



FEDERAL UNIVERSITY OF SANTA CATARINA
TECHNOLOGY CENTER
AUTOMATION AND SYSTEMS ENGINEERING DEPARTMENT
UNDERGRADUATE COURSE IN CONTROL AND AUTOMATION ENGINEERING

Guilherme Mertens de Andrade

Scoring monitor quality: A machine learning approach

New York City
2025

Guilherme Mertens de Andrade

Scoring monitor quality: A machine learning approach

Final report of the subject DAS5511 (Course Final Project) as a Concluding Dissertation of the Undergraduate Course in Control and Automation Engineering of the Federal University of Santa Catarina.
Supervisor: Prof. Pascale Le Gall, Dra.
Bertrand Waldburger
Co-supervisor: Hans Kuder, Eng.

New York City
2025

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Mertens de Andrade, Guilherme

Scoring monitor quality : a machine learning approach /
Guilherme Mertens de Andrade ; orientadora, Pascale Le
Gall, orientador, Bertrand Waldburger, coorientador, Hans
Kuder, 2025.

45 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Centro Tecnológico,
Graduação em Engenharia de Controle e Automação,
Florianópolis, 2025.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Observability.
3. Monitoring. 4. Machine Learning. 5. Big Data. I. Le
Gall, Pascale. II. Waldburger, Bertrand. III. Kuder, Hans.
IV. Universidade Federal de Santa Catarina. Graduação em
Engenharia de Controle e Automação. V. Título.

Guilherme Mertens de Andrade

Scoring monitor quality: A machine learning approach

This dissertation was developed under the Double Degree agreement between the Universidade Federal de Santa Catarina and CentraleSupélec. It was presented to an examining committee at CentraleSupélec and validated in the context of the subject DAS5511 (Course Final Project) by the Undergraduate Course in Control and Automation Engineering

Florianópolis, December 11, 2025.

Examining Board:

Prof. Pascale Le Gall, Dra.
Advisor
CentraleSupélec

Bertrand Waldburger
Advisor
CentraleSupélec

Hans Kuder, Eng.
Supervisor
Company Datadog

Minneapolis, December 8, 2025

As representative of Datadog, in which the present work was carried out, I declare this document to be exempt from any confidential or sensitive content regarding intellectual property that may keep it from being published by the Federal University of Santa Catarina (UFSC) to the general public, including its online availability in the Institutional Repository of the University Library (BU). Furthermore, I attest knowledge of the obligation by the author, as a student of UFSC, to deposit this document in the said Institutional Repository, for being it a Final Program Dissertation ("Trabalho de Conclusão de Curso"), in accordance with the Resolução Normativa nº 126/2019/CUn.

A handwritten signature in black ink, appearing to read 'Hans Kuder', with a stylized flourish at the end.

Hans Kuder
Engineering Manager
Datadog, Inc

ABSTRACT

Datadog, a leading B2B Software-as-a-Service (SaaS) observability platform co-founded by two CentraleSupélec alumni, provides developers with critical tools for monitoring, analytics, and security. Within its Alerting organization, our team is responsible for a core feature: monitors, which allow users to track metrics, logs, and other telemetry and receive notifications when predefined thresholds are crossed.

The primary challenge addressed during this internship was the unmeasured quality of user-configured monitors. Without guidance or quality metrics, users often create suboptimal monitors (e.g., averaging latency instead of using percentiles like p90) or set poorly calibrated thresholds. This leads to significant negative consequences: alert fatigue from excessive false positives, critical alerts being ignored, and observability gaps when users disable noisy monitors. These issues degrade the user experience and increase operational risks for clients.

The mission was to define, measure, and score the quality of a monitor. The initial problem was ambiguous, with the core difficulty being the complete absence of “ground truth” labels to train a supervised model. The chosen approach involved leveraging domain expertise to establish a clear definition of monitor quality and employing weak supervision techniques to programmatically generate labels.

This process culminated in the development of a machine learning model that assigns a quality score from 0 to 1 to every monitor. Critically, the model also provides an explainable reason when a monitor is flagged as low-quality. The initial deployment of this model revealed that approximately 30% of all active monitors could be classified as “low-quality,” quantifying the problem’s scale and highlighting significant potential for system-wide improvements.

This project delivers immediate value by enabling the creation of tools to help users improve their monitoring configurations. Furthermore, it serves as a foundational component for a new suite of intelligent alerting features, such as automated monitor recommendations and the detection of monitoring gaps. This work has helped catalyze the formation of a dedicated data science function within the Alerting organization, paving the way for future data-driven product innovations.

Keywords: Monitoring. Machine Learning. Data Quality. Observability.

RESUMO

A Datadog, uma plataforma líder de observabilidade B2B Software-as-a-Service (SaaS) cofundada por dois ex-alunos da CentraleSupélec, fornece aos desenvolvedores ferramentas críticas para monitoramento, análise e segurança. Dentro de sua organização de Alerting (Alertas), nossa equipe é responsável por um recurso central: os monitores, que permitem aos usuários rastrear métricas, logs e outras telemetrias, recebendo notificações quando limiares predefinidos são ultrapassados.

O principal desafio abordado durante este estágio foi a qualidade não mensurada dos monitores configurados pelos usuários. Sem orientação ou métricas de qualidade, os usuários frequentemente criam monitores subótimos (por exemplo, calculando a média da latência em vez de usar percentis como o p90) ou definem limiares mal calibrados. Isso leva a consequências negativas significativas: fadiga de alertas devido a excessivos falsos positivos, alertas críticos sendo ignorados e lacunas de observabilidade quando os usuários desativam monitores ruidosos. Esses problemas degradam a experiência do usuário e aumentam os riscos operacionais para os clientes.

A missão consistiu em definir, medir e pontuar a qualidade de um monitor. O problema inicial era ambíguo, sendo a dificuldade central a completa ausência de rótulos de referência (ground truth) para treinar um modelo supervisionado. A abordagem escolhida envolveu alavancar o conhecimento de domínio para estabelecer uma definição clara de qualidade de monitor e empregar técnicas de supervisão fraca (weak supervision) para gerar rótulos programaticamente.

Esse processo culminou no desenvolvimento de um modelo de aprendizado de máquina que atribui uma pontuação de qualidade de 0 a 1 para cada monitor. De forma crítica, o modelo também fornece uma razão explicável quando um monitor é sinalizado como sendo de baixa qualidade. A implantação inicial deste modelo revelou que aproximadamente 30% de todos os monitores ativos poderiam ser classificados como "baixa qualidade", quantificando a escala do problema e destacando um potencial significativo para melhorias em todo o sistema.

Este projeto entrega valor imediato ao permitir a criação de ferramentas para ajudar os usuários a melhorar suas configurações de monitoramento. Além disso, serve como um componente fundamental para um novo conjunto de recursos de alertas inteligentes, como recomendações automatizadas de monitores e a detecção de lacunas de monitoramento. Este trabalho ajudou a catalisar a formação de uma função dedicada de ciência de dados dentro da organização de Alerting, abrindo caminho para futuras inovações de produtos baseadas em dados.

Palavras-chave: Monitoramento. Aprendizado de Máquina. Qualidade de Dados. Observabilidade.

LIST OF FIGURES

Figure 1 – Datadog’s Logo	8
Figure 2 – Overview of the Datadog’s Platform	8
Figure 3 – Example of the Monitor Status Page. For more information on how to configure a monitor, see the official Datadog documentation: Getting Started with Monitors.	10
Figure 4 – Example of the Monitor Quality Page.	16
Figure 5 – Diagram of the Monitor Events Extraction Pipeline.	17
Figure 6 – Good Monitors are the intersection of the Well Configured Monitors and Relevant Monitors Set.	18
Figure 7 – Distribution of Time to Resolution for 4 samples.	22
Figure 8 – Fitted distributions in some sample data.	23
Figure 9 –	24
Figure 10 –	25
Figure 11 –	26
Figure 12 – Example of feature distributions for selected labels.	27
Figure 13 – Performance metrics of the supervised model.	28
Figure 14 – Feature importance and contribution based on SHAP values.	29
Figure 15 – Distribution of predicted probabilities on the unlabeled data.	30
Figure 16 – Evidence of data drift between labeled and unlabeled data.	30
Figure 17 – Isolation Forest results comparison	31
Figure 18 – Score distribution when using “bad” data as the normal class.	32
Figure 19 – Score distribution when using “good” data as the normal class.	32
Figure 20 – Resulted Sigmoids for the $S_{critical}$ score	38
Figure 21 – Resulted Sigmoids for the metadata score	38
Figure 22 – S_{events} ML model results	39
Figure 23 – Feature importance and contribution based on SHAP values.	39
Figure 24 – Score after uncertainty layer, improving scores with lower session duration.	40
Figure 25 – Example of the Spreadsheet Used to Validate Results.	41

CONTENTS

1	INTRODUCTION	8
1.1	CONTEXT OF THE INTERNSHIP	8
1.2	PROBLEM STATEMENT	9
1.3	OBJECTIVES AND SCOPE	12
2	PROBLEM ANALYSIS AND APPROACH	14
2.1	DEEPENING THE PROBLEM	14
2.2	INITIAL STATE AND EXISTING DATA	15
2.3	CHOSEN APPROACH AND JUSTIFICATION	18
3	DEVELOPMENT OF THE MONITOR QUALITY SCORE	20
3.1	THE DATA PIPELINE	20
3.2	THE EXPERIMENTAL PHASE: TRIALS AND ITERATIONS	27
3.2.1	Data Analysis	27
3.2.2	Supervised Learning	28
3.2.2.1	Anomaly Detection	31
3.2.2.2	Positive Unlabelled Learning	33
3.2.2.3	Weak Labels	33
3.3	THE MACHINE LEARNING FINAL MODEL	34
3.3.1	Model Overview	34
3.3.1.0.1	Part 1: The Critical Score ($S_{critical}$)	34
3.3.1.0.2	Part 2: The Extra Score (S_{extra})	35
3.3.1.0.3	Training Strategy for the ML Component	37
3.3.1.0.4	Interpretation	37
4	RESULTS AND EVALUATION	38
4.1	QUANTITATIVE RESULTS	38
4.2	QUALITATIVE ANALYSIS	41
4.3	ANALYSIS OF ERRORS AND LIMITATIONS	41
4.4	ADDED VALUE FOR THE COMPANY	42
5	CONCLUSION	44
5.1	SUMMARY OF ACHIEVEMENTS	44
5.2	PERSONAL AND PROFESSIONAL LEARNINGS	44
5.3	ETHICAL CONSIDERATIONS	44
	References	45

1 INTRODUCTION

This initial chapter establishes the context of the internship project. It will present Datadog's position as a leader in the tech industry, detail the specific role of the Alerting organization and the Monitor Notifications team, and culminate in a precise formulation of the problem this project aims to solve.



Figure 1 – Datadog's Logo

1.1 CONTEXT OF THE INTERNSHIP

Datadog is a global leader in the observability and security SaaS market, providing a unified platform for monitoring cloud-scale applications. Its client roster includes industry giants such as **OpenAI**, **Airbnb**, and **Github**. From an engineering perspective, Datadog's most compelling characteristic is its immense scale: the platform processes over **100 trillion events daily** and generates hundreds of terabytes of new data. Operating at this level necessitates a highly resilient, distributed, and efficient architecture, offering an unparalleled learning environment for an early-career software engineer.

