A Practical Framework for Flaky Failure Triage in Distributed Database Continuous Integration

Jun-Peng Zhu[*]
Northwest A&F University
PingCAP
China
zjp.dase@nwafu.edu.cn

Qizhi Wang[*] [†]
PingCAP
China
qizhi.wang@pingcap.com

Yulong Zhai
PingCAP
China
zhaiyl@pingcap.com

Yishen Sun
PingCAP
China
sunyishen@pingcap.com

Sen Chen[†]
Nankai University
China
senchen@nankai.edu.cn

Xu Kai
PingCAP
China
xukai@pingcap.com

Peng Cai
East China Normal University
China
pcai@dase.ecnu.edu.cn

Hongming Zhang
Northwest A&F University
China
zhm@nwsuaf.edu.cn

Heng Long
PingCAP
China
lh@pingcap.com

Liu Tang
PingCAP
China
tl@pingcap.com

Qi Liu
PingCAP
China
liuqi@pingcap.com

Abstract

Flaky failure triage is crucial for keeping distributed database continuous integration (CI) efficient and reliable. After a failure is observed, operators must quickly decide whether to auto-rerun the job as likely flaky or escalate it as likely persistent, often under CPU-only millisecond budgets. Existing approaches remain difficult to deploy in this setting because they may rely on post-failure artifacts, produce poorly calibrated scores under telemetry and workload shifts, or learn from labels generated by finite rerun policies. To address these challenges, we present SCOUT, a practical **S**tate-aware **C**ausal **O**nline **U**ncertainty-calibrated **T**riage framework for distributed database CI. SCOUT uses only strict-causal features, including pre-failure telemetry and strictly historical data, to make online decisions without lookahead. Specifically, SCOUT combines lightweight state-aware scoring with optional sparse metadata fusion, applies post-hoc calibration to support fixed-threshold decisions across temporal and cross-domain shifts, and introduces a posterior-soft correction to reduce label bias induced by finite rerun budgets. We evaluated SCOUT on a benchmark of 3,680 labeled failed runs, including 462 flaky positives, and 62 telemetry/context features. Further, we studied the feasibility of SCOUT on TiDB v7/v8 and a large GitHub Actions metadata-only trace. The experimental results demonstrated its effectiveness and usefulness. We deployed SCOUT in the production environment, achieving an end-to-end P95 latency of 1.17 ms on CPU.

[*]These authors contributed equally to this work.
[†]Corresponding author.

Conference'17, Washington, DC, USA
2026. ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

CCS Concepts

• **Software and its engineering → Software testing and debugging**.

Keywords

Distributed database CI; Flaky failure triage

ACM Reference Format:
Jun-Peng Zhu, Qizhi Wang, Yulong Zhai, Yishen Sun, Sen Chen, Xu Kai, Peng Cai, Hongming Zhang, Heng Long, Liu Tang, and Qi Liu. 2026. A Practical Framework for Flaky Failure Triage in Distributed Database Continuous Integration. In . ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Continuous integration (CI) is a core mechanism for maintaining software quality, but in distributed database systems, CI failures are often linked to transient runtime state rather than to persistent defects alone. Lock contention spikes, replication lag, storage stalls, and queue bursts can cause a run to fail even when the underlying code or test is not consistently faulty. As a result, once a failure is observed, operators must quickly decide whether to auto-rerun the job as likely flaky or escalate it as likely persistent. This decision is operationally important because unnecessary escalation wastes debugging effort, while unnecessary reruns delay diagnosis and consume scarce CI resources.

Prior work on flaky tests and CI failures has made important progress, but it does not directly solve this deployment setting. Some approaches study flaky test prediction using code evolution and test history [12, 14], while others detect or categorize failures from post-failure symptoms and logs [1]. These works are valuable, but they address different pain points in the CI lifecycle. The history-based methods are more naturally used before execution, whereas symptom-matching methods typically depend on post-failure artifacts. Meanwhile, systems and AIOps research increasingly use multimodal logs and metrics for diagnosis and operations management [5, 15, 24]. However, such approaches are generally aimed at rich diagnosis or root-cause analysis rather than an immediate rerun-versus-escalate decision under strict latency and resource constraints.

This gap matters because online flaky-failure triage is not simply a matter of training a larger model to recognize flakiness. Instead, it raises three deployment challenges. **(C1) Leakage-sensitive features**. Feature leakage is easy to introduce. Post-failure logs, error strings, or future-run information can improve apparent predictive performance while violating the online decision protocol. **(C2) Threshold-dependent decisions**. The downstream action depends on a thresholded decision, not merely on ranking quality. Post-hoc calibration and calibration under shift are well studied [19, 20], and recent work further emphasizes calibration as a decision primitive under explicit costs [8, 11]. In our setting, this means that a model with acceptable PR-AUC can still behave poorly if its scores are miscalibrated under temporal or cross-domain shifts. **(C3) Policy-generated labels**. Rerun-based flakiness labels are policy-generated. With a finite rerun budget, some flaky failures are systematically under-observed, making the problem closely related to positive-unlabeled learning under selection bias [4, 9] and to broader censored-outcome settings studied in econometrics and survival analysis [7, 16, 22]. Modern selective-prediction and uncertainty-control methods [2, 3, 6, 21, 23] are also relevant here, since real deployments must often hedge when calibration data are scarce, or target overlap is poor.

To address these challenges, we present SCOUT, a practical **S**tate-aware **C**ausal **O**nline **U**ncertainty-calibrated **T**riage framework for distributed-database CI. SCOUT addresses **C1** by using only strict-causal inputs, including pre-failure telemetry and strictly historical data, to make online rerun-versus-escalate decisions without lookahead. It addresses **C2** through lightweight state-aware scoring with optional sparse metadata fusion and post-hoc calibration, enabling a single cost-derived threshold to transfer across shifts. It addresses **C3** through a posterior-soft correction that reduces the label bias induced by finite rerun budgets.

We evaluate SCOUT on a benchmark of 3,680 labeled failed runs, including 462 flaky positives, and use 62 telemetry/context features. We also evaluate the feasibility of TiDB v7/v8 and a large GitHub Actions metadata-only trace. The results show that low-level runtime telemetry carries most of the actionable signal in this setting, while calibrated probabilities matter more than heavier model families for portable actions. Under temporal shift, isotonic calibration reduces fixed-threshold decision cost from 765.4 to 496.9. Posterior-soft correction substantially improves calibration to the larger-budget oracle label, reducing ECE from 0.320 to 0.027 and Brier score from 0.213 to 0.096, while remaining robust under correlated reruns. SCOUT also satisfies practical deployment constraints, achieving an end-to-end P95 latency of 1.17 ms on CPU.

To summarize, this paper makes the following contributions:

(1) We formulate flaky-failure triage in distributed database CI as a strict-causal online decision problem, where the system must decide whether to auto-rerun a failed run as likely flaky or escalate it as likely persistent under CPU-only millisecond budgets.

(2) We propose SCOUT, a practical framework for this setting that combines strict-causal feature extraction, lightweight state-aware scoring, and thresholded online decision-making using only pre-failure telemetry, governed pre-run metadata, and strictly earlier-run history.

(3) We develop two deployment-oriented techniques for reliable triage: decision-portable calibration for fixed-threshold actions under temporal and cross-domain shift and posterior-soft rerun-budget correction for mitigating the label bias induced by finite rerun policies.

(4) We conduct a comprehensive evaluation on synthetic benchmarks, a real TiDB v7/v8 trace, and a large GitHub Actions metadata-only trace, and further demonstrate that SCOUT is practical in production, achieving an end-to-end P95 latency of 1.17 ms on CPU.

2 Preliminaries

In this section, we introduce the preliminaries of this paper.

Online flaky-failure triage. Consider a failed primary CI run i. Let x_i denote the evidence available at decision time, and let $y_i \in \{0,1\}$ denote whether the observed failure is flaky, where $y_i = 1$ means that the failure is non-reproducible under reruns and $y_i = 0$ means that it is persistent. A triage policy outputs a score

$$p_i \approx P(y_i = 1 \mid x_i),$$

and maps this score to an action

$$a_i \in \{\texttt{rerun}, \texttt{escalate}\}.$$

The action `rerun` means that the failed job is automatically retried, while `escalate` means that the failure is surfaced for debugging or downstream diagnosis.

Strict-causal features. Each run has a telemetry window around the failure timestamp. Our main benchmark uses only the pre-failure window $[-120\text{s}, 0]$. We treat $[-120\text{s}, +30\text{s}]$ as an explicit leakage ablation (post-failure evidence).

A CI failure triggers strict-causal feature extraction from pre-failure telemetry and history. Lightweight models produce a score, which is optionally calibrated for decision portability and then mapped to an action via a fixed threshold. Labels are generated by a finite rerun policy, motivating rerun-budget correction.

We aggregate only pre-failure telemetry $[-120\text{s}, 0]$ into dense features and use history computed only from earlier primary runs. Post-failure artifacts (including error strings/logs) are blocked and used only in explicit leakage ablations. Also used are pre-run metadata tokens.

