

Using Recommendation Techniques to Reduce Cloud Monitoring Fatigue

Priyanka Mishra¹, Siddharth Shroff²

¹Senior Product Manager, Microsoft, California, USA

²Principal Product Manager, Amazon, California, USA

Abstract - Modern cloud ecosystems produce enormous volumes of telemetry-metrics, logs, and traces that commonly inundate engineering teams with noisy or low-priority alerts. In this paper, we discuss the use of recommender system concepts in cloud observability for filtering and prioritizing operational data selectively. Using techniques such as collaborative filtering and pattern finding, observability platforms can bring the most pertinent signals to the forefront, mitigate alert fatigue, and accelerate root cause analysis. The article defines the crossroads of monitoring and AI recommendation, proposing a practical strategy for technical leaders to build more responsive, human-centric, focused, and efficient operations workflows.

Key Words: Observability, Cloud Computing, Recommender System, Recommender Systems, Monitoring Automation, AI-driven Operations, Noise Reduction in Monitoring, Anomaly Detection, AIOps

1. INTRODUCTION

Cloud computing enables enormous scale with incredible flexibility, but it has also led to a deluge of monitoring data and alerts. However, these large-scale systems are complex and have many moving parts; even a small issue can quickly snowball into a bigger problem. Modern observability tools generate a firehose of telemetry, often triggering excessive alerts – many low-priority or false alarms, resulting in alert fatigue. This happens when engineers, bombarded with notifications, start tuning them out, much like ignoring car alarms that cry wolf too often. Consequently, important warnings can get missed [4]. Imagine a search system that starts serving completely irrelevant results because a data pipeline silently broke hours ago, but the alert for it was buried among dozens of less critical ones. As a result, incident response slows down while customers have a poor experience, and the team scrambles under pressure. Over time, this takes a toll, burnout becomes real, and trust in the alerting system erodes.

Meanwhile, in a completely different domain, recommender systems have evolved to tackle information overload for users. From e-commerce to streaming media,

recommendation algorithms sift through millions of options to highlight the few items most relevant to each person. They excel at filtering noise and prioritizing what matters to the end-user.

This raises an intriguing question: can we apply the same recommendation techniques to filter and prioritize observability data, thereby reducing cloud monitoring fatigue? This paper explores applying these principles from recommender systems to enhance cloud observability, aiming to reduce monitoring fatigue and sharpen operational focus for technical leaders.

1.1 What is Cloud Observability?

Cloud observability is the practice of understanding the internal state of systems by examining the telemetry (metrics, logs, traces, etc.) they produce. Unlike traditional monitoring which checks a few preset metrics, observability provides a comprehensive, proactive view into system behavior. It allows engineers to detect, diagnose, and understand issues efficiently. Observability is investigative and helps teams move beyond “Is the system up?” to “Why is the system behaving this way, and what does that mean for customers?”

There are three key inputs to a powerful observability system- (1) Metrics are numerical measurements (e.g. CPU utilization, request latency, error rates) that show how a system is performing over time. They’re great for spotting trends, setting alerts, and tracking service health. (2) Logs are detailed notes, timestamped records of events (e.g. an application error, warnings, or a user action) that help pinpoint exactly what happened, when, and why. (3) Traces track and reconstruct the end-to-end flow of a single request or transaction through distributed components, which is invaluable for debugging in microservice architectures. They help pinpoint where slowdowns or failures happen in complex, distributed applications.

Together, these telemetry streams give a rich, high-cardinality view of a cloud system’s operation so teams can quickly detect issues, understand root causes, and improve performance. Modern cloud observability

platforms aggregate and analyze this data in real time, presenting it via dashboards and alerts. The overarching aim is to maintain reliable, high-performance infrastructure and applications—engineers use observability data to ensure everything is running optimally, to rapidly troubleshoot incidents, and to continuously improve systems. In summary, cloud observability moves beyond basic monitoring by offering deep insights and context, enabling teams to understand why systems behave certain ways and the impact on customers.

