For Risk-Aware Deployment, Feature Control, And Automated Recovery

Bhargava Chowdary Musunuru

Principal Android Developer / Tech Lead. Fidelity Investments, email: bhargava.musunuru@gmail.com

Abstract

Enterprise mobile applications are continuously updated to introduce new functionality, improve performance, address security issues, and respond to changing business requirements. Conventional CI/CD pipelines primarily depend on predefined quality gates, static test results, and manually configured deployment policies. Although these mechanisms provide essential safeguards, they may not adequately capture dynamic operational risks that emerge after deployment. This paper proposes Adaptive Release Intelligence (ARI), an AI-driven framework that transforms release management from a static gate-based process into a predictive and adaptive decision system. The proposed framework integrates application telemetry, crash information, performance indicators, user-impact signals, deployment history, feature-level behavior, device and network conditions, and CI/CD metadata to estimate release risk. Based on the predicted risk, ARI dynamically controls deployment progression, feature exposure, monitoring intensity, rollback decisions, and recovery actions. A Release Risk Score (RRS) is introduced to represent the estimated probability and severity of release-related failures. The framework further incorporates explainable AI to support governance and provide evidence for release decisions. The proposed approach aims to reduce uncontrolled release exposure, detect degradation at an early stage, improve recovery speed, and establish a continuous feedback loop between development, deployment, production monitoring, and subsequent release decisions.

Keywords: Adaptive Release Intelligence, Enterprise Mobile Applications, Artificial Intelligence, Risk-Aware Deployment, Feature Control, CI/CD, Automated Rollback, Release Engineering, Predictive Analytics, Explainable AI.

1. Introduction

Enterprise mobile applications have become important components of digital business ecosystems, supporting customer services, employee operations, financial transactions, communication, logistics, healthcare, and enterprise resource management. The increasing frequency of application releases has created a need for deployment processes that can maintain reliability while allowing organizations to deliver new capabilities rapidly. Mobile applications operate across heterogeneous devices, operating-system versions, network conditions, backend services, and user environments. Consequently, an application release that performs correctly during pre-production testing may still experience unexpected failures after exposure to real users.

Traditional CI/CD pipelines generally evaluate releases using predefined conditions such as successful builds, test completion, code-quality thresholds, security checks, and deployment approvals. These gates are valuable but are largely static. They typically determine whether a release satisfies predefined conditions rather than continuously estimating how operational risk changes during deployment. Predictive maintenance research has demonstrated the value of combining historical and real-time data with artificial intelligence to identify emerging failures before they become critical [1,2]. Machine-

learning-based predictive approaches have also been investigated extensively for automotive and industrial systems [3–5].

The same predictive principle can be extended to software release engineering. Instead of treating deployment as a binary decision deploy or do not deploy an intelligent release system can continuously evaluate the condition of the application and determine how much of the user population should be exposed, whether additional monitoring is required, whether a feature should be restricted, and whether rollback or recovery should be initiated. Automotive cyber-physical health-management research has similarly emphasized integrated monitoring and intelligent health assessment across complex systems [6]. Vehicle and fleet monitoring studies further demonstrate the importance of continuous telemetry and predictive analysis for operational decision-making [7,8].

This paper proposes Adaptive Release Intelligence (ARI) as a framework for enterprise mobile applications. ARI integrates application runtime signals, crash behavior, performance degradation, deployment characteristics, feature-level telemetry, and CI/CD information into an AI-based release-risk assessment mechanism. The resulting risk estimate is translated into adaptive deployment actions, including progressive rollout, feature exposure control, intensified monitoring, rollback, and automated recovery.

The primary contributions of the proposed work are:

1. An AI-driven Adaptive Release Intelligence architecture for enterprise mobile applications.
2. A Release Risk Score (RRS) for continuously estimating deployment risk.
3. A dynamic mechanism for risk-aware deployment progression and feature exposure control.
4. An automated rollback and recovery decision mechanism based on observed release risk.
5. An explainable governance layer that provides evidence for AI-assisted release decisions.
6. A continuous feedback mechanism that uses post-release behavior to improve future release-risk prediction.

2. Literature Review

The increasing complexity of connected and software-intensive systems has encouraged the use of artificial intelligence, machine learning, IoT, and predictive analytics for reliability improvement. Existing research has primarily focused on predictive maintenance, fault diagnosis, condition monitoring, and automotive health management. These studies provide the theoretical foundation for developing an adaptive release-intelligence approach for enterprise mobile applications.

2.1 AI and Predictive Approaches for Reliability

Predictive maintenance through the integration of artificial intelligence and IoT has demonstrated the potential of intelligent systems to analyze operational information and identify potential failures proactively [1]. A systematic review of predictive maintenance in Industry 4.0 highlighted the transition from conventional maintenance practices toward data-driven and predictive decision-making [2]. These studies establish that continuously collected operational data can be transformed into predictive information for improving system reliability. However, their primary focus is physical assets rather than software-release management.

2.2 Predictive Maintenance in Automotive Systems

The integration of Industry 4.0 technologies with condition-based and predictive maintenance has demonstrated the importance of connected monitoring and proactive decision-making in automotive environments [3]. Fault detection and failure prediction using unsupervised learning and simulation have further demonstrated the usefulness of intelligent analytical approaches for identifying abnormal system conditions [4]. Machine-learning-enabled predictive maintenance in the automotive industry has also shown the potential of ML techniques for identifying patterns associated with future failures [5]. These findings are relevant to release intelligence because mobile applications associated with connected automotive ecosystems similarly generate continuous operational data that can be analyzed for emerging reliability risks.

