Labeling the Invisible: A Scalable Framework for Labeling Fail-Slow Failures in Cloud Storage Systems

Yongxin Zhao[1], Yuxin Sun[1], Shenglin Zhang[1*], Xiang Liu[2], Jiaqi Luan[1], Runzhou Wang[1], Wenwei Gu[1], Yongqian Sun[1], Yidan Li[1], Dan Pei[3]

[1]Nankai University [2]Huawei Cloud [3]Tsinghua University

Abstract

Fail-slow failures, characterized by prolonged performance degradation without complete system crashes, pose a significant threat to the reliability of large-scale cloud storage systems. However, the scarcity of high-quality labeled datasets limits the development of effective detection and diagnosis techniques, while manual labeling at cloud scale is labor-intensive and error-prone. In this paper, we present *SlowSight*, a novel tool designed to enable efficient and accurate labeling of fail-slow failures in cloud environments. *SlowSight* employs a multi-view framework that jointly models temporal deviations of individual components and behavioral divergence among peer components. To reduce spurious detections, a knowledge-driven filtering module removes anomaly candidates inconsistent with fail-slow meanings. Furthermore, *SlowSight* adopts pattern-centric aggregation to capture multi-metric degradation patterns and propagate consistent labels across similar anomaly instances, improving labeling accuracy and consistency. We deploy and evaluate *SlowSight* at *Huawei Cloud* on two production-scale disk fail-slow datasets, achieving F_1-scores of 0.851 and 0.903, and on a fail-slow dataset generated via failure injection with an F_1-score of 0.945, demonstrating its effectiveness and generalizability. To the best of our knowledge, *SlowSight* is the first framework designed for scalable fail-slow failure labeling in cloud storage systems. Additionally, we have publicly made the code available to facilitate further research.

1 Introduction

With the rapid expansion of cloud computing, cloud storage systems have become critical infrastructure across various industries and are widely deployed in numerous application domains [4, 27]. Ensuring the high reliability of cloud storage systems is of great importance for service providers [17, 27, 35]. However, system stability is frequently threatened by various failures [39], which degrade the user experience and incur substantial mitigation costs for service providers [27]. Despite significant efforts in failure diagnosis and mitigation, completely avoiding catastrophic failures remains challenging.

Before catastrophic failure, hardware typically enters a fail-slow phase, where system performance deteriorates without complete failure [20, 21]. In cloud storage systems, storage requests traverse multiple components along the I/O path; thus, performance degradation in a single component can propagate to upper layers, prolonging service degradation and complicating failure localization [21]. Consequently, early detection and mitigation of fail-slow failures are essential to prevent cascading degradation and maintain long-term system reliability [9]. However, existing detection methods [18, 28, 31] primarily target generic time-series anomalies and lack mechanisms to capture hardware degradation patterns and peer-component performance comparisons, both of which are critical for accurate fail-slow diagnosis in storage environments [21, 24]. Moreover, such methods are often environment-specific and generalize poorly across heterogeneous cloud deployments with diverse workloads, hardware configurations, and failure patterns [11]. This highlights a key bottleneck: the scarcity of high-quality, environment-specific labels needed to train and adapt robust fail-slow detectors.

Although several studies have explored hardware failure, selecting and deploying suitable diagnostic algorithms in practical scenarios remains a formidable challenge. On the one hand, there are only a few public fail-slow failures datasets [9, 11, 13, 21], the limited scale and diversity of existing datasets restrict their applicability across different cloud environments. On the other hand, given the complexity of cloud storage systems, fail-slow failures exhibit significant variability across different scenarios [11], making it challenging to generalize existing research findings to real-world production systems. Furthermore, high-quality labeled data is crucial for enhancing fail-slow diagnosis models, yet the scarcity of large-scale, diverse datasets significantly limits research progress and practical applications.

Given the critical role of labeled datasets in advancing

*Shenglin Zhang is the corresponding author (zhangsl@nankai.edu.cn).

machine learning techniques, such as ImageNet's impact on computer vision [8], the field of fail-slow failure detection could similarly benefit from large-scale and diverse datasets. However, acquiring high-quality real-world fail-slow failures datasets presents several challenges. First, cloud service providers often restrict access to operational datasets due to security and privacy concerns [11]. Second, a universally accepted definition of fail-slow failures has yet to be fully established [14, 21], leading to inconsistencies in expert evaluations and labeling. Additionally, cloud storage systems generate vast volumes of hardware performance metrics [11, 25], making manual labeling labor-intensive and error-prone.

To address these problems and alleviate the burden of manual labeling, an automatic labeling tool is proposed to assist operators in improving the efficiency and accuracy of fail-slow failure identification in cloud storage systems. However, building such a tool entails several key challenges:

1) **Massive Data Scale:** Modern cloud infrastructures generate enormous volumes of monitoring data [25], rendering manual labeling methods impractical. For instance, *Huawei Cloud*'s cloud storage system comprises dozens of service clusters per region, each containing hundreds of thousands of hardware components. Each component monitors multiple performance metrics, collectively producing over 10 billion daily data records. Efficiently identifying fail-slow failures from such a vast data pool poses a primary challenge for labeling systems.

2) **Complex Failure Patterns:** Fail-slow failures do not exhibit a uniform manifestation pattern. In practice, they may appear either as transient performance degradation or as persistent deterioration, during which affected hardware components deviate from their own historical normal behavior or from the normal operational profiles of their peers. Consequently, approaches that rely on a single analytical perspective, whether temporal self-deviation or cross-component peer divergence, are inherently inadequate to identify all potential fail-slow instances comprehensively.

3) **Ambiguous Anomalies:** In large-scale systems, anomalous behaviors are pervasive but inherently ambiguous, as many deviations arise from benign workload fluctuations or transient disturbances rather than genuine fail-slow degradations. Therefore, manually labeling each anomaly is prohibitively costly and difficult to scale. Furthermore, the absence of clear semantic boundaries between benign anomalies and fail-slow failures renders such labeling subjective and inconsistent, often resulting in unreliable labels across different operators and operational contexts.

To address the above challenges, *SlowSight*, informed by lessons learned from our earlier efforts, introduces an efficient framework that minimizes manual effort in identifying and labeling fail-slow failures in large-scale cloud storage systems. To capture complex failure dynamics, *SlowSight* adopts a multi-view analysis approach that jointly models long-term temporal degradation trends and cross-metric behavioral divergence, enabling accurate characterization of diverse fail-slow candidate manifestations. Moreover, to reduce ambiguity in anomaly interpretation and the cost of manual labeling, *SlowSight* reformulates result labeling as a representative anomaly pattern identification problem. By first filtering out candidates that are inconsistent with fail-slow behaviors based on their physical meanings, the framework enables operators to label only a small set of canonical patterns, with labels systematically propagated to structurally similar candidates, thereby ensuring both scalability and labeling consistency. Through our deployment at *Huawei Cloud*, we distill three key insights: (1) multi-view modeling is required, since a single view leaves certain fail-slow failures undetected; (2) transparent and interpretable outputs are essential for user trust and adoption; and (3) a semi-automated design is preferable, combining the efficiency of automation with the reliability of operator expertise. Notably, *SlowSight* is primarily designed as a semi-automated fail-slow labeling framework for constructing high-quality, large-scale labeled datasets. Meanwhile, its embedded multi-view analysis modules can also serve as an online fail-slow detector, enabling seamless integration into production monitoring pipelines.

Our key contributions are summarized as follows:

1) **Semi-automated fail-slow labeling framework for cloud storage systems.** We present *SlowSight*, the first framework that combines algorithmic methods with operator expertise to label fail-slow failures in cloud environments. This semi-automated design overcomes the inefficiency of fully manual labeling, enabling operators to assign fail-slow labels efficiently and accurately.

2) **Anomaly candidate extraction via multi-view analysis.** *SlowSight* employs a multi-view approach to extract candidate anomalous intervals automatically. By integrating temporal deviations with peer-component divergences, *SlowSight* identifies potential fail-slow candidates under diverse workloads, enhancing downstream labeling accuracy.

3) **Pattern-centric labeling for ambiguity reduction.** *SlowSight* reformulates result labeling as a representative anomaly pattern identification problem, shifting from individual ambiguous anomalies to structurally consistent patterns. After filtering out anomaly candidates inconsistent with the physical meaning of fail-slow failures, *SlowSight* labels only cluster centroids and propagates labels to all members, making the overall process scalable.

4) We evaluate *SlowSight* on two production-scale disk fail-slow datasets. On an internal dataset from *Huawei Cloud*, *SlowSight* achieves an F_1-score of 0.851, substantially outperforming state-of-the-art baselines, and attains an F_1-score of 0.903 on a public dataset [21], demonstrating its real-world effectiveness. To further assess generalizability, we evaluate *SlowSight* on a fail-slow dataset generated via failure injection, achieving an F_1-score of 0.945.

