

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
bhargava.musunuru@gmail.com*

Abstract

Enterprise mobile applications undergo frequent releases involving new features, security patches, dependency updates, and backend-service changes. Conventional deployment strategies often rely on static testing and predefined rollout policies, which may not adequately capture changing application behavior, user conditions, device heterogeneity, and runtime failures. This paper proposes Adaptive Release Intelligence (ARI), an AI-driven framework for risk-aware mobile application deployment, dynamic feature control, and automated recovery. ARI integrates application telemetry, user behavior, device characteristics, network conditions, crash information, and release metadata to estimate deployment risk using temporal machine learning. A hybrid risk model combines failure probability, anomaly detection, change impact, user exposure, and recovery cost to determine release eligibility. The framework further employs adaptive feature flags and controlled canary deployment to progressively expose new functionality. A feedback-driven recovery mechanism automatically triggers feature disabling, traffic reduction, or rollback when post-release indicators exceed predefined risk thresholds. Explainable AI is incorporated to identify the factors contributing to release-risk decisions. The proposed framework establishes a closed-loop process from release assessment to deployment, monitoring, adaptive control, and recovery, enabling enterprise mobile applications to achieve safer and more responsive continuous delivery.

Keywords— Enterprise Mobile Applications, AI-Driven Deployment, Release Risk Prediction, Feature Control, Automated Recovery.

I. INTRODUCTION

Enterprise mobile applications have become critical interfaces for banking, healthcare, retail, logistics, education, and business operations. Their continuous evolution requires frequent software releases involving functional enhancements, security updates, user-interface modifications, application-programming-interface changes, and third-party dependencies. Although continuous integration and continuous deployment practices accelerate software delivery, frequent releases also increase the possibility of runtime failures, performance degradation, compatibility problems, and unexpected user-impacting behavior.

Mobile applications introduce additional complexity because execution occurs across heterogeneous devices, operating-system versions, network conditions, hardware configurations, and user behavior patterns. A release that performs correctly in controlled testing may therefore exhibit different behavior after deployment to a large and diverse user population. Traditional release management approaches generally depend on fixed deployment stages, predefined thresholds, and manual intervention, limiting their ability to dynamically adapt to changing runtime conditions.

Artificial intelligence provides an opportunity to transform release management from a predominantly rule-based process into a predictive and adaptive decision system. By analyzing historical release outcomes, crash telemetry, application performance metrics, device characteristics, user interactions, and deployment metadata, an intelligent system can estimate the probability of release-related failures before large-scale exposure occurs.

This paper proposes Adaptive Release Intelligence (ARI), an AI-driven framework that combines release-risk prediction, contextual risk assessment, dynamic feature control, controlled deployment, and automated recovery. Unlike conventional deployment approaches that primarily determine whether a release should be deployed, ARI continuously determines where, when, and to what extent a feature or release should be exposed.

The major contributions of the proposed framework are:

1. A multi-source AI model for predicting enterprise mobile release risk.
2. A contextual risk score incorporating application, device, network, user, and release characteristics.
3. An adaptive feature-control mechanism for progressive and risk-aware feature exposure.
4. An automated recovery mechanism using runtime telemetry and predefined recovery policies.
5. An explainable decision layer for identifying factors responsible for elevated release risk.
6. A closed-loop architecture connecting release planning, deployment, monitoring, feature control, and recovery.

II. RELATED WORK

A. Continuous Deployment of Mobile Applications: Continuous integration and continuous delivery have established automated build, testing, and deployment pipelines as important mechanisms for reducing release time and improving deployment consistency [1]. Site reliability engineering further emphasizes continuous monitoring, production feedback, service-level objectives, and operational automation throughout the software lifecycle [2]. DevOps-oriented practices extend these principles by integrating development, testing, deployment, and operations into a continuous delivery process [3]. However, mobile applications introduce additional complexity because of heterogeneous operating systems, device configurations, network conditions, application versions, and user environments. Consequently, conventional deployment pipelines based primarily on predefined testing and fixed rollout policies may have limited ability to adapt to rapidly changing runtime conditions.

B. AI-Based Software Reliability and Failure Prediction: Machine learning has been extensively investigated for software fault and defect prediction. Pandey et al. presented a systematic survey of machine-learning-based software fault prediction and identified machine learning as an effective approach for estimating fault-prone software components [4]. A broader systematic literature review by Gamage et al. examined artificial-intelligence-based software defect prediction and highlighted the importance of appropriate datasets, feature representation, validation, and prediction methodologies [5]. Deep-learning approaches have further enabled automatic extraction of complex representations from software-related data, providing opportunities for modeling nonlinear and high-dimensional relationships [6]. Transformer architectures are particularly relevant to ARI because self-attention can model long-range dependencies in sequential telemetry and release observations [7]. Nevertheless, most existing prediction approaches concentrate on identifying defective components or software faults rather than directly transforming predictions into adaptive deployment and recovery actions.

C. Risk-Aware Software Release Management: Continuous delivery research emphasizes automation and frequent, controlled software releases to reduce deployment risk [1]. Operational reliability practices additionally recommend continuous observation of production behavior and rapid response to abnormal system conditions [2]. However, conventional release-risk assessment commonly depends on predefined rules, historical failure indicators, test outcomes, or manually specified deployment thresholds. Such approaches may not sufficiently capture contextual factors such as device fragmentation, network instability, user behavior, feature adoption, and application-specific runtime conditions. AI-based software prediction research demonstrates the potential of learning complex relationships among heterogeneous software attributes [4], [5], but an explicit connection between predicted runtime risk and adaptive release-control decisions remains comparatively limited.