1.2 What are Recommender systems?

Recommender systems (or RecSys) are machine learning and information filtering systems that predict what users would find relevant to their interests [3]. They intelligently sort through a large pool of content, highlighting the most pertinent items for each user. Recommender systems are omnipresent—for example, they drive song suggestions, product recommendations, social media feeds, and news article suggestions. These systems often use signals from user activity (clicks, views, purchases, ratings) combined with content metadata to make personalized recommendations. For example, Netflix's recommender system suggests movies and TV shows based on what you've watched before, ratings, and what similar users have enjoyed. If you watch a lot of action movies, it will recommend more films in that genre or similar shows you haven't seen yet.

Overall, recommendation systems not only personalize the experience but also, and maybe more importantly, reduce cognitive load and save time for the user.

The real-world impact of effective recommender systems has been striking. Netflix famously revealed that roughly 80% of the content watched on Netflix is influenced by their recommendation algorithms [2]. Their Chief Product Officer even estimated the recommender system to be worth over \$1 billion per year to the business, due to its effect on user retention and engagement. Another classic example is Pinterest's Pixie Recommender System. Pinterest developed a real-time recommender system called **Pixie** to personalize content for its users. By analyzing user interactions and the vast network of pins, Pixie delivers tailored recommendations that significantly boost user engagement. According to a study on Pixie [1], experiments showed that recommendations provided by Pixie lead to 50% higher user engagement when compared to the previous Hadoop-based production system. These examples demonstrate how powerful recommendation techniques have become a mainstay of technology businesses across entertainment, e-commerce, travel, and social media. By increasing engagement,

conversion, and user satisfaction, top-tier recommender systems confer a significant competitive advantage.

The success of these systems in taming information overload suggests their underlying principles hold significant potential for improving cloud observability.

2. What sparked the idea?

While seemingly disparate domains—one focused on infrastructure health, the other on user preferences—cloud observability and recommender systems share fundamental challenges and offer conceptual synergies upon closer examination. This realization sparked the core idea of this paper.

First, both areas grapple with large data sets at immense scale. Modern observability platforms ingest vast, high-velocity streams of metrics, logs, and traces, mirroring how recommender systems must process millions of user interactions and item attributes daily. Both must efficiently filter signals from noise in real-time. This shared challenge means techniques refined in one area, such as efficient real-time analytics or anomaly detection, often find relevance in the other. Furthermore, the rise of AI in operations (AIOps) directly parallels the machine learning-centric nature of modern RecSys. AIOps aims to evolve observability beyond mere data collection towards intelligent insight generation—detecting complex patterns, predicting incidents, and recommending actions—much like RecSys algorithms learn user preferences to guide choices.

Second, although their primary focus differs—observability traditionally centers on system health (CPU, latency, error rates), while RecSys prioritizes user experience (relevant content, engagement)—these goals are deeply intertwined. A seamless user experience demands both reliable infrastructure and relevant interactions. Poor system performance directly degrades the user experience, just as irrelevant recommendations frustrate users. Conversely, platforms offering both high reliability and precise personalization foster loyalty and drive business success. Thus, optimizing both infrastructure and experience through intelligent data handling is crucial for positive customer outcomes.

Most importantly, both fields confront a core problem of filtering and prioritization. Observability platforms must surface the few critical alerts from a potential flood of notifications to prevent operator fatigue; recommender systems must highlight the most relevant items from vast catalogs to capture user attention. Algorithmically, both involve ranking and filtering to reduce noise. This parallel is key: could algorithms designed for determining content relevance also excel at determining operational alert

relevance? For example, a recommendation engine might use user behavior patterns to rank content by relevance; similarly, an “intelligent monitoring” system could use past incident patterns to rank alerts by likely importance. Imagine a monitoring tool that says, “Incidents like this were often solved by checking metric X and log Y”—just like a recommender system. By applying RecSys principles to ops data, we can surface useful signals, reduce noise, and help engineers focus faster by learning from past patterns in large datasets.