The source code of our *SlowSight* prototype is available at: https://github.com/AIOps-Lab-NKU/SlowSight.

2 Background and Motivation

2.1 The Unique Challenges of Detecting Fail-Slow Failures in System Reliability

Unlike fail-stop failures that cause immediate and complete outages, fail-slow failures, also referred to as gray failures [14] or limping hardware [9, 15], exhibit partial functionality and performance degradation without explicit system crashes [14]. As modern systems scale in size and complexity, the frequency and impact of fail-slow failures have increased [11], potentially causing persistent performance degradation and cascading disruptions that compromise reliability.

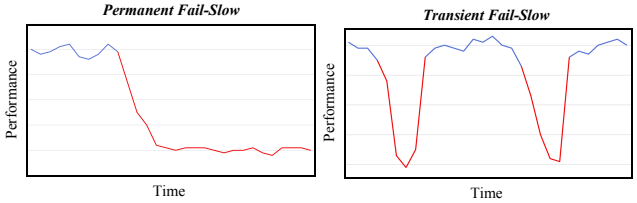
Given the operational risks posed by fail-slow failures, cloud service providers have prioritized their accurate and timely detection [9, 13, 21, 24]. However, most existing detection techniques are designed for certain environments or failure manifestations. Such scenario-specific methods often suffer from poor generalizability, as the manifestations of fail-slow failures can vary substantially across hardware platforms, workloads, and deployment conditions [11].

A fundamental bottleneck hindering the advancement of generalizable detection methods is the lack of large-scale, high-quality datasets containing accurately labeled fail-slow failures. Due to their rarity, subtle manifestations, and dependence on specific contexts, fail-slow failures are challenging to capture and label. This data scarcity severely limits the ability to train, benchmark, and validate robust detection models that can operate reliably across heterogeneous systems. Therefore, accurately identifying and labeling fail-slow failures is foundational for enabling the development of generalized detection techniques and the maintenance of system reliability.

2.2 The Bottlenecks of Manual Labeling for Fail-Slow Failures

Currently, the labeling of fail-slow failures remains predominantly manual, presenting substantial challenges in real-world production environments. Enterprise-scale systems typically collect metrics at fine-grained intervals (*e.g.*, every 1–5 minutes) [12, 23], resulting in massive volumes of multivariate time series data. Identifying fail-slow failures requires operators to carefully examine this data for subtle, often non-obvious anomalies that signal performance degradation [5]. This process is labor-intensive, time-consuming, and prone to subjectivity, particularly in distinguishing fail-slow behavior from normal variability.

Further complicating this challenge is the significant heterogeneity in fail-slow manifestations across diverse system configurations. Effective diagnosis typically necessitates comparative analysis between the suspected component and a substantial population of peer components operating under similar workloads and hardware profiles [24]. Within large-scale cloud environments, the number of components can vary from several hundred to several thousand instances. For even experienced operators, manually conducting such comprehensive comparisons is not only inefficient but also operationally prohibitive due to the scale and complexity involved.



(a) The symptom of permanent fail-slow failures.

(b) The symptom of transient fail-slow failures.

Figure 1: The symptoms of fail-slow failures in hardware components.

The combination of massive data volumes, variable failure patterns, and the need for contextual comparisons renders manual labeling fundamentally unscalable. This creates an urgent need for automatic labeling tools that can reduce the human effort involved, improve labeling accuracy, and accelerate the development of fail-slow detection systems.

3 Characterization, Goals, and Lessons

In this section, we first characterize the symptoms of fail-slow failures in *Huawei Cloud*'s scenario. We then articulate the design goals that guide our fail-slow failure labeling framework. Finally, we review prior unsuccessful attempts based on existing methods and distill the key insights. The corresponding experimental results are summarized in Table 2.

3.1 Characteristics of Fail-Slow Failures

As illustrated in Figure 1, the symptoms of fail-slow failures in *Huawei Cloud* can be broadly classified into two categories:
Permanent Fail-Slow. The first category, shown in Figure 1a, is permanent fail-slow. In this case, once a fail-slow failure occurs, the component remains in a persistently degraded performance state and cannot recover to normal condition until the underlying issue is resolved.
Transient Fail-Slow. The second category, shown in Figure 1b, is transient fail-slow. Unlike the permanent form, this symptom is characterized by performance fluctuating between degraded and normal states. The degraded state is typically short-lived, often lasting only a few minutes, before temporarily recovering.

3.2 Design Goals of the Labeling Framework

A practical strategy for labeling fail-slow failures must begin with their accurate identification. Accordingly, effective detection methods are expected to satisfy several key requirements:

Non-intrusiveness. The method should rely on external observability data, since service providers neither control user software nor can require application modifications.
Accuracy. The method should achieve high precision and recall, thereby minimizing the operational risks associated with undetected fail-slow failures while avoiding unnecessary overhead caused by false positives.
Generality. The method should generalize across diverse classes of hardware fail-slow failures without relying excessively on case-specific empirical thresholds.
Efficiency. The labeling framework should reduce reliance on manual effort while preserving high accuracy, supporting automated and scalable labeling of fail-slow failures.

3.3 Lessons from Previous Unsuccessful Attempts

3.3.1 Attempt 1: Threshold-Based Detection

Threshold-based detection represents an intuitive and non-intrusive approach to identifying hardware performance anomalies. For instance, Lu et al. [22] employ predefined rules (*e.g.*, $3rd_quartile + 2IQR$) to detect fail-slow disks.

However, such methods face significant limitations in accuracy. In large-scale cloud environments, performance metrics exhibit substantial variability across workloads, making it difficult to define applicable thresholds. Furthermore, threshold selection inherently involves a trade-off: loose thresholds improve recall but reduce precision, leading to excessive false positives, while strict thresholds enhance precision but risk missing less severe fail-slow failures. In addition, threshold configuration relies heavily on operator expertise, and tuning thresholds for heterogeneous hardware across diverse workload clusters remains time-consuming and labor-intensive.

3.3.2 Attempt 2: Time-Series Anomaly Detection

A distinguishing characteristic of fail-slow failures is the gradual deviation of a system component's performance metrics from their historical norms. As illustrated in Figure 2, metrics such as disk latency and utilization remain within a narrow and consistent range during healthy states. In contrast, during a fail-slow failure, these metrics demonstrate significant deviations from their historical distributions. Motivated by this observation, unsupervised time-series anomaly detection methods [18, 28, 31] can identify fail-slow failures in a non-intrusive manner, without requiring manual data labeling.

These methods are effective for detecting transient fail-slow failures when sufficient normal historical data are available, as deviations appear as short-lived anomalies. However, their robustness deteriorates when historical data are sparse or contaminated. For permanent fail-slow failures, prolonged degradation progressively pollutes the baseline of normal behavior, causing the learned notion of normality to drift toward faulty behavior. In addition, most existing approaches require per-component model training, resulting in substantial computational overhead in large-scale systems.

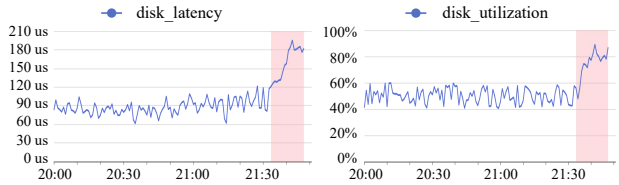


Figure 2: Evolution of key disk metrics during a fail-slow failure. The red shaded region denotes the fail-slow period.

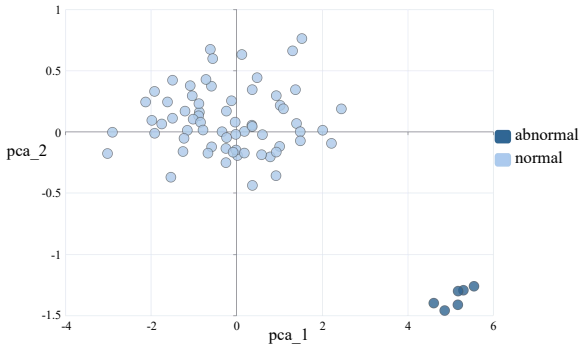


Figure 3: Clustering analysis of peer disks during a fail-slow failure.

Finding 1: Unsupervised time-series anomaly detection is effective for transient fail-slow failures given sufficient normal historical data, but degrades for permanent fail-slow failures due to baseline contamination from prolonged degradation.

3.3.3 Attempt 3: Peer-Based Fail-Slow Detection

Another defining characteristic of fail-slow failures is their component-specific manifestation, whereby affected components exhibit behavior that markedly diverges from that of their peers [21, 24]. As shown in Figure 3, disks experiencing fail-slow failures form distinct clusters, clearly separating from the majority of unaffected disks. This behavioral divergence provides compelling empirical support for peer-based anomaly detection in identifying fail-slow failures. A key advantage of peer-based approaches is that they operate without long-term historical baselines, instead identifying anomalies through contemporaneous cross-component comparisons. This property makes them particularly effective for detecting permanent fail-slow failures when reliable normal historical data are unavailable or contaminated.