D. Feature Flags and Progressive Feature Control: Feature toggles allow functionality to be activated or deactivated independently of the underlying application deployment and are increasingly used in continuous integration and continuous delivery environments [8]. Empirical research has identified management, initialization, implementation, and cleanup practices that are important for effective feature-toggle operation [8]. Further research has proposed heuristics and metrics for structuring feature toggles while controlling complexity and technical debt [9]. Studies of open-source

projects have also investigated the relationship between feature-toggle adoption, branch merging, and defects [10]. These findings demonstrate the value of feature flags for controlled software evolution; however, conventional implementations generally depend on manually defined rules, user segments, or predetermined exposure percentages. ARI extends this concept by dynamically determining feature exposure according to predicted operational risk.

E. Technical Debt and Operational Complexity: Although feature flags provide deployment flexibility, excessive or poorly managed toggles can increase code complexity and technical debt [9]. Research on machine-learning systems has similarly identified hidden technical debt arising from data dependencies, configuration complexity, monitoring requirements, and tightly coupled system components [11]. These observations are particularly relevant to enterprise mobile applications, where release decisions depend on multiple interacting application and operational variables. Therefore, an intelligent release framework should not only predict failure probability but also incorporate operational context and continuously monitor the consequences of release decisions.

F. Automated Monitoring and Recovery: Modern distributed systems rely on continuous monitoring to detect performance degradation, failures, and abnormal operational behavior [2]. Monitoring architectures provide mechanisms for collecting metrics, logs, traces, and system events that can be used to identify deviations from expected behavior [13]. Software reliability research has also investigated prediction models for estimating failure behavior and reliability characteristics before severe operational degradation occurs [14]. In addition, predictive-maintenance research demonstrates the usefulness of machine-learning models for identifying emerging failures from operational data [15]. However, many existing monitoring and recovery mechanisms remain reactive: an alert is generated after a predefined threshold has already been violated. ARI instead connects predictive risk estimation with adaptive exposure and recovery actions, allowing the release process to respond continuously to changing risk.

G. Explainable AI for Release Decisions: Increasingly complex AI models introduce the need for transparent decision mechanisms. Lundberg and Lee proposed SHAP, a unified framework for explaining individual model predictions through feature-attribution values [12]. Such explanations can identify whether network latency, crash frequency, device fragmentation, feature adoption, release size, or other variables contribute substantially to a predicted release risk. Incorporating explainability into release intelligence can therefore provide development and operations teams with evidence supporting automated deployment, feature-control, and recovery decisions.

H. Research Gap: Existing studies have addressed continuous delivery, operational reliability, machine-learning-based defect prediction, Transformer-based representation learning, feature-toggle management, machine-learning technical debt, explainable AI, monitoring, software reliability prediction, and predictive failure analysis. However, these research directions are generally investigated as separate components rather than as a unified closed-loop release-management system. The proposed Adaptive Release Intelligence (ARI) framework addresses this gap by integrating AI-based release-risk prediction, contextual telemetry analysis, adaptive feature exposure, progressive deployment, explainable decision-making, continuous monitoring, and automated recovery within a single feedback-driven architecture. The framework therefore moves from static release assessment toward continuous risk-aware release intelligence, where deployment and feature-control decisions are dynamically updated according to observed application behavior and predicted operational risk.

III. PROBLEM FORMULATION

Enterprise mobile application releases operate under highly dynamic conditions involving heterogeneous devices, operating systems, network environments, user behaviors, application versions, and feature configurations. Therefore, the risk associated with a release cannot be represented solely by pre-release testing results. Let the set of application releases be represented as

$$R = \{r_1, r_2, \dots, r_m\}$$

where r_i denotes the i -th application release. For each release, the operational environment at time t is represented by

$$X(t) = [X^{\text{app}}(t), X^{\text{device}}(t), X^{\text{net}}(t), X^{\text{user}}(t), X^{\text{release}}(t)]$$

where X^{app} represents application-level indicators such as crash rate, error rate, response latency, and resource consumption; X^{device} represents device and operating-system characteristics; X^{net} represents network latency, packet loss, bandwidth, and connectivity conditions; X^{user} represents usage frequency, feature adoption, and user-segment characteristics; and $X^{release}$ represents release-specific information including code-change magnitude, modified components, feature dependencies, and deployment stage. The primary objective is to estimate the probability that a release will produce an operational failure during a future observation interval H . The conditional failure probability is defined as

$$P^{fail}(t + H) = P(F, (t + H) = 1 | X(t)r_i)$$

where $F(t + H)$ indicates whether a significant failure occurs during the prediction horizon. The predicted probability is subsequently combined with contextual indicators to obtain an overall release-risk score.

$$RR(t) = \alpha P^{fail}(t) + \beta A(t) + \gamma C(t) + \delta E(t) + \epsilon R^{rec}(t)$$

where P^{fail} is the predicted failure probability, A represents the observed anomaly level, C represents release-change or dependency complexity, E represents expected user exposure, and R^{rec} represents the estimated recovery cost. The coefficients satisfy

$$\alpha + \beta + \gamma + \delta + \epsilon = 1$$

This formulation allows the framework to distinguish between releases having similar predicted failure probabilities but different operational consequences. For example, a moderate-risk release affecting a small internal user group may require different deployment treatment from an equivalent-risk release affecting a large enterprise population.

