Proactive Change Risk Detection in Production Cloud Systems: ByteDance's Experience

Jinyang Liu
ByteDance
jinyang.liu@bytedance.com

Yichen Li
ByteDance
liyichen.325@bytedance.com

Tieying Zhang*
ByteDance
tieying.zhang@bytedance.com

Binbin Chen
ByteDance
chenbinbin.1996@bytedance.com

Xiao He
ByteDance
xiao.hx@bytedance.com

Zhihan Jiang
ByteDance
zhihan.jiang@bytedance.com

Haipeng Zhang
ByteDance
zhanghaipeng.zhp@bytedance.com

Gang Wu
ByteDance
wugang.chewu@bytedance.com

Yi Li
ByteDance
liyi.ly@bytedance.com

Abstract

Modern cloud services rely on a high volume of changes for rapid innovation, yet these changes are a primary cause of production incidents. To manage this risk, industry practice employs tiered change management pipelines that concentrate static rules and human reviews. However, our analysis of ByteDance's cloud platform (Volcano Engine) reveals a critical long tail problem: while high-risk changes undergo extensive scrutiny, 78.1% of change-induced incidents originate from the sheer volume of low-risk changes that receive minimal review. At this scale, exhaustive manual review is fundamentally infeasible. To address this, we present Aegis, a novel knowledge-driven system that provides interpretable change risk assessment. Aegis automatically constructs a knowledge base from historical operational data, distilling it into generalized risk heuristics and identifying relevant precedent cases. When a new change is proposed, Aegis generates human-readable warnings grounded in this historical evidence, explaining why a change is risky. We have deployed Aegis in Volcano Engine's production environment for five months, where it processed tens of thousands of change requests. Its risk escalations achieved a 75% acceptance rate from production engineers and successfully prevented multiple potential incidents.

CCS Concepts: • **Software and its engineering** → **Risk management**; • **Computing methodologies** → *Artificial intelligence.*

Keywords: change management, risk detection, cloud systems, LLM, production deployment

ACM Reference Format:
Jinyang Liu, Yichen Li, Tieying Zhang, Binbin Chen, Xiao He, Zhihan Jiang, Haipeng Zhang, Gang Wu, and Yi Li. 2026. Proactive Change Risk Detection in Production Cloud Systems: ByteDance's Experience. In *European Conference on Computer Systems (EUROSYS '26), April 27–30, 2026, Edinburgh, Scotland Uk.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3767295.3803614

*Corresponding author.

EUROSYS '26, Edinburgh, Scotland Uk

ACM ISBN 979-8-4007-2212-7/26/04
https://doi.org/10.1145/3767295.3803614

1 Introduction

Modern cloud services operate at a massive scale, with thousands of engineers pushing a constant stream of changes. These include software deployments [12, 18, 22, 36], configuration modification [13, 41, 42], and infrastructure updates [8, 19] intended to maintain competitive pace. While essential for innovation, each change introduces significant risk. Indeed, post-mortems from major cloud providers like Google [4], AWS [2], and Microsoft Azure [3] consistently reveal that a large percentage of production incidents are caused by changes to live systems [16, 25], making robust change management a critical pillar of production reliability. At large-scale services like ByteDance, a large number of such changes are executed daily. Even a single problematic change can cause widespread disruptions, potentially impacting service availability and user experience.

To mitigate risk, industry practice has converged on formal change management processes [26, 29]. Proposed modifications must first pass a series of automated pre-flight checks within the deployment pipeline. These checks include static analysis, linters, and schema validators, which verify syntactic correctness and screen for obvious implementation bugs. Only after passing this initial gate does the change operator submit a formal Request for Change (RFC), which includes a self-assessed risk level (*e.g.*, Low, Medium, High), for formal human review and approval in the Change Management Platform (CMP).

This approach deliberately shifts the focus of human reviewers from verifying functional correctness to evaluating operational risk. For instance, a configuration update that disables a rate limiter might pass all syntax checks and static validations. While technically correct, deploying it during peak traffic could overwhelm backend services and trigger a cascading failure. This illustrates how operational risk often escapes detection by automated correctness checks alone.

However, our findings show that relying on manual assessment for complex operational risks exposes a critical scalability challenge. An analysis of 12 months of production data from ByteDance's cloud platform reveals a striking imbalance between review coverage and incident impact. High-risk changes receive a median of over 10 hours of review time but account for just 3.8% of change-induced production incidents. In contrast, low-risk changes receive a median of only 7 minutes of review time yet are responsible for 78.1% of change-induced production incidents. This is not due to negligence. Instead, it reflects a fundamental constraint: the volume of low-risk RFCs is so large that deep manual review for each one is infeasible. As a result, the long tail of routine, lightly reviewed changes becomes the largest aggregate source of risk.

There are two existing approaches to automating change risk decisions: static rule-based policies and machine learning (ML) classifiers, but both fall short in practice. Our platform is already equipped with a static, rule-based policy engine that enforces basic safeguards. For example, it can block any change scheduled during a major holiday. However, this method proves inherently inadequate, as such rules are superficial, blind to the rich context within an RFC's unstructured text, and slow to adapt to new failure modes. More advanced ML classifiers, while often more accurate, come with a critical drawback: they lack interpretability. Our early attempts to deploy a semantic classification model were turned down by Site Reliability Engineers (SREs). Their feedback was clear: if a system cannot explain why it flags a change, it becomes an uninterpretable tool that people cannot trust or learn from. This established interpretability as a core requirement for real-world adoption. The need for clear, human-readable explanations naturally led us to explore Large Language Models (LLMs), which have demonstrated strong reasoning and language generation capabilities across many tasks [15, 40, 43]. These models are well-suited to generating justifications that humans can understand.

However, our investigation showed that using a general-purpose LLM in our setting is not straightforward. These models lack an organization's internal operational knowledge. As a result, they can produce inaccurate or misleading explanations (*i.e.*, hallucination [34]). The problem is even greater in a large-scale product cloud, where thousands of services each carry different risks and operational patterns. Knowledge in such environments is inherently complex and highly fragmented. For instance, assessing whether a change is safe may rely on subtle, service-specific heuristics that are undocumented and scattered across teams. In such an environment, it is difficult for a general model to provide reliable and context-aware assessments. The key challenge is to *harness the reasoning power of LLMs while grounding their outputs in trusted, domain-specific knowledge.*

Solution. We address these challenges through Aegis, a knowledge-driven system that provides interpretable change risk assessment by automatically extracting and applying organizational operational knowledge. The system processes historical change data to construct a structured knowledge base containing two complementary forms of evidence: generalized risk heuristics that capture recurring risk patterns, and specific precedent cases that provide context-aware warnings based on operational similarity rather than textual resemblance. When analyzing new changes, Aegis aggregates evidence from both sources to generate human-readable assessments grounded in concrete historical examples. Rather than producing binary classifications or numerical risk scores, the system presents reviewers with specific warnings accompanied by relevant precedent cases that demonstrate why caution is warranted.

Deployment Experience. We have deployed Aegis for five months in ByteDance's Volcano Engine. During this period, the system processed tens of thousands of change requests across hundreds of services. It issued risk escalations with an acceptance rate of around 75% from production engineers. The system also successfully prevented multiple potential incidents, such as a stress test that could have impacted unprepared downstream services. Aegis continuously improved by updating its knowledge base based on operational feedback, helping reduce false positives over time. We provide detailed analyses of both escaped incidents and rejected escalations, discuss cross-organization applicability, and identify open research problems. All these results show the effectiveness and practicality of Aegis in a real-world, large-scale cloud environment.

The primary contributions of this work include:

- We present an in-depth analysis of a large-scale industrial change management system, empirically identifying and quantifying the long tail problem where low-risk changes cause the majority of production incidents.
- We detail the design and implementation of Aegis, a novel system that provides interpretable risk assessment by automatically distilling heuristics and identifying analogous precedents from historical operational data.
- We report on our experience deploying Aegis into a live, high-velocity production environment. Over five months, Aegis has processed tens of thousands of RFCs, and its risk escalations have achieved an overall acceptance rate of ~75% from production engineers, demonstrating its real-world effectiveness and practicality.

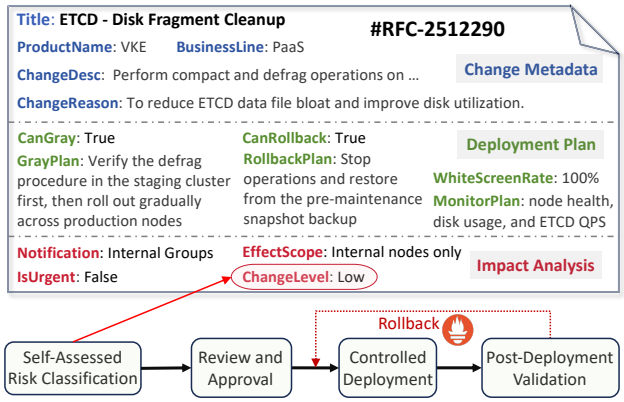


Figure 1. A Sample of Request for Change (RFC)

2 Background and Motivation

In large-scale cloud environments, maintaining service reliability while enabling rapid innovation is a paramount concern [28]. To manage the inherent risk of modifying complex production systems, the software industry has widely adopted formal change management processes [1, 27]. These processes provide a structured framework for proposing, reviewing, and deploying changes, with the primary goal of preventing service-impacting incidents [16, 17]. This section describes the standard artifacts and life-cycle of this process at Volcano Engine.