Rerun labels and finite-budget semantics. In practice, flaky-failure labels are generated by a finite rerun policy rather than observed directly. For a failed primary run, let R be the available rerun budget. We define the budget-R label as

$$y_i^{(R)} = \begin{cases} 1, & \text{if at least one rerun passes within } R \text{ attempts,} \\ 0, & \text{otherwise.} \end{cases}$$

Under this definition, a failure is labeled flaky if it becomes non-reproducible within the available rerun budget. The label is monotone in the budget:

$$R_1 \le R_2 \implies y_i^{(R_1)} \le y_i^{(R_2)}.$$

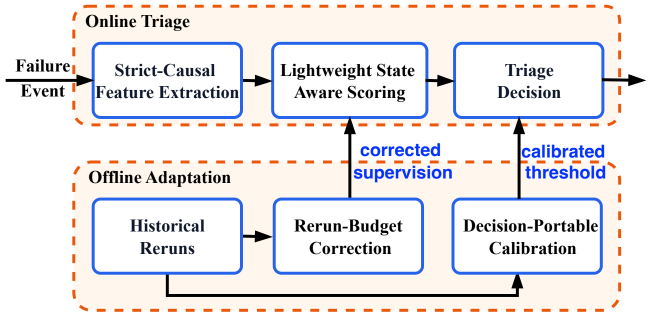


Figure 1: The architecture of SCOUT.

This monotonicity exposes an important deployment issue, where smaller rerun budgets systematically under-observe flaky failures because some failures that would pass on later reruns remain labeled as persistent under a limited budget. Hence, the observed label depends not only on the failure behavior itself but also on the rerun policy used to generate supervision. This one-sided censoring effect motivates the rerun-budget correction proposed later in the paper.

Thresholded decisions. The deployment objective in SCOUT is a thresholded action, not ranking quality alone. Let $\tau \in [0,1]$ be the decision threshold. The policy takes action:

$$a_i = \begin{cases} \texttt{rerun}, & p_i \geq \tau, \\ \texttt{escalate}, & p_i < \tau. \end{cases}$$

Suppose that automatically rerunning a persistent failure incurs cost c_{fp}, missing a flaky failure incurs cost c_{fn}, and issuing an automatic rerun itself incurs overhead c_{auto}. Under calibrated probabilities, the Bayes-optimal threshold is

$$\tau^* = \frac{c_{\text{auto}} + c_{\text{fp}}}{c_{\text{fp}} + c_{\text{fn}}}.$$

3 Overview of SCOUT

This section gives a high-level introduction to SCOUT. Figure 1 illustrates the overall framework of SCOUT. At a high level, SCOUT consists of two tightly coupled paths, which are an *online triage* path for immediate rerun-versus-escalate decisions after a failure is observed and an *offline adaptation* path that improves future online decisions using historical rerun results. The online path contains three components, namely *Strict-Causal Feature Extraction*, *Lightweight State-Aware Scoring*, and *Triage Decision*. The offline path is driven by *historical reruns* and contains two adaptation modules, including *Rerun-Budget Correction* and *Decision-Portable Calibration*. Together, these components form a deployment-oriented framework in the production environment.

The online path starts from a *Failure Event*, which serves as the trigger signal of SCOUT. Once a primary CI run fails, the system activates *Strict-Causal Feature Extraction*. This component is responsible for constructing the model input under a strict-causal protocol. Concretely, it uses only information available no later than the failure observation time, including pre-failure telemetry, pre-run metadata, and historical statistics computed from earlier primary runs. By explicitly blocking post-failure artifacts such as error strings, failure logs, and future rerun information, this component ensures that the inference-time feature space is aligned with the intended online deployment setting.

The extracted feature vector is then passed to *Lightweight State-Aware Scoring*, which forms the predictive core of SCOUT. This component estimates the flaky probability of the failed run from runtime state signals under millisecond-scale, CPU-only serving constraints. The design is intentionally lightweight. SCOUT does not rely on heavy diagnosis pipelines or large multimodal models. It uses compact models over aggregated telemetry, context, and optional sparse metadata features. The goal of this component is not only predictive accuracy but also low-latency, stable, and interpretable online scoring.

The output score is consumed by the *Triage Decision* component. Rather than treating the model output as a ranking signal alone, SCOUT maps the score to a concrete operational action using a fixed decision threshold. In this way, the online path directly implements the deployment objective of the system, which decides whether a failed run should be automatically rerun as likely flaky or escalated as likely persistent. Therefore, this component operationalizes the probability into an actionable policy.

The offline path continuously improves this online policy using *Historical Reruns*. This data source records past rerun results and provides the supervision signal for model adaptation. SCOUT uses it in two ways. First, the *Rerun-Budget Correction* module addresses the fact that observed flakiness labels are generated under a finite rerun budget. Since smaller rerun budgets systematically under-observe flaky failures, this module constructs corrected supervision for training, making the scorer more robust to policy-induced label bias. Its output is fed back to the scoring component as corrected supervision for model updating.

Second, the *Decision-Portable Calibration* module improves the portability of the online decision rule under temporal and cross-domain shifts. Given historical rerun outcomes, this module calibrates model scores so that a single threshold can remain useful across changing workloads, versions, and environments. Instead of requiring per-domain retuning, SCOUT uses this module to update the decision threshold in a way that preserves action quality when the score distribution shifts. Therefore, its output is fed back to the *Triage Decision* component as a calibrated threshold.

Overall, SCOUT separates immediate online inference from slower offline adaptation while connecting them through supervision and decision updates. The online path guarantees strict-causal, low-latency triage for each failed run, and the offline path improves both what the system learns from and how it acts on the predicted flaky probability. This architecture makes SCOUT a practical framework for state-aware flaky-failure triage in a distributed database CI.

4 Methodology

4.1 Strict-Causal Feature Extraction

The *Strict-Causal Feature Extraction* component of SCOUT constructs the input representation for online triage under a strict-causal deployment protocol. Let the time of failure be $t = 0$. SCOUT uses only information available no later than this time and blocks all post-failure artifacts from the main pipeline. As a result, feature

Algorithm 1: Strict-Causal Feature Extraction

```
Input: Failed primary run $i$ observed at failure time $t_i$
Output: Strict-causal feature vector $x_i$
1  Initialize $x_i \leftarrow \emptyset$
2  Collect telemetry in the pre-failure window $[t_i - 120\text{s},\ t_i]$
3  foreach telemetry family $m \in \mathcal{M}$ do
4      collect all observations of $m$ in $[t_i - 120\text{s},\ t_i]$
5      compute window aggregates {mean, max, std, p95}
6      append the aggregates of $m$ to $x_i$
7  Extract governed pre-run metadata tokens and append them
     to $x_i$
8  Compute history features from earlier primary runs only
     and append them to $x_i$
9  foreach feature $f$ with missing telemetry do
10     impute $f \leftarrow 0$
11     record the missingness ratio of $f$
12 if key-feature missingness exceeds the predefined threshold
     then
13     mark the run as escalation-preferred
14 return $x_i$
```

Algorithm 2: Lightweight State-Aware Scoring

```
Input: Strict-causal feature vector $x_i$, scorer type
         $s \in \{\text{StateLR}, \text{TextLR}, \text{SDJLR}\}$
Output: Flaky probability $\hat{p}_i$
1  Split $x_i$ into dense features $x_i^{(d)}$ and sparse metadata tokens
     $x_i^{(s)}$
2  Standardize dense features to obtain $\tilde{x}_i^{(d)}$
3  Construct the sparse TF-IDF vector $v_i^{(s)}$ from $x_i^{(s)}$
4  if $s = \textit{StateLR}$ then
5      set $h_i \leftarrow \tilde{x}_i^{(d)}$
6      compute $\hat{p}_i \leftarrow \sigma(w^\top h_i)$
7  else
8      if $s = \textit{TextLR}$ then
9          set $h_i \leftarrow v_i^{(s)}$
10         compute $\hat{p}_i \leftarrow \sigma(w^\top h_i)$
11     else
12         concatenate $h_i \leftarrow [\tilde{x}_i^{(d)}; v_i^{(s)}]$
13         compute $\hat{p}_i \leftarrow \sigma(w^\top h_i)$
14 return $\hat{p}_i$
```

extraction is constrained to pre-failure telemetry, pre-run metadata, and historical statistics computed from earlier primary runs.

SCOUT extracts three categories of strict-causal features. The first category is *dense telemetry*, which captures the runtime state immediately before failure. The second category is *pre-run metadata tokens*, which provide cheap CI-side context without introducing post-failure leakage. The third category is *strict-causal history*, computed only from earlier primary runs of the same test identity. This design ensures that the input representation remains causally valid for online rerun-versus-escalate decisions. Algorithm 1 summarizes the extraction procedure.

For dense telemetry, SCOUT uses a fixed pre-failure window $[-120\text{s}, 0]$ (line 2). Within this window, it collects low-level metrics from 14 telemetry families, including lock/transaction, Raft, LSM/storage, resource, and queue signals. Concretely, these families cover CPU usage, I/O wait, disk latency, network RTT, lock-wait ratio, transaction retry rate, deadlock count, LSM compaction/flush/write-stall, Raft apply/proposal latency and leader changes, and queue wait. Each metric family is aggregated into window-level summary statistics {mean, max, std, p95} before appending them to the feature vector (lines 3-6). Compared with raw logs or per-timestep sequences, this aggregation produces a compact state representation that is efficient to compute and stable for lightweight online inference.