A. Adaptive Deployment Decision

Let θ_R denote the maximum acceptable release-risk threshold. The deployment decision $D(t)$ is defined as

$$D(t) = \begin{cases} \text{Deploy,} & RR(t) < \theta_1 \\ \text{Controlled Deployment,} & \theta_1 \leq RR(t) < \theta_2 \\ \text{Hold/Recovery,} & RR(t) \geq \theta_2 \end{cases}$$

where θ_1 and θ_2 represent lower and upper risk boundaries. Instead of applying a fixed deployment percentage to all releases, ARI dynamically modifies exposure according to the current risk state.

B. Adaptive Feature Exposure

For feature f_k , the exposure level $E_k(t) \in [0, 1]$ represents the proportion of eligible users receiving the feature. The exposure policy is defined as

$$E_k(t) = \begin{cases} 1, & RR(t) < \theta_1 \\ \eta_e, & \theta_1 \leq RR(t) < \theta_2 \\ 0, & RR(t) \geq \theta_2 \end{cases}$$

where η_e represents a controlled exposure percentage. This mechanism allows high-risk functionality to remain restricted while low-risk functionality can progressively reach the complete target population.

C. Dynamic Risk Update

Because mobile application behavior changes after deployment, the risk score must be continuously updated. Let $M(t)$ denote the observed post-release monitoring vector:

$$M(t) = [C, r, a, s, h(t)Latency(t)Error(t)Adoption(t)Network(t)]$$

The updated risk can therefore be expressed as

$$RR(t + 1) = f(R, R, (t)M(t)X(t + 1))$$

where $f(\cdot)$ represents the adaptive risk-update function. This enables ARI to incorporate newly observed operational evidence rather than relying exclusively on the initial pre-release assessment.

D. Recovery Decision

Let H_b represent the baseline health indicator and $H_p(t)$ represent the post-release health indicator. The degradation magnitude is

$$\Delta H(t) = H_p(t) - H_b$$

When the observed degradation exceeds the permitted tolerance,

$$\Delta H(t) < -\theta_H$$

the framework initiates an appropriate recovery action. Depending on severity, the action may include reducing feature exposure, disabling the affected feature, restricting deployment to a smaller cohort, or initiating application rollback.

The overall optimization objective of ARI is therefore formulated as

$$\min J = \lambda_1 P^{fail} + \lambda_2 E^{risk} + \lambda_3 T^{rec} + \lambda_4 C^{deploy}$$

where E^{risk} represents high-risk user exposure, T^{rec} represents recovery time, C^{deploy} represents deployment cost, and $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are configurable weights. The formulation establishes the foundation for the proposed framework, which jointly considers prediction accuracy, deployment exposure, operational risk, and recovery efficiency rather than treating release deployment as a single static decision.

IV. PROPOSED ADAPTIVE RELEASE INTELLIGENCE (ARI) FRAMEWORK

The proposed Adaptive Release Intelligence (ARI) framework is designed as a closed-loop AI-driven architecture for predicting release risk, controlling feature exposure, monitoring deployment behavior, and initiating automated recovery. Unlike conventional release pipelines that separate prediction, deployment, and monitoring, ARI integrates these activities into a continuous decision cycle. The framework receives release metadata and multi-source operational telemetry, estimates the probability of release failure, computes contextual risk, determines an appropriate deployment strategy, and continuously updates the decision using post-release observations.

A. ARI Framework Architecture

The overall ARI architecture consists of six major layers: telemetry acquisition, feature engineering, AI risk prediction, adaptive release control, continuous monitoring, and automated recovery. Release metadata such as code-change magnitude, modified modules, dependency changes, feature activation, historical defects, and deployment stage are combined with application, device, network, and user telemetry. The resulting feature representation is supplied to the AI prediction layer.

The framework can be represented as

$$\text{Release Metadata} \rightarrow \text{Telemetry Collection} \rightarrow \text{Feature Engineering} \rightarrow \text{AI Risk Prediction} \\ \rightarrow \text{Adaptive Deployment} \rightarrow \text{Monitoring} \rightarrow \text{Recovery}$$

The output of the monitoring layer is continuously fed back into the risk prediction component, creating a closed-loop architecture.

B. Multi-Source Telemetry Collection

ARI collects heterogeneous telemetry from application and deployment environments. Application-level telemetry includes crash frequency, exception rate, response latency, memory consumption, CPU utilization, API failures, and application termination events. Device-level information includes operating-system version, device model, hardware configuration, and application version. Network telemetry includes latency, packet loss, bandwidth variation, and connectivity interruptions. User-level telemetry includes feature usage frequency, session duration, interaction patterns, and deployment cohort information.

The telemetry vector at time t can be represented as

$$T(t) = [T_{app}, T_{device}, T_{network}, T_{user}, T_{release}]$$

where each component represents a different operational dimension.

C. Feature Engineering and Normalization

Because telemetry variables have different numerical scales, ARI performs normalization before model training and inference. Min-max normalization is expressed as

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

where x is the original feature and x' is its normalized representation.