2.1 The Request for Change (RFC) Artifact

The central artifact in our change management process is the *request for change (RFC)*. An RFC is a highly structured document that serves as the source for any proposed modification to the production environment. Its purpose is to make the intent, implementation, and potential impact of a change transparent to all stakeholders. At Volcano Engine, an RFC is a rich data object with numerous fields that can be grouped into several key categories as shown in Figure 1:

Change Metadata This section defines the basic information about the change. It includes fields such as *title*, *change description*, and *change reason*, which explain what the change does and why it is needed. Contextual fields like *product name* and *business line* help triage reviewers and identify the team of change initiators.

Deployment Plan: This section outlines the rollout and recovery strategies. *can gray* indicates whether the change supports a canary rollout. If so, the *gray plan* describes how the change will be staged and gradually deployed. *can rollback* shows whether the change can be safely reversed, and the *rollback plan* provides concrete steps for doing so, such as restoring from a snapshot. The *monitor plan* describes how success and system health will be verified, such as checking ETCD QPS and disk usage. *white screen rate* reflects the maturity of the deployment procedure. A value of 100% means the entire operation is automated through a GUI interface, while lower values suggest command-line or manual steps, which may introduce risk.

Impact Analysis: This section assesses the change's potential effects. *effect scope* outlines the impact's key dimensions: whether it is internal or external, the number of users affected, the potential duration of service interruptions, and its visibility to end users. Other fields capture urgency, notification scope, and customer-facing impact such as visibility and expected duration. This information supports the self-assessed risk level and helps reviewers understand the possible blast radius.

2.2 The RFC Life-cycle in Practice

Every Request for Change (RFC) follows a formal life-cycle in our CMP designed to manage risk through structured stages. This ensures the level of scrutiny for each change matches its potential impact. The stages are as follows:

Self-Assessment and Classification. When an engineer (the *Operator*) creates an RFC, they must first perform a self-assessment to classify its risk as Low, Medium, or High. This initial judgment is critical, as it determines the entire subsequent review and deployment path.

Review and Approval. The RFC then proceeds to a review stage where requirements are based on its risk level:

- *Low-Risk:* For changes with no customer impact (*e.g.*, in test environments). Requires minimal approval from the component owner and SRE. No customer notification is needed, and deployment is flexible.
- *Medium-Risk:* For changes with manageable customer impact. Requires approval from a stability architect and the head of SRE, a 3-day customer notification, and deployment during weekdays.
- *High-Risk:* For changes with significant potential impact (*e.g.*, to core infrastructure). Demands senior leadership approval, a 7-day customer notification, and deployment strictly confined to off-peak hours. A formal technical review may also be required.

Controlled Deployment. After receiving all necessary approvals, the change is scheduled for deployment. The execution is governed by risk-based controls, such as mandatory notification periods and pre-defined time windows.

Post-Deployment Validation. Following deployment, the system is actively monitored to verify success and stability. If the change fails, the pre-planned rollback procedure is executed immediately to restore service.

2.3 Motivation

Operating large-scale cloud services requires a delicate balance between engineering agility and production stability. While a high velocity of changes is essential for innovation and feature delivery, each change introduces a potential risk

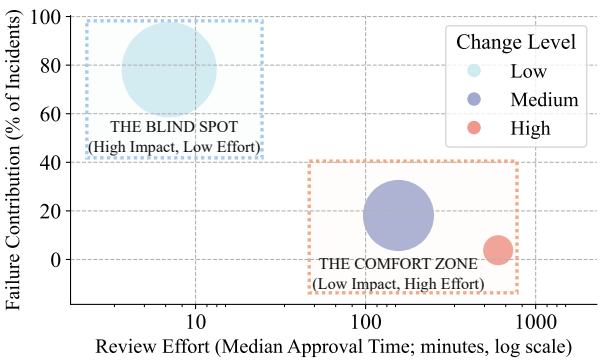


Figure 2. The Misallocation of Review Effort in Change Management (Note: The size of each bubble is proportional to the total volume of changes in that category.)

of causing a service-impacting incident. To diagnose the precise nature of this challenge in a modern operational environment, we analyzed 12 months of production change management data from our Volcano Engine. Figure 2 plots each change category by its failure contribution (y-axis) against the human effort invested in its review (x-axis). The findings are stark and expose a critical flaw in traditional, tiered review processes:

The Blind Spot (High Impact, Low Effort): Changes manually designated as low risk occupy the top-left quadrant. They are responsible for a staggering 78.1% of all change-induced production incidents, yet they receive a median approval time of only 7 minutes. The enormous size of this bubble highlights that this category also represents the overwhelming majority of all changes, making it the system's primary area of vulnerability. For example, an engineer submitted a change to reduce the timeout value for a network client library from 500ms to 200ms. The change passed all automated checks and was approved quickly due to its perceived low complexity. However, under peak load, the downstream service routinely responded in 300–400ms, causing widespread timeouts and a cascading failure across dependent services.

The Comfort Zone (Low Impact, High Effort): In contrast, high risk changes sit in the bottom-right quadrant. They consume immense review effort (a median of 602 minutes (over 10 hours) per change) but are responsible for a mere 3.8% of change-induced incidents. As their small bubble size indicates, they are also relatively rare. For example, a core infrastructure team planned a version upgrade of a widely shared internal RPC library. Due to its broad dependency surface, the change was manually classified as high risk, triggering an extensive review process: manual validation of compatibility reports, integration tests across multiple services, canary deployments, and rollout coordination. The release took days to complete and caused no production issues. This outcome illustrates why such cases fall into the comfort zone: the operational risk is real, but the high effort ensures that the change is executed safely.

Takeaway: As shown in Figure 2, our analysis highlights a striking misalignment between perceived and actual risk. A large portion of production incidents stem from changes labeled as low risk, which typically receive minimal review. This reflects a classic long-tail pattern, where the sheer volume of routine changes accumulates into a significant source of failure. While the current strategy of focusing review effort on high-risk changes is reasonable, it cannot scale to cover the full risk surface. Exhaustively reviewing every change is impractical. Rather than attempting to replace human judgment, our system focuses on pinpointing the small subset of low-risk changes that are likely misclassified and warrant deeper inspection. This targeted escalation shifts human attention to where it has the greatest impact, making expert review both focused and scalable.

3 The Pitfalls of Traditional Automation

Given the clear and costly blind spot in our manual review process, the immediate step was to explore automated solutions. Our goal was to create a system that could automatically identify risky changes that had been incorrectly submitted as low-risk. However, we quickly found that standard automation approaches were not effective for the complex and critical task of managing production changes. This section describes these initial attempts, explains why they failed, and discusses the important lessons we learned.

3.1 Change Platform Control

Before the development of a more intelligent system, the primary method for automated risk control within our change management platform is a static, rule-based policy engine. The mechanism is straightforward: the change platform provides a web form where engineers must describe and categorizes their proposed change using a series of dropdown menus and checkboxes (*e.g.*, selecting the affected service or objects, change type, risk level). When the form is submitted, a backend system validates these structured inputs against a set of hard-coded rules. These rules are manually curated by a committee of experienced engineers and site reliability experts, who distill lessons from past major incidents into explicit policies. For example, a typical set of rules might include: BLOCK any change scheduled within 48 hours of a major holiday or company-wide event; Add more reviewers if the applicant intentionally categorize the change as high-risk; Assign a minimum medium-risk to any change lacking a 100% white screen rate.

Limitations. While the rule-based system is effective at preventing simple, predictable errors and enforcing organizational policies, it has two major limitations that expose the organization to more complex risks. (1) Its analysis is superficial. The engine operates only on structured, categorical data selected by the user and is completely blind

to the critical context within unstructured text fields like the *change description*. (2) The system is static and reactive. Because its rules are manually coded, it can only protect against failure modes that have been previously identified. This design makes the system slow to adapt to new risks, requiring a resource-intensive manual and engineering process to update its rules after a new type of failure occurs.

3.2 Strawman Models for Automated Escalation

To enhance our risk platform, we focused on the critical decision of whether to escalate a change submitted as low-risk. Our analysis showed that the greatest organizational risk originates from the long tail of changes misclassified in this category (Section 2.3). We therefore formulated this as a machine learning classification task: for each low-risk submission, the model predicts if it requires escalation. This system complements our existing static rules. Specifically, we designed the following two models.

Strawman #1: Structured-Field Classifier. Our initial approach established a baseline by relying solely on the structured, machine-readable data available in the RFC form. The goal was to examine whether risk patterns could be identified through this formal metadata alone. The feature set included categorical data from the change metadata section, such as *ProductName* (*e.g.*, VKE) and *BusinessLine* (*e.g.*, PaaS). It also incorporated boolean flags from the deployment plan, including *CanGray* and *CanRollback*. Additionally, we used pre-defined fields from the impact analysis section, such as the *Notification* scope and the *IsUrgent* flag. To prepare this data for a standard classifier (a Gradient Boosted Decision Tree, GBDT [44]), we transformed these discrete inputs into a high-dimensional numerical format using one-hot encoding, creating a sparse binary vector that represents the selected options for each change. While this approach effectively captured the formal taxonomy of a change, it remained entirely blind to the rich, unstructured text where engineers describe the actual substance and nuance of their work.