SCOUT further augments the dense telemetry representation with strict-causal context and history features. Pre-run metadata tokens encode cheap CI-side descriptors such as workload, version, and other governed metadata fields (line 7). Historical features are computed only from earlier primary runs and exclude any lookahead into future reruns or future failures (line 8). In the current design, these features include the test-duration baseline and the historical failure rate. Because sparse metadata can easily leak identity, SCOUT treats feature governance as part of extraction itself. Raw identifiers and post-failure artifacts are excluded from the main protocol.

The component is also designed to be robust to incomplete telemetry. In telemetry-based deployments, some metrics may be missing or delayed within the extraction window. Therefore, SCOUT imputes missing aggregates with 0 and records per-feature missing ratios (lines 9-11). When key features exceed a missingness threshold, the system can disable auto-rerun and fall back to escalation (lines 12-13). In this way, the component provides a compact, auditable, and deployment-compatible feature vector for the downstream scoring module (line 14).

4.2 Lightweight State-aware Scoring

Lightweight state-aware scoring maps the strict-causal feature vector to a flaky probability for online triage. The design goal of this component is to preserve the most actionable pre-failure runtime state while remaining compatible with CPU-only, millisecond-scale deployment. To this end, SCOUT uses a lightweight logistic scoring family rather than heavy sequence or multimodal models.

SCOUT supports three lightweight scoring variants. The first is a *dense state model*, denoted as **State LR**, which applies logistic regression to standardized dense telemetry and context features. The second is a *sparse metadata model*, denoted as **Text LR**, which operates on a lightweight TF-IDF representation of cheap pre-run CI metadata. The third is a *sparse+dense fusion model*, denoted as **SDJ-LR**, which concatenates the standardized dense features with the sparse metadata representation and fits a single logistic regression model. Although these variants use different input representations, they share the same scoring form. For a failed run i, SCOUT predicts

$$\hat{p}_i = \sigma(w^\top h_i),$$

where $\sigma(\cdot)$ is the logistic function and h_i is the representation used by the selected variant.

Algorithm 2 summarizes the scoring procedure. Given a strict-causal feature vector x_i, the scorer first splits it into dense features and sparse metadata tokens (line 1). It then standardizes the dense channel and constructs the sparse TF-IDF representation for the metadata channel (lines 2-3). Next, it applies one of three scoring variants: State LR on the dense channel (lines 4-5), Text LR on the sparse channel (lines 6-7), or SDJ-LR on the concatenated sparse+dense representation (lines 8-10). Finally, it returns the predicted flaky probability (line 11).

The dense state model uses standardized telemetry/context features as input and computes

$$h_i = \tilde{x}_i^{(d)},$$

where $\tilde{x}_i^{(d)}$ denotes the standardized dense telemetry and context features (lines 4-6). The purpose of State LR is to capture the transient runtime state directly from the strict-causal dense representation. Because the dense features have already been aggregated over the pre-failure window, this model avoids expensive sequence processing while remaining suitable for online inference.

For the sparse metadata variant, SCOUT sets

$$h_i = v_i^{(s)},$$

where $v_i^{(s)}$ is the TF-IDF representation of governed pre-run metadata tokens (lines 7-10). The sparse channel is not intended to replace telemetry-based state information. Instead, it provides cheap CI-side context, such as workload, version, and other governed metadata fields, that can complement the dense state representation when available.

For the fusion variant, SCOUT concatenates the two channels (lines 11-13)

$$h_i = [\tilde{x}_i^{(d)}; v_i^{(s)}].$$

In the current design, SDJ-LR uses the SAGA logistic predictor over the combined representation, which preserves interpretability and predictable serving cost while allowing dense runtime state and sparse metadata to contribute jointly to the flaky score.

4.3 Decision-Portable Calibration

The component of SCOUT is to make a single decision threshold transferable across temporal and cross-domain shifts. As defined in the preliminaries, SCOUT uses a fixed cost-derived threshold τ^* to map a calibrated flaky probability to a rerun-versus-escalate action. In SCOUT, the scorer outputs a raw flaky probability $\hat{p}_i$, but the deployment objective is not ranking quality alone. Instead, the system must map this score to a concrete rerun-versus-escalate action under a fixed cost model. Accordingly, SCOUT applies post hoc calibration so that the score used for decision-making is aligned with the deployment threshold.

The main challenge is that a calibrator fitted in one source domain may transfer poorly to a target domain with different workloads, versions, or operating conditions. In this setting, using target labels to tune calibration is often unrealistic. Therefore, SCOUT adopts an overlap-aware, source-only calibration procedure, denoted as **OA-Cal**. Its design follows three principles. First, target labels are never used for calibrator selection. Second, importance

Algorithm 3: Decision-Portable Calibration (OA-Cal)

```
Input: Base scorer $f$, source calibration set $\mathcal{D}_{\text{src}}^{\text{cal}}$, unlabeled
       target inputs $\mathcal{X}_{\text{tgt}}$, candidate calibrators $\mathcal{A}$, threshold
       $\tau^*$
Output: Calibrated mapping $C^*(\cdot)$
1  Compute source scores $\{\hat{p}_j^{\text{src}}\}$ and target scores $\{\hat{p}_k^{\text{tgt}}\}$ using
     $f$
2  Fit a domain classifier $g$ on source vs. target scores
3  Estimate clipped importance weights and effective sample
     size (ESS) from $g$
4  if ESS is low or source-target separability is extreme then
5      disable importance weighting
6      use uniform weights on the source calibration set
7  else
8      enable importance weighting on the source calibration
         set
9  foreach candidate calibrator $C \in \mathcal{A}$ do
10     perform repeated source-only cross-fits on $\mathcal{D}_{\text{src}}^{\text{cal}}$
11     fit $C$ on one split using the selected weights
12     evaluate fixed-threshold decision cost at $\tau^*$ on the
         held-out split
13     record the mean cost $\mu_C$ and standard deviation $\sigma_C$
14 Select
       $C^* = \arg\min_{C \in \mathcal{A}} (\mu_C + \kappa\sigma_C)$
     using Cost-UCB
15 Fit $C^*$ on the full source calibration set with the selected
     weighting scheme
16 return $C^*(\cdot)$
```

weighting is applied only when source-target overlap is adequate. Third, calibrator selection is driven by the deployment objective at the fixed threshold τ^*, rather than by generic calibration fitness alone.

Algorithm 3 summarizes the procedure. Given a trained base scorer, a labeled source calibration set, and unlabeled target inputs, OA-Cal first computes source and target raw scores (line 1). It then fits a domain classifier on source versus target scores and derives overlap diagnostics, including clipped importance weights and effective sample size (lines 2-3). When overlap is poor, weighting is disabled, and the procedure falls back to unweighted source calibration (lines 4-8). Next, OA-Cal evaluates a set of candidate calibrators using repeated source-only cross-fits (lines 9–13). In each cross-fit, a calibrator is fitted on one split and evaluated on the other split by the fixed-threshold decision cost at τ^*, yielding a mean cost and a variance estimate. Finally, SCOUT selects the calibrator using a Cost-UCB criterion, fits it on the full source calibration set, and returns the resulting calibrated mapping (lines 14-16).

In the current SCOUT design, the candidate set includes lightweight post-hoc mappings such as sigmoid scaling, isotonic regression, beta calibration, and compact tree- or binning-based calibrators. When the available calibration set is small, or overlap diagnostics are poor, SCOUT favors smoother low-variance mappings and

Algorithm 4: Rerun-Budget Correction via Posterior-Soft Target

```
Input: Observed first-$R$ rerun outcomes for run $i$, oracle
       budget $R' > R$, Beta prior parameters $(\alpha, \beta)$
Output: Corrected soft target $\tilde{y}_i^{(R \to R')}$
1 if at least one pass is observed in the first R reruns then
2     set $\tilde{y}_i^{(R \to R')} \leftarrow 1$
3 else
4     update posterior: $q \mid \text{first } R \sim \text{Beta}(\alpha, \beta + R)$
5     compute
          $m_i \leftarrow \prod_{j=0}^{R'-R-1} \frac{\beta + R + j}{\alpha + \beta + R + j}$
      as the posterior probability of no pass in the remaining
      $R' - R$ reruns
6     set
          $\tilde{y}_i^{(R \to R')} \leftarrow 1 - m_i$
      as the posterior-soft any-pass target
7 return $\tilde{y}_i^{(R \to R')}$
```

excludes brittle high-variance ones. In this way, this component turns the raw scorer output into a deployment-compatible probability that can support portable rerun-versus-escalate decisions without per-target threshold retuning.

4.4 Rerun-Budget Correction

The component of SCOUT addresses the label bias induced by finite rerun policies. As introduced in the preliminaries, the observed flaky label depends on the available rerun budget, i.e., with a smaller budget R, some failures that would be labeled flaky under a larger budget $R' > R$ remain labeled as persistent because no pass is observed within the first R reruns. Consequently, the training label is policy-generated and one-sidedly censored. Therefore, the component replaces the finite-budget hard label with a larger-budget soft target that better reflects the decision-time oracle of interest.