Several frameworks have been proposed to detect fail-slow failures using peer evaluation. IASO [24], for example, primarily leverages software-level timeout signals, which it transforms into a stable and accurate fail-slow metric. However, IASO requires intrusive modifications to application code, limiting its practicality in production environments, and its

node-level detection granularity is insufficient for precise diagnosis. In contrast, PERSEUS [21] employs lightweight latency-versus-throughput (LvT) regression models to rapidly localize and analyze fail-slow failures at the drive level. Nevertheless, in our context, LvT distributions frequently violate the assumptions underlying PERSEUS, resulting in suboptimal performance, which is explicitly noted in the study [21].
Finding 2: Peer-based comparison methods leverage contemporaneous cross-component contrasts to identify deviations without requiring extensive historical data; however, their efficacy degrades when anomalies affect a substantial proportion of the peer group.

3.3.4 Attempt 4: Combined Time-Series and Peer-Based Fail-Slow Detection

Combining time-series and peer-based fail-slow detection methods facilitates the identification of potential anomalies or outliers. However, not all detected anomalies correspond to genuine fail-slow failures. Many deviations result from benign workload fluctuations or transient operational effects, and a naive combined approach therefore yields excessive false positives rather than actionable fail-slow labels.

For example, in the context of disk-related fail-slow failures, a persistent increase in utilization and service time accompanied by a decline in throughput typically indicates a degradation in the device's processing capacity. In contrast, simultaneous increases in utilization and throughput under high-load conditions often reflect expected system behavior, where elevated latency arises from queuing effects rather than hardware failures. These examples underscore that accurate interpretation of anomalies requires domain knowledge and operational experience: operators must examine and label representative anomalies to determine which truly correspond to fail-slow behavior. Manually labeling all detected anomalies, however, is prohibitively time-consuming and labor-intensive, particularly in large-scale cloud environments.
Finding 3: Detected anomalies do not necessarily correspond to operationally meaningful fail-slow failures; effective labeling therefore depends on expert judgment to identify a limited set of representative fail-slow patterns, rather than exhaustively labeling all anomalous instances.

4 Design of SlowSight

4.1 Overview

This paper presents *SlowSight*, a novel tool designed for the accurate and efficient identification and labeling of fail-slow failures. As shown in Figure 4, *SlowSight* comprises three key modules:
1. Data Preparation (§4.2). *SlowSight* periodically collects runtime monitoring metrics from hardware components through a dedicated collection tool. A preprocessing step ensures the consistency and reliability of the collected data, preparing it for subsequent analysis.
2. Anomaly Candidate Extraction (§4.3). This module processes raw monitoring data to extract fine-grained degradation signals, identifying statistically significant deviations that may indicate potential fail-slow behaviors and serve as input for subsequent labeling
3. Pattern-Centric Labeling (§4.4). This module formulates fail-slow labeling as a representative anomaly pattern identification problem. It first applies knowledge-driven filtering to remove candidates inconsistent with fail-slow characteristics. By constructing component-level multi-metric anomaly segments and clustering structurally similar segments, *SlowSight* reduces labeling to a small set of canonical patterns, whose labels are propagated to all corresponding instances. An interactive visualization interface further supports efficient human-in-the-loop inspection, enabling scalable and consistent labeling in large-scale cloud environments.

4.2 Data Preparation

Reliable real-time monitoring is fundamental to accurately identifying and labeling fail-slow failures in large-scale cloud storage systems. In production environments, however, monitoring data are inherently imperfect due to the system's scale, heterogeneity, and operational complexity. Based on longitudinal observations across multiple production clusters, we identify two recurring data-quality artifacts: (1) missing observations caused by transient network disruptions or monitoring agent restarts, and (2) invalid metric values that violate basic physical constraints (*e.g.*, negative resource utilization). These artifacts arise from inconsistencies in the monitoring pipeline rather than genuine performance degradation. Treating them as anomalies would introduce spurious signals and consequently bias downstream analysis.

Although infrequent, such artifacts can substantially degrade the effectiveness of long-horizon, high-dimensional anomaly detection. To mitigate this issue, *SlowSight* incorporates a data preprocessing stage to restore signal validity and temporal continuity. Invalid values are filtered using predefined feasibility constraints, while linear interpolation is applied only to short isolated gaps, preserving continuity without obscuring anomalies. To further address scale heterogeneity across metrics, *SlowSight* applies min–max normalization before analysis, ensuring that detected deviations reflect relative variation rather than absolute magnitude and enhancing the robustness of subsequent detection and labeling.

4.3 Anomaly Candidate Extraction

We adopt a multi-view anomaly candidate extraction design that unifies two complementary perspectives: (1) a temporal view that sensitively captures subtle deviations from a component's own historical dynamics, and (2) a peer view

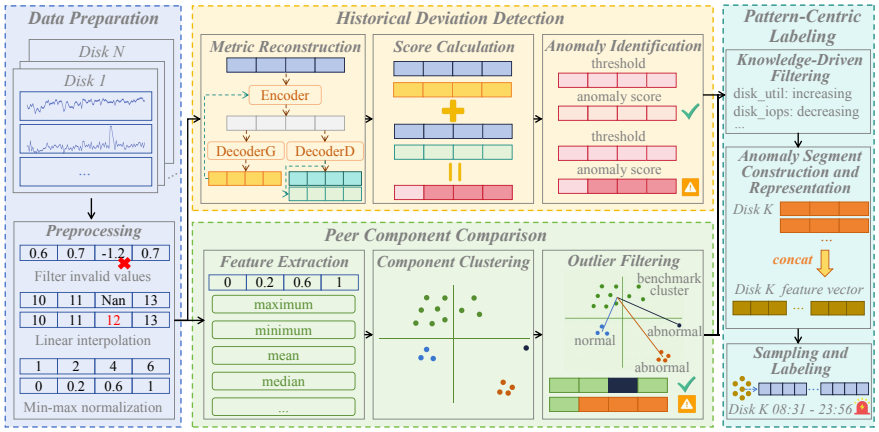


Figure 4: The overall framework of *SlowSight*.

that highlights components deviating from contemporaneous behavior within a homogeneous group.

This design is motivated by the failure modes identified in production telemetry: temporal unsupervised detectors can be highly sensitive but degrade when historical data are scarce or already contaminated by anomalies (Finding 1), whereas peer-based comparisons remain effective without long histories yet become unreliable under cluster-wide disturbances that shift all peers together (Finding 2). *SlowSight* therefore models these views as complementary, conditionally reliable sources of anomaly evidence and combines them at the level of candidate intervals rather than final labels.

4.3.1 Historical Deviation Detection

Reconstruction-based models are widely adopted for time-series anomaly detection [1, 28, 30, 34, 36], as they learn normal behavior and detect anomalies via reconstruction errors. They are particularly suitable for fail-slow detection due to sensitivity to distribution shifts. Despite their effectiveness, accurately detecting fail-slow failures remains challenging. Effective detection requires joint reasoning across multiple metrics [21], yet many existing methods struggle to capture complex inter-metric dependencies. Moreover, fail-slow failures often appear as weak but persistent deviations [11], limiting the effectiveness of detectors based solely on instantaneous reconstruction errors. To address these challenges, *SlowSight* introduces a *Historical Deviation Detection* module that integrates variate-level attention with adversarial training to enhance sensitivity to gradual, multi-metric deviations.

Variate Attention-Based Encoder. Fail-slow failures typically emerge as gradual shifts in dependencies among performance metrics rather than abrupt anomalies in individual metrics. *SlowSight* therefore employs a variate-centric attention mechanism that treats each metric as a token and computes attention across variates rather than over time. This design explicitly models inter-metric dependencies and their evolution, enabling effective detection of correlation anomaly that is difficult to capture using temporal attention alone [18].

Adversarial Dual-Decoder. Unlike conventional single-decoder designs, *SlowSight* employs a dual-decoder architecture consisting of a Generator (*Decoder_G*) and a Discriminator (*Decoder_D*). The Generator reconstructs variates from latent representations, while the Discriminator distinguishes original from reconstructed variates. The training objective of each decoder can be formulated as:

$$\mathcal{L}_G = \frac{1}{n}\|\mathbf{X} - \mathcal{G}(\mathcal{E}(\mathbf{X}))\|_2 + \frac{n-1}{n}\|\mathbf{X} - \mathcal{D}(\mathcal{E}(\mathcal{G}(\mathcal{E}(\mathbf{X}))))\|_2 \tag{1}$$

$$\mathcal{L}_D = \frac{1}{n}\|\mathbf{X} - \mathcal{D}(\mathcal{E}(\mathbf{X}))\|_2 - \frac{n-1}{n}\|\mathbf{X} - \mathcal{D}(\mathcal{E}(\mathcal{G}(\mathcal{E}(\mathbf{X}))))\|_2 \tag{2}$$

where $\mathbf{X}$ represents an input window of variates, $\mathcal{E}(\cdot)$ denotes the encoder, $\mathcal{G}(\cdot)$ denotes the Generator Decoder, $\mathcal{D}(\cdot)$ denotes the Discriminator Decoder, and n is the current training epoch.