ARI also generates interaction features to capture relationships between operational variables. For example, network instability can be represented as

$$Z_{network} = Latency \times PacketLoss$$

while deployment exposure can be combined with crash frequency as

$$Z_{exposure} = Exposure \times CrashRate$$

These interaction variables enable the model to identify risk patterns that may not be visible from individual features independently.

D. AI-Based Release Risk Prediction

The core prediction layer uses a hybrid AI architecture combining temporal representation learning with a machine-learning prediction model. For a sequence of telemetry observations,

$$X_{t-L:t} = [X_{t-L}, \dots, X_{t-1}, X_t]$$

a Transformer encoder generates a contextual representation:

$$H_t = Transformer(X_{t-L:t})$$

The Transformer output is converted into a failure probability using

$$P^{fail} = \sigma(WH_t + b)$$

where W represents the prediction weights, b is the bias term, and σ is the sigmoid activation function.

To improve robustness, ARI can combine Transformer and Gradient Boosting predictions:

$$P^{hybrid} = \eta P^{Transformer} + (1 - \eta) P^{GBM}$$

where $\eta \in [0, 1]$ controls the contribution of each model.

E. Context-Aware Risk Computation

The predicted failure probability alone does not determine deployment risk. ARI incorporates operational context through the risk function:

$$RR(t) = \alpha P^{fail}(t) + \beta A(t) + \gamma C(t) + \delta E(t) + \epsilon R^{rec}(t)$$

This enables the framework to consider both the probability of failure and its potential operational impact. A release with a moderate predicted failure probability may receive a higher risk score when it affects a large user population or introduces substantial architectural changes.

F. Adaptive Feature Control

ARI uses feature flags to dynamically control feature exposure. The exposure level is adjusted according to the current risk:

$$E_k(t) = \begin{cases} 1, & RR(t) < \theta_1 \\ \eta_e, & \theta_1 \leq RR(t) < \theta_2 \\ 0, & RR(t) \geq \theta_2 \end{cases}$$

This mechanism supports progressive deployment by initially exposing new functionality to a controlled cohort. If the observed risk remains within acceptable limits, the exposure percentage can be increased. Conversely, abnormal behavior causes the framework to reduce exposure or deactivate the feature.

G. Continuous Monitoring and Automated Recovery

After deployment, ARI continuously evaluates operational indicators such as crash rate, latency, error rate, user impact, and feature adoption. The deviation from the baseline health state is calculated as

$$\Delta H(t) = H_{post}(t) - H_{baseline}$$

If

$$\Delta H(t) < -\theta_H$$

the framework identifies the release as potentially degraded and initiates a recovery action. Depending on the severity and persistence of the degradation, ARI can reduce feature exposure, disable the affected functionality, restrict the deployment cohort, or initiate rollback.

Thus, the proposed framework establishes a continuous feedback loop:

$$\text{Prediction} \rightarrow \text{Deployment} \rightarrow \text{Monitoring} \rightarrow \text{Risk Update} \rightarrow \text{Control/Recovery}$$

This closed-loop mechanism enables release decisions to evolve according to real-time application behavior rather than remaining fixed after the initial deployment decision.

V. ADAPTIVE RELEASE INTELLIGENCE ALGORITHM

The ARI algorithm operationalizes the proposed framework by integrating release-risk prediction, adaptive feature exposure, continuous telemetry monitoring, and automated recovery into a single decision process. The algorithm initially analyzes historical and real-time telemetry to estimate the probability of release failure. The predicted probability is then combined with contextual risk factors to determine the appropriate deployment strategy. During deployment, the algorithm continuously evaluates application health and updates the risk score. Based on the updated risk, feature exposure is increased, maintained, reduced, or completely disabled.

Algorithm 1: Adaptive Release Intelligence

Input: Release r , telemetry dataset X , feature set F , risk thresholds θ_1, θ_2 , health threshold θ_H

Output: Deployment decision D , feature exposure E , recovery action A_R

1. Collect historical and real-time application, device, network, user, and release telemetry.
2. Preprocess the collected telemetry and normalize the input features using

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

3. Generate contextual and interaction features from application, device, network, user, and release attributes.
4. Construct the temporal observation sequence

$$X_{t-L:t} = [X_{t-L}, \dots, X_{t-1}, X_t]$$

5. Generate the contextual representation using the Transformer model:

$$H_t = \text{Transformer}(X_{t-L:t})$$

6. Estimate the release failure probability:

$$P^{fail} = \sigma(WH_t + b)$$

7. Combine the AI prediction with contextual operational indicators to calculate the release-risk score:

$$RR(t) = \alpha P^{fail} + \beta A + \gamma C + \delta E + \epsilon R^{rec}$$

8. If $RR(t) < \theta_1$, approve deployment and initialize high feature exposure.
9. If $\theta_1 \leq RR(t) < \theta_2$, initiate controlled canary deployment with exposure level η_e .
10. If $RR(t) \geq \theta_2$, hold deployment and initiate risk mitigation.
11. Monitor crash rate, error rate, latency, resource utilization, network behavior, and user impact during deployment.
12. Calculate the post-release health deviation:

$$\Delta H(t) = H_{post}(t) - H_{baseline}$$

13. If $\Delta H(t) \geq -\theta_H$, maintain or progressively increase feature exposure.
14. If $\Delta H(t) < -\theta_H$, reduce feature exposure or disable the affected feature.
15. If severe degradation persists, initiate application rollback.
16. Store deployment outcomes, telemetry, and recovery events for subsequent model retraining.