SCOUT adopts a simple posterior-imputation strategy. Let y_R denote the any-pass label under rerun budget R, and let $R' > R$ denote the oracle rerun budget used as the correction target. Suppose the first R reruns are observed. If at least one pass already appears in this prefix, then the larger-budget label is deterministically positive, and the soft target is set to 1. Otherwise, all observed reruns fail, and SCOUT places a Beta prior on the per-rerun pass probability $q \sim \text{Beta}(\alpha, \beta)$. After observing an all-fail prefix of length R, the posterior becomes

$$q \mid \text{first } R \sim \text{Beta}(\alpha, \beta + R).$$

The posterior-soft target for the larger-budget any-pass label is then

$$\tilde{y}_i^{(R \to R')} = P(y_{R'} = 1 \mid \text{first } R) = 1 - E[(1 - q)^{R'-R}],$$

which has the closed form

$$\tilde{y}_i^{(R \to R')} = 1 - \prod_{j=0}^{R'-R-1} \frac{\beta + R + j}{\alpha + \beta + R + j}$$

in the all-fail case. Thus, SCOUT replaces the censored hard label with the posterior probability that a larger rerun budget would observe at least one pass.

Algorithm 4 summarizes the correction procedure. Given the first R rerun outcomes and an oracle budget R', SCOUT first checks whether any pass is already observed in the prefix (lines 1-2). If so, the posterior-soft target is set to 1. Otherwise, it updates the Beta posterior under the all-fail prefix, computes the probability that no pass occurs in the remaining $R' - R$ reruns via the closed-form Beta moment (lines 3-5), and converts it to the posterior-soft any-pass target (line 6). Finally, the resulting corrected supervision is returned for model training (line 7). This is why the correction is both explicit and lightweight. It reduces to a small number of scalar operations per example.

In the current design, the prior parameters (α, β) are estimated by empirical Bayes on the training set using a method-of-moments beta-binomial fit. This keeps the correction tied to the observed rerun regime while avoiding example-specific latent variable inference. Importantly, the corrected target is used only in the offline adaptation path. It does not alter the online decision protocol, but instead changes the supervision used to update the lightweight scorer.

5 Experiment Evaluation

To evaluate the effectiveness of SCOUT, we aim to answer the following questions:

- **RQ1:** How effective is SCOUT for flaky-failure triage compared with lightweight history-, metadata-, tree-, and sequence-based baselines on strict causal features?
- **RQ2:** Is calibration, rather than heavier model classes, the key to portable fixed-threshold decisions under temporal and cross-domain shift?
- **RQ3:** Does rerun-budget correction mitigate the label bias induced by finite rerun policies and improve learning toward a larger-budget oracle label?
- **RQ4:** Is SCOUT practical for deployment under strict-causal, CPU-only online triage constraints?
- **RQ5:** Do the main conclusions of SCOUT transfer to real services and real CI traces beyond the synthetic benchmark?

5.1 Experimental Setup

5.1.1 Benchmarks and Data Sources. Our main benchmark contains 3,680 labeled failed runs, including 462 flaky positives (12.55%), and 62 telemetry/context features. It is generated by a controllable simulator of the distributed database CI with 12,000 primary runs. For each primary run, we sample a workload type, test identity, database version, and fault condition. The simulator covers representative fault families, including network delay, packet loss, CPU pressure, disk I/O noise, and leader change, together with a no-fault case. Each fault is assigned a severity level and translated into structured perturbations over pre-failure telemetry, such as network RTT, retry rate, CPU utilization, disk latency, and Raft-related indicators. These perturbations affect both the primary-run failure probability and a run-specific rerun-pass probability. For every fail-like primary run, we simulate a rerun budget of $R = 8$ and assign a flaky label if at least one rerun succeeds; otherwise, the failure is labeled persistent. We use this benchmark as the primary testbed for studying strict causal feature extraction, distribution shift, fixed-threshold decision portability, and rerun-policy-induced label bias.

To reduce over-alignment to a fixed simulator feature table, we also construct a fully generative synthetic benchmark in which dense telemetry is sampled from a nonlinear manifold with localized failure modes and temporal drift in the latent mixture. This benchmark is used as a stress test for whether the main conclusions continue to hold under a more flexible data-generating process.

For real-service feasibility evaluation, we deploy TiDB v7 and v8 clusters in Docker together with Prometheus monitoring and execute controlled SQL workloads under both container-level and workload-level fault injection. The container-level faults include stopping the TiDB service and restarting a TiKV node, while the workload-level failures include timeout-inducing sleeps, write conflicts, lock-wait timeouts, and deadlocks. For each failed primary run, we perform a rerun budget of $R = 5$ and assign a rerun-based flaky label using the same any-pass rule as in the main benchmark. This produces a real-service dataset of 1,600 runs, 341 labeled failures, and 194 positives under the rerun-based label. Telemetry is collected from both Prometheus queries and Docker container statistics.

For large-scale CI validation, we collect public GitHub Actions workflow-run metadata, which has 157,807 workflow runs (157,639 completed) from 36 repositories spanning distributed databases and data systems. We derive SHA-group failure episodes by grouping runs by `(repo, workflow_id, head_sha)`, taking the first completed non-success run as the primary failure, and searching for up to three subsequent completed runs within a 24-hour horizon. This yields 36,241 episodes, including 3,953 labeled-valid episodes with at least one observed rerun and 32,288 censored episodes with no observed rerun. An episode is labeled valid if at least one subsequent run is observed; otherwise, it is treated as censored. For labeled-valid episodes, we assign a rerun-based label indicating whether any observed subsequent run succeeds. Features in this benchmark are strictly metadata-only, including workflow/event/branch/timestamp information and strict causal history features.

5.1.2 *Environment.*

All experiments are conducted in a CPU-only setting, consistent with the deployment target of SCOUT. For latency evaluation, we use an Apple M4 machine as the main local measurement platform and additionally report a conservative `Linux/AMD64` environment. This separation allows us to distinguish development-time latency from a more deployment-like setting. The online path is evaluated under repeated calls so that the reported latency reflects steady-state serving behavior rather than one-time initialization cost.

5.2 Effectiveness on Strict-Causal Features (RQ1)

5.2.1 *Setup.*

We evaluate the main triage task on the synthetic benchmark of 3,680 labeled failed runs. All models use only pre-failure telemetry from the window $[-120\text{s}, 0]$, together with governed pre-run metadata and history features computed only from earlier primary runs. For the main ranking comparison, we use a temporal split that trains on the earliest 80% of the runs and tests on the latest 20%. We report PR-AUC as the primary metric under class imbalance, together with ROC-AUC, ECE(10), and Brier score. We compare SCOUT against several lightweight baselines and model families. These include a *History+churn LR* baseline with four cheap features, *State LR* on dense strict-causal telemetry/context features, *Seq LR* on flattened per-timestep telemetry, *Text LR* on sparse metadata-only tokens, and *SDJ-LR* as sparse+dense fusion. We further include tuned tree baselines, namely HistGB, Random Forest (RF), and LightGBM, to test whether slightly heavier tabular models provide substantial additional value under the same strict causal feature extraction.

Table 1: Temporal split on strict-causal features (window $[-120\text{s}, 0]$). Sparse baselines use metadata-only tokens (no injected fault tags). Seq LR flattens per-metric per-timestep telemetry ($5 \times 14 = 70$ dims). Rows shaded in light orange and marked with † denote SCOUT scoring variants; unshaded rows are baselines.

Model	PR-AUC	ROC-AUC	ECE(10)	Brier
History+churn LR	0.124	0.535	0.386	0.247
State LR (dense)†	0.199	0.637	0.345	0.226
Seq LR (timepoint, dense)	0.179	0.639	0.346	0.226
Text LR (TF-IDF)†	0.123	0.515	0.363	0.245
SDJ-LR (dense+sparse)†	0.206	0.628	0.324	0.221
HistGB (tuned sweep)	**0.213**	**0.645**	0.339	0.214
RF (tuned sweep)	0.194	0.613	**0.207**	**0.149**
LightGBM (tuned sweep)	0.199	0.627	0.261	0.180

Table 2: Temporal 60/20/20: fixed-τ^* decision cost across models after post-hoc calibration (lower is better). SDJ-LR uses governed sparse metadata tokens (no commit/test identifiers). Rows shaded in light orange and marked with † denote SCOUT scoring variants; unshaded rows are baselines.

Model	PR-AUC	Uncal cost	Sigmoid cost	Isotonic cost
State LR (dense)†	0.191	765.4	538.4	**496.9**
Spline GAM (dense)	0.236	766.4	531.0	503.0
Tiny GRU (dense seq)	0.197	765.4	544.5	521.2
Tiny CNN (dense seq)	0.201	765.4	**523.0**	558.9
SDJ-LR (dense+sparse, no-id)†	0.196	764.2	536.0	508.0
HistGB (dense)	0.209	765.4	546.1	557.2
RF (dense)	0.184	**756.9**	570.0	547.0
LightGBM (dense)	0.215	766.9	575.5	534.4

5.2.2 *Results.*

Table 1 reports the temporal-split ranking results under the strict-causal window. The *History+churn LR* is weak (PR-AUC 0.124), and the metadata-only *Text LR* is similarly limited (PR-AUC 0.123), indicating that cheap history-only or metadata-only signals are insufficient in this setting. In contrast, dense strict-causal telemetry substantially improves triage quality, i.e., *State LR* reaches PR-AUC 0.199 and ROC-AUC 0.637. The sequence-aware but still *lightweight Seq LR* baseline underperforms the window-aggregated dense model (PR-AUC 0.179 vs. 0.199), suggesting that simple pre-failure window statistics capture most of the usable dynamics for

Table 3: Calibration-transfer baselines for decision portability at a single global threshold τ^* (lower is better). Cells report cost/ECE(10) on the target domain at fixed τ^*. OACal uses overlap-gated weighting and source-only Cost-UCB selection. Rows shaded in light green and marked with † denote the SCOUT-specific calibration method; unshaded rows are baselines.