Strawman #2: Hybrid Semantic Classifier. To address the clear limitations of the first model, our second approach was designed to incorporate the critical context contained within the free-text fields. The central hypothesis was that the unstructured description and reasoning sections of an RFC contain the strongest signals of latent risk, which are not captured by dropdown menus or checkboxes. To incorporate unstructured textual signals, we first merged all free-text fields in the RFC into a single input string. This text was embedded using a pre-trained Doubao encoder [33], producing a 2048-dimensional semantic vector. The embedding captures the overall intent of the change. For example, it places similar phrases such as "restarting the auth service" and "bouncing the authentication pods" close together in the vector space. To improve efficiency and reduce noise, we applied PCA [5] to reduce the embedding's dimensionality.

Table 1. Performance of Strawman Escalation Detectors.

Model	Precision	Recall	F1-Score
#1 Structured-Field	52.9%	50.0%	51.4%
#2 Hybrid Semantic	68.5%	59.5%	63.8%

This semantic vector was then concatenated with the one-hot encoded features from structured fields. The resulting hybrid feature vector combines metadata and textual context. We used the same GBDT model for classification, as it supports high-dimensional input and learns interactions across both types of features.

Evaluation. We evaluated two strawman models on a binary classification task using one year of historical change records. The positive class contained all changes that engineers originally filed as medium- or high- risk. The negative class included changes marked as low-risk.

We tested both models on a separate test set. Our goal was to detect changes that were actually non-low risk but filed as low. As shown in Table 1, the Structured-Field model reached an F1-score of 51.4%. This shows that structured metadata alone is not very effective. The Hybrid Semantic model performed better, with an F1-score of 63.8%, confirming that free-text fields contain important signals about risk.

Although the hybrid model improved accuracy, it reduced transparency. With the structured model, the reason behind each prediction was clear. For example, the model might highlight that the change type was a driver update or that rollback was not supported. These explanations were easy to understand. In contrast, the hybrid model relied on embedding dimensions that are hard to interpret. Explanations based on feature importance often pointed to abstract values like *embedding_dimension_247*, which do not help reviewers understand why the model made its decision.

User Feedback. This issue became clear during pilot testing with senior site reliability engineers, who immediately raised concerns and chose not to adopt the model in production. Their reasoning was straightforward. If a system cannot explain why a change is flagged as risky, it cannot be trusted. In operational settings, engineers need to understand each alert, verify its cause, and learn from it. A model that only produces risk scores, without context or explanation, fails to support this workflow. This feedback revealed a key requirement for deployment: interpretability must be built in from the start. Risk assessments should provide clear, human-readable justifications based on real operational evidence. In response, we shifted away from black-box classification and began developing Aegis, a knowledge-driven system focused on clarity, domain relevance, and trust.

4 The Aegis System Design

The failures of traditional automation yielded a crucial insight: in high-stakes operational environments, a risk assessment system's trustworthiness is as important as its accuracy.

This led us to formulate a new core principle: Interpretable by design. Unlike a model that produces only a risk score, every assessment must be grounded in explicit evidence. To achieve this, we leverage large language models (LLMs) not merely as a predictor but as a reasoning engine that produces human-readable explanations grounded in evidence.

Challenge and Opportunity. The fundamental challenge of ensuring change reliability in modern cloud environments stems from their sheer scale and complexity. These systems evolve too quickly for any single engineer, team, or automated system to maintain a complete, up-to-the-minute understanding of all potential risks. Consequently, operational knowledge becomes fragmented, buried within the collective memory encoded in every change ever made.

Historical RFCs offer a valuable source of such knowledge. High- and medium-risk RFCs are often reviewed thoroughly or have previously led to incidents. In practice, these high-risk changes are usually reliable, precisely because they attract extensive scrutiny. Applicants are less likely to submit flawed changes when they know additional review effort will be involved. Low-risk RFCs, on the other hand, may receive less attention individually. Yet, when observed in large volumes, they offer valuable statistical patterns. Their consistency and scale can help us infer norms and detect deviations, supporting better risk assessment across the board.

To address this challenge and seize the opportunity, we present Aegis, a knowledge-driven system for proactive change risk assessment. Aegis is designed to capture the vast and fragmented operational knowledge embedded in historical changes and transform it into actionable guidance to improve production stability.

The system's architecture, illustrated in Figure 3, is composed of two primary stages: (1) an offline operational knowledge base construction that processes historical raw operational data into a structured knowledge base, and (2) an online risk inference phase that analyzes new change requests in real-time. To provide a holistic assessment, the online phase collects evidence from two complementary modules before unifying the findings in a final synthesis step.

The first heuristics-based risk triage module automatically learns recurring patterns by contrasting past high/medium- and low-risk changes. It generates risk heuristics to flag common failure modes and exemption heuristics to filter out safe, routine actions. To capture context-specific risks missed by these general rules, a precedent-informed assessment module retrieves analogous historical incidents, allowing engineers to evaluate risk using concrete precedents. Finally, a Risk Synthesis stage integrates, prioritizes, and contextualizes these findings, transforming broad evidence into a single, focused, and customized recommendation for the specific change. To ensure the knowledge base remains accurate and relevant over time, a feedback-driven update mechanism is employed. Evidence, whether a heuristic or a precedent, that is repeatedly rejected by operators is automatically pruned as a false positive. The knowledge base is continuously updated with new RFCs to reflect emerging patterns. Teams from different products can also extend it by adding custom heuristics tailored to their specific services and risks.

5 Aegis Implementation

Below, we detail the core technical problems for each dimension of analysis and describe our solutions.

5.1 Heuristics-Based Risk Triage

Human reviewers often rely on implicit rules of thumb when assessing the risk of a change. For example, recognizing that modifying a shared API is riskier than updating internal metadata. However, such reasoning is rarely formalized and is difficult for automated systems to learn directly from noisy, high-variance RFC descriptions.

Our Solution: We designed a multi-stage pipeline to systematically extract, validate, and refine operational heuristics from historical RFC data. The process first groups operationally similar changes, then uses the LLM to discover the differentiating patterns between safe and risky operations within those groups. Critically, the pipeline includes an automated refinement loop and a final human-in-the-loop step to ensure the resulting heuristics are trustworthy.

Hierarchical Grouping for Contextual Analysis. To effectively discover recurring patterns that distinguish risky changes from safe ones, we group RFCs with similar operational intent. This is done in two stages. First, we perform a coarse-grained partitioning based on product lines. Each product line within Volcano Engine functions as an independent operational domain, with distinct architectures, workflows, and team practices. Partitioning by product line ensures that the subsequent analysis is contextually relevant and avoids false generalizations across unrelated systems. Second, within each product-line partition, we apply fine-grained semantic clustering to group RFCs with similar intent. We encode the free-text description of each RFC using the pre-trained Doubao-embedding model [33], generating a 2048-dimensional semantic vector.

We then use the HDBSCAN algorithm [31] to identify dense regions of similar changes in the embedding space, which yields a set of coherent clusters. This semantic grouping is highly effective for identifying broad operational topics. As illustrated in Figure 4, these clusters allow us to extract discriminative heuristics by comparing risk patterns in these similar changes.

Generating Heuristics via Contrastive Prompting. We leverage the reasoning capabilities of LLM as a heuristic *Extractor*, tailoring the prompt based on the risk composition and topic of each cluster. To ensure quality, we discard small clusters that lack sufficient data. The approach varies across three types of clusters:

Homogeneous Low-Risk Clusters: For clusters containing exclusively low-risk RFCs, which represent established safe

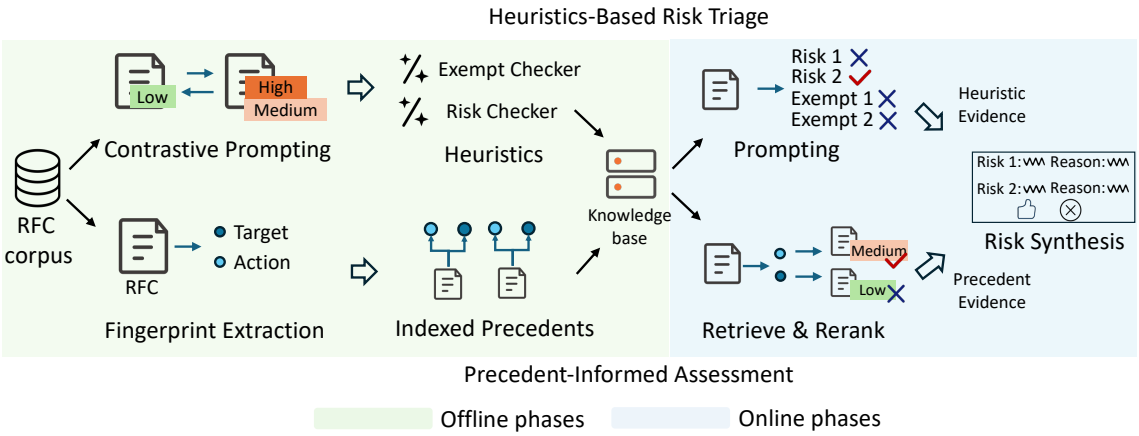


Figure 3. Overall Design of Aegis

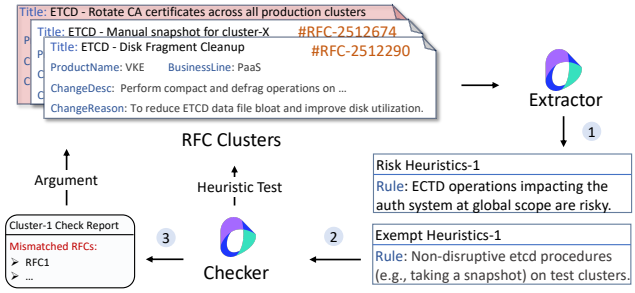


Figure 4. Distilling Heuristics with Validation

procedures, the LLM is prompted to generalize these patterns into Exempt Heuristics.