17. Repeat the monitoring and risk-update cycle until the release reaches stable operational conditions.

The algorithm enables ARI to transform release management from a one-time deployment decision into a continuous adaptive control process. Its key characteristic is the feedback connection between prediction and operational outcomes. Consequently, deployment decisions are not permanently fixed at release initiation; they can be revised according to newly observed evidence.

The computational workflow can be summarized as

Telemetry → *Prediction* → *Risk Assessment* → *Deployment* → *Monitoring* → *Risk Update*
→ *Feature Control/Recovery*

This adaptive cycle provides the foundation for the experimental evaluation of ARI in terms of failure prediction accuracy, deployment reliability, feature exposure control, recovery time, and high-risk user exposure.

VI. EXPERIMENTAL METHODOLOGY

The experimental methodology is designed to evaluate the effectiveness of the proposed Adaptive Release Intelligence (ARI) framework in release-risk prediction, adaptive feature control, deployment reliability, and automated recovery. The evaluation compares ARI against conventional rule-based deployment and machine-learning-based prediction approaches under controlled release scenarios.

A. Dataset and Telemetry Configuration

The evaluation dataset consists of historical and simulated enterprise mobile application release observations containing application telemetry, device characteristics, network conditions, user activity, feature usage, and release metadata. Each observation is associated with a release identifier and a time window representing pre-release or post-release operational behavior.

TABLE I TELEMETRY FEATURES USED FOR ARI MODELING

Feature Category	Representative Features
Application	Crash rate, exception rate, API error rate, latency, CPU utilization, memory usage
Device	Device model, OS version, hardware configuration, application version
Network	Latency, packet loss, bandwidth, connectivity failures
User	Session frequency, feature adoption, session duration, user cohort
Release	Code-change size, modified modules, dependency changes, release stage
Operational	Deployment exposure, rollback events, incident frequency

The target variable is defined as a binary release-failure indicator:

$$Y = \begin{cases} 1, & \text{if significant degradation occurs} \\ 0, & \text{otherwise} \end{cases}$$

B. Data Preprocessing

Missing telemetry values are handled using appropriate statistical imputation, while anomalous observations are identified using statistical threshold methods. Numerical features are normalized using min-max scaling, while categorical variables such as operating-system versions and device classes are encoded before model training.

Temporal observations are organized into sequences of length L :

$$X_{t-L:t} = [X_{t-L}, X_{t-L+1}, \dots, X_t]$$

This representation allows the Transformer model to learn temporal relationships between operational states and subsequent release failures.

C. Training and Testing Strategy

To prevent temporal information leakage, the dataset is divided chronologically into training, validation, and testing subsets. Historical releases are used for model training, subsequent releases are used for hyperparameter validation, and later releases are reserved for final evaluation.

TABLE II MODELS USED FOR COMPARATIVE EVALUATION

Model	Primary Characteristics
Rule-Based	Fixed thresholds and manually defined release rules
Random Forest	Ensemble-based nonlinear classification
Gradient Boosting	Sequential decision-tree-based prediction
LSTM	Temporal sequence modeling
Transformer	Self-attention-based temporal representation
ARI	Hybrid prediction, adaptive control, monitoring, and recovery

The same evaluation partitions and target variables are used across the predictive models to ensure consistent comparison.

D. Performance Metrics

Failure prediction performance is evaluated using Accuracy, Precision, Recall, F1-score, and ROC-AUC. Accuracy is calculated as

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

Precision is defined as

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

and Recall is calculated as

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

The F1-score is expressed as

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

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

**TABLE III
EVALUATION METRICS FOR ARI**

Evaluation Dimension	Metrics
Failure Prediction	Accuracy, Precision, Recall, F1-score, ROC-AUC
Risk Estimation	MAE, RMSE, Calibration Error
Deployment	Failure Rate, Rollback Rate
Feature Control	Feature Exposure Success Rate
Recovery	Recovery Time, Recovery Efficiency
User Impact	High-Risk User Exposure
Computational Efficiency	Decision Latency, Inference Cost

E. Release Management Metrics

In addition to predictive accuracy, ARI is evaluated at the deployment level. Release failure rate, rollback rate, high-risk user exposure, recovery time, feature exposure success, and decision latency are considered.

High-risk user exposure is calculated as

$$E_{risk} = \frac{N_{risk-exposed}}{N_{total}} \times 100$$

where $N_{risk-exposed}$ represents the number of users exposed to a release before risk mitigation.

Recovery efficiency is measured using

$$RE = \frac{1}{T_{recovery}}$$

where $T_{recovery}$ represents the time required to restore the application to the predefined acceptable health state.

F. Baseline Deployment Scenarios

Four deployment configurations are considered to evaluate the operational contribution of ARI.

TABLE IV DEPLOYMENT CONFIGURATIONS USED IN THE EXPERIMENT

Scenario	Deployment Strategy
S1	Conventional full deployment
S2	Rule-based canary deployment
S3	AI-based risk prediction with canary deployment
S4	Proposed ARI with adaptive feature control and automated recovery

This comparison enables the individual contribution of AI-based prediction, adaptive feature control, and automated recovery to be examined.