3. Key Idea

The key idea is to combine observability principles with recommendation system techniques to focus engineering attention efficiently on both system health and user experience. This can mean two major approaches:

1. Apply recommendation-style machine learning to prioritize and reduce noise in observability data.
2. Apply observability methods to measure and improve the outcomes of the recommender system.

This paper focuses on the former: using recommendation logic to smooth out cloud observability. We also emphasize intuitive, flowing dashboards that guide users from high-level metrics to detailed diagnostics—a form of storytelling with data. This relates to recommendation principles by personalizing the view and prioritizing information flow based on context. For example, the top of a dashboard might show a customer outcome metric (“% of users clicking a recommended item today”) and an infrastructure metric (“API error rate”) side by side. If an anomaly is seen, subsequent graphs might break the data down by region or device type, and then further down by microservice or algorithm component. The goal is that an engineer can follow the “flow” to pinpoint where a problem lies, much as a user interface guides a user. This reduces fatigue by removing the need to manually correlate data from dozens of disparate charts – the dashboard does that heavy lifting in a cohesive way. In essence, we are applying user experience design to observability, making it easy to navigate the data.

Furthermore, we leverage proactive alerts and centralized logging, enhanced with recommendation logic. Instead of static thresholds, anomaly detection models identify deviations, predictive analytics forecast issues, and recommendation-based alerts could even personalize notifications based on the responder's role or context. For instance, a junior engineer might get more explanatory alerts versus a senior engineer getting terse, technical ones – an idea explored by recent research [4]. Centralized logging enables correlation, and we can envision log recommendation: suggesting related log entries or potential root causes based on historical incident patterns

(‘Users who investigated this error also looked at these logs...’), analogous to product recommendations.

Modern observability platforms do offer centralized solutions for consolidating historical data—such as logs, metrics, and traces—to aid in troubleshooting current issues by referencing past incidents. To make observability systems smarter, we can borrow techniques from recommendation algorithms to cut through the noise and surface what matters most. Just as recommenders filter content for users, observability platforms can learn from past incidents to highlight the most relevant signals—reducing alert fatigue and helping teams focus faster. This approach brings intelligence to traditional monitoring by using data-driven suggestions to guide operators. In the next section, we’ll explore how this can be implemented step by step.

4. Key Insights

With the key idea in mind, let’s explore potential opportunities and efficiency gains that can be realized by applying recommender system principles to observability:

Infrastructure Monitoring - System m Hygiene: Effective infrastructure monitoring is foundational. Beyond tracking standard system metrics (CPU, memory, etc.), next-generation monitoring should leverage AI, akin to recommendation engines, when onboarding new services. Based on workload type and dependencies, it could automatically recommend relevant metrics, baseline thresholds, and alerting logic. These recommendations should adapt over time, learning from performance patterns, surfacing anomalies, and prompting human review when significant deviations or changes (like a new, slower dependency) are detected, ensuring monitoring stays current and relevant.

Distributed Tracing- Enhanced Debugging: In a cloud-based recommender system, traffic might pass through an API gateway, caching layer, application service, database and, ML model service. Distributed tracing follows each transaction showing where time is spent or errors arise. Augmenting tracing with automated analysis recommends likely bottlenecks or flags anomalous spans based on historical performance, integrating seamlessly with alerting, highlighting narrow latency issues that might go unnoticed in aggregated metrics.