Adversarial training between the Generator and Discriminator decoders enables the model to capture subtle second-order reconstruction discrepancies arising from gradual inter-metric dependency drift during fail-slow degradation. This objective amplifies weak yet systematic signals that would otherwise be masked by noise, thereby enhancing sensitivity to early-stage fail-slow behaviors.

Anomaly Score Calculation and Threshold Selection. Anomaly scores are computed by quantifying the reconstruction errors between the input variates and their reconstructions. Inspired by adversarial learning methods, we design a dual-term anomaly scoring mechanism that explicitly balances precision and recall in anomaly detection. Formally,

$$Score(\mathbf{X}) = \frac{1}{2}\|\mathbf{X} - \mathcal{G}(\mathcal{E}(\mathbf{X}))\|_2 + \frac{1}{2}\|\mathbf{X} - \mathcal{D}(\mathcal{E}(\mathcal{G}(\mathcal{E}(\mathbf{X}))))\|_2 \tag{3}$$

To address the limitations of static thresholds in dynamic cloud environments [38], *SlowSight* employs the Streaming

Peaks-Over-Threshold (SPOT) algorithm [26], which adaptively determines detection thresholds based on Extreme Value Theory.

Transfer Learning. Training separate models for each hardware component is costly in large-scale systems [29]. *SlowSight* mitigates this via transfer learning: a base model is trained on representative components, and only a subset of parameters is fine-tuned for new components while others remain fixed. Transfer is applied within homogeneous component types (*e.g.*, disk-to-disk) but not across heterogeneous types (*e.g.*, disk and NIC), balancing scalability and accuracy.

4.3.2 Peer Component Comparison

In large-scale cloud infrastructures, prolonged fail-slow degradation may leave insufficient or contaminated normal data for training reconstruction-based models. To complement historical modeling, *SlowSight* adopts peer-based comparison, leveraging behavioral similarity among components under comparable workloads [21] for spatial anomaly identification via clustering with adaptive filtering, which confines comparison to groups of similar components. The adaptive parameterization enables *SlowSight* to effectively balance detection sensitivity and robustness across diverse operational settings and failure characteristics.

Clustering-Based Outlier Identification. *SlowSight* begins by transforming raw multivariate time-series data into compact statistical feature vectors using sliding windows, thereby preserving essential behavioral characteristics while reducing temporal redundancy. Principal Component Analysis (PCA) is then applied to project the feature space into a lower-dimensional representation, improving pattern separability and reducing computational overhead. The resulting embeddings are clustered using DBSCAN, which identifies dominant behavioral clusters and potential outliers without requiring a predefined number of clusters, making it well suited for heterogeneous production environments.

Adaptive Outlier Filtering. To enhance detection accuracy and minimize false positives, *SlowSight* implements an adaptive outlier filtering mechanism comprising three stages:

1) **Cluster Categorization:** Clusters are categorized as either (1) Benchmark Clusters, representing dominant behavioral patterns with component proportions exceeding a predefined threshold α, or (2) Candidate Clusters, smaller clusters with fewer components ($< \alpha$) that may correspond to secondary patterns or potential outliers.

2) **Distance-Based Outlier Determination:** Components within Candidate Clusters are evaluated based on their Euclidean distances to all Benchmark Cluster centroids. A component is classified as anomalous if its distance to any Benchmark Cluster centroid exceeds β times that cluster's radius, where β controls detection sensitivity.

3) **Temporal Consistency Validation:** To mitigate the impact of transient fluctuations and ephemeral operational variations, a component must consistently exhibit anomalous behavior across N consecutive sliding windows to be definitively confirmed as an outlier, where N is a scenario-dependent parameter determined by the target labeling requirements.

4.4 Pattern-Centric Labeling

The objective of pattern-centric labeling in *SlowSight* is to identify fail-slow components from numerous detected anomalies while minimizing manual effort. In production environments, anomalous metrics are frequent and often benign; labeling all anomalies is therefore neither scalable nor reliable. Instead, *SlowSight* formulates labeling as a representative pattern identification problem: structurally similar anomaly segments are grouped, and only representative patterns require manual inspection.

4.4.1 Knowledge-Driven Filtering

Given the anomaly candidates identified by the upstream module, *SlowSight* first determines potential fail-slow components to maximize recall. Statistical deviation alone is insufficient, as fail-slow behavior must also align with the physical semantics of performance degradation. To address this, *SlowSight* introduces a knowledge-driven filtering stage grounded in metric semantics. Specifically, TSFRESH [6] extracts trend-related features (*e.g.*, slope, correlation coefficient, and statistical significance) to characterize the dominant evolution pattern of each metric. Based on these trends, *SlowSight* applies meaning-aware rules to eliminate candidates inconsistent with fail-slow characteristics. For example, a sustained decrease in disk I/O latency under stable throughput indicates performance improvement and is excluded. In contrast, progressively increasing latency, persistent throughput degradation, and prolonged resource contention are retained, as they reflect fail-slow patterns.

Importantly, the knowledge-driven filtering incurs minimal overhead. The rules derive from widely accepted operational principles and can be obtained from historical failure analyses, troubleshooting guides, or industry best practices [11], rather than ad hoc dataset-specific tuning, without requiring extensive expert labeling or system-specific customization. After filtering, the remaining candidates are statistically significant and semantically consistent with fail-slow behavior, and are forwarded for anomaly segment construction. We acknowledge that no static rule set can achieve complete coverage: unmatched candidates are passed through to the clustering stage for operator review, with confirmed patterns incorporated back as new rules.

4.4.2 Anomaly Segment Construction and Representation

For each retained fail-slow candidate, *SlowSight* identifies a component-level anomaly onset time and centers a fixed-

length window on this point to capture the local evolution of degradation. The same window is applied to all monitored metrics, producing a temporally aligned multi-metric anomaly segment. By anchoring segments to a unified onset, *SlowSight* preserves cross-metric temporal consistency and captures both the emergence and early progression of anomalous behavior. This design facilitates distinguishing gradual fail-slow degradation from transient fluctuations and enables meaningful comparison across components and workloads.

Clustering of multi-metric time-series segments is hindered by high dimensionality and noise. To derive compact and discriminative representations, *SlowSight* applies TSFRESH [6] to each segment for feature extraction. For each metric, TSFRESH computes statistical, temporal, and frequency-domain features (*e.g.*, distributional statistics, correlation, and trend descriptors), which are concatenated into a unified feature vector representing the component-level anomaly segment. The feature-based abstraction suppresses low-level noise while preserving degradation patterns, enabling robust similarity measurement across anomaly segments.

4.4.3 Scalable Sampling and Labeling

To group anomaly segments with similar degradation behaviors, *SlowSight* applies hierarchical agglomerative clustering (HAC) to the extracted feature vectors. Unlike partition-based methods, HAC does not require a predefined number of clusters, making it well-suited for exploratory settings where failure pattern diversity is unknown. Clustering is performed using Euclidean distance in the feature space. To select the appropriate granularity, *SlowSight* evaluates clustering quality using the Davies–Bouldin (DB) index, which balances intra-cluster compactness and inter-cluster separation. The configuration minimizing the DB index is chosen. Each cluster represents a canonical anomaly pattern shared across components, allowing operators to label only the cluster centroid and propagate the label to all members, thereby substantially reducing manual effort while preserving consistency.

To improve scalability, *SlowSight* further adopts a workload-aware sampling strategy. Components are first grouped by workload similarity using the Peer Component Comparison module. Within each group, a random subset of anomaly segments is sampled for clustering and labeling. This approach preserves pattern diversity while significantly reducing computational overhead and labeling cost.

4.4.4 Interactive Labeling Interface

To facilitate efficient human-in-the-loop labeling, *SlowSight* provides an interactive graphical interface that enables operators to explore anomaly clusters, inspect representative prototypes, and refine labeling results when necessary. The interface supports synchronized visualization of multi-metric anomaly segments, comparative analysis across components and clusters, and management of labeled datasets, thereby ensuring transparency and usability in practical deployment.

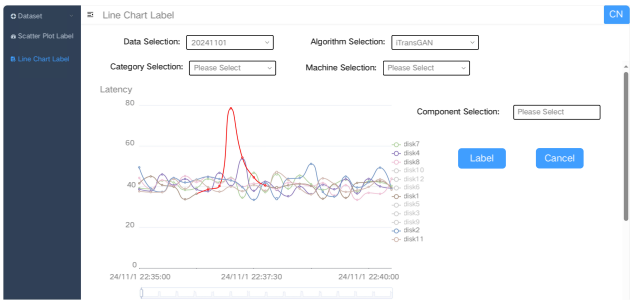


Figure 5: The interface of *SlowSight*.