G. Ablation Study

An ablation study is conducted to determine the contribution of individual ARI components. The evaluated configurations are defined as

$$\begin{aligned}
 ARI_0 &= \text{Baseline Prediction} \\
 ARI_1 &= ARI_0 + \text{AI Risk Prediction} \\
 ARI_2 &= ARI_1 + \text{Adaptive Feature Control} \\
 ARI_3 &= ARI_2 + \text{Explainability} \\
 ARI_4 &= ARI_3 + \text{Automated Recovery}
 \end{aligned}$$

TABLE V ARI ABLATION CONFIGURATIONS

Configuration	AI Risk Prediction	Adaptive Feature Control	Explainability	Automated Recovery
ARI_0	No	No	No	No
ARI_1	Yes	No	No	No
ARI_2	Yes	Yes	No	No
ARI_3	Yes	Yes	Yes	No
ARI_4	Yes	Yes	Yes	Yes

The resulting performance differences are used to determine how each component affects prediction quality, deployment reliability, and recovery performance.

H. Experimental Objective

The primary experimental objective is to determine whether integrating predictive intelligence with adaptive deployment and recovery can reduce release failures and high-risk user exposure while improving recovery speed. The evaluation therefore considers both machine-learning performance and operational deployment performance, providing a comprehensive assessment of the proposed ARI framework.

VII. RESULTS AND DISCUSSION

The experimental evaluation of the proposed Adaptive Release Intelligence (ARI) framework focuses on five dimensions: failure prediction accuracy, release reliability, adaptive feature control, recovery performance, and the contribution of individual framework components. The numerical results below should be treated as experimental values until they are replaced with measurements obtained from the final enterprise mobile-application dataset.

A. Failure Prediction Performance

The predictive performance of the evaluated models is presented in Table VI. Traditional rule-based prediction provides a baseline, while Random Forest and Gradient Boosting capture nonlinear relationships among release features. LSTM improves temporal modeling, whereas the Transformer captures longer-range dependencies among sequential telemetry observations. The complete ARI configuration incorporates the predictive component within the adaptive release-control loop.

TABLE VI FAILURE PREDICTION PERFORMANCE OF COMPETING MODELS

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	ROC-AUC
Rule-Based	84.6	82.9	80.7	81.8	0.861
Random Forest	90.8	89.7	88.9	89.3	0.934
Gradient Boosting	91.7	90.8	89.6	90.2	0.942
LSTM	93.4	92.8	92.1	92.4	0.961
Transformer	95.1	94.6	94.0	94.3	0.973
ARI	96.8	96.3	95.9	96.1	0.982

The results indicate that temporal deep-learning models provide stronger predictive performance than static rule-based approaches. The Transformer configuration demonstrates improved capability to model long-range relationships within telemetry sequences. The ARI configuration further incorporates contextual release information and adaptive decision variables, resulting in the highest performance across the evaluated metrics.

B. Deployment and Recovery Performance

The operational impact of the proposed framework is evaluated using release failure rate, rollback rate, recovery time, and high-risk user exposure. The results are presented in Table VII.

TABLE VII DEPLOYMENT AND RECOVERY PERFORMANCE

Deployment Strategy	Failure Rate (%)	Rollback Rate (%)	Recovery Time (min)	High-Risk User Exposure (%)
Conventional Release	6.2	5.1	42	8.7
Rule-Based Canary	4.1	3.4	31	5.9
AI-Based Canary	2.8	2.3	24	4.1
Proposed ARI	1.7	1.4	16	2.6

The results show that controlled deployment reduces failure exposure compared with conventional full deployment. The addition of AI-based prediction further reduces exposure by identifying potentially high-risk releases before broader deployment. ARI additionally monitors post-release behavior and dynamically adjusts feature exposure, which contributes to lower high-risk user exposure and shorter recovery time in the evaluation.

C. Adaptive Feature-Control Analysis

The feature-control mechanism is evaluated by progressively increasing exposure when operational risk remains below the predefined threshold and reducing exposure when degradation is detected. The relationship between risk and feature exposure is summarized in Table VIII.

TABLE VIII RISK-BASED FEATURE EXPOSURE POLICY

Risk Condition	Risk Range	Feature Exposure	Deployment Action
Low Risk	$RR < \theta_1$	100%	Progressive/full deployment
Moderate Risk	$\theta_1 \leq RR < \theta_2$	η_e	Controlled canary deployment
High Risk	$RR \geq \theta_2$	0%	Hold, disable, or rollback

This mechanism prevents a fixed deployment percentage from being applied uniformly to all releases. Instead, the exposure level is continuously associated with the estimated operational risk. Consequently, the framework can maintain limited exposure during uncertain deployment conditions and expand deployment when the observed application health remains stable.

D. Ablation Analysis

The contribution of individual ARI components is evaluated using the ablation configurations defined in Section VI. The results are presented in Table IX.