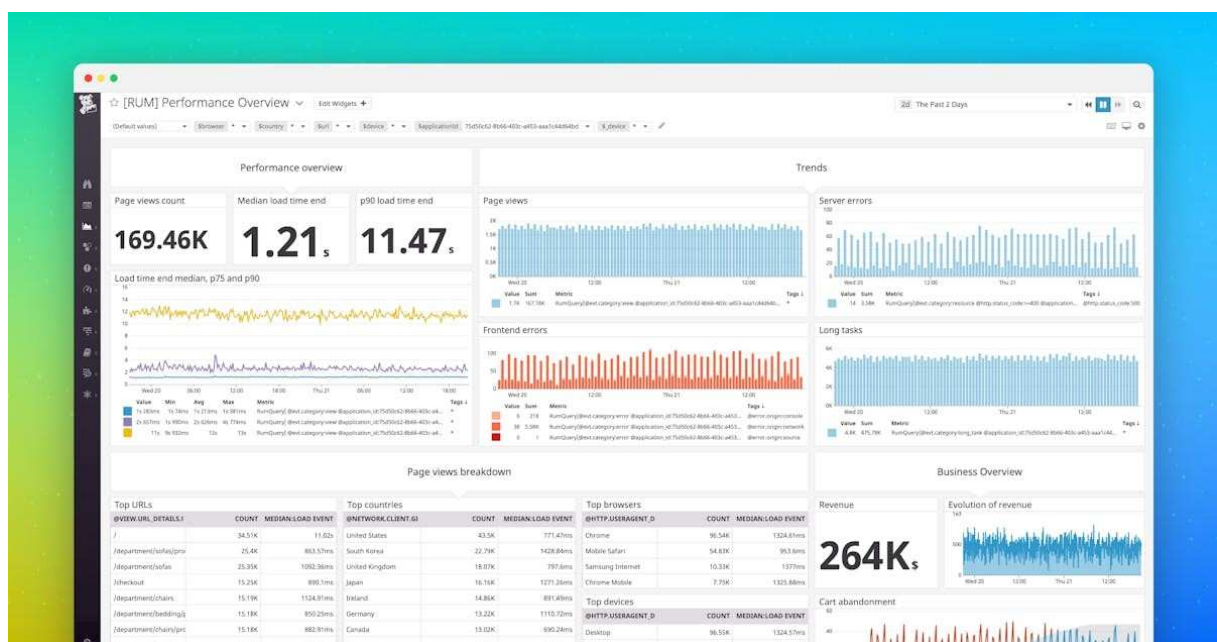


Figure 2 – Overview of the Datadog's Platform

The internship took place within the **Alerting organization**, a core pillar of the Datadog platform. This organization is responsible for the entire lifecycle of the **monitor**, a feature that enables clients to move from reactive dashboard observation to proactive, automated issue detection. Alerting's responsibilities span:

- Event ingestion and real-time query evaluation
- The delivery of notifications to end-users

My team, **Monitor Notifications**, operates at the final stage of this alerting pipeline. We own a critical service called **Alerting State Notifications (ASN)**, whose primary function is to consume monitor state transitions (e.g., from OK to ALERT) and enrich these events before dispatching them for delivery. This enrichment process includes:

- Generating dynamic messages based on user-defined templates and variables
- Attaching context-rich snapshots, such as graphs of the metric at the time of the alert
- Including relevant data samples, for example, the specific log lines that triggered a log-based monitor

Beyond this core service, the team is also responsible for maintaining the system's overall **robustness and reliability**. Operating at such scale introduces numerous potential failure modes. To mitigate them, the team owns and maintains several auxiliary services:

- **Avalanche Protection:** Prevents notification systems from being overwhelmed when a single issue causes a high cardinality of alerts (e.g., a single failing server affecting thousands of distinct customers)
- **Notification Delivery Tracker:** Ensures at-least-once delivery semantics, confirming that critical alerts reach their destinations even in the face of transient network failures

1.2 PROBLEM STATEMENT

To understand the core problem, one must first understand the anatomy of a Datadog monitor. In essence, a monitor is an automated check that continuously runs against a stream of data.

The Anatomy of a Monitor

While users can stare at dashboards to find issues, this is inefficient and impractical. **Monitors** are the solution, allowing users to define rules that automatically detect anomalies (Datadog, 2024b). A monitor configuration consists of several key components:

- **Query Definition:** The user defines what data to track. For a metric monitor, this involves selecting a critical metric (e.g., `api.request.latency`) and applying transformations. The query can be highly complex, involving aggregations (`avg`, `sum`, `max`, `p99`), mathematical operations between multiple metrics, and grouping by tags (e.g., `group by: endpoint, host`). This grouping is crucial, as it allows a single monitor definition to track the metric's behavior across hundreds of distinct entities (e.g., API endpoints, servers). Each unique group (e.g., `endpoint:/cart`, `endpoint:/login`) is tracked as an independent state.
- **Alert Conditions:** The user sets a threshold (e.g., `> 100ms`). When the query's output value crosses this threshold for a sustained period, the monitor's state for that specific group transitions from OK to ALERT or WARN.
- **Notification Configuration:** This defines what happens upon a state transition. The user crafts a title and message, ideally including a playbook or steps for remediation. They then specify the handles (the destinations for the notification) such as a Slack channel (`@slack-team-firefighters`), a PagerDuty service, or an email address.

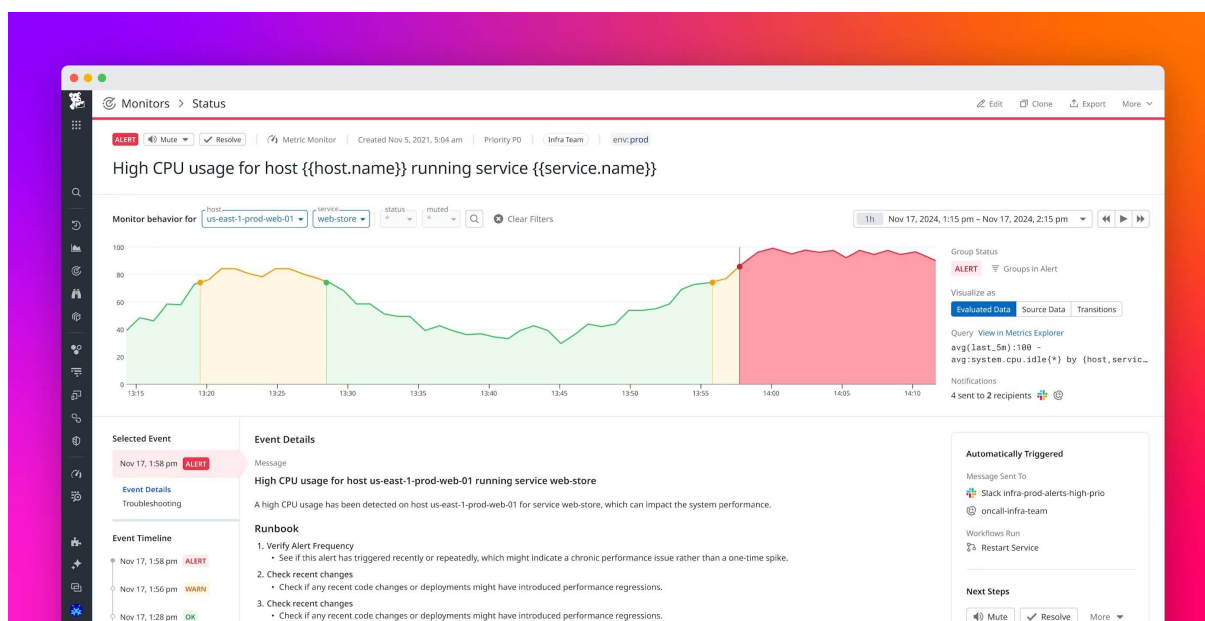


Figure 3 – Example of the Monitor Status Page. For more information on how to configure a monitor, see the official Datadog documentation: [Getting Started with Monitors](#).

The Emergence of “Monitor Debt”

Because monitors are entirely user-defined, their quality and utility vary drastically. This flexibility leads to a series of systemic problems, collectively referred to as **monitor debt**:

- **Suboptimal Query Design:** Users may not know the best way to monitor a service. A common mistake is tracking the average latency (`avg:api.request.latency`) instead of a high percentile such as p90 or p99. The average can easily hide latency spikes affecting a subset of users, rendering the monitor ineffective at catching real user-facing issues.
- **Poorly Calibrated Thresholds:** A threshold set too low results in a **flappy monitor** that generates a constant stream of false positives, leading to alert fatigue where engineers begin to distrust and ignore notifications. Conversely, a threshold set too high causes the monitor to miss legitimate incidents, resulting in false negatives.
- **Configuration Rot and Silent Failures:** Over time, monitors can become useless. Some may have no notification handle, making them completely silent. Others might send alerts to decommissioned channels (e.g., an inactive Slack workspace or PagerDuty service). Even when active, their messages may lack actionable details or playbook links, increasing time to resolution during real incidents.

The cumulative effect is a large and growing population of “**zombie**” monitors across Datadog, monitors that are actively configured and consuming computational resources, but providing zero value. They are muted, ignored, or fundamentally broken.

This issue has a critical secondary consequence: it pollutes our **training data**. As Datadog increasingly incorporates Machine Learning into the platform, for features like automated monitor generation and monitoring coverage analysis, the presence of thousands of poorly configured monitors severely limits our ability to build effective models. The data becomes biased towards bad user habits, meaning any model trained on it risks replicating those same mistakes.

Therefore, my mission was born from this dual need:

- **Improve user experience** by identifying and remediating poorly configured monitors.
- **Clean the foundational dataset** to enable more accurate and reliable machine learning initiatives.

1.3 OBJECTIVES AND SCOPE

Given the challenges of **monitor debt** outlined previously, the central goal of this internship was to design and implement a quantitative framework for assessing monitor quality. The project was driven by two primary, interconnected objectives:

- **For Datadog Clients:** To provide users with a clear, data-driven signal indicating the health of their monitoring setup. The ultimate goal is to empower them with actionable insights to improve their monitors, thereby reducing alert fatigue, minimizing false positives, and closing critical observability gaps.
- **For Internal Datadog Initiatives:** To create a foundational layer of intelligence for the Alerting platform. By programmatically identifying high-quality monitors, we can curate a clean and reliable dataset essential for future machine learning applications, such as automated monitor recommendations or anomaly detection.

To achieve these objectives, the project's scope was defined by the development of a **Monitor Quality Score**. This score was designed with the following key characteristics:

- **Quantitative and Bounded:** The output must be a numerical score normalized between 0 (lowest quality) and 1 (highest quality). This allows for objective comparison, thresholding, and tracking of quality improvements over time.
- **Explainable:** A low score must be accompanied by a set of clear, diagnostic reasons. This is critical for making the score actionable for the end-user (e.g., "This monitor is flapping frequently," or "The notification handle for this monitor is broken").

The analysis to generate this score leveraged a comprehensive set of data signals available within the Datadog ecosystem, which can be categorized as follows:

- **Static Configuration Data:** Information derived from the monitor's definition itself, including the query structure, the choice of metric and aggregations, the presence and content of the notification message, and the configuration of its notification handles.
- **Dynamic Behavioral Data:** Metrics describing how the monitor behaves over time, such as its trigger frequency (how often it alerts), its recovery time (how long it stays in an ALERT state), and its *flap rate* (how often it rapidly transitions between states).

- **User Engagement and Interaction Data:** Signals reflecting how users interact with the monitor, such as whether it is viewed after triggering, if it is frequently muted, or if it is edited shortly after an alert. This also includes data on the health of notification channels (e.g., whether notifications are being successfully delivered).

While the internship also involved work on several smaller tickets related to bug fixes and service improvements, this report will focus exclusively on the **Monitor Quality project**, as it represented the core strategic mission. A brief summary of these secondary contributions is included in the appendix for completeness.

2 PROBLEM ANALYSIS AND APPROACH

With the project's objectives defined, this chapter delves into the core analytical challenges. It begins by dissecting the inherent ambiguity of "monitor quality" to demonstrate why simple heuristics are insufficient. It then outlines the existing tools and data sources that served as the foundation for a more sophisticated, data-driven solution.

2.1 DEEPENING THE PROBLEM

The core objective of this project is to predict whether a monitor is "**good**" or "**bad**". However, these labels are far from straightforward. This is not a simple binary classification problem. To illustrate the complexity, consider the following questions and examples.

What constitutes a "**bad**" monitor?

- **Alert Frequency:** Does a monitor that alerts too frequently qualify as "bad"?
 - At what point does it become *too frequent*?
 - Is a daily alert always undesirable, or could it accurately reflect a chronically unstable service?
- **Alert Distribution:** Which is worse:
 - A monitor that triggers once every day for two months, indicating persistent noise; or
 - One that is silent for 58 days but triggers 50 times in a single day?
- **Resolution Time:** A monitor that stays in an ALERT state for days appears problematic. But where is the threshold where "long" becomes "bad": an hour, a day, or a week?
- **Muting Behavior:** A muted monitor suggests user abandonment.
 - How long must it be muted to be considered "bad"?
 - What if it's muted intentionally for scheduled maintenance?
- **Configuration Quality:** Some problems are objective, for instance a monitor with a broken notification handle fails at its core function. However, even a technically perfect monitor (good query, reasonable threshold) might still be of low quality if it lacks a descriptive message or ownership tags.

What constitutes a “good” monitor?

- **Alert Frequency and Silence:** A monitor that alerts infrequently may seem “good,” but how infrequent is acceptable? A monitor that remains in an OK state for months might be observing a stable system or it might be misconfigured with overly lenient thresholds.
- **Resolution Time:** A healthy monitor might have a reasonable time-to-resolution, so neither instantaneous nor excessively long. But what if one incident lasts far longer than usual? Should a single outlier mark the monitor as “bad”?
- **User Engagement:** Engagement can serve as a proxy for usefulness.
 - Frequent views after alerts might indicate importance.
 - Conversely, low engagement might reflect an excellent setup. Perhaps the notification message or playbook is so effective that no one needs to check the monitor page at all.
- **Business Relevance:** Even a technically perfect monitor may be watching an irrelevant or deprecated metric. If it provides no meaningful business value, can it still be considered “good”?

As these examples show, defining monitor quality is deeply **contextual and subjective**. This leads to the project’s most fundamental challenge: **the absence of ground-truth labels**. Unlike a typical machine learning dataset with predefined categories, Datadog monitors lack explicit quality annotations. Any attempt to rely on simplistic heuristics, such as `alerts_per_day < 1`, would be overly rigid and biased.

Compounding this difficulty is the presence of **unstructured data**, such as free-text queries and notification messages, which must also be analyzed to extract meaningful quality indicators.