Logging- Aggregated Insights from Detailed Events: Logs are the finest-grained data we have, and in a complex system they can be overwhelming. Centralized logging is a must – collecting logs from all services (including the recommender system components, databases, message queues, etc.) into one searchable repository. Once centralized, we can perform analytics on logs to derive

insights. We might also aggregate error logs: if a particular error message (“Null pointer in RecommendationsService”) started appearing right before CTR dropped, that’s a clue. Modern logging systems apply ML for clustering or anomaly detection, effectively recommending groups of related log messages (‘Error X, Y, Z often appear together’). Some platforms offer log query recommendations (‘Users searching for ‘out of memory’ often also query ‘garbage collection logs’ from the same timeframe’). Ticket chatter, bugfix information, and other documentation automatically become part of historical record allowing it to be recommended alongside future similar alerts, accelerating troubleshooting. For eg- IBM Cloud Logs allows users to analyze historical observability data. By leveraging tools like IBM Watson Studio and Apache Spark, organizations can examine past logs and metrics to identify patterns, which is crucial for proactive issue resolution and capacity planning [5].

Alerting and Incident Management – Automation and Smart Triage:

Alerts must be actionable. Recommendation techniques can power smart routing and prioritization, learning from past incidents to suppress noise or aggregate related alerts. A recent study introduced a machine learning filtering mechanism that hides non-actionable alerts and only shows the important ones, achieving over 90% accuracy in reducing alert noise (espace2.etsmtl.ca). AI can group related alerts into single incidents, recommending a unified view. Automation can assist remediation by suggesting or triggering runbooks for known issues, potentially even recommending likely root causes based on historical data (‘Similar alert patterns previously correlated with database connection pool exhaustion’). Agentic AI workflows represent a sophisticated application of this: upon detecting an anomaly, an AI agent could analyze historical data and recommend specific diagnostic steps or telemetry views most relevant to that pattern, personalizing the initial response guidance, identifying (and executing) the best runbook for the situation at hand - reducing the need for human intervention to the most complex of scenarios. This approach can significantly cut down the operational burden on engineers, unlocking headspace for increased business value.

Each of these components builds on established observability practices but supercharges them with intelligent filtering, personalization, and data-driven guidance – the hallmarks of recommender systems. The result is a monitoring ecosystem that not only captures everything (to be fully observable) but also curates and presents information in a way that an operator can consume efficiently, very much like how a streaming service presents just a few shows out of thousands that you are likely to enjoy.

5. Risks & Challenges

While the approach is promising, implementing recommendation techniques in cloud monitoring is not without pitfalls. It’s important to recognize these hurdles:

Data Preparation and Quality: High-quality, well-curated data is the bedrock of effective machine learning. Observability data can be messy – logs have unstructured text, metrics may change meaning with new software versions, etc. Preparing this data for machine learning (e.g. labeling past alerts as “useful” or “noise” to train a model) is a significant challenge. Unlike product recommendations where user clicks provide implicit labels of preference, determining which alerts are truly important requires expertise and careful retrospective analysis. There is a risk of *garbage in, garbage out*: if we train on poorly curated incident data, the recommendations will be unreliable.

Cold Start and Evolving Systems: Recommender systems often struggle with the “cold start” problem (what to recommend when there’s no prior data for a new user or item). Similarly, a monitoring recommendation system faces a cold start when a new service launches or a new metric is introduced – the ML model has no history to learn from. Also, cloud environments evolve rapidly (new deployments, scaling events, architectural changes), causing patterns to shift. The models need to adapt quickly. This requires continuous learning or periodic retraining, which is complex to manage. If not handled, a previously effective alert filtering model might become outdated and start misclassifying alerts when the system’s behavior changes.

Accuracy: No ML system is infallible. An AI-driven filter might incorrectly flag benign events as critical (false positives) adding noise, or worse, suppress or deprioritize genuinely critical alerts (false negatives). In operational monitoring, missing a critical alert can lead to undetected outages, security breaches, or significant business impact. This risk might be mitigated through careful validation (e.g., running the AI filter in shadow mode initially) and often requires retaining human oversight or fallback mechanisms, ensuring critical alerts are never fully suppressed automatically, especially in sensitive domains like healthcare or aviation.