Dataset	#Components	#Fail-slow Failures	#Records
D1	1,261	22	50,843,520
D2	42,017	299	671,244,629

Table 1: Dataset information (For each component, any single metric at a given timestamp is represented as one record).

5 Evaluation

We aim to answer the following research questions (RQs):
RQ1: How well does *SlowSight* perform in fail-slow failures detection?
RQ2: Does each module of *SlowSight* contribute significantly to *SlowSight*'s performance?
RQ3: Does *SlowSight* produce higher-quality fail-slow labels than the threshold-based approach?

5.1 Experimental Setup

Dataset. To evaluate the performance of *SlowSight*, we conduct experiments on two production-scale datasets, D1 and D2, collected from distinct environments, as summarized in Table 1. Both datasets consist of component-level monitoring metrics sampled at one-minute intervals. D1 is chronologically divided into training and test subsets, where the first 25,421,760 records form the training set and the subsequent 25,421,760 records constitute the test set; the training set contains only normal data. D2 is a publicly available dataset that contains test data only.

Production Dataset (D1) comprises telemetry from *Huawei Cloud*'s distributed Object Storage Service. To ensure workload and hardware diversity, we sampled a representative subset from November 2024 across multiple regional clusters, spanning 1,261 disks (HDDs and SSDs). For each disk, we collect six metrics at 1-minute resolution: read/write latency, service time, read/write IOPS, and utilization. D1 contains 22 confirmed fail-slow incidents; the resulting fail-slow rate

(1.7%) is consistent with the 1.4% [21] for similar device types, suggesting that D1 is broadly aligned with industry field evidence. To establish reliable ground truth, potential anomalies flagged by a loose-threshold monitoring mechanism are independently cross-verified by two O&M experts using historical and peer comparisons. Only unanimously confirmed cases are labeled, with the dataset scale constrained primarily by this rigorous manual labeling process.

Public Dataset (D2) is derived from PERSEUS [21] and contains performance logs from approximately 42,017 HDDs and SSDs in production, including 297 confirmed fail-slow drives. As D2 provides only test data, it cannot be used to evaluate approaches requiring training (*e.g.*, OutSpot [28], TimesNet [31], iTransformer [18], or the Historical Deviation Detection module of *SlowSight*).

Implementation. We implement *SlowSight* with Python 3.8. All experiments are conducted on a Linux server with 8 × Intel(R) Xeon(R) Gold 6266C CPU @ 3.00GHz, 62.7 GB RAM, and without GPU support. For the hyperparameters, the input sequence length is 38, the number of heads is 8, and the dimension of the model is 128, respectively. We repeat each experiment three times and average the results to minimize the effect of randomness.

Baselines. To evaluate the effectiveness of *SlowSight*, we compare it with representative state-of-the-art methods for fail-slow detection and time-series anomaly analysis. For all baselines, we follow the original implementations and parameter settings to ensure fairness and reproducibility. Specifically, we include: (1) Lu et al. [22], a threshold-based method for fail-slow disk detection; (2) PERSEUS [21], which uses polynomial regression over latency and throughput distributions to derive adaptive thresholds; (3) OutSpot [28], which combines hierarchical clustering with a conditional variational autoencoder for outlier detection; (4) iTransformer [18], an attention-based model with dimensional inversion for anomaly detection; and (5) TimesNet [31], which converts one-dimensional time series into multi-periodic two-dimensional representations to enhance detection performance.

Evaluation Metrics. We employ Precision, Recall, and F_1-score as the evaluation metrics to assess the effectiveness of fail-slow failure detection, which are widely used in anomaly detection tasks [28, 31]. For evaluation, a detection is considered a True Positive (TP) if the reported component is correct and its identified time interval overlaps with the labeled failure duration, as operators are more concerned with identifying anomalous segments as a whole rather than isolated anomalous points.

Evaluation Criteria Fail-slow anomalies typically manifest as contiguous temporal segments, and a component may experience multiple failure–recovery cycles. We therefore adopt the component–interval pair as the evaluation unit and follow the widely used point-adjusted method [12, 23, 33]. Under this method, detecting any portion of a labeled segment counts the entire segment as one TP. Consecutive spurious detections, or detections within three minutes, are merged into a single FP interval. For instance, if a fail-slow segment spans 10:00–10:05 and a detection is triggered at 10:02, the full segment is recorded as one TP. Similarly, spurious detections at 11:00, 11:01, and 11:03 are merged into a single FP interval.

True Negatives (TNs) are excluded because the point-adjusted method operates at the segment level. Given the dominance of normal periods, counting all normal points as TNs would distort performance assessment. Consistent with prior work [21], we therefore report Precision, Recall, and F_1-score, which more accurately reflect the practical objectives of identifying and localizing fail-slow failures.

5.2 Overall Performance (RQ1)

Method	D1			D2		
	Precision	Recall	F_1-score	Precision	Recall	F_1-score
SlowSight	**0.8**	**0.909**	**0.851**	0.896	0.911	0.903
Lu et al. [22]	0.789	0.682	0.732	**1.0**	0.52	0.684
PERSEUS [21]	0.692	0.409	0.514	0.997	**1.0**	**0.998**
OutSpot [28]	0.778	0.636	0.7	-	-	-
iTransformer [18]	0.45	0.409	0.429	-	-	-
TimesNet [31]	0.458	0.5	0.478	-	-	-

Table 2: Effectiveness of fail-slow failure detection.

As shown in Table 2, *SlowSight* achieves the best performance on D1 with an F_1-score of 0.851. It delivers higher precision than the strongest baseline while maintaining high recall to capture early-stage degradations. The slight margin of recall over precision stems from the Historical Deviation Detection module's sensitivity to transient workload spikes, a challenge largely mitigated by our knowledge-driven filter. Specifically, on permanent and transient fail-slow failures, *SlowSight* achieves a recall of 1.00 and 0.875, respectively, compared with 0.83 and 0.25 for PERSEUS [21], validating its efficacy in detecting diverse fail-slow dynamics.

Lu et al. [22] identify fail-slow failures using latency quartiles as thresholds, which performs well when latency follows an approximately normal distribution but suffers significant precision and recall degradation under non-normal conditions. PERSEUS [21] models the latency–throughput relationship via polynomial regression; however, it may misclassify normal latency increases caused by high concurrency as fail-slow failures. Both approaches rely on expert-defined scoring functions and static system assumptions, limiting their adaptability to workload variability. Moreover, they cannot detect fail-slow failures in real time based on duration, potentially missing transient failures.

Although effective for general time series anomaly detection, OutSpot [28], iTransformer [18], and TimesNet [31] are limited for fail-slow failure detection: they fail to capture the distinctive temporal degradation patterns of fail-slow failures and lack mechanisms for peer-component comparison and false positive filtering, both of which are essential for accurate detection in large-scale systems.

To further evaluate the robustness of *SlowSight*, we also conduct experiments on the D2 dataset [21]. Since D2 lacks training data and the baseline implementations are closed-source, we compare directly with the results reported in PERSEUS [21]. While PERSEUS [21] performs marginally better on D2, *SlowSight* demonstrates superior robustness across both datasets. This performance gap highlights dataset discrepancies: D1 contains continuous telemetry under complex, heterogeneous workloads, whereas D2 is selectively sampled (only three hours per day) to satisfy PERSEUS's strict Latency-versus-Throughput (LvT) assumption. Because *SlowSight* is distribution-free and does not rely on the LvT assumption, it generalizes consistently to real-world, unconstrained environments where such assumptions fail.

5.3 Contribution of Key Modules (RQ2)

To show the effectiveness of each critical module in *SlowSight* (*i.e.*, Historical Deviation Detection, Peer Component Comparison, and Knowledge-Driven Filtering), we construct three ablated variants, denoted as **C1**–**C3**, each omitting a specific module. Specifically, (1) **C1** excludes the Historical Deviation Detection module, utilizing only Peer Component Comparison and Knowledge-Driven Filtering; (2) **C2** removes the Peer Component Comparison module, incorporating only Historical Deviation Detection and Knowledge-Driven Filtering; and (3) **C3** omits the Knowledge-Driven Filtering module, combining Historical Deviation Detection and Peer Component Comparison.

Method	D1			D2		
	Precision	Recall	F_1-score	Precision	Recall	F_1-score
SlowSight	0.8	0.909	**0.851**	**0.896**	**0.911**	**0.903**
C1	**0.867**	0.591	0.703	-	-	-
C2	0.8	0.727	0.762	-	-	-
C3	0.362	**0.955**	0.525	0.785	**0.911**	0.843

Table 3: The evaluation results of ablation study.

The results in Table 3 show that *SlowSight* consistently outperforms all ablated variants, demonstrating the necessity of each module and the effectiveness of their integration. By integrating these complementary mechanisms, *SlowSight* achieves accurate and robust fail-slow detection, supporting stability and reliability management in large-scale cloud storage systems.