2.3 Integrated Monitoring and Connected Systems

Integrated health-management approaches for automotive cyber-physical systems emphasize the combination of multiple sources of health information for intelligent assessment [6]. Remote vehicle health monitoring and prognostic maintenance further demonstrate the value of continuously collected operational information for proactive reliability management [7]. IoT-based predictive maintenance for fleet management shows how distributed operational data can support predictive decision-making across connected systems [8].

These studies demonstrate that reliability assessment can be treated as a continuously evolving process rather than a one-time evaluation. This principle is particularly important for enterprise mobile applications, where the risk associated with a release can change after deployment because of actual user behavior, device diversity, network conditions, and backend interactions.

2.4 Intelligent Diagnostics and Data-Driven Failure Detection

Intelligent vehicular diagnostic systems using OBD-II and IoT technologies demonstrate how operational data can support automated diagnosis [9]. A systematic review of machine-learning methods applied to predictive maintenance highlights the use of different ML approaches for identifying and predicting failures [10]. Data-driven fault diagnostics and prognostics further demonstrate the transformation of operational data into actionable information for predictive reliability [11].

For an adaptive release framework, these studies support the use of multiple application signals rather than relying on a single CI/CD quality gate. Release telemetry, runtime performance, crash behavior, network conditions, and deployment characteristics can similarly be integrated into a unified predictive model.

2.5 Data Preprocessing and Machine-Learning-Based Analysis

Intelligent data preprocessing and analysis are important components of predictive-maintenance applications [12]. This is particularly relevant to mobile application release intelligence because telemetry is generated from heterogeneous devices, operating systems, networks, application versions, and deployment stages. Inconsistent, incomplete, or noisy data can negatively affect predictive performance.

A systematic mapping of machine-learning applications for predictive maintenance demonstrates the increasing adoption of ML techniques for intelligent failure identification [13]. These findings support the use of machine-learning models in ARI for identifying relationships among release characteristics and subsequent operational failures.

2.6 Model-Based Prediction and Automated Fault Detection

Model-based predictive maintenance techniques applied to the automotive industry demonstrate the relevance of predictive models for anticipating system degradation [14]. Automatic fault detection using cause-and-effect relationships in in-vehicle networks further highlights the importance of relationships among multiple system signals in detecting abnormal conditions [15].

Together, these studies indicate that intelligent reliability systems can move beyond simple threshold-based monitoring by considering relationships among multiple signals. Such an approach is valuable for release intelligence because a deployment decision may depend on the combined effect of crash growth, latency changes, feature failures, deployment characteristics, and historical release behavior.

2.7 Research Gap

The literature represented by the 15 selected studies demonstrates substantial progress in AI-based predictive maintenance, automotive reliability, intelligent diagnostics, fault detection, and continuous monitoring [1–15]. However, these approaches primarily address physical assets, automotive components, production systems, or vehicle networks. They do not directly provide an integrated mechanism for AI-driven software-release risk assessment, dynamic feature exposure, adaptive deployment progression, automated rollback, and recovery governance in enterprise mobile applications.

3. Proposed Adaptive Release Intelligence Framework

The proposed Adaptive Release Intelligence (ARI) framework is designed to transform conventional CI/CD-based mobile application deployment into a predictive and adaptive release-management

process. Instead of depending only on predefined quality gates, the framework continuously evaluates application, deployment, feature, and operational signals to estimate release risk and dynamically determine the appropriate deployment and recovery strategy. The framework follows a closed-loop architecture in which production outcomes are fed back into the predictive model for subsequent releases.

3.1 Framework Architecture

The overall ARI framework consists of the following stages:

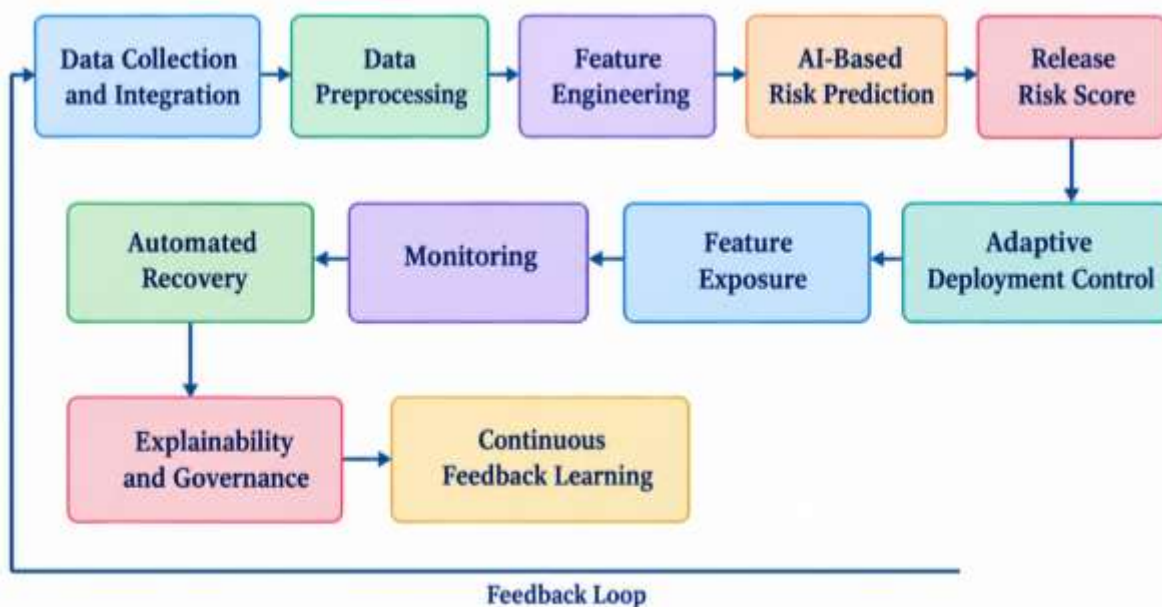


Figure 1. Proposed Adaptive Release Intelligence Architecture

The architecture enables release decisions to be continuously updated according to newly observed evidence.

3.2 Data Collection and Integration

The first stage collects information from both the development pipeline and the production environment. The framework considers runtime performance, crash behavior, API responses, network conditions, device characteristics, feature-level telemetry, release history, and CI/CD information.

The major data categories include application crash frequency, response latency, CPU and memory utilization, API failure rate, network errors, feature-specific failures, deployment frequency, failed deployment history, code-change intensity, testing outcomes, and previous release behavior. Combining these signals provides a broader representation of release health than conventional build and testing gates.

The integration of heterogeneous operational information is consistent with the broader predictive-reliability literature, where multiple data sources are used to support intelligent failure prediction and diagnosis [1,6,7,11].

3.3 Data Preprocessing

Collected telemetry may contain missing observations, duplicated records, inconsistent timestamps, abnormal values, and measurements generated at different frequencies. Therefore, preprocessing is performed before predictive modelling.

The preprocessing stage consists of data cleaning, temporal synchronization, missing-value handling, normalization, and aggregation. For a feature X , normalized representation can be expressed as:

$$X' = \frac{X - \mu_X}{\sigma_X + \epsilon}$$

where μ_X is the mean, σ_X is the standard deviation, and ϵ is a small constant used to avoid division by zero.

Intelligent preprocessing is important because the quality of input data directly influences the reliability of predictive models.

3.4 Feature Engineering

The processed data are transformed into release-level features. The feature vector at time t can be represented as:

$$X_t = [C_t, L_t, M_t, C, P, U_t, N_t, F_t, D_t, T_t, H_t]$$

where:

- C_t = crash and error intensity;
- L_t = application/API latency;
- M_t = memory utilization;
- CPU_t = CPU utilization;
- N_t = network/API failure indicators;
- F_t = feature-level reliability indicators;
- D_t = deployment characteristics;
- T_t = testing and CI/CD indicators;
- H_t = historical release information.

Temporal characteristics are also considered. Crash growth can be calculated as:

$$CG_t = \frac{C_t - C_{t-1}}{C_{t-1} + \epsilon}$$

Latency deviation can be expressed as:

$$LD_t = \frac{L_t - \mu_L}{\sigma_L + \epsilon}$$

These features allow the framework to identify not only abnormal conditions but also rapid deterioration following a new release.

3.5 AI-Based Release Risk Prediction

The feature vector is provided to machine-learning models to estimate the probability of a release-related failure. Candidate models include Random Forest, XGBoost, LightGBM, Support Vector Machine, and temporal learning approaches.

The prediction function is represented as:

$$P(\text{Failure} | X_t) = f(X_t)$$

where $P(\text{Failure} | X_t)$ denotes the estimated probability that the current release state will lead to a significant reliability problem.

The use of machine learning for predictive failure identification is supported by previous studies on predictive maintenance and intelligent fault diagnosis. However, ARI applies the predictive principle specifically to software-release decision-making.

3.6 Release Risk Score

To make the model output easier to interpret operationally, the predicted probability is converted into a Release Risk Score (RRS):

$$RRS_t = 100 \times P(\text{Failure} | X_t)$$

The resulting score ranges from 0 to 100.

RRS	Risk Level	Release Action
0–20	Low	Normal deployment
21–40	Moderate	Progressive rollout

41–60	Elevated	Restricted exposure and enhanced monitoring
61–80	High	Pause deployment and perform additional validation
81–100	Critical	Block or rollback release

The thresholds are configurable and can subsequently be optimized using validation results.

3.7 Adaptive Deployment Control

The primary innovation of ARI is the conversion of predicted risk into deployment actions. Instead of applying a single static deployment decision, the framework dynamically adjusts release progression according to the current risk.

For low-risk releases, deployment can proceed normally. Moderate-risk releases can be introduced gradually to a limited user population. Elevated-risk releases can remain under restricted exposure with intensified monitoring. High-risk releases can be paused for additional validation, while critical releases can be blocked or rolled back.

Thus, deployment becomes a controlled process:

Predict, Deploy Partially, Monitor, Reassess, Expand-Restrict-Rollback

This approach uses continuous operational evidence rather than relying exclusively on pre-deployment conditions.

3.8 Dynamic Feature Exposure

ARI also introduces feature-level control to prevent a localized problem from affecting the complete application.

The exposure of feature f at time t is represented as:

$$E_f(t) \in [0, 1]$$

where $E_f(t)$ represents the proportion of users exposed to that feature.