Method	Temporal (60/20/20)	Cross-workload	$v7.x \to v8$	$v8 \to v7.5$
Uncal	765.4/0.337	1224.5/0.336	1244.9/0.326	1290.5/0.323
Sigmoid	538.4/0.006	869.5/0.025	1153.9/0.041	930.6/0.007
BetaCal	524.7/0.016	**839.3**/**0.024**	995.9/0.038	**930.5**/0.023
Isotonic	**496.9**/0.017	870.5/0.035	875.5/0.033	998.8/0.066
CalibTree	532.5/0.025	1004.5/0.062	934.9/0.045	1091.4/0.073
Venn–Abers	501.5/0.019	867.4/0.029	**867.9**/**0.018**	1012.8/0.056
BBQ-lite	556.7/0.034	941.0/0.033	899.9/0.027	985.9/0.034
Cluster-TS	748.9/0.310	1224.5/0.317	1246.0/0.326	1280.2/0.309
OA-Cal†	**496.9**/0.017	**839.3**/**0.024**	875.5/0.033	**930.5**/0.023

online triage. Adding governed sparse metadata on top of dense telemetry yields a modest improvement, i.e., *SDJ-LR* reaches PR-AUC 0.206 and improves over both dense-only and sparse-only logistic baselines.

Among the tuned tree models, *HistGB* achieves the highest PR-AUC (0.213), followed by *LightGBM* (0.199) and RF (0.194). However, the gain over SDJ-LR is small in absolute terms, and the tree models do not provide a decisive advantage in the ranking setting. This already suggests that low-level runtime telemetry is the main source of useful features, while model-class complexity contributes only marginal additional benefit.

To evaluate whether this observation still holds when decisions are made through a fixed global threshold, we further compare model families under a disjoint 60/20/20 train/calibration/test split. As shown in Table 2, uncalibrated costs are uniformly high across model classes. In particular, isotonic-calibrated *State LR* attains the lowest decision cost (496.9), outperforming calibrated tree baselines such as *HistGB* (557.2), *RF* (547.0), and *LightGBM* (534.4). *SDJ-LR* remains competitive after calibration (508.0), while the ms-feasible temporal encoders *Tiny GRU* and *Tiny CNN* do not improve fixed-threshold decision cost despite slightly stronger sequence-modeling capacity.

Under the strict-causal protocol, SCOUT achieves effective flaky-failure triage with lightweight state-aware models. Dense pre-failure telemetry provides the dominant source of actionable features, clearly outperforming history-only and metadata-only alternatives, while simple window-level aggregation already captures most of the useful runtime dynamics. Sparse metadata contributes only a modest complementary gain when fused with dense telemetry. Although slightly heavier models may offer small ranking improvements, these gains do not translate into clear advantages in deployment-oriented fixed-threshold decisions. Overall, the results validate the core design choice of SCOUT, i.e., practical online triage is better served by strict causal runtime state and lightweight scoring than by increasing model complexity.

Table 4: OA-Cal gate ablations for decision portability (cells show cost/ECE(10) at fixed τ^*). The light-green column marked with † denotes the final SCOUT calibration method; unshaded columns are ablations of OA-Cal.

Scenario	n_{cal}	OA-Cal†	+force weights	+allow isotonic
Temporal	736	496.9/0.017	496.9/0.017	496.9/0.017
Cross-workload	497	**839.3**/**0.024**	1004.5/0.062	870.5/0.035
$v8 \to v7.5$	246	930.5/0.023	930.5/0.023	930.5/0.023

5.3 Decision Portability Under Shift (RQ2)

5.3.1 Setup. To isolate the effect of calibration from the effect of model-family choice, we fix the base scorer to the dense *State LR* model, which attains the lowest calibrated fixed-threshold decision cost among the model families evaluated in Table 2. We compare the following calibration baselines: *Uncal*, *Sigmoid*, *BetaCal*, *Isotonic*, *CalibTree*, *Venn–Abers*, *BBQ-lite*, *Cluster-TS*, and the proposed *OA-Cal*. Evaluation is performed in four settings: a temporal 60/20/20 split, a cross-workload transfer, and two cross-version transfers ($v7.x \to v8$ and $v8 \to v7.5$). In all cases, a single global threshold τ^* is fixed from the deployment cost model and is not re-tuned on the target domain. We report fixed-threshold decision cost at τ^* and ECE(10) as the primary metrics. For OA-Cal, target labels are never used. The calibrator selection is based only on the source calibration split and unlabeled target scores, using overlap diagnostics, optional importance weighting, and Cost-UCB selection.

5.3.2 Results. Table 3 shows that calibration is already essential in the in-domain temporal setting. On the temporal 60/20/20 split, the uncalibrated scorer incurs a decision cost of 765.4 with ECE 0.337. All calibrators reduce this cost substantially, with isotonic reaching 496.9/0.017 and OA-Cal matching the same result by selecting isotonic on the large calibration set. Thus, even before cross-domain transfer is considered, raw scores are not reliable enough to support a fixed-threshold deployment rule.

The effect becomes stronger under a shift. In the cross-workload setting, the uncalibrated decision cost rises to 1224.5 with ECE 0.336, showing that naive threshold reuse is unsafe. The best portable cost is 839.3/0.024, achieved by BetaCal and matched by OA-Cal, while isotonic degrades to 870.5/0.035 and CalibTree further degrades to 1004.5/0.062. Cluster-TS fails to improve over the uncalibrated score at all. In the $v7.x \to v8$ transfer, Venn-Abers achieves the lowest cost (867.9/0.018), while OA-Cal matches isotonic at 875.5/0.033. In the harder $v8 \to v7.5$ transfer, smoother mappings are clearly preferable. OA-Cal attains 930.5/0.023, matching the best portable cost and improving substantially over isotonic (998.8/0.066). Taken together, these results show that no single calibrator dominates across all shifts, but well-chosen calibration is consistently more important than naive threshold reuse.

In the cross-workload setting, forcing importance weighting under extreme source-target separability sharply worsens portability, raising cost/ECE from 839.3/0.024 to 1004.5/0.062 as shown in Table 4. Allowing isotonic in the same poor-overlap setting also increases cost from 839.3 to 870.5. These ablations show that it is necessary to avoid weighting and high-variance calibrators when transfer is difficult. By contrast, in the temporal split and

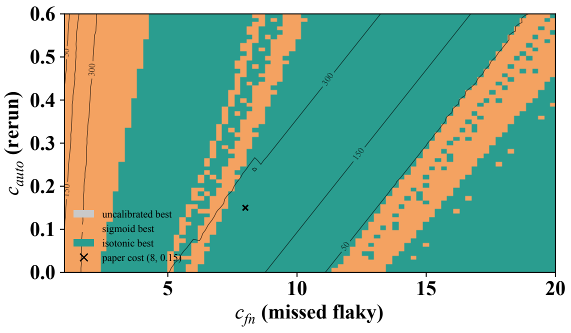


Figure 2: Cost dominance map over $(c_{\text{fn}}, c_{\text{auto}})$ with c_{fp}=1 on the temporal 60/20/20 split. Uncalibrated scores are never best; isotonic dominates in 71% of grid points and sigmoid in 29%.

Table 5: Robustness to cost misspecification at fixed τ^* on the temporal 60/20/20 split (c_{fp}=1). Even under 2× errors in c_{fn} or c_{auto}, calibrated probabilities typically reduce decision cost relative to uncalibrated scores.

Scenario	τ^*	Uncal	Sigmoid	Isotonic
baseline	0.128	765.4	538.4	496.9
$c_{fn}/2$	0.230	698.4	324.0	305.9
$2 \times c_{fn}$	0.068	765.4	770.9	757.1
$2 \times c_{auto}$	0.144	874.5	533.5	537.1

the moderate-overlap $v8 \longrightarrow v7.5$ setting, the OA-Cal gates do not hurt because the procedure naturally falls back to the safer choice. Figure 2 further confirms this pattern. The importance weighting helps only under moderate overlap and becomes harmful under extreme separability, while the cost-UCB parameter is stable over a broad range.

The calibration benefit is also robust to moderate misspecification of the cost model. Figure 3 shows that uncalibrated scores are never optimal over the explored $(c_{\text{fn}}, c_{\text{auto}})$ grid, whereas isotonic dominates most of the grid and sigmoid dominates the remainder. Table 4 shows the same trend numerically. On the temporal split, calibrated scores remain preferable under 2× perturbations of c_{fn} or c_{auto}. For example, when $c_{\text{fn}}/2$, isotonic reduces decision cost from 698.4 to 305.9; when $2 \times c_{\text{auto}}$, it reduces cost from 874.5 to 537.1. These results indicate that the gain from calibration is not an artifact of a single threshold choice.