TABLE IX ABLATION ANALYSIS OF ARI COMPONENTS

Configuration	F1-Score (%)	Release Failure Rate (%)	Recovery Time (min)
ARI ₀ : Baseline	88.4	5.8	39
ARI ₁ : + AI Risk Prediction	93.7	3.9	31

ARI ₂ : + Adaptive Feature Control	95.0	2.8	25
ARI ₃ : + Explainability	95.4	2.5	23
ARI ₄ : + Automated Recovery	96.1	1.7	16

The ablation analysis indicates that the addition of AI-based risk prediction contributes to improved failure identification. Adaptive feature control further reduces release exposure by dynamically controlling the affected functionality. Explainability provides transparency regarding the factors contributing to the risk prediction, while automated recovery contributes primarily to reducing recovery time and limiting continued exposure following degradation.

E. Computational Performance

The computational behavior of the framework is also considered because release-control decisions must be generated within a practical operational timeframe.

TABLE X COMPUTATIONAL COMPLEXITY OF ARI COMPONENTS

Component	Computational Complexity
Telemetry preprocessing	$O(Nd)$
Feature interaction generation	$O(Nd^2)$
Transformer inference	Approximately $O(L^2d)$
Risk computation	$O(d)$
Feature-control decision	$O(1)$
Recovery decision	$O(1)$

where N represents the number of telemetry observations, d represents the feature dimension, and L represents the temporal sequence length. The principal computational cost is associated with Transformer-based temporal representation learning. Since model training can be performed offline, deployment environments primarily require inference and lightweight risk-control operations.

F. Discussion

The evaluation demonstrates the intended relationship between predictive intelligence and adaptive release management. Traditional deployment mechanisms primarily react to predefined conditions, whereas ARI combines predicted failure probability with contextual risk and real-time operational feedback. The results suggest that integrating AI prediction, adaptive feature control, explainability, and automated recovery can simultaneously address prediction accuracy and deployment-level reliability.

Importantly, the reported numerical values are not claimed as measured results from a real enterprise dataset. For the final IEEE submission, these values should be replaced with experimentally obtained results, confidence intervals, statistical significance analysis, and dataset-specific observations. This distinction is necessary to ensure that the final paper reports reproducible experimental evidence rather than assumed performance.

VIII. COMPUTATIONAL COMPLEXITY AND PRACTICAL DEPLOYMENT

The computational design of ARI is intended to support continuous risk assessment without introducing excessive overhead into the mobile application release pipeline. The framework separates computationally intensive model training from lightweight inference and deployment-control operations. Training can be performed centrally using historical telemetry, while the trained model can subsequently provide risk predictions to the release-control layer.

A. Computational Complexity

Let N denote the number of telemetry observations, d the feature dimension, and L the temporal sequence length. Telemetry preprocessing requires approximately

$$O(Nd)$$

operations. If pairwise feature interactions are generated, the corresponding complexity is approximately

$$O(Nd^2)$$

For the Transformer component, self-attention has an approximate computational complexity of $O(L^2d)$

where L^2 results from the pairwise attention calculation between elements in the temporal sequence. The final risk calculation requires only a weighted combination of the prediction and contextual variables:

$$RR = \sum_{i=1}^k w_i x_i$$

and therefore has complexity

$$O(k)$$

where k represents the number of risk factors. The feature-control and recovery decisions require constant-time threshold evaluation and therefore have approximately

$$O(1)$$

complexity.

TABLE XI COMPUTATIONAL COMPLEXITY OF ARI COMPONENTS

Component	Complexity
Telemetry preprocessing	$O(Nd)$
Feature interaction generation	$O(Nd^2)$
Transformer inference	$O(L^2d)$
Risk-score computation	$O(k)$
Feature-control decision	$O(1)$
Recovery decision	$O(1)$

B. Training and Inference Architecture

ARI separates offline model training from online release intelligence. Historical release telemetry is used to train and periodically update the predictive model. During deployment, only feature extraction, model inference, risk calculation, and control decisions are executed in the online pipeline.

The operational workflow can therefore be represented as

$$\text{Historical Data} \rightarrow \text{Model Training} \rightarrow \text{Model Validation} \rightarrow \text{Model Deployment} \rightarrow \text{Real - Time Inference}$$

This separation reduces the computational burden of the production release environment while allowing the prediction model to be periodically retrained using newly accumulated release observations.

C. Practical Deployment

ARI can be integrated with enterprise mobile DevOps pipelines without requiring modification of the core mobile application architecture. The risk engine can operate as an independent service between the continuous integration/continuous deployment pipeline and the application distribution layer.

A practical deployment architecture consists of the following logical components:

TABLE XII PRACTICAL ARI DEPLOYMENT COMPONENTS

Component	Primary Function
CI/CD Pipeline	Build, test, and package application releases
Telemetry Collector	Collect application and operational observations
Feature Engineering Layer	Transform raw telemetry into model features
AI Risk Engine	Predict release failure probability
Risk Controller	Calculate contextual release risk
Feature-Flag Service	Control feature exposure
Monitoring Layer	Track post-release application behavior
Recovery Controller	Trigger feature disabling or rollback
Model Repository	Store trained and validated AI models

The risk engine can initially evaluate a release before deployment and subsequently perform periodic reassessment during canary deployment. The feature-control layer receives the updated risk score and modifies exposure accordingly. This enables the framework to operate without requiring complete redeployment of the application whenever a feature needs to be disabled.

D. Scalability Considerations

Enterprise mobile environments may generate telemetry from thousands or millions of application sessions. Therefore, ARI should support scalable stream-processing and distributed data-storage mechanisms. Telemetry can be processed in batches for model retraining and as streams for real-time release monitoring.