2.2 INITIAL STATE AND EXISTING DATA

Prior to this project, Datadog had a basic **Monitor Analytics** (Datadog, 2024a) page that provided high-level health indicators. While a valuable first step, its heuristics were limited and served more as a list of potential issues than a comprehensive quality assessment. The four main indicators were:

- **Monitors in Continuous Alert:** Flags monitors that have been in an ALERT state for the entire previous 30 days.
- **Monitors with Broken Handles:** Identifies monitors failing to deliver notifications.

- **Noisy Monitors:** Flags monitors whose notification count in the past 30 days exceeds the 90th percentile for their organization.
- **Muted Monitors:** Lists monitors that have been muted for the entire previous 60 days.

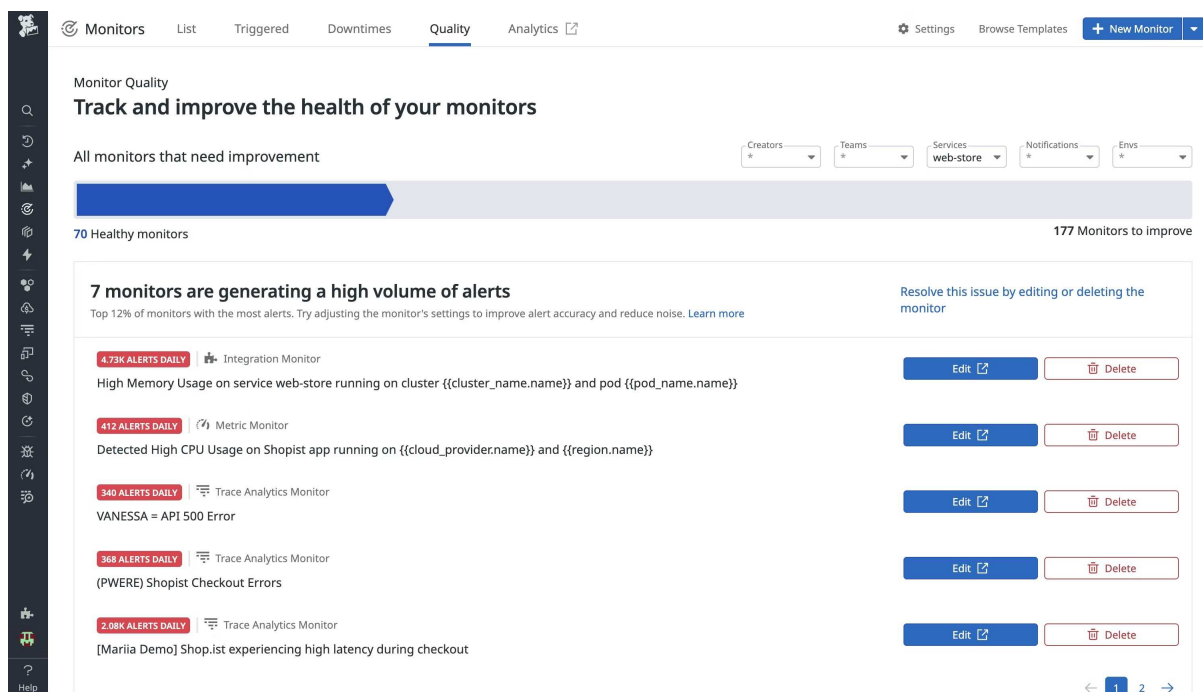


Figure 4 – Example of the Monitor Quality Page.

While useful, this system lacked the granularity required for our objectives. It provided a binary, often lagging, classification and could not differentiate between a slightly noisy monitor and a severely misconfigured one. For instance, a monitor that recovered yesterday after being in alert for 29 days would not appear, despite being problematic. This existing toolset confirmed the need for a more robust solution, one that could leverage the rich datasets available in the **Alerting Data Lake**.

The **Alerting Data Lake** is a centralized cloud storage repository (an AWS S3 for instance, but depends on the datacenter) where structured data from the production alerting systems is saved daily. This provided the raw material for the project. The key existing datasets are:

- **Monitor Definitions:** The master dataset containing a daily snapshot of the complete configuration for every monitor on the platform. It details the query, message text, tags, threshold settings, and notification handles. This dataset answers the question: *“What is this monitor supposed to do?”*

Key Columns: `monitor_id`, `org_id`, `monitor_type`, `query`, `message`, `tags`, `handles`.

- **Monitor Events:** An immutable log of every state transition for every monitor. Each row represents a single event, such as a monitor group transitioning from OK to ALERT. It is the primary source for behavioral analysis and answers the question: *“What did this monitor do, and when?”*

Key Columns: `monitor_id`, `event_timestamp`, `transition_name` (OK to ALERT), `time_to_resolution_in_seconds`, `monitor_group`.

- **Monitor Notifications:** Tracks the delivery status of every notification sent. It allows verification of whether an alert successfully reached its intended destination (e.g., Slack channel or PagerDuty service). This dataset answers: *“Did the right people get notified?”*

Key Columns: `monitor_id`, `delivery_status` (Success, Failure), `integration_name` (e.g. the slack channel, pagerduty service).

- **Monitor Downtimes:** Contains records of all scheduled downtimes (or “mutes”). It provides precise start and end times for any period a monitor was intentionally silenced by a user. This dataset answers: *“When were users intentionally ignoring this monitor?”*

Key Columns: `monitor_id`, `start_date`, `end_date`, `is_recurring`.

- **User Activity:** Sourced from Datadog’s own product analytics (RUM), this dataset logs user interactions with monitor pages in the Datadog application. It records every view, click, and edit action performed by a user on a specific monitor’s page. This dataset answers: *“How are users engaging with this monitor?”*

Key Columns: `monitor_id`, `datadog_user_id`, `action_type` (view, click), `action_name` (Edit Monitor, Mute), `action_timestamp`.

Together, these datasets provide a **360-degree view** of a monitor’s lifecycle, from its static configuration to its dynamic behavior and user engagement, forming the foundation upon which a sophisticated monitor quality score could be built.

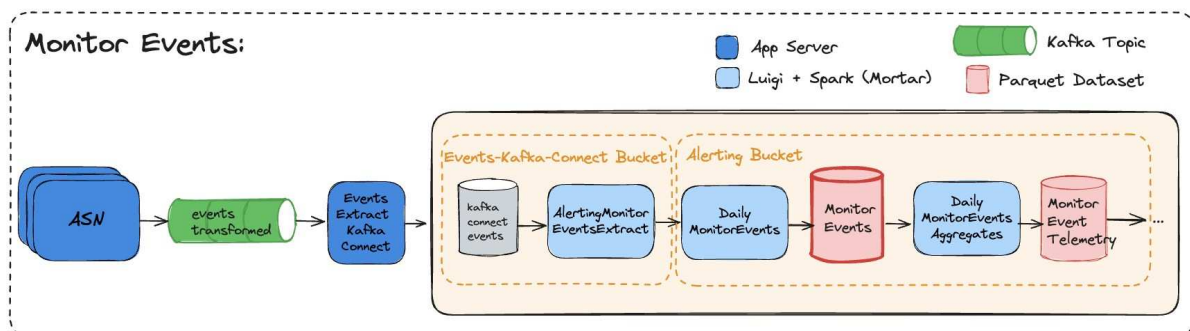


Figure 5 – Diagram of the Monitor Events Extraction Pipeline.

2.3 CHOSEN APPROACH AND JUSTIFICATION

The inherent ambiguity of **monitor quality** required a structured, methodical approach. The first critical step was to decompose this broad concept into two distinct, measurable dimensions:

- **Configuration Quality:** Answers the question, *“Is this monitor technically well-constructed?”* This dimension concerns measurable behaviors such as signal-to-noise ratio, alert stability, and user engagement patterns. A monitor with high configuration quality is one that is not “flappy,” resolves in a timely manner, and is acted upon when it triggers.
- **Relevance:** Answers the question, *“Is this monitor tracking something that is truly important to the system?”* This is a more abstract dimension related to the criticality of the monitored metric and its contribution to the overall observability coverage of a service. For example, a monitor tracking database writes for a write-heavy application is likely more relevant than one tracking cache hit rates on a non-critical component.

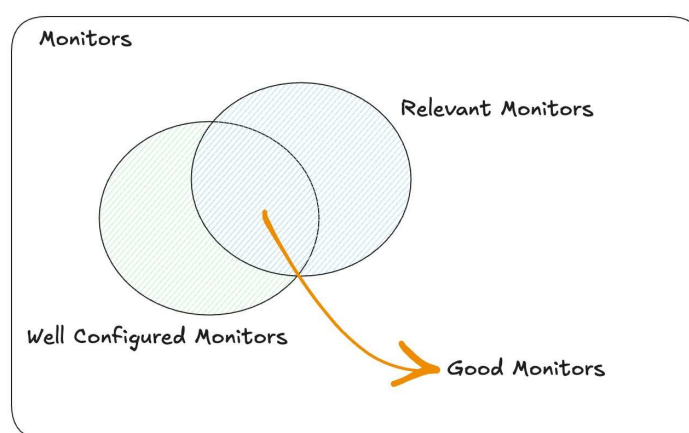


Figure 6 – Good Monitors are the intersection of the Well Configured Monitors and Relevant Monitors Set.

For the scope of this internship, the primary focus was placed on developing the **Configuration Score**. This decision was strategic: it addressed the most pressing user pain point: alert fatigue from poorly configured monitors; and was more directly measurable using available behavioral and engagement datasets.

Inferring Quality from Outcomes, Not Definitions

One potential approach to scoring configuration quality would be to analyze a monitor’s definition directly, using Natural Language Processing (NLP) to parse its

query and message text. However, this method introduces significant complexity and ambiguity.

A more pragmatic and robust approach was adopted: to **infer configuration quality from observable outcomes**. The central hypothesis is that a poorly configured monitor will inevitably produce poor results. For instance, it will be noisy, have long resolution times, or be ignored by users. Therefore, the model was designed to learn from a monitor's **behavioral and engagement data** (e.g., events, notifications, user activity) rather than from its configuration text or query. This provides a strong foundation upon which future models could build to learn the specific characteristics of well-written queries or messages.

Justification for Supervised Machine Learning

Given the nuances explored previously, it was clear that a system based solely on simple, hand-tuned thresholds would be too brittle and biased. A **machine learning** approach was therefore selected to capture the complex, non-linear relationships between the various data signals influencing monitor quality.

The next design choice was between **unsupervised** and **supervised learning**. While an unsupervised clustering model could potentially identify groups of “bad” monitors, this project represented a high-stakes application:

- A false positive, incorrectly flagging a critical monitor as low-quality, could lead a user to delete it, introducing dangerous observability gaps.
- Such a mistake could cause a client to miss a major outage, potentially resulting in substantial revenue loss.

For these reasons, a **supervised learning** approach was chosen. It provides several key advantages:

- **Control and Reliability:** Enables explicit definition of what constitutes a “good” or “bad” monitor, providing control over the model's objective.
- **Explainability:** Simpler supervised models offer greater transparency into why a prediction was made, crucial for a user-facing system.
- **Performance:** Supervised models generally achieve higher accuracy on well-defined classification tasks.

The primary obstacle for a supervised approach, however, was the **absence of ground-truth labels**. The following chapter will describe the **weak supervision strategy** developed to overcome this challenge and programmatically construct the labeled dataset necessary for training the model.

3 DEVELOPMENT OF THE MONITOR QUALITY SCORE

This chapter details the technical implementation of the monitor quality project, from raw data processing to the creation of a labeled dataset ready for machine learning and the training of these models. While an experimental phase with various approaches was conducted, only the last subsection will contain the final successful implementation.

3.1 THE DATA PIPELINE

The data pipeline was developed on top of the alerting data lake previously described. In this section, rather than focusing on the specific logic behind feature creation (such as joins or transformations), we will discuss the key features that were engineered and their motivations.

Given the dataset exceeded 100 GB, traditional Python libraries such as **pandas** were impractical for large-scale processing. To address this, we leveraged Apache Spark, which supports distributed processing across multiple cores. By utilizing a virtual machine with 32+ CPUs, the entire pipeline became computationally feasible and significantly faster than a single-core workflow.

Session Definition

A monitor in Datadog can be created, edited, or deleted. Since our goal is to score each monitor based on its configuration, we must consider its definition at every point in time. For example, if a user changes a threshold, the monitor's quality score may change substantially.

To handle this, we introduced the concept of a *session*, defined as the time interval between two consecutive monitor editions (including creation and deletion). For instance, a monitor created on **2025-01-10** and edited on **2025-05-01** would produce two sessions: one from **2025-01-10** to **2025-05-01**, and another from **2025-05-01** to the present. The Spark pipeline developed generated all such sessions, enforcing a minimum session duration of seven days. This threshold was hard-coded, as shorter periods were deemed insufficient to properly evaluate a monitor's performance.

It is worth noting that the alerting data lake retained one year of data in Amazon S3. Consequently, if a session's start date preceded the earliest available timestamp, it was truncated to the minimum date present in the dataset.