If the feature remains stable:

$$E_f(t + 1) > E_f(t)$$

If the feature demonstrates increasing failure risk:

$$E_f(t + 1) < E_f(t)$$

This allows the system to progressively increase or decrease feature exposure based on observed reliability. Consequently, a problematic feature can potentially be restricted without immediately reverting the entire application release.

3.9 Continuous Monitoring and Risk Reassessment

After deployment begins, ARI continuously monitors production signals. The new observations are incorporated into the risk model to determine whether the release remains safe.

A simplified risk-update process is:

$$RRS_{t+1} = f(X_{t+1})$$

If the updated risk decreases, deployment can progress. If the risk increases, feature exposure or deployment progression can be reduced. This makes release management adaptive rather than predetermined.

3.10 Automated Rollback and Recovery

When the predicted risk exceeds a validated threshold, ARI can initiate an appropriate recovery action.

$$Rollback = \begin{cases} 1, & RRS_t \geq \tau_R \\ 0, & RRS_t < \tau_R \end{cases}$$

where τ_R represents the validated rollback threshold.

The recovery sequence can be represented as:

Risk Increase, Deployment Pause, Feature Restriction, Failure Analysis, Rollback, Verification, Controlled Redeployment

The mechanism should also consider the persistence of abnormal behavior to prevent temporary fluctuations from causing unnecessary rollback.

3.11 Explainability and Governance

Because ARI uses AI to influence deployment decisions, the framework incorporates an explainability and governance layer. Each release decision should retain information regarding the predicted risk, major contributing factors, selected deployment action, monitoring observations, and recovery outcome.

The explanation can be represented as:

$$Explanation(RRS) = \{(f_1, c_1), (f_2, c_2), \dots, (f_n, c_n)\}$$

where f_i denotes a contributing feature and c_i denotes its contribution to the risk prediction.

This enables release teams to understand why a release was classified as high risk and provides an auditable basis for AI-assisted deployment decisions.

4. Research Methodology

The proposed research adopts a data-driven methodology to evaluate the effectiveness of the Adaptive Release Intelligence (ARI) framework for enterprise mobile application deployment. The methodology integrates release-level information, application telemetry, feature behavior, CI/CD indicators, and historical deployment outcomes to develop an AI-based release-risk prediction model. The predicted risk is subsequently mapped to adaptive deployment, feature-control, and recovery actions.

4.1 Data Collection

The dataset consists of historical enterprise mobile application release records collected from development, testing, deployment, and production environments. Each release instance is associated with operational and release-related attributes such as crash frequency, application latency, resource utilization, API failures, network errors, feature-level failures, code-change intensity, deployment history, testing outcomes, and release success or failure.

4.2 Data Preprocessing

The collected data are processed to ensure consistency and suitability for machine-learning analysis. The preprocessing procedure includes removal of duplicate observations, treatment of missing values, temporal synchronization, abnormal-value handling, and feature normalization.

For a numerical variable X , normalization is expressed as:

$$X' = \frac{X - \mu_X}{\sigma_X + \epsilon}$$

where μ_X represents the mean, σ_X represents the standard deviation, and ϵ is a small constant used to avoid division by zero.

4.3 Feature Engineering

The processed data are transformed into release-level features representing application reliability and deployment characteristics. The feature vector for release state t is defined as:

$$X_t = [C_t, L_t, M_t, C, P, U_t, N_t, F_t, D_t, T_t, H_t]$$

where C_t represents crash intensity, L_t latency, M_t memory utilization, CPU_t CPU utilization, N_t network/API failures, F_t feature-level reliability, D_t deployment characteristics, T_t CI/CD and testing information, and H_t historical release information.

Temporal features are also generated to capture changes following deployment. Crash growth is calculated as:

$$CG_t = \frac{C_t - C_{t-1}}{C_{t-1} + \epsilon}$$

Latency deviation is represented as:

$$LD_t = \frac{L_t - \mu_L}{\sigma_L + \epsilon}$$

These features help distinguish stable releases from releases exhibiting rapid degradation.

4.4 Predictive Model Development

The engineered features are supplied to different machine-learning algorithms to estimate release-failure probability. The candidate models include Random Forest, XGBoost, LightGBM, and Support Vector Machine.

The predictive function is expressed as:

$$P(Failure | X_t) = f(X_t)$$

where $P(Failure | X_t)$ denotes the probability of a release-related failure for the current application state.

Historical release outcomes are used as target information for supervised learning. Chronological separation of training, validation, and testing data is applied to reduce temporal information leakage.

4.5 Release Risk Estimation

The predicted failure probability is converted into a Release Risk Score (RRS):

$$RRS_t = 100 \times P(Failure | X_t)$$

The score provides a standardized representation of release risk from 0 to 100.

RRS Range	Risk Level	Decision
0–20	Low	Normal deployment
21–40	Moderate	Progressive rollout
41–60	Elevated	Restricted feature exposure
61–80	High	Pause and additional validation
81–100	Critical	Block or rollback

The thresholds are treated as configurable parameters and can be optimized using validation data.

4.6 Adaptive Deployment Decision

The RRS is used to determine the appropriate deployment strategy. Instead of applying a fixed deployment policy to every release, ARI dynamically selects an action according to the estimated risk. For a low-risk release, normal deployment is permitted. Moderate-risk releases undergo progressive rollout, while elevated-risk releases are maintained under restricted exposure and enhanced monitoring. High-risk releases are paused for further validation, and critical releases are blocked or rolled back.