Calibration is the key to portable fixed-threshold decisions under shift. The main performance gains come from transforming raw scores into reliable probabilities, not from further increasing model complexity. However, no single calibrator is uniformly best across all domains. The isotonic is strongest with a large in-domain calibration set, whereas smoother mappings are safer under scarce calibration data or poor source-target overlap. OA-Cal captures this trade-off without using target labels by gating importance weighting and selecting low-variance calibrators when transfer is risky. Therefore, SCOUT should treat calibration as part of the

Table 6: Selective-labeling stress test (synthetic): only some error signatures receive reruns/labels. We compare naive training on the censored y_3 label against posterior-soft correction with an unweighted empirical-Bayes (EB) prior and an IPW-corrected EB prior. Lower ECE is better; evaluation is against the oracle y_8 label. Columns shaded in light green and marked with † denote SCOUT-specific rerun-budget correction variants.

Selection	Naive ECE	Posterior-soft ECE†	Posterior-soft+IPW ECE†
none	0.338	0.010	0.010
mild	0.304	0.010	0.014
strong	0.377	0.022	0.031

Table 7: Rerun-budget correction on the synthetic benchmark: simulate $R = 3$ labels and evaluate against the oracle $R = 8$ any-pass label. The empirical-Bayes prior is $\alpha = 0.79$ and $\beta = 42.20$. Cost is measured at the fixed τ^* induced by $(c_{\text{fp}}, c_{\text{fn}}, c_{\text{auto}}) = (1, 8, 0.15)$. Rows shaded in light green and marked with † denote the SCOUT rerun-budget correction method; unshaded rows are baselines.

Method	ECE(10)	Brier	Cost@ τ^*	Auto@ τ^*
Naive train on y_3	0.320	0.213	757.4	0.99
Posterior-soft target (ours)†	**0.027**	0.096	591.7	0.49
Direct-q GLM (binom)	0.028	0.096	525.5	0.38
Direct-q (early-stop naive)	0.028	**0.095**	**521.0**	0.37
Discrete hazard (censored@$R = 3$)	0.039	0.096	551.5	0.44
PU correction (Elkan–Noto)	0.598	0.495	765.4	1.00

Table 8: Rerun-budget correction stress tests (mean over 5 seeds): sticky Markov correlation (ρ), attempt drift (cooldown/backoff vs. contention), an intervention regime, and a hidden-state warm-cache process. Columns shaded in light green and marked with † denote SCOUT posterior-soft correction results.

Scenario	Naive ECE	Posterior-soft ECE†	Posterior-soft Brier†
$\rho = 0.0$	0.309	0.017	0.105
$\rho = 0.3$	0.329	0.019	0.085
$\rho = 0.6$	0.355	0.008	0.059
cooldown ($\gamma = 0.01$)	0.139	0.045	0.225
contention ($\gamma = 0.01$)	0.309	0.023	0.037
intervention ($p = 0.3, \Delta = +0.2$)	0.161	0.026	0.219
warm-cache ($p = 0.4, \Delta = +0.2$)	0.243	0.069	0.197

deployment policy itself, rather than as an optional post-processing step.

5.4 Rerun-budget Correction (RQ3)

5.4.1 Setup. We simulate finite-budget supervision by truncating rerun labels to $R = 3$ and evaluate all methods against the larger-budget oracle label y_8 on the benchmark. To isolate the effect of label correction, we keep the downstream strict-causal scoring pipeline fixed and vary only how the training target is constructed. We

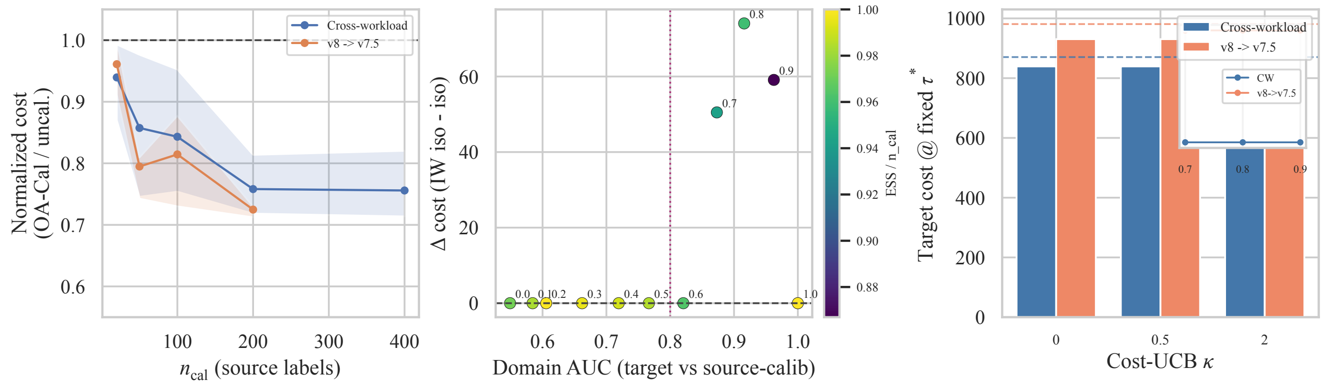


Figure 3: OA-Cal sensitivity for decision portability. Left: label-scarce learning curve (OA-Cal normalized fixed-τ^* cost versus calibration-set size n_{cal}) for two transfer scenarios. Middle: controlled overlap sweep (mix target with source; annotate mix fraction; color = ESS/n_{cal}); importance weighting helps only under moderate overlap. Right: fixed-τ^* cost versus Cost-UCB κ (bars) with an inset sweeping the AUC gate threshold.

Table 9: Real-service rerun censoring on TiDB: simulate $R = 3$ labels and evaluate decisions against the oracle $R = 5$ any-pass label. Rows shaded in light green and marked with † denote the SCOUT rerun-budget correction method; unshaded rows are baselines.

Method	Cost@ τ^*	ECE(10)	Brier
Naive train on y_3	60.0	0.195	0.269
Posterior-soft target†	**45.8**	0.157	0.257
Discrete hazard (censored@$R = 3$)	52.0	**0.106**	**0.233**

compare the baselines, including naive training on y_3, the proposed posterior-soft target, a direct-q Binomial GLM, a direct-q early-stop naive variant, a discrete hazard model for censored reruns, and a PU correction baseline. We report ECE(10), Brier score, and fixed-threshold decision cost at τ^*, together with the auto-rerun rate when relevant. The empirical-Bayes prior used by the main posterior-soft model is $\alpha = 0.79$ and $\beta = 42.20$.

To stress-test robustness beyond the main synthetic setting, we further evaluate three additional scenarios. First, we simulate *selective labeling*, where reruns are observed only for some error signatures, and compare posterior-soft correction with and without IPW-corrected empirical Bayesian priors. Second, we evaluate *correlated rerun* and *attempt-drift* regimes, including sticky Markov dependence, cooldown/contention drift, intervention, and warm-cache hidden-state processes. Third, we test *real-service rerun censoring* on the TiDB trace by simulating $R = 3$ labels and evaluating against the oracle $R = 5$ any-pass label.

5.4.2 Results. Table 6 shows that posterior-soft correction remains highly stable under selective labeling. Across none, mild, and strong selection regimes, naive training on the censored y_3 label yields ECE values of 0.338, 0.304, and 0.377, whereas posterior-soft reduces these to 0.010, 0.010, and 0.022, respectively. The IPW-corrected empirical-Bayes variant is also effective but does not improve over the unweighted posterior-soft baseline and becomes slightly worse under stronger selection. This suggests that the main benefit comes from correcting the finite-budget label semantics itself, rather than from additional reweighting.

Table 7 reports the main synthetic comparison against the oracle y_8 label. The proposed posterior-soft target substantially improves calibration relative to naive training on y_3, reducing ECE from 0.320 to 0.027 and Brier score from 0.213 to 0.096, while also lowering fixed-threshold decision cost from 757.4 to 591.7. Direct-q alternatives achieve slightly lower decision cost (525.5 and 521.0), but only when the first-R pass-count information is fully observed and the Binomial count model is appropriate. In contrast, many CI systems stop on the first pass, which censors rerun counts and makes direct-q methods less natural without additional attempt-level instrumentation or censored-likelihood modeling. The discrete hazard baseline is also competitive, but the PU correction baseline performs very poorly, indicating that the finite-rerun problem is not well captured by a generic positive-unlabeled treatment.

The robustness results in Table 8 further support the correction design. Across sticky Markov dependence ($\rho \in \{0.0, 0.3, 0.6\}$), attempt-drift regimes, intervention, and warm-cache hidden-state processes, posterior-soft consistently yields much lower ECE than the naive censored-label baseline. In particular, under stronger correlation ($\rho = 0.6$), naive ECE rises to 0.355, whereas posterior-soft drops to 0.008. This indicates that the method remains effective even when reruns are not i.i.d., which matches the intended use of posterior-soft as a lightweight first-order correction rather than a brittle exact model.

Finally, Table 9 shows that the same pattern extends to a real-service trace. On TiDB, posterior-soft reduces fixed-threshold decision cost from 60.0 to 45.8 and lowers ECE from 0.195 to 0.157 relative to naive training on y_3. The hazard baseline attains the lowest ECE and Brier score on this trace, but its decision cost remains higher than posterior-soft (52.0 vs. 45.8). Thus, posterior-soft provides the best fixed-threshold action quality on the real-service correction task, even when a more specialized censoring model can achieve a slightly better calibration fit.

Table 10: Leakage guard evaluation: PR-AUC under disjoint splits with identifier tokens removed. Rows shaded in light orange and marked with † denote SCOUT scoring variants.