Monitor Definition Features

Several features were extracted directly from each monitor's definition, reflecting both configuration quality and best practices. Although not all were used in the final

model, the following represent key examples linked to monitor quality and alerting outcomes:

- **Length of the message:** Short alert messages often lack context or runbooks, reducing their actionability.
- **Number of tags:** A higher number of tags generally reflects better organization and context.
- **Presence of service/team tags:** Monitors with explicit `service:` or `team:` tags typically indicate ownership and accountability.
- **Created via API:** Monitors managed via Infrastructure-as-Code (e.g., Terraform) are often better maintained.
- **Session length / days since creation:** Provide time-based normalization and indicate monitor maturity.
- **Deleted flag:** Useful for downstream label generation (e.g., detecting abandoned monitors).
- **Number of handles:** A missing or excessively large number of handles (e.g., > 20) may indicate misconfiguration.
- **Contains “test” in name:** Often signals non-production or temporary monitors, relevant for filtering or labeling.
- **High urgency:** Monitors using high-urgency PagerDuty handles (e.g., those paging at night) are typically critical and well-tuned.

Notification Features

From the notifications dataset, we computed several session-level metrics by aggregating all notification events associated with a monitor:

- **Number of Failed Notifications:** Capture configuration health.
- **Number of Successful Notifications:** Capture volume of notifications.
- **Notifications to least-notified handle:** Identifies how many notifications the handles receive.

Downtime Features

Downtime data was joined with the session windows to compute the total muted duration per session. Mutes (downtimes without end timestamps) were handled explicitly, as they represent monitors intentionally silenced due to alert fatigue.

User Activity Features

To capture engagement, we calculated:

- The percentage of days with at least one alert that received a user view.
- The number of distinct users who accessed the monitor page during the session.

These features were crucial for linking the alerting datasets to user behavior and engagement patterns.

Event Features

The *events dataset* was the richest source of behavioral insight, providing detailed information about alert occurrences, resolutions, and their temporal patterns. To design the right set of features, we first performed an exploratory data analysis (EDA) on a sample of the data using **pandas** in Python. The objective of this analysis was to understand how alerting behavior varied across monitors and to identify meaningful statistical summaries that could later be scaled using Spark.

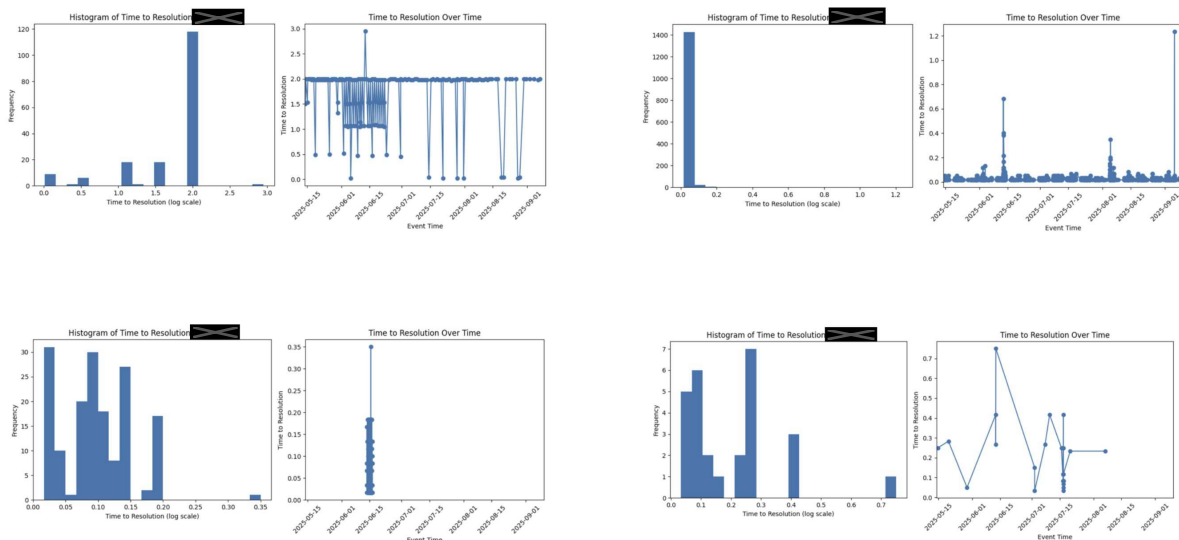
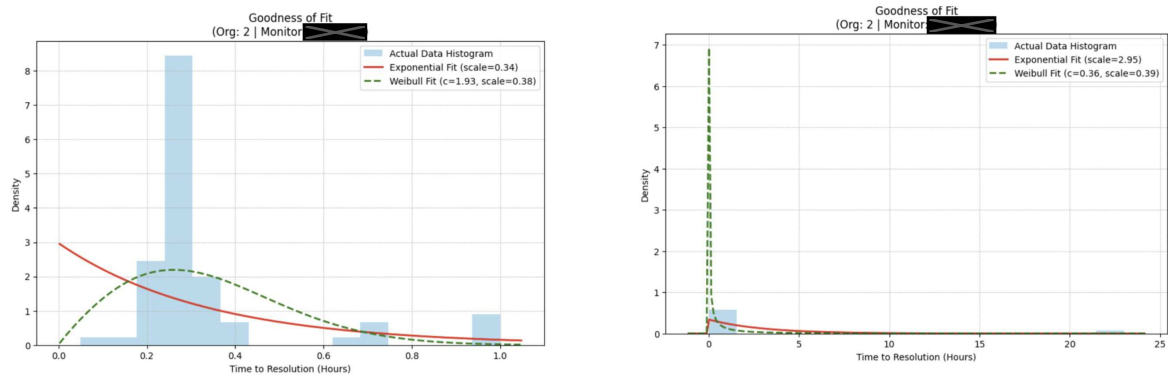


Figure 7 – Distribution of Time to Resolution for 4 samples.

By examining the plots generated during this analysis, we could visualize how the *time to resolution* (TTR) was distributed, as well as the total number of alerts per monitor. For instance, monitor  typically resolved alerts within approximately two hours but exhibited some near-zero outliers, while monitor  showed a highly noisy pattern where most TTR values were below 0.1 hours. This kind of behavior suggests an over-alerting configuration or an automated resolution process. It is important that a monitor's TTR remains above a minimum threshold, around 10 minutes in our case, to indicate that alerts were not automatically resolved, but also

not so long (e.g., several days) that they reflect ignored or low-priority issues. Another example, monitor  showed all alerts occurring within a single day before abruptly stopping, possibly because the monitor was muted or deleted.

From these observations, it became evident that TTR values did not follow a normal distribution. Consequently, using simple statistics such as the mean and variance would misrepresent the data. Instead, we fitted two alternative probability distributions that are well-suited for modeling time-to-event data: the *exponential* and the *Weibull* distributions. After comparison, the Weibull distribution proved to better capture the heavy-tailed nature of the TTR data. However, due to the absence of suitable built-in statistical functions in Spark's Scala environment, this fitting step could not be implemented efficiently at scale.



Next, we analyzed the *time between failures* (TBF): the elapsed time between consecutive alerts for a monitor. Ideally, well-behaved monitors should have a relatively high TBF (e.g., greater than one day), as frequent alerting can cause alert fatigue. However, we also accounted for sample size effects: a low TBF (e.g., five minutes) might not necessarily indicate a bad monitor if it corresponds to only two closely spaced alerts. Similar to TTR, the TBF distributions were non-normal and required a Weibull fit, since traditional metrics like mean or median are not reliable under strong skewness or small sample sizes.

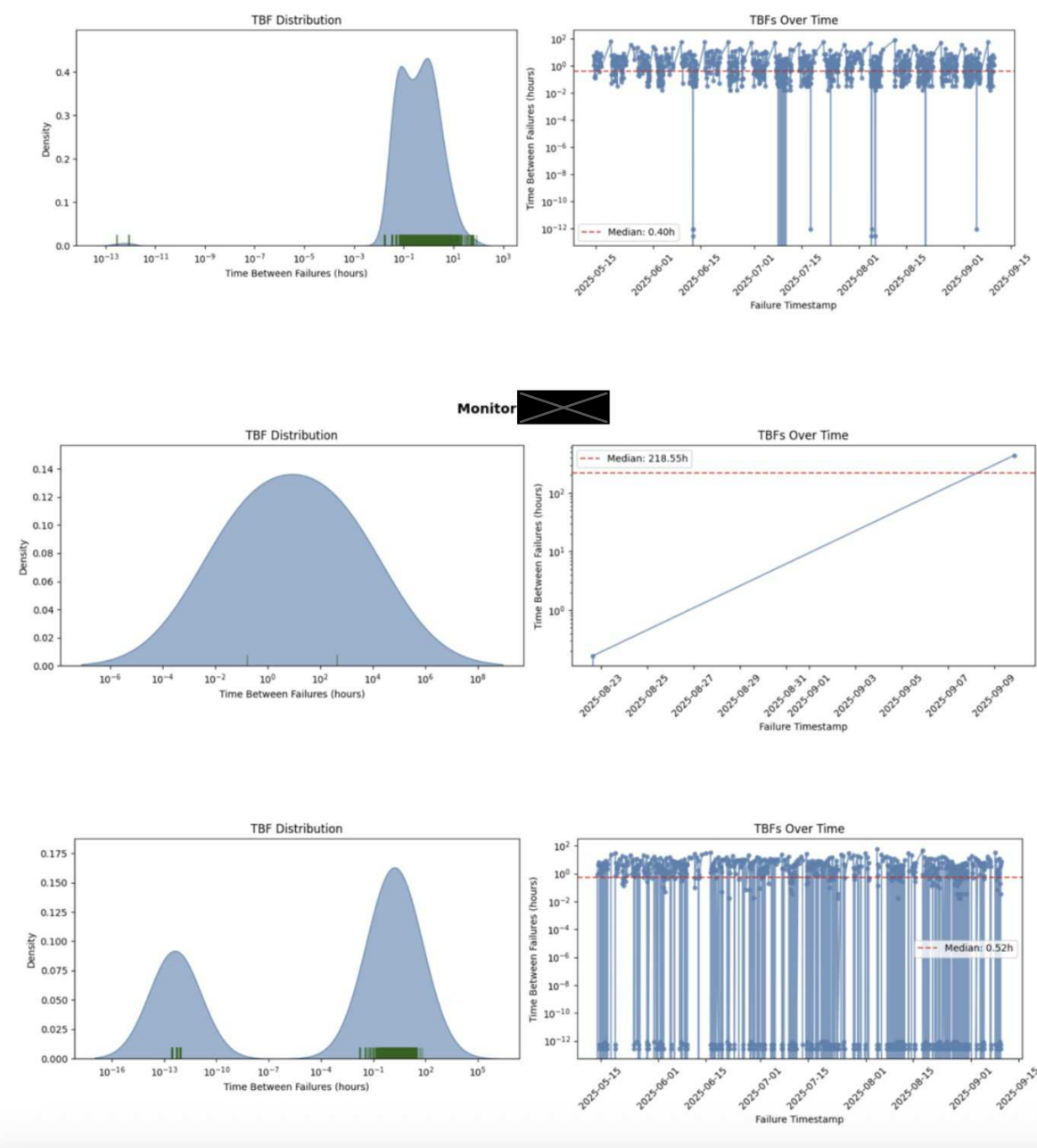


Figure 9 – .

After deriving these features, we applied the HDBSCAN clustering algorithm to identify behavioral patterns. The algorithm revealed three primary clusters:

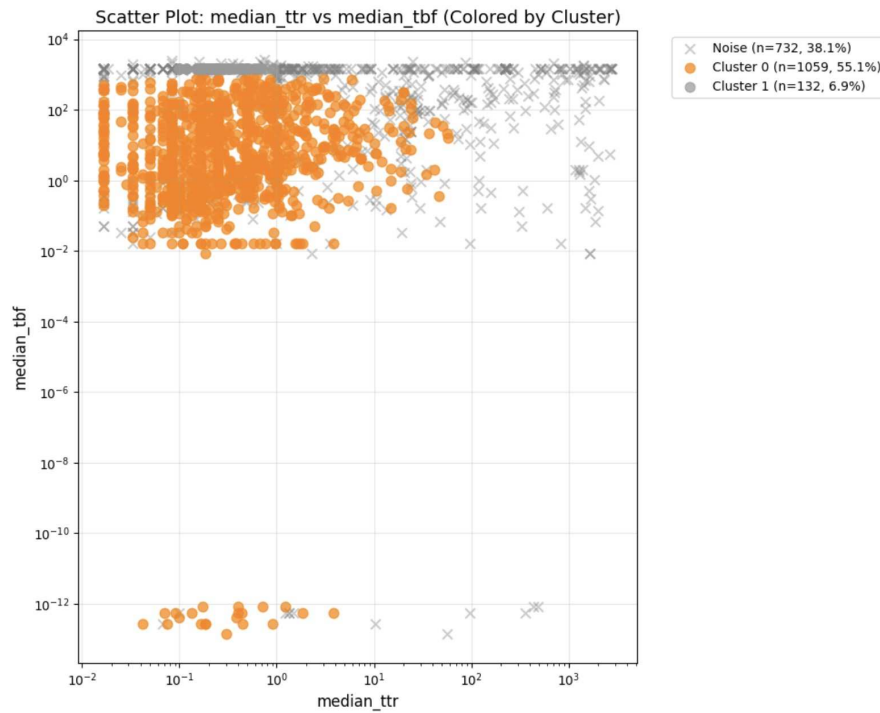


Figure 10 – .