The decision function can be represented as:

$$A_t = g(RRS_t)$$

where A_t represents the deployment action selected for release state t .

4.7 Dynamic Feature Exposure

Feature-level control is incorporated to reduce the impact of localized failures. The exposure level of feature f is represented as:

$$E_f(t) \in [0, 1]$$

A stable feature can gradually receive increased exposure, whereas a feature showing abnormal behavior can have its exposure reduced. This provides a containment mechanism without necessarily requiring complete application rollback.

4.8 Automated Recovery

ARI continuously evaluates post-deployment telemetry. If the release risk exceeds a predefined recovery threshold, the framework initiates an appropriate recovery process.

$$Recovery_t = \begin{cases} 1, & RRS_t \geq \tau_R \\ 0, & RRS_t < \tau_R \end{cases}$$

where τ_R represents the validated recovery threshold.

The recovery sequence consists of:

Risk Detection, Deployment Pause, Feature Restriction, Recovery Decision, Rollback, Verification, Controlled Redeployment

4.9 Explainability and Governance

The methodology incorporates explainability to identify the factors contributing to a release-risk prediction. The explanation can be represented as:

$$Explanation(RRS) = \{(f_1, c_1), (f_2, c_2), \dots, (f_n, c_n)\}$$

where f_i represents a contributing feature and c_i represents its contribution to the predicted risk.

For governance, each release decision records the risk score, major contributing factors, deployment action, observed outcome, and recovery decision. This creates a traceable relationship between AI prediction and deployment action.

4.10 Model Evaluation

The predictive models are evaluated using Accuracy, Precision, Recall, F1-score, ROC-AUC, and False Positive Rate.

$$\begin{aligned} Accuracy &= \frac{TP + TN}{TP + TN + FP + FN} \\ Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN} \\ F1 &= 2 \frac{Precision \times Recall}{Precision + Recall} \end{aligned}$$

Recall is particularly important because failure to identify a high-risk release can result in wider user exposure and increased recovery effort.

4.11 Comparative and Ablation Analysis

ARI is compared with conventional static CI/CD gates and individual machine-learning models. The comparison focuses not only on predictive accuracy but also on release-management outcomes such as failed-release rate, rollback frequency, recovery time, user exposure before failure detection, and manual intervention.

Ablation experiments are conducted by removing individual signal categories, including runtime telemetry, crash information, CI/CD information, feature-level telemetry, and network indicators. The resulting performance differences are used to determine the contribution of each information source to adaptive release-risk prediction.

5. Experimental Design and Performance Evaluation

The experimental design is developed to evaluate whether the proposed Adaptive Release Intelligence (ARI) framework can improve release-risk prediction and deployment decision-making compared with conventional static CI/CD gates and individual machine-learning approaches. The evaluation considers both predictive performance and operational effectiveness, with particular emphasis on early risk identification, adaptive deployment, feature exposure control, and automated recovery.

5.1 Experimental Dataset

The experimental dataset is organized around individual mobile application releases. Each release record contains information from the development, testing, deployment, and production stages. The dataset includes crash indicators, application latency, resource utilization, API and network failures,

feature-level errors, code-change characteristics, CI/CD outcomes, deployment history, and post-release failure information.

Each release is represented by a feature vector:

$$X_t = [C_t, L_t, M_t, C, P, U_t, N_t, F_t, D_t, T_t, H_t]$$

where the variables represent crash behavior, latency, memory, CPU utilization, network/API behavior, feature reliability, deployment characteristics, CI/CD information, and historical release behavior.

The corresponding target variable represents whether the release experienced a significant reliability problem within the defined observation period.

5.2 Experimental Models

Several models are evaluated to determine the effectiveness of the proposed approach. Conventional threshold-based monitoring is used as the primary baseline. Machine-learning baselines include Random Forest, XGBoost, LightGBM, and Support Vector Machine.

The proposed ARI model combines predictive risk estimation with adaptive deployment and recovery decisions. Therefore, the comparison evaluates not only classification capability but also the operational decisions generated by each approach.

5.3 Training and Testing Strategy

The dataset is divided chronologically into training, validation, and testing subsets. Chronological partitioning is preferred because future releases should not influence predictions for earlier releases.

The training set is used to develop the predictive model, while the validation set is used to optimize model parameters and risk thresholds. The final test set is reserved for unbiased evaluation.

The prediction process is represented as:

$$P_t = P(\text{Failure} \mid X_t)$$

followed by:

$$RRS_t = 100P_t$$

The resulting RRS is converted into an appropriate release-management action.

5.4 Evaluation Metrics

Predictive performance is evaluated using Accuracy, Precision, Recall, F1-score, ROC-AUC, and False Positive Rate.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \frac{Precision \times Recall}{Precision + Recall}$$

Recall is given particular importance because an incorrectly classified high-risk release can expose a large number of users to application failures.

ROC-AUC is used to evaluate the model's ability to distinguish between reliable and high-risk releases across different decision thresholds.

5.5 Baseline Comparison

The proposed framework is compared against conventional and machine-learning-based approaches.

Approach	Risk Prediction	Adaptive Rollout	Feature Control	Automated Recovery
Static CI/CD Gates	No	No	No	Limited
Threshold Monitoring	Limited	Limited	Limited	Rule-based

Random Forest	Yes	No	No	No
XGBoost	Yes	No	No	No
LightGBM	Yes	No	No	No
Proposed ARI	Yes	Yes	Yes	Yes

The comparison is intended to determine whether integrating prediction with deployment control provides advantages over using prediction or static gates independently.