5.4 Effectiveness of Labeling (RQ3)

To assess the quality of fail-slow labels produced by *SlowSight*, we compare it with a representative threshold-based method, Lu et al. [22]. Experiments are conducted on the D2 dataset, which includes both normal and fail-slow disks. We randomly sample 70% of disks from each class to form the labeling pool, which is independently labeled using *SlowSight* and Lu et al. The remaining 30% of disks are reserved as an unseen test set. Using the datasets labeled by each method, we train two supervised classifiers, *i.e.*, XGBoost [3] and Random Forests (RF) [2], under identical feature representations, hyperparameter tuning budgets, and training protocols. This controlled setup ensures that performance differences stem solely from the quality of labeling.

Method	*SlowSight*-Labeled			Lu et al. [22]-Labeled		
	Precision	Recall	F_1-score	Precision	Recall	F_1-score
XGBoost [3]	**1.0**	0.989	0.994	**1.0**	0.4	0.571
RF [2]	**1.0**	**1.0**	**1.0**	**1.0**	0.344	0.512

Table 4: Effectiveness of fail-slow labeling methods in downstream classification tasks.

As listed in Table 4, classifiers trained on *SlowSight*-labeled data consistently achieve higher F_1-scores than those trained on threshold-labeled data. These results indicate that *SlowSight* produces higher-quality fail-slow labels that translate into superior downstream classification performance. This closes the loop from labeling to online detection, confirming that better labeling directly enables more accurate fail-slow detection in production.

6 Deployment of SlowSight

6.1 Workflow

SlowSight is deployed in a production storage cluster at *Huawei Cloud* that supports diverse internal workloads, including transactional services, data analytics, and periodic backups. Currently running across more than 80,000 disks, *SlowSight* successfully distilled tens of thousands of threshold-based alarms into 1,754 actionable anomalies over a three-month deployment period, directing operators to verified components for timely eviction or replacement scheduling.

In terms of system overhead, *SlowSight* imposes negligible resource costs. Monitored storage nodes incur zero additional compute overhead beyond standard telemetry collection. On the centralized analysis server, which is the same server described in Section 5.1, *SlowSight*'s background execution only modestly increases CPU utilization (from 53% to 65%) and memory utilization (from 32% to 37%). Operationally, *SlowSight* is highly efficient: the transfer learning phase requires approximately 15 minutes per model, and labeling a representative cluster centroid takes only 11.2 s on average. Given that telemetry at *Huawei Cloud* is sampled at one-minute granularity, *SlowSight* delivers detection results within one minute. Combined with a production F_1-score exceeding 0.85, these results demonstrate its viability for real-world, low-latency fail-slow detection.

Within the operational stack, *SlowSight* serves as a decision-support component rather than an autonomous failure detector.

Prior to its integration, operators manually inspected numerous repetitive alerts from threshold-based and statistical detectors. This manual triage was highly labor-intensive, requiring cross-node metric correlation, and was prone to inconsistency. *SlowSight* restructures this workflow by introducing a semi-automated, pattern-centric abstraction. Instead of individual segment triage, *SlowSight* automatically clusters candidate anomalies based on structural similarity, allowing operators to focus exclusively on representative centroids (see §6.3). This abstraction drastically reduces redundant inspection effort while preserving the semantic distinctions of different failure modes. This human-in-the-loop design not only fosters operator trust but also ensures practical usability. Crucially, *SlowSight* aggregates these expert-verified labels to incrementally update and optimize its embedded detection models, forming a closed feedback loop that continuously enhances algorithmic accuracy.

6.2 Generalizability

To evaluate the generalizability of *SlowSight*, we construct an experimental dataset (D3) at *Huawei Cloud* focusing on NICs. In distributed storage systems, NIC fail-slow behaviors impair inter-node communication, severely degrading storage performance and potentially causing service interruptions [16,24]. Using ChaosBlade [1], we inject 339 NIC failures covering common degradation modes documented by Gunawi et al. [11], including latency inflation, packet loss, and packet corruption. Failure durations are randomly sampled between five and ten minutes, with inter-failure intervals ranging from 30 to 50 minutes [2].

Method	D3		
	Precision	Recall	F_1-score
SlowSight	**0.910**	0.982	**0.945**
Lu et al. [22]	0.683	**0.991**	0.809
PERSEUS [21]	0.499	0.319	0.389
OutSpot [28]	0.464	0.876	0.607
iTransformer [18]	0.210	0.260	0.232
TimesNet [31]	0.824	0.982	0.896

Table 5: Effectiveness of NIC-related fail-slow failure detection.

As summarized in Table 5, *SlowSight* achieves an F_1-score of 0.945, significantly outperforming all baseline methods. This result demonstrates *SlowSight*'s efficacy on NIC degradations, highlighting its robust cross-component generalizability. In contrast, existing fail-slow detection methods [21,22] are tailored to disk-centric failure characteristics and do not readily generalize to other hardware components, thereby limiting their applicability in cloud-scale environments. More fundamentally, *SlowSight* operates on multivariate metrics without imposing distribution assumptions, which makes it agnostic to the underlying component type. For instance, CPU fail-slow manifested in frequency and utilization metrics [11], as well as GPU fail-slow reflected in training throughput and SM utilization [32], both align with the degradation patterns that *SlowSight* is designed to capture, suggesting that its applicability extends beyond the components evaluated here.

6.3 Overall Labeling Workload

In traditional fail-slow labeling, operators examine long-term performance histories across multiple metrics to establish a baseline under varying workloads. Suspicious intervals are then inspected sequentially and compared with historical periods and peer components to distinguish true degradation from benign fluctuations before assigning labels at fine temporal granularity. In practice, most manual effort lies not in the final labeling decision but in repeatedly scanning long time series, correlating metrics, and cross-checking patterns across components. At cloud scale, where thousands of components are monitored over extended periods, such per-anomaly analysis becomes time-consuming and cognitively demanding.

To intuitively demonstrate the effectiveness of *SlowSight*, we conducted a controlled user study involving a team of four graduate researchers specializing in Artificial Intelligence for IT Operations (AIOps). The participants were randomly divided into two groups, with each operator independently labeling the same dataset (D3). The first group performed labeling using our developed labeling interface without *SlowSight*, serving as the baseline to measure the workload of conventional manual labeling, while the second group used the same interface with *SlowSight* integrated.

Empirically, manually labeling the 366 detected anomaly segments required an average of approximately 61 minutes. In contrast, when assisted by *SlowSight*, the average labeling time was reduced to around 5 minutes. This substantial reduction is attributed to *SlowSight* clustering the anomaly segments into 27 representative patterns, such that operators only needed to label the corresponding cluster centroids. These results indicate that *SlowSight* reduces manual labeling time by over 90%, demonstrating its effectiveness in alleviating labeling overhead while maintaining labeling consistency. The interactive labeling interface allows operators to relabel or split clusters when needed, thereby preserving labeling accuracy.

6.4 Lessons Learned

Lesson 1: Single-View Detection Is Fragile. Early variants relying solely on temporal deviation or peer comparison performed adequately in controlled settings but proved unreliable in production. Temporal-only analysis misclassified workload-

[1] https://github.com/chaosblade-io/chaosblade

[2] The injection strategy follows the gray failure paradigm adopted by GrayScope [37]. The experimental scale is comparable to existing industry benchmarks, *e.g.*, GrayScope [37], which evaluates 16 instances at *Huawei*.

induced shifts as fail-slow events, while peer-only comparison missed correlated degradations across components. These observations indicate that fail-slow behavior manifests jointly as temporal deviation and peer divergence. Integrating both views substantially improved robustness under heterogeneous and dynamic workloads.

Insight: Practical fail-slow detection requires multi-view modeling to avoid systematic blind spots.

Lesson 2: Interpretability Drives Adoption. Although early versions of *SlowSight* primarily emphasized detection accuracy, operator feedback revealed that interpretability was equally critical for practical deployment. Initially, *SlowSight* reported detected fail-slow segments through automated alerts without providing supporting visual context. In practice, operators were often reluctant to rely on these alerts because they could not easily determine whether the reported anomalies reflected genuine hardware degradation or transient workload fluctuations. To address this limitation, we introduced an interactive interface that presents anomaly segments together with peer comparisons and temporal performance trends. By visually examining the latency evolution of the affected disk alongside that of its peers, operators could more readily validate whether sustained degradation was present. This enhancement significantly improved operator confidence and reduced the effort required for manual verification.

Insight: In operational environments, transparent and interpretable outputs are essential for user trust and adoption, often outweighing marginal improvements in detection accuracy.

Lesson 3: Human-in-the-Loop Design Is Necessary. Fully automated fail-slow labeling was initially considered desirable. However, correlated degradations, ambiguous borderline cases, and monitoring noise frequently required contextual interpretation beyond algorithmic inference. Maintaining human validation as part of the workflow proved critical to avoiding overconfident mislabeling. For instance, during a firmware update rollout, several disks exhibited mild yet synchronized latency increases. Although the resulting degradation pattern resembled fail-slow behavior, subsequent operator investigation determined that the effect was transient and caused by background firmware activities rather than genuine hardware degradation.

