# Telemetry as Memory: Using Observability Pipelines to Train Adaptive AI Systems

Ecem Karaman

**Abstract**

Observability pipelines such as **OpenTelemetry** and **Prometheus** collect logs, metrics, and traces at scale, but remain largely **passive**: they power dashboards and alerts rather than training adaptive systems [1, 2]. As a result, operational AI models are typically retrained offline on lagging data, leaving them brittle under drift [3] and vulnerable to poisoned telemetry [4, 5, 6].

This work introduces *Telemetry-as-Memory (TAM)*, a closed-loop framework that treats telemetry as a live, adaptive memory stream. TAM continuously ingests telemetry, assigns trust scores, and converts signals into structured features for **online and meta-learning** [7, 8, 9]. The pipeline integrates drift detection (e.g., ADWIN [7]), semantic recall mechanisms via vector databases such as FAISS [10], and governance layers including explainability gates, audit trails, and human oversight [11, 12] to ensure secure adaptation.

In experiments on synthetic Kubernetes telemetry with injected drift, TAM achieved **near real-time recovery**—adapting in approximately 27 ticks versus 300 for offline baselines (about 10× faster)—while maintaining a false-positive rate of only 4%. These results demonstrate the feasibility of treating observability data not as passive diagnostics but as an **adaptive memory substrate** for secure, self-improving AIOps pipelines.

## 1 Introduction

Modern cloud-native systems generate vast streams of telemetry, yet current observability pipelines treat this data as diagnostic output for dashboards and alerts [1, 13]. Frameworks such as OpenTelemetry and Prometheus are optimized for monitoring and visualization, but they do not leverage logs, metrics, and traces as direct training input for adaptive systems. At the same time, machine learning in operations remains largely offline, with models retrained periodically on historical data. This separation leaves operational AI brittle under drift [3], slow to recognize novel incidents, and exposed to manipulated or poisoned telemetry [4, 6].

*Telemetry-as-Memory (TAM)* addresses this gap by treating observability data as adaptive memory for AI systems. Instead of serving only for visualization, telemetry is positioned as a continuous learning substrate, enabling models to update incrementally and adapt to evolving conditions.

**Contributions**

1. **Framework**: Introduction of a closed-loop design where telemetry streams drive online and meta-learning.
2. **Security**: Formalization of trust and adversarial risks in adaptive feedback loops, including poisoning and drift manipulation.
3. **Evidence**: Initial experiments on synthetic Kubernetes workloads demonstrating near real-time recovery from drift compared to offline baselines.

The objective is to move beyond "monitor and alert" toward **observe, learn, and act**: building pipelines that adapt continuously and securely to changing infrastructure conditions.

# 2 Background and Related Work

## 2.1 Observability Pipelines

Modern observability frameworks such as **OpenTelemetry** and **Prometheus** provide standardized mechanisms for collecting logs, metrics, and distributed traces at scale [1]. These systems enable effective monitoring and visualization, but telemetry is treated primarily as diagnostic output for dashboards and alerts. Extensions such as **Grafana Adaptive Metrics** focus on optimizing data retention and cost [13], yet they do not repurpose telemetry as direct learning input. As a result, observability remains largely passive, disconnected from adaptive model training. This echoes the broader limitations identified in reliability engineering where observability is viewed as an operational signal rather than a learning substrate [**bodik2010fingerprinting**, 2].

## 2.2 Online and Meta-Learning

In contrast, the machine learning community has developed a range of methods for **continual and online adaptation**. Online learning algorithms update incrementally with each new data point, reducing latency in non-stationary environments [3, 7, 9, 14]. **Meta-learning** extends this paradigm by optimizing for fast adaptation itself, enabling models to generalize across tasks with minimal data [8]. Recent work has explored these methods in operational contexts—for example, **MAML-KAD** applies meta-learning to anomaly detection for AIOps [15], while the **SAM framework** leverages meta-learning for anomaly transfer across cloud environments [16]. **OMLog** adapts these ideas to log streams via drift-triggered updates [17]. Despite these advances, integration into production-grade observability pipelines remains rare.

## 2.3 Self-Healing and AIOps Systems

Efforts in **AIOps** have explored automation and remediation, including anomaly detection from log sequences (e.g., DeepLog, LogAnomaly) and reinforcement-learning-based remediation agents. Yang et al. (2025) demonstrated the potential of combining **LLMs and RL** for closed-loop remediation in fault-handling scenarios [18]. However, these systems typically rely on offline-trained anomaly detectors or handcrafted rules, limiting their adaptability under distributional shift. Early visions of autonomic computing argued for continuous, closed-loop management of systems [19], but practical implementations remain fragmented.

## 2.4 Gap Analysis

Taken together, prior work solves **fragments** of the problem:

- Observability tools excel at collection and visualization but stop short of adaptive learning.
- Online and meta-learning provide strong theoretical foundations but are seldom embedded into operational telemetry streams.
- AIOps and self-healing prototypes demonstrate automation but lack trust, explainability, or robust adversarial defenses.