Balancing Engineer Trust against Skill Atrophy: Engineers are trained to trust their dashboards and alerts. Introducing AI-driven recommendations into an established DevOps/SRE workflow requires building trust. Gaining that trust requires the system to prove itself through transparency and explainability - engineers need to understand why the system made a particular

recommendation. Showing which signals contributed to a recommendation is necessary to overcome this hurdle. They should also be able to provide feedback and corrective signals when the model gets it wrong. Conversely, when engineers rely too heavily on AI, their own skills and situational awareness can atrophy over time. Over-reliance can reduce the healthy skepticism and investigative instinct that often catches weird edge-case issues, for example, if the system labels something as low priority, engineers might discount it even if their gut feeling says otherwise. Maintaining a balance where AI augments and enables, rather than replaces, human expertise is crucial. To counter this, some organizations encourage periodic drills without AI assistance to keep the team's manual skills sharp (much like pilots training without autopilot to maintain proficiency).

Bias and Feedback Loops: Algorithms learn from historical data, which can embed existing biases or suboptimal practices. For example, if certain alerts were historically ignored during off-hours simply due to staffing, the AI might learn to suppress them, reinforcing a potentially harmful pattern. This creates a feedback loop where the AI reinforces past behavior, not best practices. It can also lead to "filter bubble" effects (in recommender systems, discovery is a challenge), where the system focuses only on known failure modes derived from past data, potentially missing future novel failure modes ("unknown unknowns").

Tuning and Maintenance Complexity: The AI-driven observability system is itself a complex system that requires care. Defining appropriate thresholds for anomaly detection, choosing the right filtering models, and keeping them performant adds a new layer of operational responsibility and specialization required for SREs and platform teams. Smaller organizations might lack the data science expertise to tune these algorithms effectively. Bugs in this system could cause chaos – e.g. a bug could drop all alerts, or flood with alerts. The complexity increases the potential points of failure during incident response. Deploying third party vendor solutions can help, but adds vendor lock-in risks.

Domain Specificity: Rolling out AI models for monitoring is not without effort; performance heavily depends on the specific system context, as the critical signals for a telecommunications network differ vastly from an e-commerce site. Achieving accuracy often necessitates engineering domain-specific features (e.g., knowing 'checkout_errors' are crucial for retail) via iterative refinement involving both domain experts (SREs) and data scientists. This significant effort means the approach isn't plug-and-play and may be overly complex or costly for smaller systems where traditional monitoring suffices.

Cost, Performance, and Compliance: Implementing and running sophisticated ML models on high-volume telemetry data can be computationally expensive and may introduce latency into the monitoring pipeline. The cost (compute resources, development effort, potential vendor lock-in) must be justified by the benefits. For smaller systems with manageable alert volumes, simpler, well-tuned traditional monitoring might be more cost-effective and reliable. Furthermore, regulatory or compliance requirements in certain industries (e.g., finance, healthcare) might restrict the use of AI for alert filtering or demand meticulous logging of all events, making automated suppression inappropriate. Misapplying these techniques where they don't fit can lead to wasted resources or compliance violations.

While the promise of using recommendation techniques to combat monitoring fatigue is high, the practical implementation faces substantial risks and challenges related to data, algorithms, human factors, and cost. Overcoming these requires careful planning, incremental rollout, significant investment, and a commitment to human-in-the-loop systems. It's a journey demanding technical rigor and cultural adaptation. While there are real risks and scenarios where this approach is not appropriate, with careful implementation and human oversight many of these risks can be managed. The potential rewards—more focused engineers and resilient systems—are driving organizations to tackle these complexities, paving the way with real-world experiences that illuminate the path forward.

6. Real World Cases

Several organizations have begun adopting strategies that blend observability with intelligent, recommendation-like analytics. Here are a few notable examples and case studies:

New Relic (Recommended Alert Conditions): New Relic, a popular observability platform, introduced an Alert Condition Recommendation service in 2021. It uses AI/ML to analyze your applications and recommend which alert metrics and thresholds [6]. Essentially, it looks at your telemetry and suggests "alert candidates" that you might not have thought of – for example, it might notice a particular custom metric is erratic and recommend setting an alert on it, or propose a baseline for CPU usage alert based on historical peaks. This is akin to a recommender system for configuration: it learns from many New Relic customers and best practices, and gives tailored suggestions for your environment. By doing so, it helps teams cover monitoring blind spots and set up meaningful alerts without extensive manual tuning. It's an example of using knowledge from large datasets (all the apps instrumented in New Relic) to benefit individual users –

very much how a movie recommender uses preferences of millions to suggest to one user.