1. **Noisy monitors:** Median TBF between 0.01 and 100 hours, and TTR below 100 hours, likely corresponding to unstable or poorly configured monitors.
2. **Outlier monitors:** Those with extremely high TTR values (above 100 hours) or near-zero TTRs, both of which suggest abnormal configurations.
3. **Healthy monitors:** A stable cluster characterized by balanced and consistent TTR and TBF values.

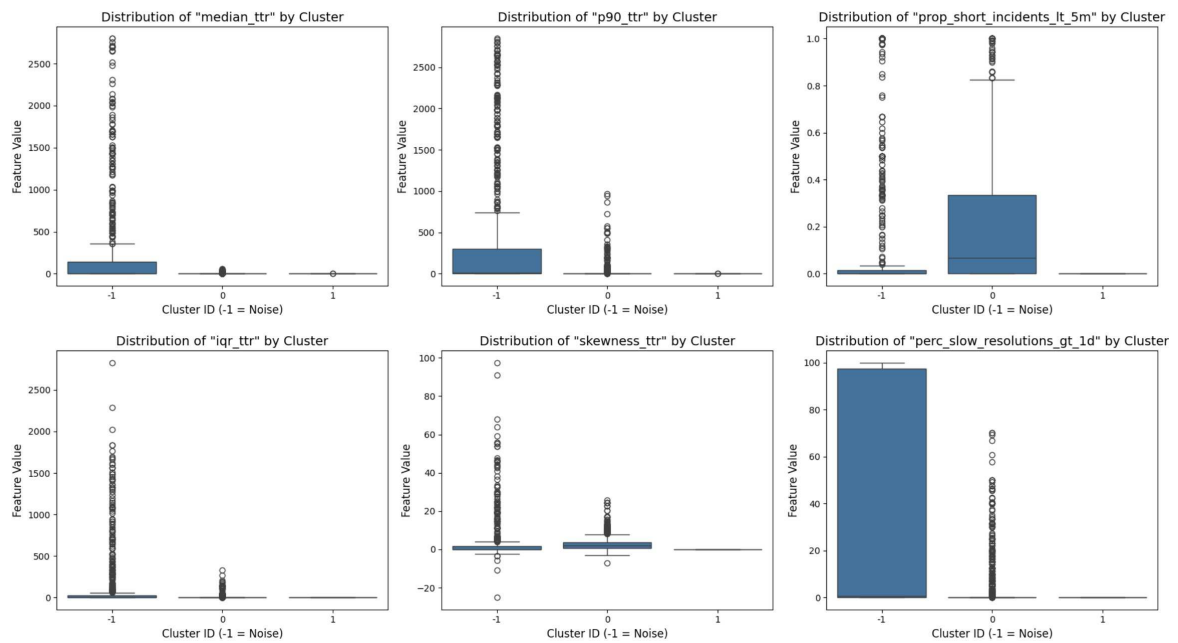


Figure 11 – .

These analyses confirmed that both TTR and TBF contain valuable behavioral signals that help distinguish between effective and ineffective alerting setups.

The engineered event features derived from this analysis were grouped as follows:

General features: total number of alerts, total number of groups, maximum alert transitions per day, and percentage of time in alert.

TTR-based features: median_ttr, p90_ttr, iqr_ttr, skewness_ttr, cv_ttr, and histogram-based proportions (e.g., hist_lt_5m, hist_1h_to_6h, hist_gt_7d).

TBF-based features: median_tbf, max_tbf, pct_short_gaps, p10_tbf, cv_tbf, and max_tbf.

While mean-based statistics were avoided due to the strong skew in both distributions, the Weibull model remains a promising direction for future computation once implemented efficiently in Spark.

Labeling Strategy

For weak supervision, we proposed several proxy labels linking features from the alerting datasets:

- **High engagement monitors:** Those with rapid user responses (average time from alert to view < 2 hours), derived by joining the notification and user activity datasets.
- **Critical monitors:** Identified by the presence of high-urgency PagerDuty handles, indicative of essential, well-configured monitors.

- **Bad monitors:** Monitors remaining in an alert state for more than 30 consecutive days, suggesting neglect or poor configuration.

Care was taken to prevent feature–label leakage (e.g., excluding the average view delay feature when labeling based on engagement). After preprocessing, approximately 100,000 sessions were selected for model training (sampled from a big dataset of monitor sessions), restricted to those containing alerts. Of these, around 5% were labeled through the above heuristics.

3.2 THE EXPERIMENTAL PHASE: TRIALS AND ITERATIONS

In this subsection, I describe the characteristics of the features used in the experiments, their distributions, the additional features I engineered, and the initial models I trained and evaluated.

3.2.1 Data Analysis

This part presents the feature distributions for some of the labels derived from the dataset. The goal is to understand how the features behave and how they differ across classes.

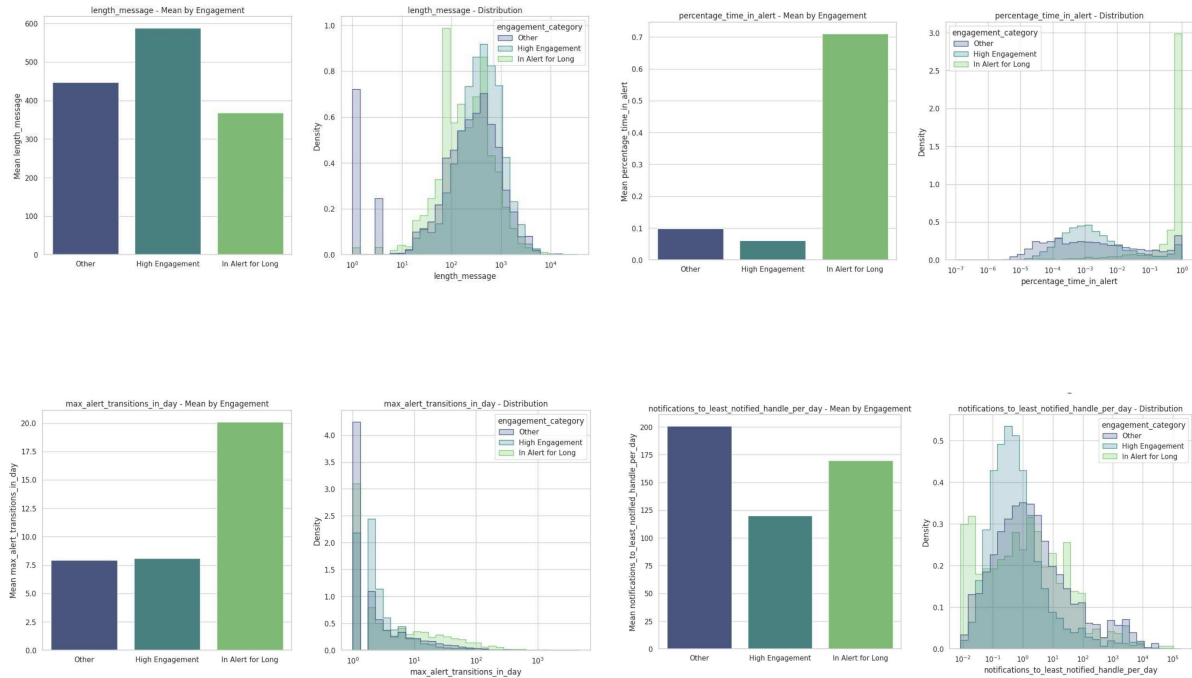


Figure 12 – Example of feature distributions for selected labels.

From the graphs, we can observe a clear difference between the *high engagement* and *constant in alert* monitor classes, which respectively represent well-performing and poorly-performing monitors. The “Other” in this case are the unlabelled

datapoints. We can also see that many features exhibit skewed distributions, which indicates the need for preprocessing, particularly if we plan to use more complex machine learning models such as neural networks.

3.2.2 Supervised Learning

In this part, I applied a classic supervised learning approach. I removed all unlabeled data from the dataset and trained a classifier using LightGBM, a widely adopted gradient boosting framework (a detailed explanation is omitted here, as it is not the focus of this report). I followed best practices such as splitting the data into training and test sets and avoided normalization, since tree-based methods do not require it. As shown in the confusion matrix and ROC curve, the model achieved near-perfect performance.

However, the objective here is not only to achieve high accuracy but also to assess the model's *explainability*: whether it is learning meaningful patterns, avoiding overfitting, and producing interpretable scores. Since this is a classification model, the predicted probabilities are interpreted as scores.

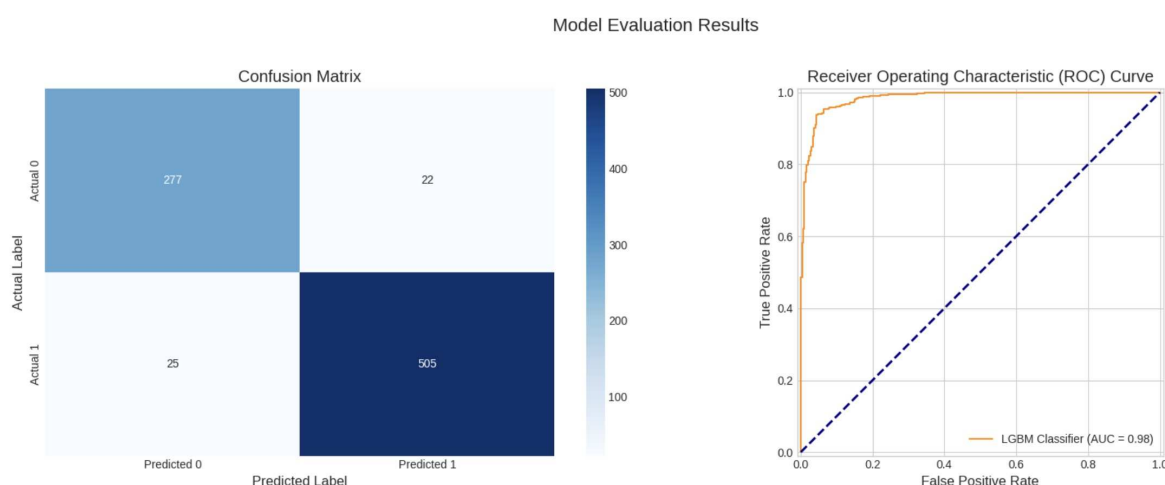


Figure 13 – Performance metrics of the supervised model.

For model explainability, I used SHAP (SHapley Additive exPlanations) values. The plot below ranks the features by importance and shows their contribution to the model's output. Features that push the prediction toward the negative class appear on the left, while those pushing toward the positive class appear on the right. Blue points correspond to low feature values, and red points correspond to high feature values.

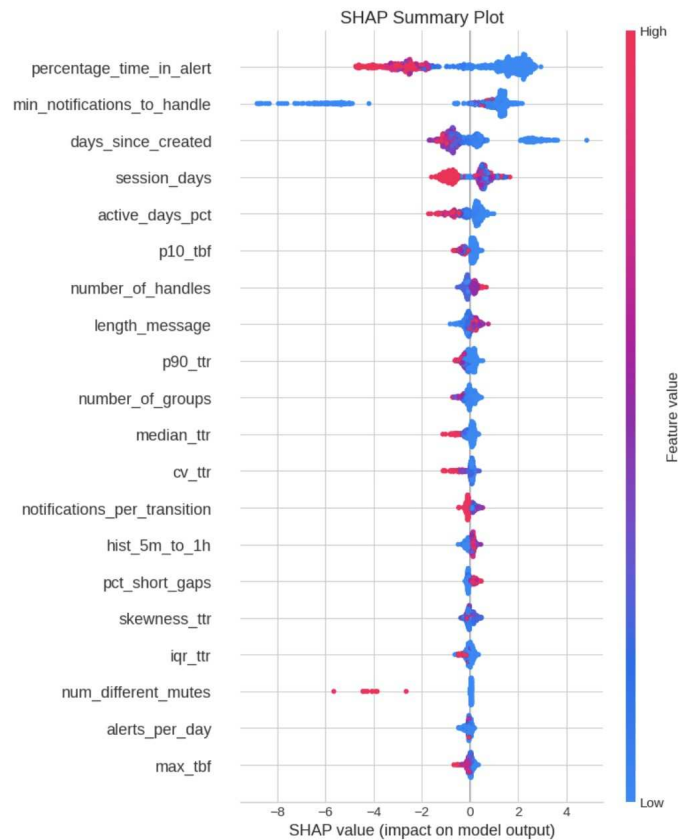


Figure 14 – Feature importance and contribution based on SHAP values.

By examining the SHAP plot, I noticed that some features did not behave as expected. For example, for the feature *notifications sent to least notified handle* (written above as `min_notifications_to_handle`), the model learned that higher values are better, whereas in reality, a value of zero is undesirable, moderate values (e.g., 1–20) are good, and excessively high values are problematic. Similarly, for the feature *p10 of TBF*, the model learned that lower values are better, but the opposite is true. Moreover, it assigned high importance to features such as *days after creation* and *session length*, which suggests potential overfitting—perhaps because older monitors in the dataset happened to correspond to poorer performance.