Insight: Fail-slow labeling in cloud storage systems benefits from structured automation, but complete automation is neither realistic nor advisable in practical environments.

7 Related Work

Fail-Slow Detection. Fail-slow failures have attracted increasing attention, motivating various detection approaches [21, 22, 24]. PERSEUS [21] uses regression-based modeling to identify slow drives, while IASO [24] relies on peer comparison to detect degraded nodes. Lu et al. [22] combine statistical device metrics with cross-drive latency comparisons for fail-slow diagnosis. Although effective, these methods depend on expert-crafted heuristics and scenario-specific parameter tuning (*e.g.*, thresholds and risk scores), limiting robustness in production. Recent systems, including Greyhound [32] for large-scale LLM training and Sieve [10] for hardware bug analysis, extend detection to compute and communication paths but require code-level instrumentation or workload assumptions. Related studies characterize similar phenomena as gray failures [14], leading to systems such as Panorama [13] and OmegaGen [19], which typically rely on source-code visibility or intrusive modifications. Other work investigates "limping" failures [9]; for example, Kasick et al. [15] compare statistical attributes across servers but rely on predefined thresholds, reducing robustness under dynamic conditions.

Fail-Slow Labeling. Despite progress in fail-slow detection, efficient labeling tools for constructing high-quality datasets remain scarce. For example, although PERSEUS [21] releases a dataset, the lack of precise failure timestamps still requires substantial manual labeling. Existing efforts mainly target related domains. LabelLess [40] reduces labeling effort for KPI anomaly datasets but focuses on single metrics, limiting applicability to multivariate fail-slow scenarios. OutSpot [28] and Curve [7] provide GUI-based labeling for time-series anomalies but largely rely on manual inspection without tight integration with automated detection. To the best of our knowledge, *SlowSight* is the first system specifically designed for fail-slow labeling. Integrating automated detection with pattern-centric labeling substantially reduces labeling effort while enabling the scalable construction of high-quality fail-slow datasets.

8 Conclusion

High-quality labeled datasets are essential for advancing fail-slow failure detection in large-scale cloud storage systems; however, the volume of monitoring data and the complexity of fail-slow behaviors render manual labeling costly and impractical. To address this challenge, we propose *SlowSight*, a novel labeling framework that integrates multi-view anomaly candidate extraction with pattern-centric labeling to enable efficient and accurate fail-slow labeling. Experimental results on two datasets show that *SlowSight* achieves F_1-scores of 0.851 and 0.903, demonstrating its effectiveness in fail-slow failure identification and labeling.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (No.2024YFB4505903), the National Natural Science Foundation of China (62272249, 62302244), the Fundamental Research Funds for the Central Universities (XXX-63253249), and the Tianjin Key Research and Development Program (Grant No. 25YFYFFG00690).

References

[1] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A Zuluaga. Usad: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, KDD '20, pages 3395–3404, New York, NY, USA, 2020. Association for Computing Machinery.

[2] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[3] Tianqi Chen. Xgboost: A scalable tree boosting system. *Cornell University*, 2016.

[4] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, et al. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, pages 674–688, New York, NY, USA, 2024. Association for Computing Machinery.

[5] Zhuangbin Chen, Jinyang Liu, Yuxin Su, Hongyu Zhang, Xiao Ling, Yongqiang Yang, and Michael R. Lyu. Adaptive performance anomaly detection for online service systems via pattern sketching. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 61–72, New York, NY, USA, 2022. Association for Computing Machinery.

[6] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W. Kempa-Liehr. Time series feature extraction on basis of scalable hypothesis tests. *Neurocomputing*, 307:72–77, 2018.

[7] Curve. Open source repository of curve, 2018.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255, Los Alamitos, CA, 2009. IEEE.

[9] Thanh Do, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tiratat Patana-anake, and Haryadi S. Gunawi. Limplock: understanding the impact of limpware on scale-out cloud systems. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, New York, NY, USA, 2013. Association for Computing Machinery.

[10] Gen Dong, Yu Hua, Yongle Zhang, Zhangyu Chen, and Menglei Chen. Understanding and detecting {Fail-Slow} hardware failure bugs in cloud systems. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pages 1127–1142, 2025.

[11] Haryadi S Gunawi, Riza O Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, et al. Fail-slow at scale: Evidence of hardware performance faults in large production systems. *ACM Transactions on Storage (TOS)*, 14(3):1–26, 2018.

[12] Zijun Hu, Pengfei Chen, Guangba Yu, Zilong He, and Xiaoyun Li. Ts-invarnet: Anomaly detection and localization based on tempo-spatial kpi invariants in distributed services. In *2022 IEEE International Conference on Web Services (ICWS)*, pages 109–119, 2022.

[13] Peng Huang, Chuanxiong Guo, Jacob R Lorch, Lidong Zhou, and Yingnong Dang. Capturing and enhancing in situ system observability for failure detection. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 1–16, Carlsbad, CA, October 2018. USENIX Association.

[14] Peng Huang, Chuanxiong Guo, Lidong Zhou, Jacob R Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. Gray failure: The achilles' heel of cloud-scale systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS '17, page 150–155, New York, NY, USA, 2017. Association for Computing Machinery.

[15] Michael P Kasick, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. Black-box problem diagnosis in parallel file systems. In *8th USENIX Conference on File and Storage Technologies (FAST 10)*, FAST'10, page 4, USA, 2010. USENIX Association.

[16] Daehyeok Kim, Amirsaman Memaripour, Anirudh Badam, Yibo Zhu, Hongqiang Harry Liu, Jitu Padhye, Shachar Raindel, Steven Swanson, Vyas Sekar, and Srinivasan Seshan. Hyperloop: group-based nic-offloading to accelerate replicated transactions in multi-tenant storage systems. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 297–312, New York, NY, USA, 2018. Association for Computing Machinery.

[17] Hongyi Liu, Xiaosong Huang, Mengxi Jia, Tong Jia, Jing Han, Ying Li, and Zhonghai Wu. Uac-ad: Unsupervised adversarial contrastive learning for anomaly detection on multi-modal data in microservice systems. *IEEE Transactions on Services Computing*, 17(6):3887–3900, 2024.

[18] Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer:

Inverted transformers are effective for time series forecasting. In *International Conference on Learning Representations*, Ithaca, NY, 2024. OpenReview.net.

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

[20] Ruiming Lu, Yunchi Lu, Yuxuan Jiang, Guangtao Xue, and Peng Huang. One-Size-Fits-None: Understanding and enhancing Slow-Fault tolerance in modern distributed systems. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 359–378, Philadelphia, PA, April 2025. USENIX Association.

[21] Ruiming Lu, Erci Xu, Yiming Zhang, Fengyi Zhu, Zhaosheng Zhu, Mengtian Wang, Zongpeng Zhu, Guangtao Xue, Jiwu Shu, Minglu Li, et al. Perseus: A {Fail-Slow} detection framework for cloud storage systems. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*, pages 49–64, Santa Clara, CA, February 2023. USENIX Association.

[22] Ruiming Lu, Erci Xu, Yiming Zhang, Zhaosheng Zhu, Mengtian Wang, Zongpeng Zhu, Guangtao Xue, Minglu Li, and Jiesheng Wu. {NVMe}{SSD} failures in the field: the {Fail-Stop} and the {Fail-Slow}. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 1005–1020, Carlsbad, CA, July 2022. USENIX Association.

[23] Minghua Ma, Shenglin Zhang, Junjie Chen, Jim Xu, Haozhe Li, Yongliang Lin, Xiaohui Nie, Bo Zhou, Yong Wang, and Dan Pei. Jump-Starting multivariate time series anomaly detection for online service systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 413–426. USENIX Association, July 2021.

[24] Biswaranjan Panda, Deepthi Srinivasan, Huan Ke, Karan Gupta, Vinayak Khot, and Haryadi S Gunawi. {IASO}: A {Fail-Slow} detection and mitigation framework for distributed storage services. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 47–62, Renton, WA, July 2019. USENIX Association.

[25] Chunhui Shen, Qianyu Ouyang, Feibo Li, Zhipeng Liu, Longcheng Zhu, Yujie Zou, Qing Su, Tianhuan Yu, Yi Yi, Jianhong Hu, Cen Zheng, Bo Wen, Hanbang Zheng, Lunfan Xu, Sicheng Pan, Bin Wu, Xiao He, Ye Li, Jian Tan, Sheng Wang, Dan Pei, Wei Zhang, and Feifei Li. Lindorm tsdb: A cloud-native time-series database for large-scale monitoring systems. *Proc. VLDB Endow.*, 16(12):3715–3727, August 2023.