Split	State LR†	Text LR†	SDJ-LR†
Temporal 80/20	0.199	0.107	0.197
Leave-one-`test_id`-out	0.241	0.114	0.237
Leave-one-`commit_id`-out	0.239	0.128	0.239

Rerun-budget correction mitigates the label bias induced by finite rerun policies, and the proposed posterior-soft target is an effective default correction for SCOUT. It markedly improves calibration relative to naive finite-budget labels and remains robust under selective labeling, correlated reruns, and a real-service trace. Although direct-q and hazard-based alternatives can be competitive or even slightly better on some metrics when stronger assumptions or richer attempt-level observations are available, posterior-soft is the most broadly compatible correction for SCOUT because it aligns with any-pass label semantics and remains well-defined under early-stop censoring. These results justify using posterior-soft correction as the default supervision update mechanism in the offline adaptation path.

5.5 Deployment Practicality (RQ4)

5.5.1 Setup. We evaluate deployment practicality from three perspectives. First, we evaluate leakage robustness by enforcing disjoint splits and removing identifier tokens from the sparse channel, and by additionally opening a post-failure window only in an explicit leakage ablation. Second, we measure latency in both a local Apple M4 and a conservative `Linux/AMD64` setting, reporting end-to-end online latency as well as model-only inference latency. Third, we assess interpretability using global feature-attribution summaries, coefficient-sign inspection, and a face-validity perturbation check on canonical state features.

5.5.2 Results. Table 10 reports the leakage-guard evaluation. Under the strict temporal 80/20 split, the SCOUT scoring variants achieve PR-AUCs of 0.199 (State LR), 0.107 (Text LR), and 0.197 (SDJ-LR). Under leave-one-`test_id`-out and leave-one-`commit_id`-out splits with identifier tokens removed, the SCOUT variants remain stable rather than collapsing, State LR reaches 0.241/0.239, and SDJ-LR reaches 0.237/0.239, respectively. These results show that the main dense strict causal feature is not driven by test- or commit-level shortcuts. The explicit post-failure leakage ablation provides further evidence. Opening the post-failure window [−120s, +30s] does not improve the dense LR baseline (PR-AUC 0.190), indicating that the main results are not explained by hidden post-failure evidence. Window-length sensitivity follows the same pattern. PR-AUC is 0.180, 0.191, and 0.199 for 30s, 60s, and 120s pre-failure windows, while extending the window to 240s does not improve further because the telemetry artifact is concentrated within the last 120 seconds.

Figure 4 summarizes the CPU-only latency breakdown. On Apple M4, the end-to-end online triage path has P95 latency 1.17 ms in the state-only pipeline, with feature extraction 0.77 ms and inference 0.40 ms. For SDJ-LR, the additional sparse processing cost

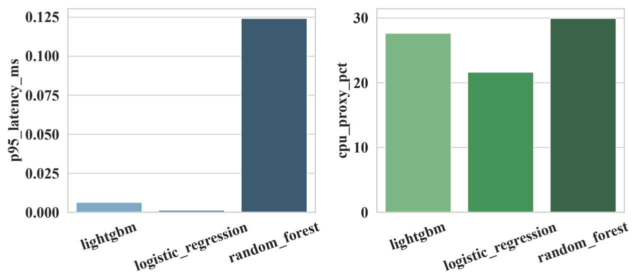


Figure 4: CPU-only latency breakdown in Apple M4. We separate model-only inference from the end-to-end serving path to avoid under-reporting deployment latency.

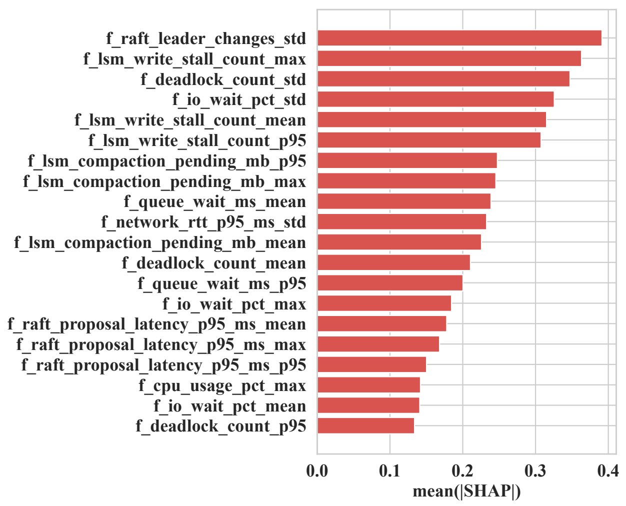
remains negligible relative to feature extraction, including vectorization, scaling, and inference, which together contribute only P95 0.0074 ms. In a `Linux/AMD64`, model inference is P95 3.0 μs/example and the strict-causal aggregation kernel is P95 0.77 ms/example. These results indicate that SCOUT is compatible with millisecond-scale CPU-only deployment as long as telemetry retrieval is not performed synchronously over remote queries. The four Prometheus instant queries used in external validation have P95 35.9 ms with keep-alive and 38.5 ms cold, which suggests that production deployment should rely on local agents, cached telemetry, or recording rules rather than per-decision remote fetching.

Figure 5 shows that the learned dense strict-causal model is interpretable at both the global and local levels. The SHAP top-20 summary highlights intuitive runtime-state drivers of flakiness, including lock-wait ratio, Raft apply latency, LSM flush duration, and queue wait. The coefficient signs are also stable and operationally plausible. The LSM compaction backlog (+0.27), queue-wait P95 (+0.16), and network RTT P95 (+0.14) increase predicted flakiness, whereas persistent severity features such as LSM write-stall count (−0.35), deadlock count (−0.17), and Raft proposal latency (−0.12) decrease it. The face-validity perturbation check further confirms this behavior. Increasing a canonical transient-state feature monotonically increases the mean predicted flaky probability, without causing unstable or counterintuitive responses.

SCOUT satisfies the key requirements of deployment practicality. Its strict causal performance does not depend on obvious identifiers or post-failure leakage; its online path remains compatible with millisecond-scale CPU-only serving; and its state-aware logistic scoring rule is sufficiently interpretable for operational debugging and auditing.

5.6 Generalization In Production (RQ5)

5.6.1 setup. For the real-service setting, we run TiDB v7 and v8 with Prometheus telemetry and controlled fault injection, producing 1,600 runs, 341 labeled failures, and 194 positives under the rerun-based label. Telemetry includes resource statistics per container, together with a small set of Prometheus queries, including TiDB QPS, PD TSO wait, and TiKV Raft apply and scheduler latch wait. We evaluate PR-AUC on labeled failures under repeated



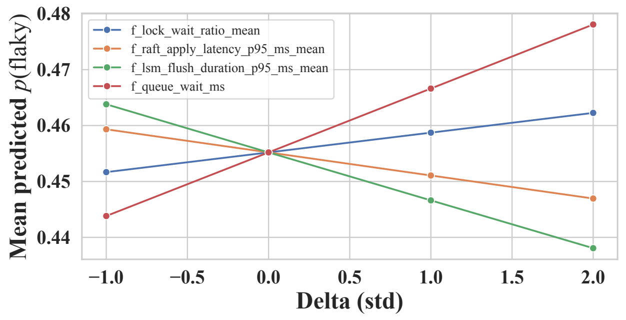


Figure 5: Top: global interpretability summary (state-only LR, strict-causal temporal split). Bottom: face-validity check via do-style perturbations on canonical state features (increase one standardized feature at a time and measure mean predicted $p(\text{flaky})$).

Table 11: TiDB external validation on real services: PR-AUC on labeled failures under repeated 5-fold CV and cross-version transfer. Symptom baselines use only short error strings (no telemetry), and DistilBERT is a frozen offline encoder baseline. Columns shaded in light orange and marked with † denote SCOUT scoring variants; unshaded columns are baselines.

Setting	State LR†	TF-IDF	DistilBERT	Dict	SDJ-LR†
Repeated CV (mean)	**0.623**	0.561	0.560	0.561	0.613
$v8 \longrightarrow v7$	0.541	0.531	0.531	0.531	**0.556**
$v7 \longrightarrow v8$	**0.633**	0.595	0.596	0.595	0.630

cross-validation and cross-version transfer. We compare the methods, including a telemetry-only State LR scorer, sparse symptom-based baselines on short error strings (TF-IDF LR and a dictionary matcher), a heavier frozen DistilBERT symptom baseline, and the sparse+dense SDJ-LR fusion model.

Table 12: GitHub Actions metadata-only validation (sha-group episodes; 36 repositories): fixed-threshold decision quality on labeled-valid episodes, together with a normalized IPW cost sensitivity analysis and a coarse time-saved proxy. Tokens are governed by dropping raw `branch_` tokens and adding coarse `branchb_` buckets and time bins. This table reports supporting calibration baselines on the governed metadata trace.

Method	PR-AUC	ECE(10)	Cost@τ^*	IPW Cost	Net saved (h)
Uncalibrated	0.591	0.140	728.80	706.43	60.4
Sigmoid	0.591	0.039	606.90	546.60	51.0
BetaCal	**0.591**	0.039	**589.45**	**541.17**	**62.5**
CalibTree	0.561	0.042	608.05	547.28	50.5
Isotonic	0.564	**0.030**	608.05	547.28	50.5

For the real-CI setting, we construct a metadata-only rerun trace from public GitHub Actions. Features are strictly metadata-only, which logs and error strings are excluded entirely, and token governance removes raw `branch_` tokens in favor of coarse `branchb_` buckets and temporal bins. Because 89.1% of episodes are censored, results on the labeled-valid subset are subject to selection bias. Therefore, we also report a normalized IPW cost as a best-effort sensitivity analysis.