A real-world example of this strategy applied is- DeFacto, a global fashion retailer. Finding it hard to monitor their large and expensive infrastructure across multiple nations, DeFacto partnered with Bion Consulting to bring on New Relic's observability features. With New Relic's APM and using AI-driven alerting, they were able to achieve increased real-time monitoring visibility, reducing incident response times by 35%. Additionally, improved data ingestion methods meant there was an automatic 50% reduction in monitoring costs, while still having accurate and actionable intelligence [7].

Datadog (Watchdog and Anomaly Detection):

Datadog's platform includes an AI engine called Watchdog, which automatically detects anomalies, outliers, and even potential root causes in your metrics and logs [8]. It runs in the background and will surface a notification like "Metric XYZ has suddenly spiked and deviated from its baseline, here's the timeline." It also tries to correlate events - for instance, if an anomaly in error rate coincides with a new deployment, it will call that out. This is being used by many companies as a way to catch issues that slip through static thresholds. An e-commerce company using Datadog, for example, might be alerted by Watchdog of a subtle increase in checkout latency that wasn't caught by existing alerts, prompting investigation before it becomes a major outage. Datadog has effectively deployed large-scale time-series analysis (with algorithms like SARIMA and others) behind the scenes to implement this across their customer base [8]. Many see it as a step towards "self-driving" monitoring, where the system not only collects data but interprets it and informs you proactively - essentially recommending where to look.

Netdata (Intelligent Alert Filtering): Netdata, a monitoring software company, conducted research and prototyped a machine learning based alert filtering mechanism in a cloud environment. Their system learns from historical alerts and administrator actions to decide which alerts to hide and which to show, effectively personalizing the alert stream for usefulness. [5] Expert evaluation of this approach showed it could mitigate alert fatigue with over 90% accuracy in identifying irrelevant alerts [4]. In practice, this means a Netdata user could enable an "AI mode" where instead of seeing 100 alerts an hour, they might see the 5 that truly need attention, with the others quietly logged. This real-world test not only demonstrated the feasibility of reducing noise but also highlighted the importance of considering the user's experience level in tuning alerting (as they did - e.g., a novice admin might get fewer, more high-level alerts vs. an expert who gets detailed diagnostics).

These examples illustrate that the industry is indeed moving in this direction. Early adopters report that they see a reduction in noise and faster incident detection. However, they also emphasize that human oversight remains important. For instance, New Relic's recommendations are optional - it aids the user but doesn't enforce changes. Datadog's Watchdog alerts are informative but you still decide how to act on them. The general trend is augmenting the operators, not replacing them. Organizations that have embraced these tools often do so incrementally: they might start with anomaly detection on a few metrics, then expand to automated alert tuning, and so on, always measuring the results.

For companies considering a similar strategy, these cases show the potential payoffs: less burnout, faster resolution, and more confidence that important issues won't slip through the cracks. They also highlight the need to tailor the approach to one's context - what works at Netflix scale might be overkill for a smaller enterprise, but selective adoption (like just using ML to detect anomalies in business metrics) can still provide value. The common theme is leveraging data and patterns (often from lots of previous incidents or from many users' behavior) to make monitoring smarter.

7. CONCLUSIONS

While originating in different contexts, Cloud Observability and Recommender Systems share the challenge of extracting meaningful signals from vast data streams. By applying the filtering, prioritization, and personalization techniques perfected in recommender systems to observability data, we can significantly reduce monitoring fatigue and enable more proactive, intelligent operations. The approach we discussed is essentially about focusing human attention where it matters most: just as a good recommender system surfaces the most relevant content for a user, a good observability system augmented with recommendation techniques surfaces the most relevant insights for an operator. This leads to more efficient operations, quicker incident response, and ultimately more reliable services for customers.