Next, I applied the model to the unlabeled data. The resulting score distribution was not smooth; instead, it showed two sharp peaks near 0 and 1. Ideally, we would expect a smoother distribution—scores around 0.3 or 0.6, for example, since only consistently poor or excellent monitors should receive extreme values. However, since the model outputs probabilities rather than calibrated scores, this behavior is not unexpected.

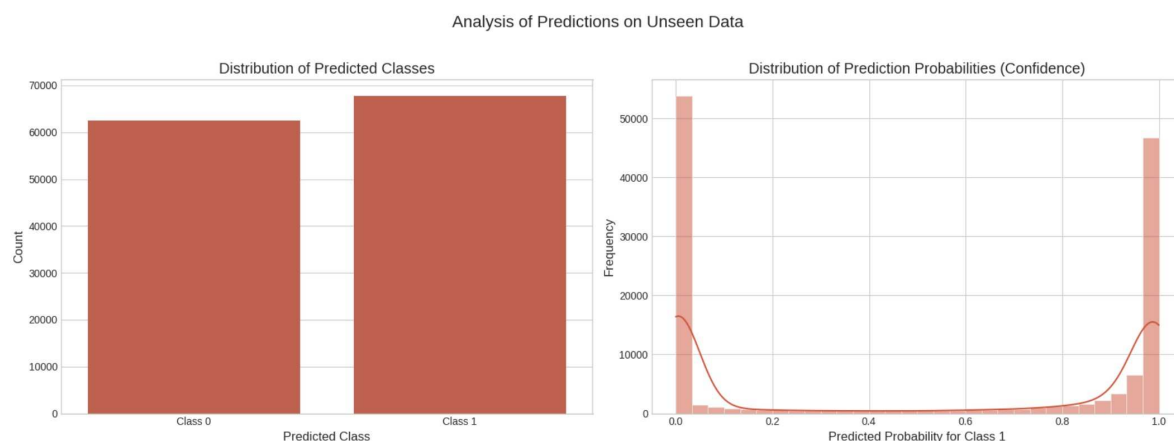


Figure 15 – Distribution of predicted probabilities on the unlabeled data.

Finally, I investigated whether excluding the unlabeled data affected the model's generalization. I trained another classifier to distinguish between labeled and unlabeled samples, and it achieved an AUC of nearly 90%. This indicates a significant difference between the two datasets, a phenomenon known as *data drift*. This is undesirable, as it suggests that we are losing valuable information by not leveraging the unlabeled data, given that it represents a distribution distinct from the labeled portion.

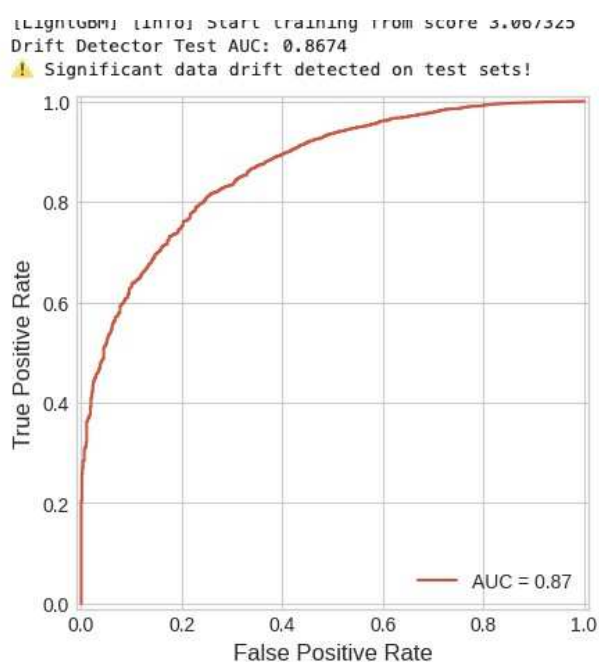


Figure 16 – Evidence of data drift between labeled and unlabeled data.

The idea of this test was to be only a baseline. So we expect that the final model will have better results.

3.2.2.1 Anomaly Detection

The next baseline approach involved training an anomaly detection model. In this setup, the model learns to characterize what constitutes *normal* behavior, and any significant deviation from this learned pattern is classified as an anomaly. We experimented with two variants: one where the *bad* data was treated as normal, and another where the *good* data was treated as normal. The model used for both experiments was an Isolation Forest.

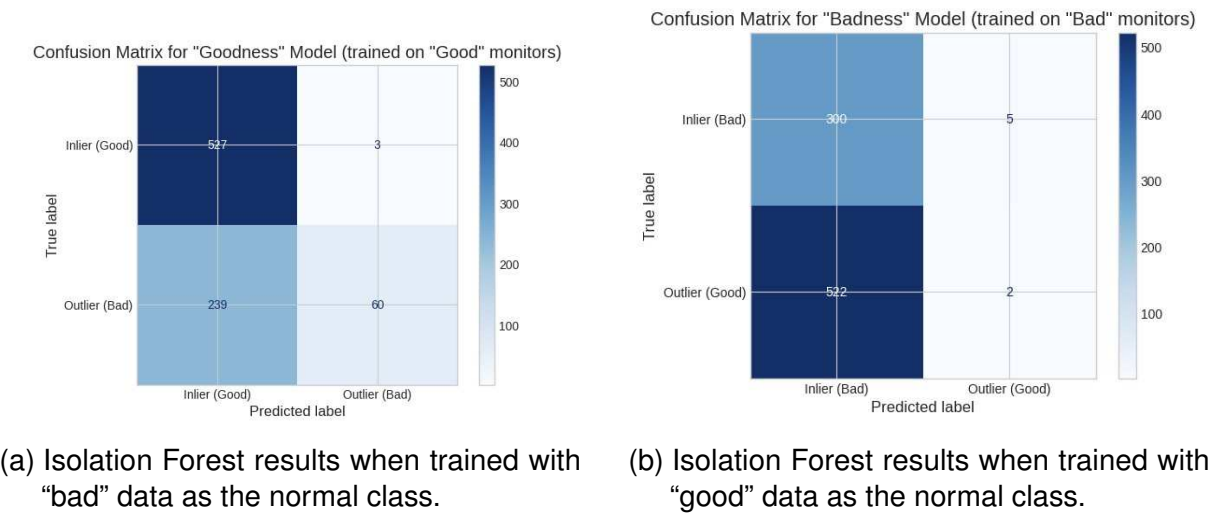


Figure 17 – Isolation Forest results comparison

Both models performed poorly, failing to effectively learn a consistent representation of what constitutes *normal* behavior. As a result, each model tended to classify most data points as belonging to the same category it was trained on. This outcome can be attributed to the inherent complexity and heterogeneity of the data.

In particular, the model trained on *bad* monitors (AUC = 0.60) struggled because the "bad" class is not homogeneous. Poor-performing monitors can fail for a variety of reasons—such as flapping, delayed alert resolution, or unstable triggering patterns—making it difficult for the model to capture a unified definition of "badness." Consequently, the model's ability to identify truly anomalous (i.e., good) samples was weak.

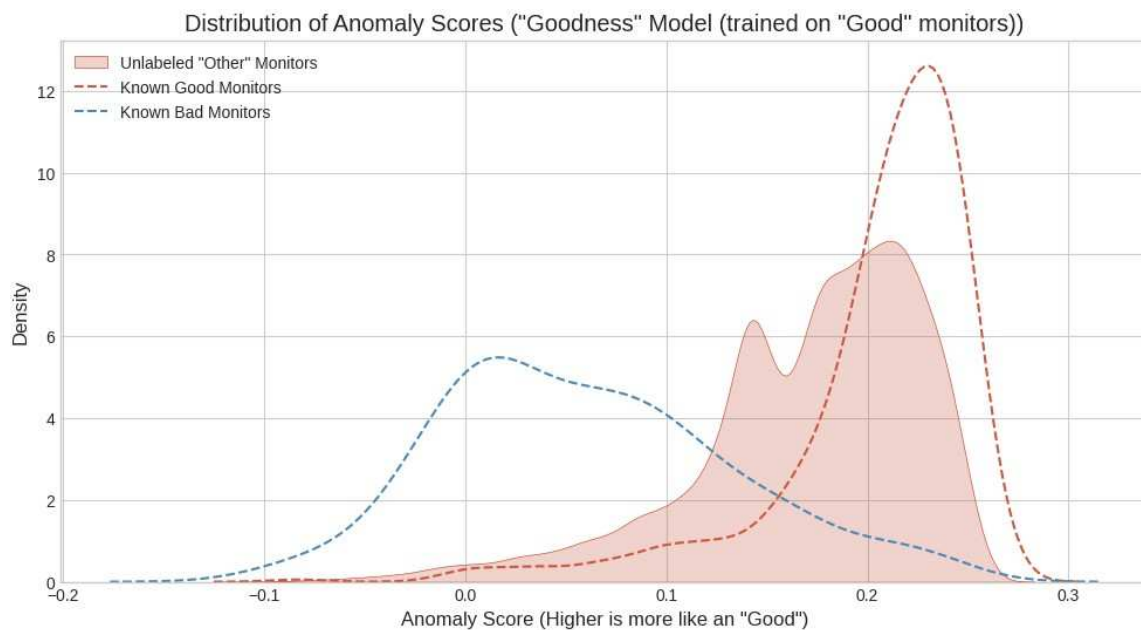


Figure 18 – Score distribution when using “bad” data as the normal class.

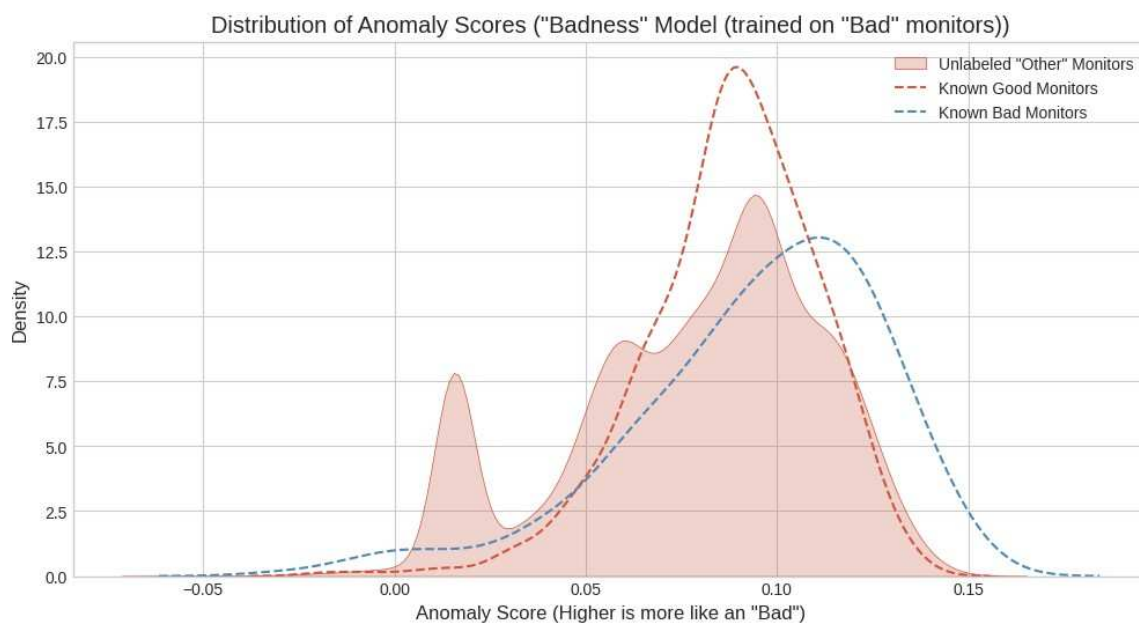


Figure 19 – Score distribution when using “good” data as the normal class.

Finally, the figures above show the score distributions on the unlabeled dataset. As observed, the distributions are poorly separated, further indicating that the model failed to capture meaningful patterns of normality or abnormality. No detailed explainability analysis was performed for this approach, as the overall results were already unsatisfactory.

3.2.2.2 Positive Unlabelled Learning

In this experiment, the objective was to extend the anomaly detection approach by incorporating **Positive Unlabeled (PU) Learning**. The goal was to leverage both labeled and unlabeled data, rather than relying solely on the limited labeled subset used in the supervised baseline.

In PU learning, we assume that only a subset of the positive examples is labeled, while the remaining data contains a mix of positive and negative examples without explicit labels. The model must therefore learn to separate the hidden negative samples within the unlabeled group. To guide this process, we provide the algorithm with an estimate of the class prior—here (Zhao *et al.*, 2022), we assumed that approximately 50% of the unlabeled data belongs to the positive class.

The training objective is then twofold:

1. Minimize the classification loss for the labeled positive samples.
2. Adjust the model's predictions on the unlabeled data such that the expected proportion of positives matches the prior (e.g., 0.5).