5.6 Ablation Analysis

An ablation study is performed to determine the contribution of different information sources to the proposed framework. Several reduced versions of ARI are evaluated.

Model Variant	Removed Information
ARI-Runtime	Runtime performance signals
ARI-Crash	Crash and exception signals
ARI-CI/CD	CI/CD and deployment information
ARI-Feature	Feature-level telemetry
ARI-Network	Network/API indicators
Full ARI	No information category removed

The performance difference between the complete framework and each reduced configuration indicates the importance of the corresponding signal category.

5.7 Risk Threshold Evaluation

The effectiveness of different RRS thresholds is evaluated using validation data. Rather than permanently assuming fixed thresholds, the values are optimized according to the trade-off between missed failures and unnecessary deployment restrictions.

For example:

$$Action(RRS) = \begin{cases} Normal, & RRS < 20 \\ Progressive, & 20 \leq RRS < 40 \\ Restricted, & 40 \leq RRS < 60 \\ Pause, & 60 \leq RRS < 80 \\ Rollback, & RRS \geq 80 \end{cases}$$

The final thresholds should be selected based on experimental validation rather than assumed as universal values.

5.8 Adaptive Deployment Evaluation

The deployment component is evaluated by measuring whether ARI can reduce unnecessary exposure to high-risk releases. A release begins with limited exposure and progresses according to observed reliability.

The evaluation considers:

- deployment progression rate;
- percentage of users exposed before failure detection;
- number of releases paused;
- number of releases requiring rollback;
- successful release completion rate;
- detection time for emerging failures.

The objective is to determine whether adaptive rollout can provide better risk containment than immediate full deployment.

5.9 Feature Exposure Evaluation

Feature-level control is evaluated by measuring the ability of ARI to isolate unstable functionality.

For feature f , exposure is represented by:

$$E_f(t) \in [0, 1]$$

A successful control mechanism should increase $E_f(t)$ when reliability remains stable and decrease $E_f(t)$ when feature-related failures increase.

The primary evaluation measures include feature failure detection time, affected-user exposure, feature containment rate, and unnecessary application-wide rollback rate.

5.10 Automated Recovery Evaluation

The recovery mechanism is evaluated by measuring its ability to respond to high-risk releases without unnecessary delay.

The primary recovery metrics include:

$$MTTR = \frac{\sum Recovery\ Time}{Number\ of\ Incidents}$$

where MTTR represents Mean Time to Recovery.

Additional measures include rollback success rate, recovery decision accuracy, recovery latency, and percentage of incidents resolved without manual intervention.

5.11 Explainability Evaluation

The explainability component is evaluated by examining whether the major factors contributing to release risk can be identified.

For each prediction:

$$Explanation(RRS) = \{(f_1, c_1), (f_2, c_2), \dots, (f_n, c_n)\}$$

The analysis identifies whether crash growth, latency degradation, network failures, feature errors, deployment characteristics, or other variables contribute significantly to the predicted risk.

This information provides release engineers with a transparent basis for accepting, restricting, or rejecting a deployment.

5.12 Operational Performance Evaluation

In addition to predictive metrics, the overall ARI framework is evaluated using release-engineering measures.

Metric	Purpose
Failed Release Rate	Measures frequency of unsuccessful releases
User Exposure Before Detection	Measures potential impact before risk identification
Rollback Rate	Measures recovery requirements
Mean Time to Recovery	Measures recovery efficiency
Deployment Completion Rate	Measures successful deployment
Manual Intervention Rate	Measures automation effectiveness
Feature Containment Rate	Measures localized failure control

These measures enable evaluation of ARI as an end-to-end release-management system rather than merely as a classification model.

5.13 Expected Experimental Outcome

The expected outcome is that the proposed ARI framework will provide more adaptive release decisions than conventional static CI/CD gates. The framework is expected to identify high-risk releases earlier, restrict exposure when uncertainty increases, isolate problematic features, and initiate recovery actions when risk becomes critical.

However, numerical improvements should only be reported after conducting the proposed experiments. No empirical performance values are assumed in this study.

6. Results and Technical Discussion

This section presents the expected analytical behavior of the proposed Adaptive Release Intelligence (ARI) framework. Since the framework is proposed for experimental validation, numerical performance

values are not fabricated. The discussion focuses on the mathematical operation of the framework, comparative evaluation structure, risk-based release decisions, adaptive feature control, and recovery behavior.

6.1 Release Risk Prediction

For each release state t , the framework generates a feature vector:

$$X_t = [C_t, L_t, M_t, C, P, U_t, N_t, F_t, D_t, T_t, H_t]$$

The AI model processes these variables and estimates the probability of release failure:

$$P_t = P(\text{Failure} \mid X_t)$$

A higher value of P_t indicates a greater likelihood that the release may produce reliability problems after deployment.

Unlike a conventional CI/CD gate, which generally produces a predefined pass/fail decision, the proposed approach generates a continuous risk estimate. This allows release decisions to be adjusted according to changing application conditions.

6.2 Release Risk Score

The predicted probability is transformed into a Release Risk Score:

$$RRS_t = 100P_t$$

The score enables the release state to be categorized into different risk levels.