Homogeneous Non-Low-Risk Clusters: For clusters composed entirely of medium- or high-risk RFCs, which signify known dangerous operations, the LLM is tasked with abstracting the underlying risk factors into Risk Heuristics.

Mixed-Risk Clusters: For clusters containing both risk levels, we use a conservative, contrastive approach. The model, acting as an expert SRE, is presented with both low-risk and medium/high-risk examples from the cluster. It is then asked to articulate only the risk heuristics that explain why the exceptional changes were fundamentally riskier. In this mixed context, we deliberately avoid generating exempt heuristics to maintain a conservative safety posture.

This contrastive approach is effective because RFCs within the same cluster share operational context (same service type, similar actions), making the risk-distinguishing factors more salient. By holding the operational domain constant, the LLM can focus on identifying the specific attributes (such as scope, timing, or affected components) that differentiate risky changes from safe ones. A known failure mode is cluster composition drift: as services evolve, the risk distribution within a cluster may shift, potentially degrading heuristic quality. We partially mitigate this by periodically injecting new RFCs into the knowledge base (Section 5.4), but proactive drift detection remains an open challenge (Section 7.7).

Automated Heuristic Validation and Refinement. To ensure the trustworthiness of generated heuristics, we employ a two-stage validation pipeline that combines automated refinement with lightweight human review.

In the first stage, we use an automated Extractor–Checker loop built on LLMs. The Extractor is responsible for generating initial heuristic candidates based on patterns observed in historical RFCs. These heuristics are then evaluated by a second LLM (the Checker), which serves as a critical peer reviewer. Next, the Checker tests the heuristic against the original RFC clusters, attempting to classify them as low-risk or non-low-risk. Besides the risk level, its output includes counterexamples for any misclassifications it observes. This feedback is then used to augment the Extractor's prompt in the next iteration. The refinement loop continues until the heuristic's F1-score on the validation set converges.

After the automated refinement loop, the resulting heuristics undergo a final manual review by a panel of senior site reliability engineers for the dedicated product. This review focuses on assessing the clarity, practicality, and potential for misinterpretation of each rule. Because the heuristics have already been iteratively refined and validated by the LLM-based pipeline, the manual step is lightweight and low-effort. Only heuristics that pass this expert check are integrated into the knowledge base.

5.2 Precedent-Informed Assessment

While heuristics effectively capture recurring risk patterns, they often miss context-sensitive edge cases. These are changes historically recognized as risky but may appear safe to newcomers. Such cases require reasoning by referring to precedent. These are prior changes that were escalated due to known risk factors that are not statistically frequent.

However, retrieving these precedents with high accuracy presents a more demanding challenge than the thematic clustering used for heuristic generation. While a standard semantic search on the full RFC text is suitable for grouping documents by general topics (Section 5.1), it struggles

with the high-precision retrieval needed for finding specific precedents. The reason is that RFC documents are often long and verbose, with detailed justifications, rollback plans, and discussions that can obscure the core operational intent. This extra text adds noise, making it difficult to tell the difference between changes that are merely on the same topic and those that perform the exact same action.

Our Solution: Operational Fingerprint Index. To address this, we draw inspiration from Retrieval-Augmented Generation (RAG), a paradigm that enhances LLM performance by grounding their outputs in retrieved context. We adapt this idea to change risk assessment and design a two-stage retrieval pipeline tailored for operational data. The first stage emphasizes *high recall* by reducing each RFC to a structured operational fingerprint, a distilled representation that captures the essential nature of the change. This allows us to filter out irrelevant details and retrieve a broad set of candidate precedents with strong signal. In the second stage, we apply *high-precision* reranking using more nuanced models that evaluate the operational similarity of each candidate, ultimately surfacing the most relevant precedents for downstream risk interpretation.

Defining and Extracting the Operational Fingerprint via LLM. This model is informed by our experience observing how senior engineers assess risk. They intuitively decompose a proposed change into what is being done and what system it affects. This *operational fingerprint* consists of two main facets:

- *The Action:* This *action* defines the core operation being performed. We identify the primary verb in the change description and normalizes it to a canonical action (*e.g.*, deploy, revert, migrate, configure, restart). For example, varied phrases such as *bumping the version*, *pushing a new build*, and *rolling out a release* are expected to be mapped to the single normalized action DEPLOY.
- *The Target:* The *target* is the specific systems, services, or components affected by the corresponding action. Typical targets include service names (auth-service, billing-pipeline), database tables (users, invoices), infrastructure components (prod-db-cluster-1, api-gateway), and configuration flags (cache.ttl, network.timeout), etc.

The (Action, Target) schema is intentionally minimal to maximize recall during retrieval. Environment-dependent factors such as system load, deployment timing, or dependent service states are not encoded in the fingerprint. Instead, these contextual factors are evaluated at the LLM reranking stage, where the model compares the full RFC context of the current change against each retrieved precedent. For example, two RFCs with identical fingerprints (CONFIGURE, redis-cluster) may differ in deployment context. One may occur during off-peak hours, while the other takes place during a traffic surge. The fingerprint match ensures retrieval, and the reranking LLM evaluates whether the differing conditions warrant different risk assessments.

Indexing and Retrieval Design. This process contains two phases: offline index construction and online retrieval.

Offline Index Construction. We extract the operational fingerprint index for each historical RFC in an offline manner, which consists of two steps. First, for each historical RFC, an operational fingerprint is extracted using a Large Language Model. The system prompts the LLM to understand the RFC's text and identify the operational fingerprint. To accurately capture complex changes, the system generates multiple action-target pairs from a single RFC when necessary. For example, an RFC to deploy a new version of the auth-service and billing-service would yield two distinct pairs: (DEPLOY, auth-service) and (DEPLOY, billing-service). After the initial extraction, a consolidation step is applied within each product's data. This process normalizes the extracted fingerprints by merging aliases and other variations. For instance, authentication-service and auth-service are both mapped to a single target, *e.g.*, auth-service. This step is critical for ensuring that the index is robust against naming inconsistencies during retrieval.

Online Retrieval. When a new RFC is submitted to Aegis, the online retrieval process find historical changes (*i.e.*, precedents) with similar operational fingerprint.

This context includes all known *Actions* and *Targets* previously seen within that product. Second, this extracted fingerprint is used for candidate retrieval. The system queries the pre-computed index to retrieve all historical RFCs that share the same (Action, Target) fingerprint(s). To ensure the most relevant precedents are considered, this query is restricted to the N most recent matches in chronological order. This step acts as a high-recall filter, rapidly surfacing structurally identical precedents for the new RFC.

Finally, an LLM-based semantic reranking step is performed to verify the relevance of the candidates. The initial retrieval, based on a structured fingerprint, may still include cases where the operational intent is different. For example, consider two RFCs that both generate the fingerprint (SCALE_SERVICE, recommendation-engine). One RFC might describe a proactive, planned scaling event to prepare for anticipated peak traffic during a major holiday sale. The other could detail an emergency, reactive scaling of the same service to mitigate a sudden performance degradation caused by a memory leak. Although their fingerprints match, the former is a routine capacity management task while the latter is a critical incident response. The LLM is prompted to act as a verifier assessing whether the new change is truly the same as the historical precedent by comparing their underlying goals, scope, and context. This produces a final, high-precision, ranked list of the most similar historical changes. These precedents serve as concrete, human-readable references that help reviewers understand past risks associated with similar changes.

5.3 Risk Synthesis

This component serves as the system's central decision-making unit. Its role is to consolidate evidence and assign one of three possible outcomes for each RFC.

- Silent Pass: If no risk evidence is found, the change proceeds without interruption.
- Comment on Risk: If risk is detected but matches an exemption heuristic (i.e., a known safe pattern), the system comments on the potential risk for the author's awareness but does not escalate, preventing unnecessary delays.
- Escalate for Review: If risk is detected and no exemption applies, the change is temporarily escalated, and experts are notified to conduct an in-depth manual assessment.