A small neural network was used for this experiment, requiring the input data to be normalized. Given the strong right-skewed distribution of several features, we applied the **Yeo-Johnson power transformation** to stabilize variance and improve feature scaling.

Although the confusion matrix obtained from this model looked promising, the **explainability analysis revealed inconsistencies** with expected feature importance patterns again. For this reason, the detailed results are not presented here; the main purpose of this subsection is to document the methodology and lessons learned.

3.2.2.3 Weak Labels

Since the supervised and PU learning models showed signs of capturing unintended patterns, I next explored an alternative labeling strategy based on **weak supervision**. The core idea of weak supervision is to automatically generate noisy labels using a large set of heuristic rules, rather than relying entirely on manually labeled data.

In this framework, multiple labeling functions are defined using domain knowledge. Each function expresses a simple, imperfect rule that maps feature values to label guesses. For instance:

- If `alerts_per_day > 10` $\Rightarrow$ label as *bad monitor*.
- If `active_days_pct > 90%` $\Rightarrow$ label as *good monitor*.
- If `cv_tbf > 1.5` $\Rightarrow$ label as *unstable*.

Individually, these heuristics are noisy and often contradictory, but by combining many of them, we can estimate a probabilistic consensus. This is achieved using a **generative model**, typically Bayesian, that learns the reliability and correlation of each labeling function, producing denoised “pseudo-labels.” These pseudo-labels can then be used to train a downstream discriminative model (e.g., a classifier).

This approach effectively encodes expert intuition directly into the data, guiding the model toward desirable behaviors without fully manual labeling. Although there is an inherent risk of bias from hand-crafted rules, this effect diminishes when a large and diverse set of labeling functions is used.

The results of this experiment are not shown here, as they did not outperform previous models; however, they provided valuable insight into how rule-driven labeling strategies can be used to align model predictions with domain expectations in future iterations.

3.3 THE MACHINE LEARNING FINAL MODEL

In this subsection, I present the final version of the model designed to assess monitor quality. The proposed solution combines deterministic rule-based logic with machine learning techniques to achieve both interpretability and predictive power. Specifically, it integrates the impact of critical, deal-breaking metrics with a data-driven evaluation of more nuanced behavioral patterns.

This hybrid approach ensures that monitors exhibiting clear signs of failure are heavily penalized, while those showing subtle differences in performance are scored more flexibly. In essence, the model balances explainability and adaptability, allowing to capture complex relationships without losing interpretability.

3.3.1 Model Overview

The final monitor quality score, denoted as S_{final} , is a value between 0 and 1. It is computed as the product of two primary components: a **Critical Score** and an **Extra Score**.

$$S_{final} = S_{critical} \cdot S_{extra} \quad (1)$$

This multiplicative structure ensures that a failure in any critical area (e.g., a monitor that is always in alert) significantly reduces the overall score, regardless of how well the monitor performs in other aspects.

3.3.1.0.1 Part 1: The Critical Score ($S_{critical}$)

The **Critical Score** captures fundamental aspects of monitor health. A low value here directly signals an unusable or unhealthy monitor. We identified three key features

that can each invalidate monitor quality:

1. Percentage of time in alert (x_{alert})
2. Percentage of time muted (x_{mute})
3. Failed notification ratio ($x_{notification}$)

Each feature is converted into a score between 0 and 1 using a sigmoid function:

$$p(x) = \frac{1}{1 + e^{-k(x-x_0)}} \quad (2)$$

Where:

- x is the input feature value.
- x_0 is the midpoint, where $p(x) = 0.5$.
- k controls the steepness of the curve.

The final critical score is the product of these three individual scores:

$$S_{critical} = p_{alert}(x_{alert}) \cdot p_{mute}(x_{mute}) \cdot p_{notification}(x_{notification}) \quad (3)$$

Fitting Strategy for Sigmoids. Rather than relying on arbitrary thresholds, the sigmoids are fitted using two complementary strategies:

- **Heuristic Fitting:** Define domain-specific thresholds based on operational knowledge. For instance, if a monitor being in alert more than 50% of the time is unacceptable, we set $x_0 = 0.1$ (10%) with a steep k that penalizes values above this level.
- **Proxy Labeling:** Train simple logistic regression models for each feature using labels such as `in_alert_for_long`. The regression parameters naturally yield the optimal k and x_0 values to separate good and bad monitors for each metric.

3.3.1.0.2 Part 2: The Extra Score (S_{extra})

The **Extra Score** evaluates more nuanced attributes of a monitor that are not inherently critical but still correlate with overall quality. It is composed of two sub-scores:

- A. **Metadata Score** ($S_{metadata}$)
- B. **Event Pattern Score** (S_{events})

A. Metadata Score ($S_{metadata}$). This score measures the quality of the monitor's configuration, such as the presence of tags, descriptive titles, or clear notification settings. Missing metadata should penalize the score but not reduce it to zero. To model this behavior, we use a weighted sum of feature-specific scores normalized between 0 and 1:

$$Influence_{meta} = w_1 \cdot score(x_{meta_1}) + w_2 \cdot score(x_{meta_2}) + \dots \quad (4)$$

The combined influence is passed through a final sigmoid transformation:

$$S_{metadata} = \frac{1}{1 + e^{-Influence_{meta}}} \quad (5)$$

B. Event Pattern Score (S_{events}). This is the most complex component. It captures dynamic behavioral patterns such as alert frequency, time-to-resolution (TTR) statistics, and overall stability. These features often exhibit non-linear and interdependent relationships that cannot be effectively captured with simple rules.

We employ a machine learning model f_{ML} to learn these relationships directly from data. Gradient Boosted Trees (e.g., LightGBM) are used due to their robustness and interpretability for tabular data.

$$S_{events} = f_{ML}(x_{groups}, x_{median_{ttr}}, x_{p90_{ttr}}, x_{cv_{ttr}}, \dots) \quad (6)$$

The model incorporates the following input features:

- number_of_groups, max_alert_transitions_in_day
- median_ttr, p90_ttr, iqr_ttr, skewness_ttr, cv_ttr
- hist_lt_5m, hist_5m_to_1h, hist_1h_to_6h, hist_6h_to_24h, hist_1d_to_7d, hist_gt_7d
- median_tbf, pct_short_gaps, p10_tbf, cv_tbf, max_tbf
- active_days_pct, alerts_per_day, event_spread_std_dev, activity_duration_pct
- notifications_to_least_notified_handle_per_day, no_handles

The final extra score is a weighted combination of the metadata and event pattern components:

$$S_{extra} = w_{meta} \cdot S_{metadata} + (1 - w_{meta}) \cdot S_{events} \quad (7)$$

3.3.1.0.3 Training Strategy for the ML Component

The core challenge in training f_{ML} is ensuring that it learns meaningful behavioral distinctions rather than merely replicating patterns already captured by the rule-based $S_{critical}$.

To achieve this, we train f_{ML} as a **classification model** using high-quality labels derived from monitor engagement patterns:

- Monitors labeled as High Engagement or High Urgency are assigned $Y = 1$.
- Monitors labeled as In Alert For Long are assigned $Y = 0$.

During training:

1. Compute $S_{critical}$ for each monitor using the pre-fitted sigmoids.
2. Use both the raw features (listed above) as model inputs.
3. Optimize using a classification loss (e.g., binary cross-entropy) to directly predict monitor quality.

This classification-oriented training allows the ML model to focus on patterns in event data that correlate with high or low engagement, rather than attempting to replicate deterministic logic. It ensures the model captures behavioral nuances (e.g., irregular alerting, timing patterns, resolution consistency) that rule-based methods cannot.

3.3.1.0.4 Interpretation

This hybrid system combines:

- An interpretable, rule-based layer ($S_{critical}$) for handling critical failures.
- A data-driven, machine learning layer (S_{events}) for nuanced behavioral modeling.

Together, they produce a single unified score:

$$S_{final} = S_{critical} \cdot S_{extra} \quad (8)$$

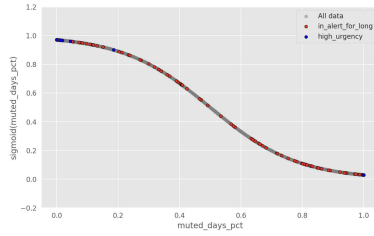
This structure yields the best of both worlds—robust interpretability and fine-grained, data-driven sensitivity to monitor quality.

4 RESULTS AND EVALUATION

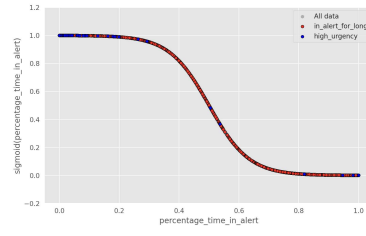
This section presents both the quantitative and qualitative evaluation of the hybrid model for monitor quality scoring. The goal is to assess how well the proposed system performs in practice, both from a numerical perspective and in real-world validation scenarios.

4.1 QUANTITATIVE RESULTS

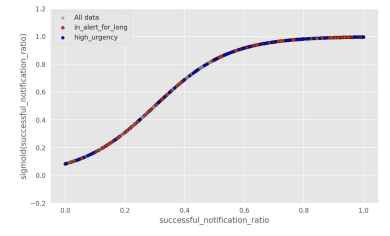
The first step of the training process involved fitting the sigmoid functions used in the computation of $S_{critical}$. These sigmoids were trained using a binary classification log-likelihood objective, ensuring that they accurately reflect the probability of a monitor being in an undesirable state.



(a) Result Sigmoid for Percentage of Days Muted Feature.



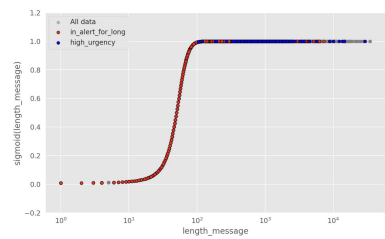
(b) Result Sigmoid for Percentage of Days in Alert Feature.



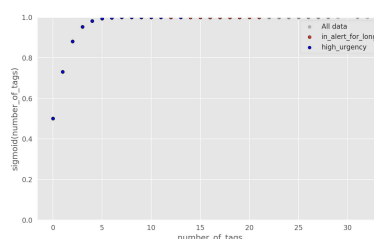
(c) Result Sigmoid for Successful Notification Ratio Feature.

Figure 20 – Resulted Sigmoids for the $S_{critical}$ score

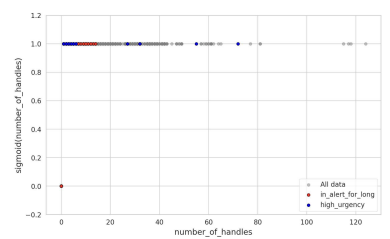
For the metadata component, a similar procedure was followed to fit each sigmoid function. Since the architecture is modular, new metadata features can easily be integrated in the future without retraining the entire model—only the corresponding sigmoid functions would need to be fitted.



(a) Result Sigmoid for Message Length Feature.



(b) Result Sigmoid for Number of Tags Feature.



(c) Result Sigmoid for Number of Handles Feature.

Figure 21 – Resulted Sigmoids for the metadata score

Next, we trained the internal machine learning model, implemented with **XG-Boost**. Its performance was comparable to that of the earlier supervised learning baseline. However, we introduced *monotonic constraints* to guide the model toward

behaviorally consistent patterns, ensuring that features such as alert duration or failure ratios influence the score in the expected direction. Additionally, careful hyperparameter tuning was applied to achieve a smoother and more uniform final score distribution across monitors.

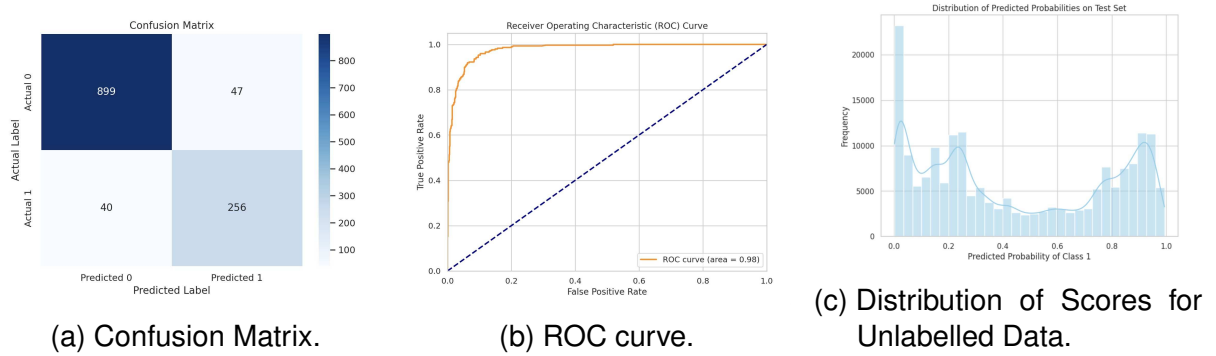


Figure 22 – S_{events} ML model results

Model explainability was analyzed using SHAP values. The results confirmed that the most influential features aligned with domain expectations, for example, time-to-resolution (TTR) metrics, alert frequency, and notifications to handles all contributed meaningfully to the predictions, following the expected pattern as well.