For a distributed environment containing q processing nodes, the approximate processing workload can be represented as

$$W_{node} \approx \frac{W_{total}}{q}$$

assuming relatively balanced workload distribution. Horizontal scaling can therefore be used to accommodate increasing telemetry volume.

E. Reliability and Fault Tolerance

Because ARI itself participates in release decisions, failure of the intelligence layer should not result in uncontrolled deployment. A fallback policy can be incorporated into the release controller. If the AI risk service becomes unavailable, the system can temporarily switch to conservative predefined deployment rules.

The fallback decision can be represented as

$$D = \begin{cases} D_{AI}, & Service_{AI} = 1 \\ D_{fallback}, & Service_{AI} = 0 \end{cases}$$

where D_{AI} represents the AI-driven decision and $D_{fallback}$ represents the predefined safe deployment policy.

F. Practical Limitations

The effectiveness of ARI depends on telemetry quality, historical release data, model calibration, and the reliability of the monitoring infrastructure. Incomplete or biased telemetry may reduce prediction accuracy, while changes in application architecture or user behavior can introduce model drift. Therefore, production deployment should incorporate periodic model validation, drift detection, threshold recalibration, and monitoring of false-positive and false-negative decisions.

Overall, the computational architecture enables ARI to combine centralized model training with lightweight online decision-making, making the framework suitable for integration with enterprise mobile release pipelines while maintaining scalability, operational resilience, and controlled deployment behavior.

CONCLUSION

The proposed Adaptive Release Intelligence (ARI) framework provides an AI-driven approach for improving the reliability and safety of enterprise mobile application releases by integrating release-risk prediction, contextual telemetry analysis, adaptive feature control, progressive deployment, explainable decision-making, continuous monitoring, and automated recovery within a unified closed-loop architecture. Unlike conventional release strategies that rely primarily on static testing and predefined deployment rules, ARI continuously evaluates application, device, network, user, and release conditions to dynamically adjust deployment exposure and recovery actions. The framework employs temporal AI modeling to estimate failure probability and combines the prediction with contextual risk factors to generate an adaptive release-risk score. Feature exposure is subsequently controlled according to the estimated risk, while post-release telemetry enables continuous reassessment and automated mitigation through feature disabling, exposure reduction, or rollback. The experimental methodology evaluates ARI using predictive, deployment, recovery, computational, and ablation metrics, providing a comprehensive basis for assessing its practical effectiveness. Future work will focus on validation using large-scale production telemetry, online learning for model adaptation, federated learning for privacy-

preserving analytics, automated concept-drift detection, and reinforcement-learning-based release optimization to further improve autonomous and context-aware enterprise mobile deployment.

References:

1. D. H. Autor, "Why are there still so many jobs? The history and future of workplace automation," *Journal of Economic Perspectives*, vol. 29, no. 3, pp. 3–30, 2015, doi: 10.1257/jep.29.3.3.
2. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.
3. S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in Neural Information Processing Systems*, vol. 30, pp. 4765–4774, 2017.
4. F. Huang and M. A. Vasarhelyi, "Applying robotic process automation (RPA) in auditing: A framework," *International Journal of Accounting Information Systems*, vol. 35, Art. no. 100433, 2019, doi: 10.1016/j.accinf.2019.100433.
5. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," OpenAI, 2019.
6. P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," *Advances in Neural Information Processing Systems*, 2020.
7. K. Pandya and M. Holia, "Automating customer service using LangChain: Building custom open-source GPT chatbot for organizations," 2023.
8. J. Ruan, Y. Chen, B. Zhang, Z. Xu, T. Bao, Q. Du, and others, "TPTU: Task planning and tool usage of large language model-based AI agents," in *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.
9. B. Y. Lin, C. Huang, Q. Liu, W. Gu, S. Sommerer, and X. Ren, "On grounded planning for embodied tasks with language models," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 11, pp. 13192–13200, 2023, doi: 10.1609/aaai.v37i11.26549.
10. S. Erdmann and K. Sandkuhl, "Robotic process automation in small enterprises: An investigation into application potential," *Complex Systems Informatics and Modeling Quarterly*, no. 34, pp. 84–105, 2023, doi: 10.7250/csimq.2023-34.04.
11. A. Perdana, W. E. Lee, and C. M. Kim, "Prototyping and implementing robotic process automation in accounting firms: Benefits, challenges and opportunities to audit automation," *International Journal of Accounting Information Systems*, vol. 51, Art. no. 100641, 2023, doi: 10.1016/j.accinf.2023.100641.
12. A. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, et al., "ChatDev: Communicative agents for software development," 2023.
13. Y. Li, Y. Yu, H. Li, Z. Chen, and K. Khashanah, "TradingGPT: Multi-agent system with layered memory and distinct characters for enhanced financial trading performance," 2023.
14. S. Yuan, K. Song, J. Chen, X. Tan, D. Li, and D. Yang, "EvoAgent: Towards automatic multi-agent generation via evolutionary algorithms," in *Workshop on Open-World Agents: Synergizing Reasoning and Decision-Making in Open-World Environments*, 2024.
15. Z. Liu, W. Yao, J. Zhang, L. Yang, Z. Liu, and J. Tan, "AgentLite: A lightweight library for building and advancing task-oriented LLM agent systems," 2023.