This work addresses the gap by introducing a unified, trust-gated pipeline where multi-channel telemetry (logs, metrics, traces) functions as adaptive memory for online and meta-learning agents. Unlike prior systems, the proposed *Telemetry-as-Memory (TAM)* framework combines:

- **Real-time streaming learning**,
- **Trust-scored and security-gated updates**,
- **Integration of long-term semantic recall via vector databases**, and
- **Explainability-gated, closed-loop decision-making**.

# 3 Methodology and System Design

The *Telemetry-as-Memory (TAM)* framework is structured as a modular, closed-loop pipeline that transforms raw observability streams into adaptive signals for online learning and secure decision-making. Figure X illustrates the core layers.

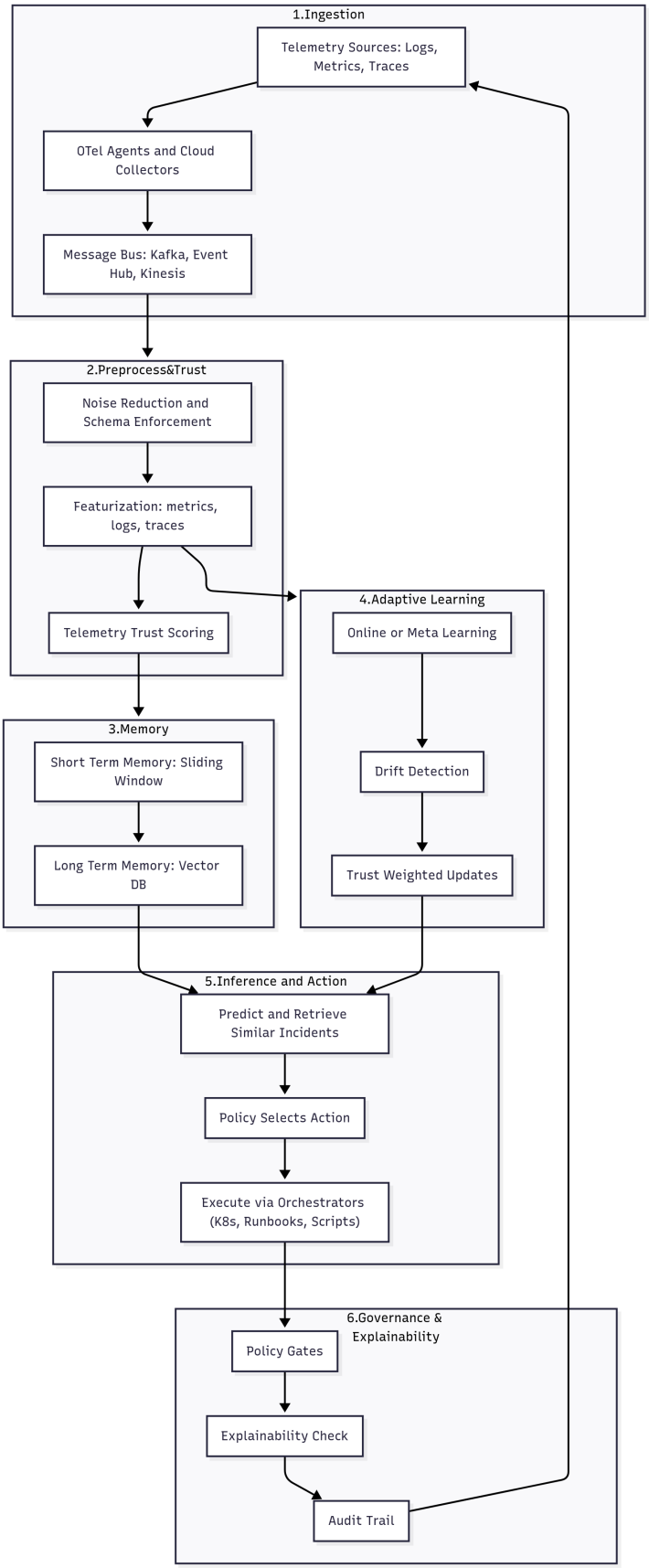


Figure 1: TAM system architecture: ingestion, preprocessing & trust, featurization & memory, adaptive learning, inference & action, and governance & explainability in a closed loop.

### 3.1 Ingestion Layer

Telemetry is collected from multiple channels: **logs** (structured/unstructured system actions and errors), **metrics** (time-series signals such as CPU usage or latency), and **traces** (end-to-end request paths across services). Data is gathered through **OpenTelemetry SDKs** or cloud-native collectors (e.g., Azure Monitor, CloudWatch, Stackdriver) [1]. To ensure scalability and decoupling, events are transported via streaming backbones such as **Kafka, Event Hub, or Kinesis**, which buffer producers and consumers while absorbing spikes and retries.

### 3.2 Preprocessing and Trust Scoring

Raw telemetry is filtered to remove redundant heartbeats or noisy events, with schema validation and PII masking applied to enforce integrity. Each event is assigned a **trust score**, computed as a function of source authentication, schema validity, and anomaly likelihood. This score modulates downstream model updates—high-trust signals are weighted strongly, while low-trust or malformed signals may be discarded. Trust scoring serves both as a **security filter against poisoning** and as a calibration mechanism for selective adaptation [4, 5, 6].