RRS	Risk Level	Recommended Action
0–20	Low	Normal deployment
21–40	Moderate	Progressive rollout
41–60	Elevated	Restricted feature exposure
61–80	High	Pause and additional validation
81–100	Critical	Block or rollback

The thresholds are configurable and should be optimized using validation experiments. The main advantage is that deployment decisions become proportional to estimated risk.

6.3 Multi-Signal Risk Assessment

ARI does not depend on a single application metric. Multiple signals are considered simultaneously to identify compound reliability risks.

A simplified risk representation can be expressed as:

$$R_t = w_1C_t + w_2L_t + w_3M_t + w_4N_t + w_5F_t + w_6D_t + w_7T_t$$

where w_i represents the contribution weight of each signal.

For example, a release with slightly increased latency may not necessarily represent a critical risk. However, if increased latency occurs simultaneously with crash growth, API failures, and high code-change intensity, the combined risk may increase significantly.

This multi-signal approach enables ARI to identify complex release conditions that may not be detected by individual static thresholds.

6.4 Temporal Risk Behavior

Release risk can change over time. Therefore, ARI evaluates temporal variations in application behavior.

Crash growth is represented as:

$$CG_t = \frac{C_t - C_{t-1}}{C_{t-1} + \epsilon}$$

Latency deviation is represented as:

$$LD_t = \frac{L_t - \mu_L}{\sigma_L + \epsilon}$$

A sudden increase in crash frequency or latency following deployment can therefore increase the predicted risk even if the absolute value remains within a conventional static threshold. This enables the framework to detect directional degradation rather than simply identifying whether a metric exceeds a fixed limit.

6.5 Adaptive Deployment Behavior

The major difference between ARI and conventional release management is the ability to modify deployment progression according to the predicted risk.

The deployment process can be represented as:

Release Candidate, Risk Prediction, Limited Deployment, Monitoring, Risk Reassessment, Deployment Expansion, Restriction, Rollback

For low-risk releases, deployment can progress normally. For moderate-risk releases, exposure can be increased gradually. Elevated-risk releases can remain under restricted deployment until sufficient evidence of stability is obtained.

This approach reduces the possibility of exposing the entire user population to an unstable release.

6.6 Dynamic Feature Exposure

ARI provides an additional control layer at the feature level.

Feature exposure is represented as:

$$E_f(t) \in [0, 1]$$

If feature f performs reliably:

$$E_f(t + 1) > E_f(t)$$

If feature-related errors increase:

$$E_f(t + 1) < E_f(t)$$

Therefore, the framework can reduce exposure to an unstable feature while retaining other stable application functions.

This provides a finer-grained response than complete application rollback.

6.7 Continuous Risk Reassessment

After deployment begins, new telemetry is continuously incorporated into the framework:

$$RRS_{t+1} = 100 \times f(X_{t+1})$$

The updated score determines whether deployment should continue, pause, expand, or reverse.

This creates an adaptive control mechanism in which deployment is continuously influenced by observed application behavior.

6.8 Automated Recovery

When release risk reaches the recovery threshold, ARI initiates an appropriate recovery action.

$$Recovery_t = \begin{cases} 1, & RRS_t \geq \tau_R \\ 0, & RRS_t < \tau_R \end{cases}$$

The recovery process follows:

Risk Detection, Deployment Pause, Feature Restriction, Rollback, Verification, Controlled Redeployment

The framework can also use persistence conditions to avoid triggering rollback because of short-lived telemetry fluctuations.

6.9 Comparative Analysis

The expected functional comparison between conventional release management and ARI is presented below.

Capability	Static CI/CD	Threshold Monitoring	ML Prediction	Proposed ARI
Release risk prediction	No	Limited	Yes	Yes
Continuous reassessment	Limited	Yes	Possible	Yes
Adaptive rollout	No	Limited	No	Yes
Feature-level control	No	Limited	No	Yes
Automated rollback	Limited	Rule-based	No	Yes
Explainable decision support	Limited	Limited	Possible	Yes
Feedback learning	Limited	Limited	Possible	Yes

The table demonstrates that ARI combines predictive intelligence with operational control rather than treating prediction as an isolated activity.

6.10 Ablation Analysis

Ablation analysis evaluates the contribution of individual data categories. The complete ARI model is compared with reduced configurations:

$$ARI_{Runtime}$$

removes runtime performance information,

$$ARI_{Crash}$$

removes crash-related information,

$$ARI_{CI/CD}$$

removes deployment and CI/CD information,

$$ARI_{Feature}$$

removes feature-level information, and

$$ARI_{Network}$$

removes network/API information.

A reduction in predictive performance after removing a particular feature group indicates that the corresponding information contributes meaningfully to release-risk estimation.

6.11 Explainability Analysis

For every risk prediction, the framework identifies the major factors contributing to the RRS.

$$Explanation(RRS) = \{(f_1, c_1), (f_2, c_2), \dots, (f_n, c_n)\}$$

For example, the dominant contributors to an elevated risk score could include increased crash growth, latency degradation, API failures, or high deployment-change intensity.

The explanation enables release engineers to understand the reason behind an AI-generated decision instead of treating the model as a completely opaque mechanism.

6.12 Release-Level Decision Chain

The complete operational behavior of ARI can be summarized as:

Release, Feature Extraction, AI Prediction, RRS Generation, Risk Classification, Deployment Decision, Feature Control, Monitoring, Recovery, Feedback

This decision chain represents the principal contribution of the proposed framework. Conventional release engineering generally separates these activities, whereas ARI integrates them into a single adaptive intelligence loop.