5.6.2 Results. Table 11 shows that SCOUT variants remain competitive on a fully real TiDB trace. Under repeated 5-fold cross-validation, telemetry-only State LR achieves the highest mean PR-AUC (0.623), while SDJ-LR remains close at 0.613. Both outperform symptom-only baselines based on TF-IDF (0.561), a dictionary matcher (0.561), and frozen DistilBERT embeddings (0.560). Under cross-version transfer, the same conclusion continues to hold. For $v8 \longrightarrow v7$, the best result is SDJ-LR (0.556), followed by State LR (0.541), both ahead of the symptom-only baselines (all 0.531). For $v7 \longrightarrow v8$, State LR achieves the best PR-AUC (0.633), with SDJ-LR close behind (0.630), while the symptom-only baselines remain around 0.595-0.596. These results show that the core SCOUT remains effective on a real distributed database service.

Table 12 evaluates fixed-threshold decisions on the labeled-valid GitHub Actions episodes. In this setting, raw scores are poorly calibrated. The uncalibrated system has ECE 0.140 and cost@τ^* 728.80. Post-hoc calibration substantially improves deployment-oriented action quality, reducing fixed-threshold cost to 606.90 with sigmoid scaling and to 589.45 with BetaCal, while also improving ECE to 0.039. The normalized IPW cost shows the same relative pattern, decreasing from 706.43 to 541.17 under BetaCal. Although PR-AUC changes little across the calibration baselines, the decision quality at the fixed threshold improves considerably, which is consistent with RQ2.

On TiDB, lightweight strict-causal telemetry is sufficient to support competitive flaky-failure triage without heavy symptom models. On GitHub Actions, even in a metadata-only regime with severe rerun censoring and substantial token drift, post-hoc calibration continues to improve fixed-threshold decision quality.

6 Related Work

Flaky failures in CI. Prior work studies flaky tests and intermittent CI failures using code evolution and test history [12], empirical analyses of flaky behavior at scale [14, 18], and symptom/log matching for just-in-time detection and categorization [1]. Our focus is narrower, which is triage *after* a failure, under CPU-only millisecond budgets, where low-level runtime state is available, and interpretability is required. *History- and churn-based approaches are* primarily useful for anticipating instability *before* execution, whereas our setting reacts *after* a concrete failure using strict-causal pre-failure telemetry and earlier-run history.

Telemetry-centric operations modeling. Systems and AIOps work increasingly uses multimodal logs and metrics for diagnosis and monitoring, including live forensics and database failure management [5, 15, 24]. Our strict-causal triage is a *front-end decision*, rerun versus escalate under millisecond CPU budgets. In this sense, it is complementary to heavier diagnosis systems and can be used to gate when such systems should be invoked. Accordingly, we avoid heavy log parsing and GPU/LLM inference and emphasize protocol rigor under deployment constraints rather than richer representation learning.

Calibration under shift. Post-hoc calibration and calibration under covariate shift are well studied [13, 17, 19, 20]. Our work is closest to this line but differs in emphasizing fixed-threshold actions under temporal and cross-domain shifts in CI triage. Rather than evaluating calibration only as a probabilistic goodness property, we study when calibrated probabilities support a single global deployment threshold across domains and compare lightweight post hoc calibrators together with transfer-oriented extensions such as cluster-level temperature scaling [10].

Cost-sensitive and decision-aware calibration. Cost-sensitive learning emphasizes that optimal actions depend on calibrated probabilities and explicit costs [8]. Recent work further formalizes calibration as a *decision* primitive and studies efficient post-hoc procedures for minimizing decision loss [11]. Our decision-portable evaluation follows this line by fixing a cost-derived threshold and studying whether thresholded decisions remain stable under a shift without target-domain threshold retuning.

Selective prediction and risk control. Abstention and reject-option classification have a long history [6]. Modern distribution-free uncertainty methods such as Venn-Abers [23] and split conformal predictors [3] provide a complementary way to trade cost versus coverage, and conformal methods under shift can use reweighting or weighted scores [2, 21]. We include these as lightweight, deployment-compatible baselines in our evaluation. Operationally, they can hedge when overlap/OOV diagnostics indicate high shift or when calibration windows are tiny.

Finite rerun budgets and PU learning. Rerun-budget labeling induces one-sided censoring, where smaller budgets systematically under-observe flakiness. This connects to learning from positive and unlabeled data under selection bias [4, 9]. More broadly, it is a censored-outcome problem related to classical econometric and survival settings, including Kaplan-Meier, Tobit, and Cox-style modeling [7, 16, 22]. Our work differs in targeting a deployable correction that uses only information available at labeling time, namely the limited rerun outcomes, rather than requiring full time-to-event supervision.

7 Conclusion

In this work, we introduced SCOUT, a practical framework for state-aware flaky-failure triage in distributed-database CI. SCOUT targets the deployment setting in which a failed run must be triaged online under strict-causal, CPU-only constraints. To support this setting, it combines strict causal feature extraction, lightweight state-aware scoring, decision-portable calibration, and rerun-budget correction. We evaluated SCOUT on both synthetic benchmarks and real-world CI traces, and the results demonstrated its effectiveness and practical usefulness for flaky-failure triage in distributed database CI.

References

[1] Gabin An et al. 2023. Just-in-Time Flaky Test Detection via Failure Symptom Matching. *arXiv preprint arXiv:2310.06298* (2023).
[2] Anastasios Angelopoulos et al. 2024. Weighted Conformal Predictive Systems under Distribution Shift. *arXiv preprint arXiv:2404.15018* (2024).
[3] Anastasios N. Angelopoulos and Stephen Bates. 2021. A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification. *arXiv preprint arXiv:2107.07511* (2021).
[4] Jessa Bekker and Jesse Davis. 2018. Learning from Positive and Unlabeled Data Under Selection Bias. In *Proceedings of the International Conference on Machine Learning.*
[5] Adan Jorge-Piezzas Chand, Calvin Pei, Colin Stewart, Shuaiwen Leon Jiang, and Keaton Jackson. 2025. AgentFM: Role-Aware Failure Management for Distributed Databases with LLM-Driven Multi-Agents. *arXiv preprint arXiv:2504.06614* (2025).
[6] C. K. Chow. 1970. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory* 16, 1 (1970), 41–46. doi:10.1109/TIT.1970.1054410
[7] D. R. Cox. 1972. Regression Models and Life-Tables. *Journal of the Royal Statistical Society: Series B (Methodological)* 34, 2 (1972), 187–202. doi:10.1111/j.2517-6161.1972.tb00899.x
[8] Charles Elkan. 2001. The Foundations of Cost-Sensitive Learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI).* 973–978.
[9] Charles Elkan and Keith Noto. 2008. Learning classifiers from only positive and unlabeled data. In *Proceedings of KDD.*
[10] Yunye Gong et al. 2021. Confidence Calibration for Domain Generalization Under Covariate Shift. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).* 8958–8967.
[11] Parikshit Gopalan et al. 2025. Efficient Calibration for Decision Making. *arXiv preprint arXiv:2511.13699* (2025).
[12] Martin Gruber et al. 2023. Practical Flaky Test Prediction with Code Evolution and Test History. *arXiv preprint arXiv:2302.09330* (2023).
[13] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *International conference on machine learning.* PMLR, 1321–1330.
[14] Guillaume Haben et al. 2023. Discerning Flaky from Fault-Triggering Test Failures: A Chromium CI Case Study. *arXiv preprint arXiv:2302.10594* (2023).
[15] Saurabh Jha et al. 2019. Live Forensics for Distributed Storage Systems. *arXiv preprint arXiv:1907.10203* (2019).
[16] Edward L. Kaplan and Paul Meier. 1958. Nonparametric Estimation from Incomplete Observations. *J. Amer. Statist. Assoc.* 53, 282 (1958), 457–481. doi:10.1080/01621459.1958.10501452
[17] Meelis Kull, Telmo Silva Filho, and Peter Flach. 2017. Beta Calibration: A Well-Founded and Easily Implemented Improvement on Logistic Calibration for Binary Classifiers. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS).*
[18] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. 2014. An empirical analysis of flaky tests. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering.* 643–653.
[19] Alexandru Niculescu-Mizil and Rich Caruana. 2005. Predicting Good Probabilities with Supervised Learning. *Proceedings of ICML* (2005).
[20] Anusri Pampari and Stefano Ermon. 2020. Calibrated Predictive Systems Under Covariate Shift. In *Proceedings of AISTATS.*
[21] Ryan J. Tibshirani et al. 2019. Conformal Prediction Under Covariate Shift. *arXiv preprint arXiv:1904.06019* (2019).

[22] James Tobin. 1958. Estimation of Relationships for Limited Dependent Variables. *Econometrica* 26, 1 (1958), 24–36.
[23] Vladimir Vovk and Ivan Petej. 2012. Venn-Abers predictors. *arXiv preprint arXiv:1211.0025* (2012).
[24] Lingzhe Zhang et al. 2025. LogDB: Multivariate Log-based Failure Diagnosis for Distributed Databases. *arXiv preprint arXiv:2505.01676* (2025).