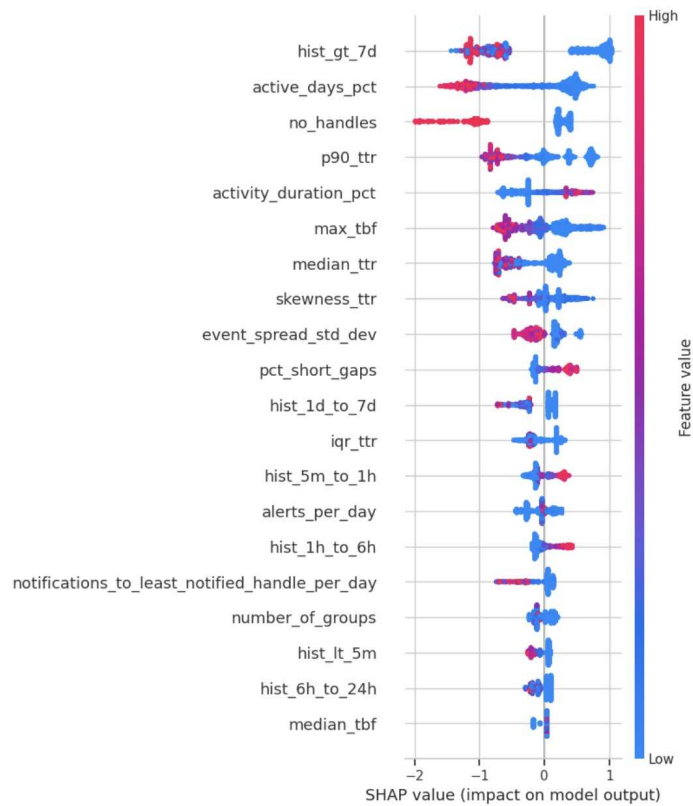


Figure 23 – Feature importance and contribution based on SHAP values.

We also incorporated a **final uncertainty adjustment layer** to account for monitors with limited activity data. For example, a monitor that has triggered only once

or has a very short observation window (e.g., 10 days) naturally carries higher uncertainty. This layer adjusts the final score by learning a correction based on the raw score, the number of alerts, and the session duration, effectively smoothing out unreliable predictions.

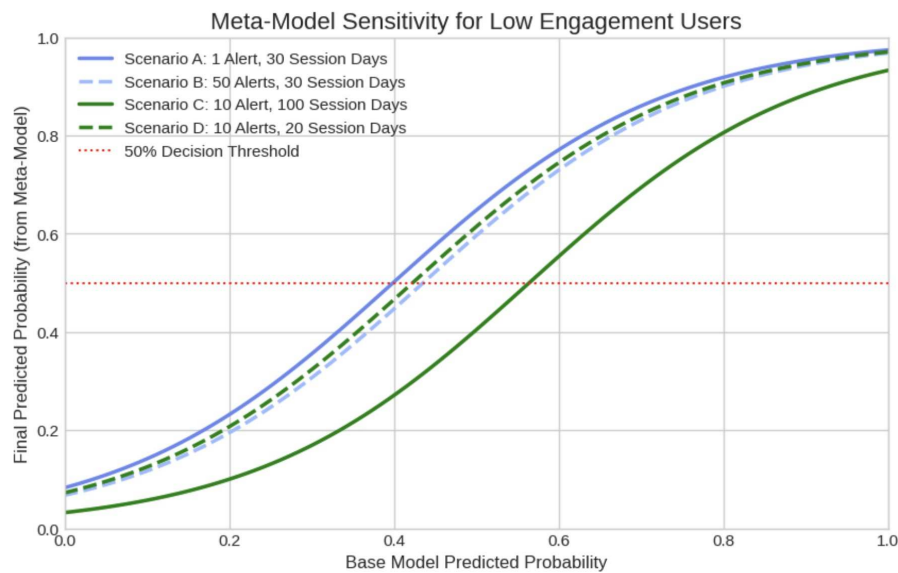


Figure 24 – Score after uncertainty layer, improving scores with lower session duration.

Finally, one of the project’s key requirements was **explainability**, providing interpretable reasons behind each score. To achieve this, we implemented a hybrid reasoning framework combining heuristics and SHAP-based explanations:

- For rule-based features (from $S_{critical}$), heuristic messages are triggered when specific thresholds are violated. For instance, if the *percentage of time in alert* score drops below 0.7, the model adds a reason such as “*too much time in alert*”.
- For metadata-related sigmoids, similar heuristics are defined—for example, missing tags or titles produce corresponding feedback messages.
- For the internal ML component (S_{events}), we use SHAP values to identify the top five most influential features contributing to a low score. Based on these, additional heuristics provide human-readable explanations. For example, if $p90_ttr$ is among the top drivers and its value exceeds 100, the reason might be “*alerts take too long to recover*”. Conversely, if $p90_ttr$ is extremely low (e.g., < 0.01), the reason might be “*monitor mostly auto-resolves*”.

This layered explainability mechanism ensures that every monitor score can be accompanied by interpretable and context-aware justifications, making the model’s outcomes actionable for users. Finally, below is the final distribution of the final scores for the unlabelled data.

4.2 QUALITATIVE ANALYSIS

While the quantitative metrics demonstrate promising performance, it was equally important to validate the model qualitatively through real-world feedback. To that end, two initiatives were launched:

1. **Internal Evaluation with Datadog Teams.** A spreadsheet was distributed to several internal teams at Datadog, listing their monitors along with the model-assigned quality scores and the corresponding explanations for low-scoring monitors. Team members were asked to review, confirm, or challenge the results. The feedback so far has been overwhelmingly positive—teams reported identifying monitors they were unaware of, which led to corrective actions such as updating, deactivating, or deleting obsolete monitors.

	A	B	C	D
1	monitor_id	service	final_score	reasons_bad_score
21	179979790	notification-intake	0.5086242142	['The monitor takes too long to recover', 'The monitor takes, on average, too many hours to resolve', 'An alert took too long to recover', 'Alerts occur too frequently']
22	145615917	notifications-backlog-processor	0.7930503065	
23	103188237	monitor-state-manager	0.9478438209	
24	119316972	alerting-state-notifications-shadow	0.3452982818	['Monitor stays constantly in alert', 'Alerts occur too frequently']
25	166014283		0.4675941429	['The monitor takes too long to recover', 'The monitor takes, on average, too many hours to resolve', 'An alert took too long to recover']
26	70340453	delancey-downtime	0.9330364722	
27	112650033	alerting-downtime-matcher-shadow	0.1592347619	['Monitor stays constantly in alert', 'The monitor takes too long to recover', 'Alerts occur too frequently', 'The monitor takes, on average, too many hours to resolve']
28	76014690	monitor-state-manager	0.0217070753	['Not enough handles', 'Monitor has a lot of failed notifications or isn't sending notifications', 'Alerts occur too frequently']
29	148464460	alerting-system-monitors	0.02226384793	['Not enough handles', 'Monitor has a lot of failed notifications or isn't sending notifications', 'Alerts occur too frequently']
30	106388576	alerting-state-notifications-shadow	0.800081852	
31	106388553	monitor-state-manager-shadow	0.8624911917	
32	179979762	notification-router	0.9129179273	
33	134386077		0.512238601	['Alerts occur too frequently']
34	105432594	alerting-downtime-matcher	0.926079915	
35	106388555	monitor-state-manager-shadow	0.8602104268	
36	106388574	alerting-state-notifications-shadow	0.02688562639	['Not enough handles', 'Monitor has a lot of failed notifications or isn't sending notifications']
37	145615912	notifications-backlog-processor	0.8742240315	

Figure 25 – Example of the Spreadsheet Used to Validate Results.

2. **Limited Production Deployment.** The model is currently being tested in a controlled production environment with a small subset of customers. This gradual rollout allows us to monitor the score's behavior under real operational conditions and gather feedback from users who directly interact with the scoring system. Insights from this phase will guide further refinements, such as adjusting heuristic rules, improving sigmoid calibration, and tuning the uncertainty layer.

Overall, both the quantitative and qualitative evaluations indicate that the hybrid model successfully balances interpretability and predictive accuracy, providing a solid foundation for a scalable monitor quality assessment system.

4.3 ANALYSIS OF ERRORS AND LIMITATIONS

While the overall results were satisfactory and the initial feedback from internal teams has been very positive, there are still several important limitations and improvement opportunities to address in future iterations of this project.

1. **Inactive Monitors.** The most significant limitation arises from the nature of Datadog’s monitor dataset: approximately 40% of existing monitors have not experienced any alert transition in the past year. Since the model relies on behavioral features such as alert transitions, resolution times, and frequency distributions, it cannot effectively score these inactive monitors. In such cases, the model would default to assigning a high score, which may not accurately reflect quality. A monitor that has remained in an “OK” state for an extended period could mean one of two things: either the monitored service has been consistently healthy, or the query or threshold is misconfigured—or, in some cases, the monitor has simply become inactive after a migration or organizational change. Addressing this limitation will require additional logic or metadata-based heuristics to distinguish these scenarios.
2. **Labeling Bias.** Although this model significantly reduces bias compared to the previous threshold-based scoring page, some bias remains, especially regarding label selection and definition. The current labeling process still depends on the initial set of heuristic and engagement-based labels, which might not capture all relevant quality dimensions. Expanding and refining the label set, perhaps by incorporating more user feedback and weakly supervised signals, would help mitigate this issue.
3. **Query Definition Gap.** Finally, the current version of the model does not evaluate the quality of the *monitor query* itself—the core logic defining what the monitor checks. As a result, the model can only assess quality based on monitor behavior (outputs and patterns) rather than on the correctness or robustness of the underlying query. Future work should explore ways to integrate query-level analysis, enabling the system to answer whether a monitor is not only behaving well but also designed appropriately.

4.4 ADDED VALUE FOR THE COMPANY

Despite these limitations, the project already provides substantial strategic and practical value for Datadog.

1. **Improved Visibility and User Insights.** The model enables Datadog to rank monitors from best to worst quality, helping users easily identify which monitors require attention and why. This transparency provides tangible value to customers by highlighting actionable issues (although the feature is not yet publicly released).
2. **Integration with the MCP Server.** We are developing an integration with the **Monitor Control Plane (MCP)** server, which will allow it to access monitor quality

scores and explanations. With this, the MCP will be able to suggest improvements, analyze problematic monitors, and assist users in optimizing their configurations—combining natural language interaction with data-driven insights.

3. **Foundation for Future Machine Learning Products.** The existence of a reliable monitor quality scoring model also enables the creation of new ML-based features on top of it. For instance:

- Automatic threshold tuning based on monitor quality and historical stability.
- Automatic monitor generation or recommendation for under-monitored services.
- Detection of monitoring gaps within customer setups.

4. **Operational Efficiency.** From a business perspective, the model's insights could lead to significant infrastructure benefits. Around 30% of monitors are currently classified as “bad,” and these tend to generate the majority of events and notifications. If users remove or fix these problematic monitors, Datadog could see a measurable reduction in unnecessary alert traffic, directly decreasing infrastructure load and operational costs.

5 CONCLUSION

5.1 SUMMARY OF ACHIEVEMENTS

During this internship, I developed and deployed a hybrid, interpretable machine learning model for scoring monitor quality at Datadog. The model combines deterministic rule-based scoring with a data-driven behavioral analysis, balancing explainability and predictive accuracy. The work resulted in a scalable, modular architecture that already provides actionable insights to internal teams and lays the groundwork for future ML-driven alerting systems.

5.2 PERSONAL AND PROFESSIONAL LEARNINGS

From a technical standpoint, this experience deepened my understanding of how machine learning operates in real-world production systems. I learned that model performance depends far more on **data quality, problem framing, and feature engineering** than on model complexity alone.

On a professional level, working at a large-scale technology company provided invaluable experience. It was my first time operating in such an environment—collaborating across teams, managing large datasets, and deploying ML models with tangible user impact. Living in New York also taught me adaptability and independence. Finally, working with Machine Learning in a completely different domain reinforced the importance of domain expertise in building meaningful AI systems.

5.3 ETHICAL CONSIDERATIONS

Applying machine learning to critical systems like monitoring involves ethical responsibility. A miscalibrated model could lead to the removal of an important monitor, potentially hiding service incidents and increasing business risk. Throughout this project, I emphasized transparency and explainability to minimize these risks. Rather than focusing solely on predictive performance, I aimed to understand *how* the model made its decisions and whether they aligned with human intuition. Future work should continue to prioritize interpretability and rigorous validation before deploying automated scoring at scale.

REFERENCES

DATADOG. **Dashboards and monitors at scale**. [S. l.: s. n.], 2024a. Available from: <https://www.datadoghq.com/blog/dashboards-monitors-at-scale/>. Visited on: 2024.

DATADOG. **Monitors**. [S. l.: s. n.], 2024b. Available from: <https://docs.datadoghq.com/monitors/>. Visited on: 2024.

ZHAO, Yunrui; XU, Qianqian; JIANG, Yangbangyan; WEN, Peisong; HUANG, Qingming. Dist-pu: Positive-unlabeled learning from a label distribution perspective, 2022.