6.13 Overall Technical Discussion

The proposed framework changes the fundamental release-management strategy from a binary deployment decision to a continuous risk-control process. A release is not considered permanently safe or unsafe based only on pre-deployment testing. Instead, its risk is continuously reassessed as new operational evidence becomes available.

The framework consequently supports three levels of intelligence:

1. **Predictive intelligence** — identifies potential release failures.
2. **Adaptive intelligence** — modifies deployment and feature exposure according to risk.
3. **Recovery intelligence** — initiates appropriate rollback and recovery actions when risk becomes unacceptable.

The integration of these capabilities provides a foundation for transforming enterprise mobile application deployment into a more predictive, controlled, and continuously improving release-engineering process.

Conclusion

The proposed Adaptive Release Intelligence (ARI) framework provides an AI-driven approach for improving the deployment and management of enterprise mobile applications. Unlike conventional CI/CD processes that primarily depend on predefined and static quality gates, ARI continuously evaluates application, release, feature, deployment, and operational information to estimate the risk associated with a software release. The framework introduces a Release Risk Score (RRS) that converts predicted failure probability into an interpretable risk measure. This risk information is used to dynamically determine deployment progression, feature exposure, monitoring intensity, deployment pauses, rollback, and recovery actions. The integration of feature-level control provides an additional mechanism for containing localized failures without necessarily reverting the complete application release. Another important contribution of ARI is its continuous feedback mechanism. Production outcomes are incorporated into subsequent risk assessments, allowing the release-intelligence process to progressively improve. The explainability and governance layer further supports transparent and traceable AI-assisted release decisions. Overall, ARI establishes a transition from static release gates to predictive and adaptive release engineering. The proposed framework integrates risk prediction, adaptive deployment, feature control, automated recovery, explainability, and continuous learning into a unified lifecycle. Future experimental validation can determine the quantitative improvements in release reliability, failure detection, deployment efficiency, user exposure reduction, and recovery time achieved by the framework.

References:

- [1] Somasekaram, P.; Calinescu, R.; Buyya, R. High-availability clusters: A taxonomy, survey, and future directions. *Journal of Systems and Software*, 2022, 187, 111208.
- [2] Pereira, P.; Araujo, J.; Melo, C.; Santos, V.; Maciel, P. Analytical models for availability evaluation of edge and fog computing nodes. *Journal of Supercomputing*, 2021, 77, 9905–9933.
- [3] Sangolli, D.R.; Ravindrarao, N.M.; Patil, P.C.; Palissery, T.; Liu, K. Enabling high availability edge computing platform. *Proceedings of the 2019 IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, 2019, pp. 85–92.
- [4] Liu, X.; Cheng, B.; Wang, S. Availability-aware and energy-efficient virtual cluster allocation based on multi-objective optimization in cloud datacenters. *IEEE Transactions on Network and Service Management*, 2020, 17, 972–985.
- [5] Toumi, N.; Bagaa, M.; Ksentini, A. Machine learning for service migration: A survey. *IEEE Communications Surveys & Tutorials*, 2023, 25, 1991–2020.
- [6] Endo, P.T.; Rodrigues, M.; Gonçalves, G.E.; Kelner, J.; Sadok, D.H.; Curescu, C. High availability in clouds: Systematic review and research challenges. *Journal of Cloud Computing*, 2016, 5, 16.
- [7] Maciel, P.; Dantas, J.; Melo, C.; Pereira, P.; Oliveira, F.; Araujo, J.; Matos, R. A survey on reliability and availability modeling of edge, fog, and cloud computing. *Journal of Reliability and Intelligent Environments*, 2022, 8, 227–245.
- [8] Hasan, M.; Goraya, M.S. Fault tolerance in cloud computing environment: A systematic survey. *Computers & Industrial Engineering*, 2018, 99, 156–172.

- [9] Kitchenham, B. Procedures for Performing Systematic Reviews; Technical Report; Keele University: Keele, UK, 2004.
- [10] Tran, D.H.; Nguyen, V.L.; Nguyen, H.; Jang, Y.M. Self-supervised learning for time-series anomaly detection in Industrial Internet of Things. *Electronics*, 2022, 11, 2146.
- [11] Kalaskar, C.; Thangam, S. Fault tolerance of cloud infrastructure with machine learning. *Cybernetics and Information Technologies*, 2023, 23, 26–50.
- [12] Gollapalli, M.; Al Metrik, M.; Alnajrani, B.; Alomari, A.; AlDawoud, S.; Almunsour, Y.; Abdulqader, M.; Aloup, K. Task failure prediction using machine learning techniques in the Google Cluster Trace cloud computing environment. *Mathematical Modelling of Engineering Problems*, 2022, 9, 545–553.
- [13] Carter, A.; Imtiaz, S.; Naterer, G.F. Review of interpretable machine learning for process industries. *Process Safety and Environmental Protection*, 2023, 170, 647–659.
- [14] Odyurt, U.; Pimentel, A.D.; Gonzalez Alonso, I. Improving the robustness of industrial Cyber–Physical Systems through machine learning-based performance anomaly identification. *Journal of Systems Architecture*, 2022, 131, 102716.
- [15] Mohammed, B.; Awan, I.; Ugail, H.; Younas, M. Failure prediction using machine learning in a virtualised HPC system and application. *Cluster Computing*, 2019, 22, 471–485.