Crucially, the system uses an LLM for contextual adaptation to make historical precedents relevant. For instance, when a new low-risk RFC is filed to expand gateway-cluster-B, the engine may recall a precedent where a similar expansion of gateway-cluster-A (RFC-2024-4120) was escalated to high risk. It then generates a tailored comment highlighting the semantic similarity:

> *The RFC proposes expanding capacity of gateway-cluster-B. A similar operation on gateway-cluster-A (RFC-2024-4120) was previously escalated to High Risk. Although the target cluster differs, the action (large-scale cluster expansion) resembles the precedent. Please reassess the risk level...*

By providing such context-aware advice, Aegis ensures that its escalations are interpretable and actionable for reviewers.

5.4 Rejection Monitoring and Knowledge Base Adaptation

To keep Aegis' knowledge base accurate and up-to-date as engineering practices evolve, we implement a feedback-driven update mechanism. We continuously monitor the rejection count of risk evidence—either heuristics or risks suggested based on precedent examples. If a specific heuristic or precedent is rejected more than a predefined threshold (*e.g.*, 5 times), we treat it as a false positive. The system will automatically remove the heuristic or delete the precedent from the database to prevent it from affecting future assessments. In addition to this reactive cleaning process, we also periodically inject new RFCs into the knowledge base to ensure it reflects the latest operational patterns. The system also supports adding custom heuristics for product-specific checks, allowing teams to extend the system with rules tailored to their unique risk profiles.

6 Evaluation

In this section, we quantitatively evaluate Aegis in a controlled setting using historical data. We aim to answer the following research questions (RQs).

- **RQ1: Overall Effectiveness.** How effective is Aegis at performing risk classification task against historical data?
- **RQ2: Component Contribution.** What is the unique contribution of each analysis phase to the overall effectiveness of Aegis?
- **RQ3: Efficiency and Token Cost.** What are the latency and LLM token costs of the analysis pipeline?

6.1 Experimental Setup

Dataset. Our evaluation is based on a comprehensive dataset of historical RFC from our production cloud platform, Volcano Engine. The whole dataset encompasses all RFCs submitted for hundreds of products over a 12-month period.

To simulate realistic operational workflows and prevent look-ahead leakage, we partition the data with chronological order. All RFCs filed before 1st Feb 2025 constitute the training set, and all subsequent RFCs form the test set. The training set is used to build the knowledge base for Aegis; It serves as the foundation for distilling risk heuristics (Section 5.1) and to build the indexed repository of historical precedents used in our analogy-based assessment (Section 5.2).

Evaluation Rationale: As discussed in Section 2.3, our goal is to identify underestimated changes incorrectly labeled as low-risk. We formulate this as a binary classification task where the positive class comprises medium/high-risk RFCs and the negative class includes low-risk RFCs. We evaluate using *Precision*, *Recall*, and the *F1-Score*. This formulation is effective because the label distribution is inherently asymmetric: engineers are conservative when applying higher risk levels, making the non-low-risk class a high-precision dataset, while the large volume of low-risk cases makes noise rates acceptable.

Baselines: We compare against six baselines. **GBDT** [44], **XGBoost** [14], and **MLP** [35] are trained on structured RFC metadata, each with a variant (w/ semantic) adding text embeddings. **Doubao (Prompt)** [11] uses zero-shot prompting, **Doubao (ICL)** [20] adds five in-context examples, and **Doubao (Finetuned)** applies LoRA [24] fine-tuning.

LLM Backbone. All LLM-based components in Aegis use Doubao-1.6 [11], a sparse Mixture-of-Experts model (23B active / 230B total parameters, 256K-token context). We host it internally for data privacy, as production RFC data cannot leave the organization. The model is accessed through a standard API and is replaceable with any capable LLM. All "Doubao" baselines also use this model.

6.2 RQ1: Overall Effectiveness in Risk Identification

Table 2 presents the results. Traditional ML models using only structured metadata performed poorly (F1 $\leq$ 56.2%), confirming that free-text fields carry essential risk signals. Adding semantic embeddings improved all ML baselines, but their reasoning remains uninterpretable. Among LLM-based methods, zero-shot prompting failed without operational context, while fine-tuning achieved a strong 72.0% F1 but cannot explain its decisions. Aegis outperformed all

Table 2. Performance Comparison of Baselines and Aegis.

Method	Precision	Recall	F1-score
GBDT	52.9%	50.0%	51.4%
XGBoost	46.7%	40.9%	43.6%
MLP	49.6%	65.0%	56.2%
GBDT w/ semantic	68.5%	59.5%	63.8%
XGBoost w/ semantic	52.4%	58.2%	55.2%
MLP w/ semantic	73.7%	63.5%	68.2%
Doubao (Prompt)	30.3%	36.1%	32.9%
Doubao (ICL)	46.4%	63.9%	53.8%
Doubao (Finetuned)	67.7%	76.9%	72.0%
Aegis	**73.1%**	**80.3%**	**76.5%**

Table 3. Ablation Study of Aegis Components.

Method	Precision	Recall	F1-score
Aegis (H)	88.2%	54.7%	67.4%
Aegis (P)	60.1%	81.8%	69.2%
Aegis	**73.1%**	**80.3%**	**76.5%**

baselines with 76.5% F1, combining superior accuracy with interpretable, evidence-grounded assessments.

Validation with Historical Incidents. Retrospective analysis shows Aegis would have flagged 64.2% of past incident-causing changes initially assessed as low-risk.

6.3 RQ2: Component Contribution

To quantify the contribution of Aegis 's two core components (heuristics-based risk triage and precedent-informed assessment), we conduct an ablation study with the following two variants: **Aegis (H)** uses only risk and exemption heuristics mined from historical RFCs via contrastive prompting. **Aegis (P)** relies solely on precedent-informed assessment, retrieving operationally analogous changes based on fingerprint matching and LLM-based reranking.

We observe that Aegis (H) achieves the highest precision (88.2%), showing strong ability to detect recurring risk patterns with high confidence. However, its recall is lower (54.7%), as it may miss context-specific edge cases not covered by learned heuristics. In contrast, **Aegis (P)** attains higher recall (81.8%) by capturing more such edge cases, but with lower precision (60.1%), likely due to retrieval noise or context drift in precedents. The full Aegis system balances both strengths, achieving the highest overall F1-score (76.5%), with solid precision (73.1%) and recall (80.3%). This confirms that heuristics and precedents provide complementary value: the former offers interpretable coverage of known risks, while the latter brings in context-sensitive signals that general rules may overlook. Although the full Aegis system has lower precision than the heuristic-only variant, this is a deliberate design choice. We prioritize higher recall because the operational cost of a false positive is minimal. A brief, non-blocking expert review is a negligible price to pay compared to the high cost of a false negative, which could become a production incident.

Table 4. Computational Performance of Aegis.

Analysis Phase	Avg. Latency (P50, s)	Latency (P99, s)
Heuristics Analysis	6.86	12.13
Precedent Retrieval	6.83	16.89
Risk Synthesis	1.67	3.17
Total (End-to-End)	**11.87**	**23.45**

6.4 RQ3: Efficiency and Token Cost

Table 4 reports end-to-end latency. The Heuristics Analysis and Precedent Retrieval stages run in parallel, yielding a median total latency of 11.87s (P99: 23.45s). For the final risk synthesis step, LLM inference overhead is low: median input is 1,297 tokens and median output is 147 tokens.

7 Production Deployment Experience

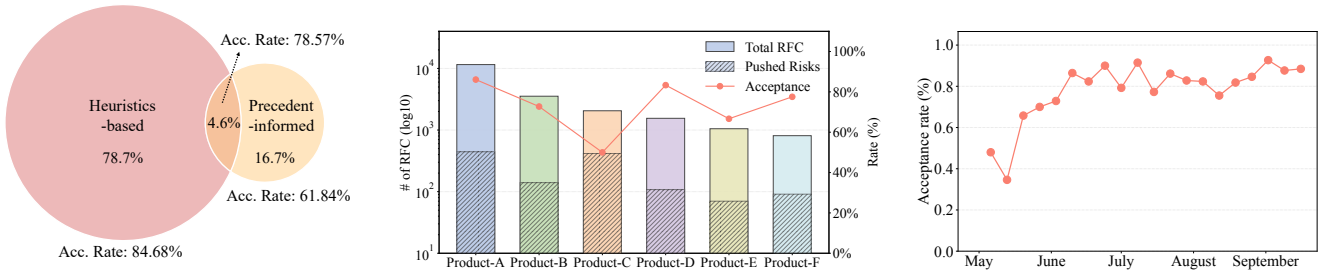
Aegis has been fully operational for 5 months, analyzing all RFCs across all major products of Volcano Engine (Note that sensitive data is anonymized or normalized per policy, but key signals are preserved to ensure evaluation quality). This section details its integration into our operational workflow, presents quantitative results from this deployment, and shares feedback from our engineers.