For Technical leaders, embracing recommendation principles within observability is a strategic move towards reducing monitoring fatigue and sharpening operational focus. This approach promises more efficient operations, faster incident response, and ultimately, more reliable services that improve customer satisfaction. Success hinges on an incremental and people-centric strategy. Start with high-impact, manageable steps: pilot intelligent alert prioritization for a key service, automate anomaly detection on critical business metrics, or introduce log pattern recommendations to aid debugging. Leaders must

involve their engineers deeply from the outset—their domain knowledge is vital for training effective models and building trust. Demonstrate value quickly through measurable improvements and maintain transparency about how the underlying algorithms work.

As confidence grows with proven results, layer in more sophisticated capabilities like automated root cause suggestions or context-aware dashboard personalization. Rigorously measure the impact against clear goals (e.g., reduced actionable alert volume, faster MTTR). It is critical to monitor the recommendation system itself as part of the infrastructure, and always maintain human oversight with clear fallback procedures to ensure safety and control.

In conclusion, using recommendation techniques to reduce cloud monitoring fatigue represents a powerful idea to reduce information overload. It acknowledges that human operators shouldn't have to sift through mountains of data unaided, and that algorithms can shoulder some of that burden. At the same time, it keeps humans in the loop for judgment and oversight. When balanced correctly, this synergy can lead to systems that are both highly observable and intelligently managed. Companies that successfully leverage it will likely see not only technical benefits but also happier engineers and happier customers – a true strategic win in an increasingly complex digital landscape.

REFERENCES

- [1] Cornell University. (2017). *Pixie: A System for Recommending 3+ Billion Items to 200+ Million Users in Real-Time*. arXiv:1711.07601. <https://arxiv.org/abs/1711.07601>
- [2] Data Science W231. (2022). *Netflix Recommendations: Beyond the \$1 Million Challenge*. Berkeley ISchool Blogs. <https://blogs.ischool.berkeley.edu/w231/2022/07/page/3/>
- [3] Coralogix. *What Are Recommender Systems? Use Cases, Types, and Techniques*. Coralogix AI Blog. <https://coralogix.com/ai-blog/what-are-recommender-systems-use-cases-types-and-techniques/>
- [4] Leivadeas, A., Kibalas, M. R., Soldatos, J., & Anagnostopoulos, C. (2024). *Mitigating Alert Fatigue in Cloud Monitoring Systems: A Machine Learning Perspective*. École de technologie supérieure. <https://espace2.etsmtl.ca/id/eprint/28822/1/Leivadeas-A-2024-28822.pdf>
- [5] IBM. *IBM Cloud Logs: Gain deeper insights from your log data*. IBM Products Blog.

<https://www.ibm.com/products/blog/ibm-cloud-logs-observability>

[6] New Relic. (2021, September 21). *How to use machine learning to get recommended alerts*. New Relic Blog. <https://newrelic.com/blog/how-to-relic/machine-learning-recommended-alerts>

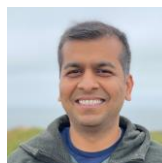
[7] Bion Consulting. *Cost-Effective Observability with New Relic: DeFacto Case Study*. Bion Consulting Case Studies. <https://www.bionconsulting.com/case-studies/cost-effective-observability-with-new-relic>

[8] Datadog. (2023, November 7). *Detect emerging issues earlier with anomaly detection in Datadog AIOps*. Datadog Blog. <https://www.datadoghq.com/blog/early-anomaly-detection-datadog-aiops/>

BIOGRAPHIES



Priyanka Mishra
Product Leader, Azure
Observability, Microsoft
California, USA



Siddharth Shroff
Product Leader, Amazon
California, USA