[26] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouet. Anomaly detection in streams with extreme value theory. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, KDD '17, pages 1067–1075, New York, NY, USA, 2017. Association for Computing Machinery.

[27] Pooja Srinivas, Fiza Husain, Anjaly Parayil, Ayush Choure, Chetan Bansal, and Saravan Rajmohan. Intelligent monitoring framework for cloud services: A data-driven approach. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '24, page 381–391, New York, NY, USA, 2024. Association for Computing Machinery.

[28] Yongqian Sun, Daguo Cheng, Tiankai Yang, Yuhe Ji, Shenglin Zhang, Man Zhu, Xiao Xiong, Qiliang Fan, Minghan Liang, Dan Pei, et al. Efficient and robust kpi outlier detection for large-scale datacenters. *IEEE Transactions on Computers*, 72(10):2858–2871, 2023.

[29] Yongqian Sun, Minghan Liang, Shenglin Zhang, Zeyu Che, Zhiyao Luo, Dongwen Li, Yuzhi Zhang, Dan Pei, Lemeng Pan, and Liping Hou. Efficient multivariate time series anomaly detection through transfer learning for large-scale software systems. *ACM Trans. Softw. Eng. Methodol.*, November 2024. Just Accepted.

[30] Zexin Wang, Changhua Pei, Minghua Ma, Xin Wang, Zhihan Li, Dan Pei, Saravan Rajmohan, Dongmei Zhang, Qingwei Lin, Haiming Zhang, et al. Revisiting vae for unsupervised time series anomaly detection: A frequency perspective. In *Proceedings of the ACM Web Conference 2024*, WWW '24, pages 3096–3105, New York, NY, USA, 2024. Association for Computing Machinery.

[31] Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. In *International Conference on Learning Representations*, Ithaca, NY, 2023. OpenReview.net.

[32] Tianyuan Wu, Wei Wang, Yinghao Yu, Siran Yang, Wenchao Wu, Qinkai Duan, Guodong Yang, Jiamang Wang, Lin Qu, and Liping Zhang. {GREYHOUND}: Hunting {Fail-Slows} in {Hybrid-Parallel} training at scale. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pages 731–747, 2025.

[33] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei,

Yang Feng, Jie Chen, Zhaogang Wang, and Honglin Qiao. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 187–196, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[34] Jiehui Xu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Anomaly transformer: Time series anomaly detection with association discrepancy. In *International Conference on Learning Representations*, Ithaca, NY, 2022. OpenReview.net.

[35] Zhaoyang Yu, Minghua Ma, Chaoyun Zhang, Si Qin, Yu Kang, Chetan Bansal, Saravan Rajmohan, Yingnong Dang, Changhua Pei, Dan Pei, et al. Monitorassistant: Simplifying cloud service monitoring via large language models. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, pages 38–49, New York, NY, USA, 2024. Association for Computing Machinery.

[36] Chaoli Zhang, Tian Zhou, Qingsong Wen, and Liang Sun. Tfad: A decomposition time series anomaly detection architecture with time-frequency analysis. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, page 2497–2507, New York, NY, USA, 2022. Association for Computing Machinery.

[37] Shenglin Zhang, Yongxin Zhao, Xiao Xiong, Yongqian Sun, Xiaohui Nie, Jiacheng Zhang, Fenglai Wang, Xian Zheng, Yuzhi Zhang, and Dan Pei. Illuminating the gray zone: Non-intrusive gray failure localization in server operating systems. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, FSE 2024, page 126–137, New York, NY, USA, 2024. Association for Computing Machinery.

[38] Shenglin Zhang, Zhenyu Zhong, Dongwen Li, Qiliang Fan, Yongqian Sun, Man Zhu, Yuzhi Zhang, Dan Pei, Jiyan Sun, Yinlong Liu, et al. Efficient kpi anomaly detection through transfer learning for large-scale web services. *IEEE Journal on Selected Areas in Communications*, 40(8):2440–2455, 2022.

[39] Xuchao Zhang, Supriyo Ghosh, Chetan Bansal, Rujia Wang, Minghua Ma, Yu Kang, and Saravan Rajmohan. Automated root causing of cloud incidents using in-context learning with gpt-4. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, FSE 2024, page 266–277, New York, NY, USA, 2024. Association for Computing Machinery.

[40] Nengwen Zhao, Jing Zhu, Rong Liu, Dapeng Liu, Ming Zhang, and Dan Pei. Label-less: A semi-automatic labelling tool for kpi anomalies. In *2019 Conference on Computer Communications*, pages 1882–1890. IEEE, 2019.

A Artifact Appendix

Abstract

The artifact includes the prototype implementation of *SlowSight*, an interactive labeling system, and the failure-injection dataset (D3) used in our evaluation, along with instructions to reproduce the main experimental results presented in the paper. *SlowSight* is a semi-automated framework for labeling fail-slow failures in large-scale cloud storage systems. It combines multi-view anomaly candidate extraction (*i.e.*, historical deviation detection and peer component comparison) with pattern-centric labeling, enabling operators to label only a small set of representative anomaly patterns whose labels are systematically propagated to structurally similar instances. This design substantially reduces manual labeling effort while preserving labeling accuracy and consistency.

Scope

This artifact enables reviewers to validate the main claims of the paper. Specifically, it supports (i) the end-to-end execution of the *SlowSight* pipeline on the released D3 dataset, from data preparation through pattern-centric labeling; (ii) the reproduction of the fail-slow labeling results that demonstrate *SlowSight*'s effectiveness and generalizability to NIC-related fail-slow failures (Section 6.2); and (iii) the qualitative inspection of the interactive labeling interface described in Section 4.4. The production dataset (D1) is proprietary and cannot be released due to security and privacy constraints; the public dataset (D2) is available from the PERSEUS [21] repository. Consequently, the artifact focuses on D3, which is sufficient to substantiate the framework's functionality and generalizability claims.

Contents

The artifact is organized as follows:

• `backEnd/`, which contains the core implementation of *SlowSight*, including the detection and labeling modules under `backEnd/SlowSight/`, pretrained models under `backEnd/model/`, intermediate and final outputs under `backEnd/result/`, and the Flask application entry point `app.py`.

• `backEnd/data/`, which ships the D3 dataset, consisting of the time-series file `D3_metric.csv` and the labeled ground truth `ground_truth.csv`.

• frontEnd/, which contains the interactive labeling interface that supports synchronized visualization of multi-metric anomaly segments, cluster inspection, and human-in-the-loop label refinement.

Hosting

The artifact is accessible from GitHub at https://github.com/AIOps-Lab-NKU/SlowSight. Please refer to the `main` branch and the latest commit version for the exact code used in this paper.

Requirements

Hardware Dependencies. *SlowSight* imposes no specialized hardware requirements and is designed to operate without GPU acceleration. All experiments reported in the paper were conducted on a Linux server equipped with eight Intel Xeon Gold 6266C CPUs @ 3.00 GHz and 62.7 GB of RAM. A commodity multi-core machine with comparable memory is sufficient to reproduce the D3 results.
Software Dependencies. The artifact is developed and tested on a Linux platform with `Python 3.8`. The backend depends on the packages listed in `backEnd/requirements.txt`. The interactive frontend requires `Node.js` and the `npm` package manager. The `git` utility is required if the repository is obtained via cloning.

Evaluation Setup

We provide detailed step-by-step instructions in the `README.md` file on GitHub to reproduce the major experiments. The core steps are summarized below for quick reference.

Clone the code repository

First, clone the code repository from GitHub:

```
$ git clone \
https://github.com/AIOps-Lab-NKU/SlowSight.git
$ cd SlowSight
```

Set up the Python environment

Next, create the Conda environment and install the backend dependencies:

```
$ cd backEnd
$ conda create -n slowSight python=3.8 -y
$ conda activate slowSight
$ pip install -r requirements.txt
```

Launch the interactive labeling system

SlowSight follows a frontend–backend separation architecture; two terminals are therefore required to run concurrently:

```
# Terminal 1: backend
$ cd frontEnd
$ set FLASK_APP=app.py
$ flask run
# Terminal 2: frontend
```

```
$ cd backEnd
$ npm install
$ npm run dev
```

Run fail-slow labeling with SlowSight

In the web interface, navigate to the "Line Chart Label" tool, select "D3" from the "Data Selection" dropdown menu and "SlowSight" from the "Algorithm Selection" dropdown menu, and then click the "Label" button to perform the labeling operation. The resulting output corresponds to the generalizability experiments presented in Table 5.

To expedite the evaluation, the pre-trained model shipped in `backEnd/model/` is used by default. If retraining is desired, please uncomment the "Train the model" section in the run function of `main.py`.

Result analysis

The detection and labeling results are stored in the `backEnd/result/` directory. Reviewers can compare the generated labels against `ground_truth.csv` to compute Precision, Recall, and F_1-score, and can further inspect the anomaly clusters and representative pattern centroids through the interactive interface to validate the labeling-effort reduction reported in Section 6.3.