7.1 Integration into the Live Change Workflow

To balance system reliability with change efficiency, Aegis is integrated into the CMP as a non-blocking, expert-in-the-loop service. The primary goal is not to automatically block changes, but to support the human review process by surfacing risks that might otherwise be missed in high-volume, low-risk changes. The workflow is shown in Figure 6: Aegis continuously listens to each RFC submission via a webhook and analyzes every RFC self-assessed as low-risk upon submission. If its analysis finds risk evidence not covered by an exempt heuristic (as described in Section 5.3), it triggers an escalation. This escalation is delivered as an interactive notification card to a dedicated Lark group containing the original reviewers and on-call SREs. Aegis then additionally invites the reliability owner, usually a senior SRE. Crucially, the card provides the *why* by presenting the synthesized evidence, such as the specific heuristic that was triggered or a link to a relevant historical precedent. This initiates a human-in-the-loop decision point where engineers review the evidence and interact with the card:

- Accept: Acknowledges the risk, prompting the reviewer to work with the change owner to add safeguards, cancel the RFC, or escalate it to a higher risk level for formal review.
- Reject: Overrides the escalation if it is inapplicable. A brief justification is optional but helps refine Aegis.

Aegis's role as a collaborative assistant, rather than an authoritative gatekeeper, is key to its adoption. This non-blocking design builds trust by allowing engineers to override escalations with minimal friction, ensuring its acceptance in our fast-paced engineering culture.



(a) Path-wise Escalations — Volume Share & Acceptance Breakdown

(b) Per-Product Scale — Total RFCs, Escalation (Risk) Rate & Acceptance

(c) Monthly Acceptance Rate (May–Sep)

Figure 5. Field Results of Aegis during Production Rollout.

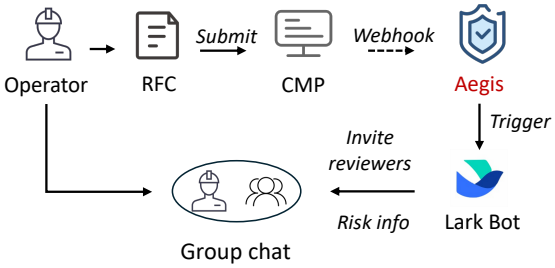


Figure 6. Deployment and Runtime Flow of Aegis

7.2 Quantitative Analysis of Production Data

During a five-month deployment period, Aegis has processed a large number of RFCs initially classified as low risk. Aegis is hooked into all RFCs across the major regions of Volcano Engine and has issued thousands of notifications to change operators. **Overall, reviewers accepted 75.03% of these notifications.** We define the *accept rate* as the percentage of notifications where human reviewers agreed with Aegis's assessment by clicking the Accept button. The detailed deployment results are presented in Figure 5.

Path-wise Analysis. Figure 5a shows that heuristics-based alerts account for 78.7% of all escalations, with an acceptance rate of 84.68%, reflecting their alignment with recurring, well-understood risk patterns. This trend is consistent with the high precision of the heuristics-based analysis reported in Table 3. Precedent-informed alerts contribute 16.7%, targeting rarer edge cases with a lower acceptance rate of 61.84%. This gap reflects a fundamental difference in targets: heuristics distill stable, recurring patterns that engineers readily recognize, while precedents target inherently noisier edge cases where context drift (such as service evolution or newly added safeguards) can make historical precedents less applicable. This lower precision is a deliberate tradeoff: our ablation study (Table 3) shows that removing precedents drops recall from 80.3% to 54.7%, confirming that precedents capture a substantial class of risks that heuristics alone would miss. The remaining 4.6% of alerts are flagged by both paths, achieving a strong 78.57% acceptance rate.

Per-product Analysis. Figure 5b illustrates the performance of Aegis across the six highest-volume products (A–F). To maintain confidentiality, the analysis uses a data sample from the total RFC volume. Bar height (left, log scale) shows total RFCs, and the hatched segment marks escalations (counts on the same log scale). The red line (right axis) is the acceptance rate for those escalations. Volumes span roughly one to two orders of magnitude. Escalation rates differ by service but are generally low. We use a log scale for clearer presentation, so hatched shares may appear large while mostly corresponding to <10%. Acceptance is high overall. Product-C shows a dip because it is evolving very fast. Its rapid changes moved faster than our heuristics and precedents. As a result, the evidence was weaker, and reviewers accepted fewer advisories.

Monthly Trend. In Figure 5c, the monthly acceptance rate increases from May to September. The upward shift follows the incorporation of feedback from multiple teams. Based on this feedback, we refined the knowledge base: we adjusted service-specific heuristics and dropped outdated precedents. Evidence became more precise and easier to verify, which reduced reviewer burden and raised acceptance.

Engineer Response Actions. To understand how engineers respond to Aegis's escalations, we examine two tiers of response. At the system level, approximately 5% of escalated changes were subsequently upgraded from low-risk to medium-risk review tracks by engineers after reviewing Aegis's assessment, triggering additional approval requirements and stricter deployment constraints. At the engineer level, we analyze the escalations where engineers provided explicit feedback through the notification card. Among these responses, approximately 75% accepted the flagged risk. These engineers agreed with Aegis's assessment and incorporated the identified risks into their review. The remaining ~25% were rejections, which we analyze in detail in Section 7.3. Beyond explicit feedback, we observed post-notification modifications to deployment plans (e.g., adding monitoring URLs, refining rollback procedures) through cross-referencing with the change management system's audit trail. The combination of automatic level escalation and voluntary engineer engagement demonstrates that Aegis's notifications serve as effective risk amplifiers within the existing change workflow.

Table 5. Decision latency from risk detection to engineer response.

Response	Median	Mean
Overall	1.2 min	3.5 h
Accept	1.2 min	3.1 h
Reject	1.1 min	4.4 h

Decision Latency. Table 5 reports the time between Aegis's risk detection and the engineer's feedback response, measured from thousands of feedback interactions during the deployment period.

The median response time of approximately one minute confirms that Aegis integrates seamlessly into the existing change approval workflow: risk notifications are delivered to the review group's Lark channel alongside the ongoing RFC discussion, and engineers respond as part of their normal review cycle. Notably, rejection responses arrive slightly faster (median 1.1 min vs. 1.2 min for accepts), suggesting that engineers who disagree with an assessment act more promptly. The low median combined with a higher mean (3.5 h) reflects a long-tailed distribution: most engineers respond immediately upon encountering the notification, while a smaller fraction reviews the assessment later.

7.3 Escape and Rejection Analysis

Escape analysis. To evaluate Aegis's coverage, we examined changes that Aegis did not escalate but were later associated with production incidents during the deployment period. None of these escaped changes resulted in major customer-impacting events. A small number of internal incidents were traced back to changes that Aegis did not escalate. We manually reviewed each escaped incident and categorize them by whether the risk was identifiable from the RFC content:

- **Execution-level** (~75%): These incidents arose from runtime factors not predictable from RFC content, such as unexpected resource contention during deployment, race conditions triggered by concurrent operations, or cascading effects across dependent services. For example, a CDN cache configuration update triggered request blocking due to an unforeseen interaction between cache invalidation timing and concurrent domain configuration delivery. This was a runtime artifact invisible in the change plan.
- **Plan-level missed** (~25%): A small fraction involved risks that were, in retrospect, identifiable from RFC content but were not captured by existing heuristics. For instance, a network device firmware upgrade did not adequately describe its blast radius across availability zones, leading to storage I/O disruptions. Each such case prompted the addition of a new heuristic, incrementally expanding Aegis's coverage.

The predominance of execution-level escapes underscores the inherent boundary of plan-level assessment: Aegis evaluates what is written in the RFC, not what happens during execution. Extending coverage to runtime signals remains an open challenge.

Rejection analysis. The overall ~25% rejection rate (where engineers overrode Aegis's escalation) warrants careful examination. We sampled rejected escalations across representative products and time periods to classify root causes and assess safety implications. Among the sampled rejections, we identified two primary categories:

- **Operational evolution** (~65%): The majority of rejections occurred because previously risky operations had been automated, hardened, or made safer through new procedures since the heuristic was created. For example, routine resource scaling and shrink operations (which historically carried risk) were frequently flagged by the "resource downgrade" heuristic even when engineers had implemented safe, automated procedures. These rejections reflect *knowledge staleness* rather than system error.
- **LLM reasoning errors** (~35%): The remaining rejections stemmed from the LLM misinterpreting RFC content. Common patterns included confusing UI-level display changes with database schema modifications, and misclassifying code-level optimizations as resource reductions.

These root causes inform the iterative refinement of both heuristic rules and LLM prompts through the feedback loop described in Section 5.4. We further illustrate representative cases in the following subsection.

7.4 Case Study

We demonstrate typical case studies to show how Aegis benefits the reliability of Volcano Engine, and discuss cases where Aegis fails.

7.4.1 Success Case. On May 21, 2025, an engineer submitted a change request to perform an online stress test on production infrastructure, primarily targeting the CBR (Cloud Backup and Recovery) service. The test involved large-scale, concurrent backup, recovery, and deletion of snapshots using a scaling number of ECS instances. While the operation included use of EBS, the request emphasized that it would run under a test account with EBS rate-limited accordingly. Therefore, the change operator initially classified it as low-risk.

Aegis processed the request and found a direct precedent: a similar stress test on the same service that had previously been classified as higher risk. This triggered an immediate escalation, and the system automatically invited a senior reliability architect to join the review. The architect's broader system knowledge was critical. They recognized that the impact on the dependent EBS service had not been communicated with the responsible team. Acting on this insight, the architect brought the EBS SREs into the discussion, who

confirmed their service was undergoing an update and was unprepared for the stress test. As a result of this expert collaboration, the change was safely postponed. This case demonstrates how Aegis drives collaboration to prevent incidents. By identifying the risk and engaging the right experts, it helped transform a dangerous change into a safe operational decision.

7.4.2 Failure Cases. We present two representative failure modes that illustrate how Aegis's feedback loop drives iterative improvement.

Knowledge staleness. Aegis persistently flagged upgrades for an ECS component as high-risk, but engineers consistently rejected these escalations. This consistent pattern of user feedback triggered an automated review. Our investigation with the ECS team revealed that they had implemented comprehensive automation and enhanced fault tolerance for their upgrade procedures. These improvements had fundamentally transformed the risk profile, converting what was previously a high-risk manual operation into a routine automated process. The heuristic Aegis was using, learned from older data, no longer reflected the current, safer operational reality. This case demonstrates how Aegis maintains knowledge base accuracy through systematic pattern detection in user feedback, as our system automatically detects grouped rejections for each heuristic and streamlines the updates (see Section 5.4).

LLM misinterpretation. Aegis flagged a frontend display update for a billing service as a "database structure modification." The change optimized invoice rendering by hiding certain fields (email, phone) and adding a new display field. The LLM read "remove email and phone fields" and matched it against the database schema change heuristic, interpreting "remove fields" literally as column deletion.

The engineer rejected the escalation since the change only modified the UI rendering template. This rejection, combined with similar patterns from other display-related changes, triggered the feedback loop. The corrective action refined the heuristic to require evidence of actual DDL statements (e.g., `ALTER TABLE`, `DROP COLUMN`) rather than keyword-level pattern matching. This case illustrates how Aegis's self-correcting feedback mechanism converts individual LLM errors into systematic knowledge improvements.

7.5 Lessons Learned

Our experience deploying Aegis at Volcano Engine yielded several generalizable lessons. The core long-tail risk is a common problem in large-scale cloud systems, and the RFC-based workflow an industry standard. We expect our insights are broadly applicable to other production environments.

Augment human judgment, not replace it. Our experience highlighted that the real challenge in change risk management in a large cloud system lies not in handling clearly high-risk changes, but in spotting the few risky ones that are mistakenly treated as low risk. These cases often blend in with routine changes and receive minimal review, yet can lead to significant issues. Rather than fully automating risk classification, we chose to focus on augmenting the existing process. Aegis is designed to work alongside human reviewers, helping to flag under-assessed changes that deserve a second look. By framing the task as a targeted escalation decision, we reallocate the human effort, complementing, not replacing, expert judgment.

Integration design determines adoption success. Technical accuracy alone does not guarantee adoption. Our non-blocking integration, automated reviewer invitation system, and interactive feedback mechanism were as crucial to Aegis's success as its risk detection capabilities. By making the system easy to interact with, providing clear escalation paths, and allowing engineers to override decisions with feedback, we created a collaborative tool rather than an enforcement mechanism. The high acceptance rate validates that engineers are willing to engage with automated systems when they enhance rather than hinder their workflow.

Fragmented knowledge is a challenge and an opportunity. Risk knowledge is scattered across team practices, and historical change data. Traditional models often overlook these details, especially when risks depend on subtle context. Aegis addresses this challenge by turning fragmented signals into structured operational knowledge. It combines generalized heuristics with similar past cases to produce clear, context-aware risk assessments. In the future, it can incorporate more types of data (such as logs, traces, or system topology) to support deeper multi-modal analysis.

Prefer liberal flagging over confidence thresholds. A natural approach to managing LLM uncertainty is to introduce confidence scores and suppress low-confidence outputs. In practice, we found LLM confidence to be poorly calibrated for operational risk assessment. The model's stated certainty did not reliably correlate with assessment quality. Instead, Aegis adopts a "flag liberally, refine via feedback" strategy: both heuristics and precedents undergo offline refinement and validation before deployment, and are continuously improved through engineer rejection feedback online. This approach avoids silently suppressing valid warnings and produces a self-improving system where assessment quality improves over time, as reflected by the rising acceptance rate in Figure 5c.

Knowledge freshness is the hardest operational challenge. As illustrated by Product-C's lower acceptance rate (Figure 5b), rapidly evolving services can outpace the knowledge base. When a service undergoes frequent architectural changes or automates previously risky procedures, historical heuristics and precedents become stale. While our feedback loop eventually corrects these through rejection-count pruning, the lag between operational evolution and KB adaptation remains the primary source of false positives in production.

7.6 Cross-Organization Applicability

A recurring reviewer concern is whether Aegis generalizes beyond ByteDance. We analyze this along three dimensions.

Transferable components. RFC-based change management is an industry standard adopted by major cloud providers including AWS [7], Azure [32], and Google Cloud [21]. Aegis's core design is largely organization-agnostic: the heuristic extraction pipeline requires only historical RFC data and a text embedding model, adapting to local formats via LLM prompting; the (Action, Target) fingerprint schema generalizes to any structured change description; and the KB refinement loop, driven by engineer feedback, operates independently of organizational specifics.

Organization-specific components. The LLM choice (Doubao-1.6) is replaceable with any capable language model. The integration surface (Lark notification cards) is adaptable to other platforms such as Slack, Teams, or PagerDuty. Specific RFC field names can be reconfigured through prompt engineering without code changes.

Deployment considerations. For new organizations, three practical concerns arise: (1) *Data privacy*: the knowledge base is fully local, requiring no data sharing across organizations; (2) *RFC field heterogeneity*: the fingerprint extraction (via LLM) adapts to different RFC schemas without structural modifications; (3) *Cold-start*: bootstrapping the knowledge base requires sufficient historical RFC data to populate heuristic clusters and the precedent index.

7.7 Limitations and Open Research Problems

We identify five key limitations of Aegis, each paired with an open research problem for the community.

(1) Plan-level vs. execution-level boundary. Aegis assesses risks visible in RFC content but cannot detect execution-level failures such as resource exhaustion, race conditions, or operational errors that emerge only at runtime. Bridging plan assessment with runtime signals (monitoring, topology, live load) requires integrating heterogeneous systems in real time, which remains an open challenge at scale.

(2) Proactive knowledge drift detection. Current KB maintenance relies on rejection feedback, which lags behind fast-evolving services (as observed with Product-C in Figure 5b). Proactively detecting operational pattern shifts across hundreds of heterogeneous services, where evolution patterns differ widely, remains unsolved.

(3) Cold-start for new services. New services or change types lack historical data for heuristic extraction and precedent retrieval. While product teams can contribute custom heuristics for immediate coverage of known risks, cross-service meta-learning to transfer knowledge between services remain open questions.

(4) LLM reasoning reliability. LLM-generated assessments occasionally contain hallucinations, such as numerical misinterpretation of thresholds or incorrect causal reasoning. In safety-critical operational contexts, reducing such errors without relying on unreliable confidence calibration remains a key challenge. We discuss our practical mitigation ("flag liberally, refine via feedback") in the lessons learned above.

(5) Cross-organization validation. Our evaluation is within a single organization. Identifying which risk heuristics are organization-agnostic versus environment-specific, and developing shared change risk benchmarks, are important directions for future work.

8 Related Work

Research on production change risk covers pre-deployment assessment, in-deployment monitoring [22, 29, 39], and post-deployment analysis [9, 15, 30, 45]. Our work focuses on the pre-deployment phase, where most systems address either artifact correctness or operational risk.

Artifact correctness verifies that code or configurations are not inherently faulty. For code, this involves predicting risk from source metrics [37] or using LLMs for Diff Risk Scores [6]. For configurations, methods include configuration-driven testing [38] and static dependency analysis [46]. These checks cannot assess broader operational impact when artifacts are correct but the deployment context is risky.

Operational risk evaluates the change execution itself. This is critical for manual script executions or interactive commands that lack artifacts but carry user-facing impact. Prior ML-based risk classification [10, 23] lacks explainability, hindering operator trust. Aegis addresses this by providing human-readable justifications grounded in historical precedents rather than opaque risk scores.

9 Conclusion

We present Aegis, an interpretable, knowledge-driven system for proactive change risk assessment in cloud platforms. It targets high-volume, self-assessed low-risk RFCs that receive minimal review yet cause a significant share of incidents, grounding each escalation in mined heuristics and precedent retrieval via operational fingerprints. Deployed as an expert-in-the-loop assistant, Aegis synthesizes multiple signals into actionable, human-auditable advisories, achieving broader coverage with more efficient use of review effort. Our production analysis reveals complementary roles of heuristics and precedents, challenges of knowledge freshness, and the effectiveness of liberal flagging with feedback-driven refinement. We identify five open research problems spanning plan-vs-execution boundaries, drift detection, cold-start bootstrapping, LLM reliability, and cross-organization transferability.

Acknowledgments

We thank the ByteBrain team and the Platform Architecture Department from VolcEngine for their support. We also thank the anonymous reviewers and our shepherd, Vijay Govindarajan, for their constructive feedback that improved this paper.

References

[1] 2024. A Quick Guide on SaaS Change Management. https://www.cloudeagle.ai/blogs/saas-change-management

[2] 2025. AWS Cloud. https://aws.amazon.com/

[3] 2025. Azure Cloud. https://azure.microsoft.com/en-us/

[4] 2025. Google Cloud. https://cloud.google.com/

[5] Hervé Abdi and Lynne J Williams. 2010. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics* 2, 4 (2010), 433–459.

[6] Rui Abreu, Vijayaraghavan Murali, Peter C Rigby, Chandra Maddila, Weiyan Sun, Jun Ge, Kaavya Chinniah, Audris Mockus, Megh Mehta, and Nachiappan Nagappan. 2025. Moving Faster and Reducing Risk: Using LLMs in Release Deployment. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 448–457.

[7] Amazon Web Services. 2024. Change Management Categories and Priorities. https://docs.aws.amazon.com/whitepapers/latest/establishing-your-cloud-foundation-on-aws/change-management-categories-priorities.html. Accessed: 2026-02-27.

[8] Pradeep Ambati, Íñigo Goiri, Felipe Frujeri, Alper Gun, Ke Wang, Brian Dolan, Brian Corell, Sekhar Pasupuleti, Thomas Moscibroda, Sameh Elnikety, et al. 2020. Providing {SLOs} for {Resource-Harvesting}{VMs} in cloud platforms. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 735–751.

[9] Mona Attariyan, Michael Chow, and Jason Flinn. 2012. X-ray: Automating {Root-Cause} diagnosis of performance anomalies in production software. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 307–320.

[10] Raghav Batta, Larisa Shwartz, Michael Nidd, Amar Prakash Azad, and Harshit Kumar. 2021. A system for proactive risk assessment of application changes in cloud operations. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 112–123.

[11] ByteDance Seed. 2025. Technical Introduction to the Seed1.6 Model Series. https://seed.bytedance.com/en/blog/introduction-to-techniques-used-in-seed1-6. Accessed: 2026-02-27.

[12] Gerardo Canfora and Luigi Cerulo. 2006. Supporting change request assignment in open source development. In *Proceedings of the 2006 ACM symposium on Applied computing*. 1767–1772.

[13] Qingrong Chen, Teng Wang, Owolabi Legunsen, Shanshan Li, and Tianyin Xu. 2020. Understanding and discovering software configuration dependencies in cloud and datacenter systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 362–374.

[14] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[15] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, et al. 2024. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 674–688.

[16] Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, et al. 2020. Towards intelligent incident management: why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1487–1497.

[17] Chi-An Chih and Yu-Lun Huang. 2015. An adjustable risk assessment method for a cloud system. In *2015 IEEE International Conference on Software Quality, Reliability and Security-Companion*. IEEE, 115–120.

[18] Olivier Crameri, Nikola Knezevic, Dejan Kostic, Ricardo Bianchini, and Willy Zwaenepoel. 2007. Staged deployment in mirage, an integrated software upgrade testing and distribution system. *ACM SIGOPS Operating Systems Review* 41, 6 (2007), 221–236.

[19] Frank Doelitzscher, Christian Fischer, Denis Moskal, Christoph Reich, Martin Knahl, and Nathan Clarke. 2012. Validating cloud infrastructure changes by cloud audits. In *2012 IEEE Eighth World Congress on Services*. IEEE, 377–384.

[20] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234* (2022).

[21] Google Cloud. 2024. Google Cloud's Approach to Change. https://cloud.google.com/docs/cloud-approach-to-change. Accessed: 2026-02-27.

[22] Boris Grubic, Yang Wang, Tyler Petrochko, Ran Yaniv, Brad Jones, David Callies, Matt Clarke-Lauer, Dan Kelley, Soteris Demetriou, Kenny Yu, et al. 2023. Conveyor:{One-Tool-Fits-All} Continuous Software Deployment at Meta. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 325–342.

[23] Sinem Güven and Karin Murthy. 2016. Understanding the role of change in incident prevention. In *2016 12th International Conference on Network and Service Management (CNSM)*. IEEE, 268–271.

[24] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations (ICLR)*.

[25] Junjie Huang, Jinyang Liu, Zhuangbin Chen, Zhihan Jiang, Yichen Li, Jiazhen Gu, Cong Feng, Zengyin Yang, Yongqiang Yang, and Michael R Lyu. 2024. Faultprofit: Hierarchical fault profiling of incident tickets in large-scale cloud systems. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. 392–404.

[26] Gregor Joeris. 1997. Change management needs integrated process and configuration management. In *Proceedings of the 6th European SOFTWARE ENGINEERING conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of software engineering*. 125–141.

[27] Rabia Latif, Haider Abbas, Saïd Assar, and Qasim Ali. 2014. Cloud computing risk assessment: a systematic literature review. *Future information technology* (2014), 285–295.

[28] Sebastien Levy, Randolph Yao, Youjiang Wu, Yingnong Dang, Peng Huang, Zheng Mu, Pu Zhao, Tarun Ramani, Naga Govindaraju, Xukun Li, et al. 2020. Predictive and adaptive failure mitigation to avert production cloud {VM} interruptions. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 1155–1170.

[29] Ze Li, Qian Cheng, Ken Hsieh, Yingnong Dang, Peng Huang, Pankaj Singh, Xinsheng Yang, Qingwei Lin, Youjiang Wu, Sebastien Levy, et al. 2020. Gandalf: An intelligent,{End-To-End} analytics service for safe deployment in {Large-Scale} cloud infrastructure. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 389–402.

[30] Chang Lou, Peng Huang, and Scott Smith. 2020. Understanding, detecting and localizing partial failures in large system software. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 559–574.

[31] Leland McInnes, John Healy, Steve Astels, et al. 2017. hdbscan: Hierarchical density based clustering. *J. Open Source Softw.* 2, 11 (2017), 205.

[32] Microsoft Azure. 2025. Administer Your Azure Cloud Estate – Manage Change. https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/manage/administer. Accessed: 2026-02-28.

[33] H Nie, ZH Long, ZJ Fang, and LQ Gao. 2025. Multimodal detection framework for financial fraud integrating LLMs and interpretable machine learning. *Journal of Data and Information Science* 10, 4 (2025), 1–25.

[34] Vipula Rawte, Amit Sheth, and Amitava Das. 2023. A survey of hallucination in large foundation models. *arXiv preprint arXiv:2309.05922* (2023).

[35] Martin Riedmiller. 2014. Machine Learning: Multi Layer Perceptrons. Machine Learning Lab, University of Freiburg.

[36] Tony Savor, Mitchell Douglas, Michael Gentili, Laurie Williams, Kent Beck, and Michael Stumm. 2016. Continuous deployment at Facebook and OANDA. In *Proceedings of the 38th International Conference on software engineering companion*. 21–30.

[37] Emad Shihab, Ahmed E Hassan, Bram Adams, and Zhen Ming Jiang. 2012. An industrial study on the risk of software changes. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. 1–11.

[38] Xudong Sun, Runxiang Cheng, Jianyan Chen, Elaine Ang, Owolabi Legunsen, and Tianyin Xu. 2020. Testing configuration changes in context to prevent production failures. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 735–751.

[39] Diane Tang, Ashish Agarwal, Deirdre O'Brien, and Mike Meyer. 2010. Overlapping experiment infrastructure: More, better, faster experimentation. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 17–26.

[40] Haopei Wang, Anubhavnidhi Abhashkumar, Changyu Lin, Tianrong Zhang, Xiaoming Gu, Ning Ma, Chang Wu, Songlin Liu, Wei Zhou, Yongbin Dong, et al. 2024. {NetAssistant}: Dialogue based network diagnosis in data center networks. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 2011–2024.

[41] Tianyin Xu, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, and Shankar Pasupathy. 2016. Early detection of configuration errors to reduce failure damage. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 619–634.

[42] Tianyin Xu and Yuanyuan Zhou. 2015. Systems approaches to tackling configuration errors: A survey. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 1–41.

[43] Chenyuan Yang, Zijie Zhao, and Lingming Zhang. 2025. Kernelgpt: Enhanced kernel fuzzing via large language models. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 560–573.

[44] Jerry Ye, Jyh-Herng Chow, Jiang Chen, and Zhaohui Zheng. 2009. Stochastic gradient boosted distributed decision trees. In *Proceedings of the 18th ACM conference on Information and knowledge management*. 2061–2064.

[45] Ding Yuan, Soyeon Park, Peng Huang, Yang Liu, Michael M Lee, Xiaoming Tang, Yuanyuan Zhou, and Stefan Savage. 2012. Be conservative: Enhancing failure diagnosis with proactive logging. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 293–306.

[46] Ennan Zhai, Ang Chen, Ruzica Piskac, Mahesh Balakrishnan, Bingchuan Tian, Bo Song, and Haoliang Zhang. 2020. Check before you change: Preventing correlated failures in service updates. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 575–589.