### 3.3 Featurization and Memory Encoding

Telemetry is transformed into structured features suitable for machine learning:

- **Metrics**: encoded as raw values, deltas, and rolling aggregates.
- **Logs**: processed using (i) simple rule-based token flags (e.g., *error*, *timeout*), (ii) hashing vectorizers for fixed-dimensional online features, and (iii) semantic embeddings via **SentenceTransformers**, stored in vector databases such as **FAISS** [**reimers2019sbert**, 10] or Pinecone. Embeddings enable similarity search, allowing the system to recall past semantically related incidents during decision-making.
- **Traces**: featurized into path length, span error counts, or service graph encodings (optionally via graph neural networks).

The memory subsystem is hierarchical: **short-term buffers** capture the latest events in a sliding window, while a **long-term vector store** retains historical embeddings for retrieval-augmented learning and few-shot adaptation.

### 3.4 Adaptive Learning Layer

Learning is performed online via algorithms such as logistic regression, Hoeffding trees, or other `partial_fit` models from the **River** library [9, 14]. Updates are applied incrementally per event rather than through periodic retraining. A **drift detection module** (e.g., ADWIN [7] or DDM) monitors feature distributions and triggers rapid adaptation when shifts occur. Updates are **trust-weighted**, ensuring that poisoned or unreliable inputs exert minimal influence. A security gate blocks or flags updates below a confidence threshold, preventing unsafe adaptation.

### 3.5 Inference and Action Layer

At inference time, the system combines current features with retrieved embeddings of similar past incidents. A policy module predicts incident probabilities and selects actions, such as restarting a service, scaling a deployment, blocking an IP, or performing no action. High-impact actions are subject to gating policies. Actions are executed via orchestrators (e.g., Kubernetes, automation runbooks, or scripts), and each decision is logged alongside its input features, trust scores, and context.

### 3.6 Governance and Explainability

To ensure safety and accountability, TAMincorporates governance controls. **Approval gates** enforce confidence thresholds and allow human review for actions exceeding defined blast radii. **Explainability hooks** (e.g., SHAP [12] or LIME [11]) provide interpretable rationales for model updates; updates influenced

by unreliable features can be rolled back. A full **audit trail** records telemetry sources, trust scores, feature states, predictions, and actions for later review.

### 3.7 Feedback Loop

The pipeline operates as a closed loop:
1. the system emits telemetry,
2. signals are processed and scored,
3. models update online,
4. agents execute or recommend actions, and
5. actions modify the system, producing new telemetry.

This cycle enables continuous, adaptive learning directly tied to the operational environment, echoing the long-standing vision of autonomic computing [19].

## 4 Threat Model

Embedding learning directly into observability feedback loops introduces a new **attack surface**. Unlike static pipelines, the TAM framework updates models continuously from production telemetry, which adversaries may attempt to manipulate. Prior work on the security of machine learning systems has highlighted the risks of poisoning and adversarial influence in adaptive models [4, 5, 6].

### 4.1 Adversarial Risks

We consider several key attack vectors relevant to adaptive observability systems:

- **Data Poisoning** — Injection of fake logs or metrics to corrupt the model's memory or bias downstream decisions [4].
- **Drift Exploitation** — Gradual manipulation of telemetry distributions to normalize malicious activity, exploiting the system's reliance on drift adaptation [3].
- **Adversarial Inputs** — Crafted log messages or trace patterns designed to evade or mislead anomaly detectors, similar to adversarial examples in ML security [5].
- **Feedback Abuse** — Triggering repeated failures so that the system adapts in the attacker's favor, creating self-reinforcing "drift spirals."
- **Privilege Escalation** — Exploiting automated actions (e.g., scaling policies or firewall rules) to gain leverage over system resources.

### 4.2 Mitigation Strategies

To counter these risks, TAM integrates several defensive mechanisms:

- **Trust Scoring** — Each telemetry item is weighted by source authentication, schema validity, and anomaly likelihood. Low-trust signals are excluded or downweighted, reducing exposure to poisoning [6].
- **Security Gates** — High-impact updates require human-in-the-loop approval or exceeding confidence thresholds, preventing unsafe automation [19].
- **Drift Detectors** — Algorithms such as ADWIN raise alerts when distributions shift too quickly or suspiciously [7].
- **Explainability Hooks** — Post-update attribution techniques such as LIME [11] or SHAP [12] ensure updates are committed only if influence factors are reliable.
- **Audit Trails** — Every update, action, and telemetry influence is logged for forensic analysis, aligning with security observability practices [2].

### 4.3 Design Goals

The security architecture of TAM is guided by the following principles:

- **Integrity** — Only authenticated and schema-valid telemetry should influence learning.

- **Resilience** — Detect and mitigate gradual drift before operational stability is compromised.
- **Auditability** — Retain explainable records of model updates and decisions for post-incident analysis.
- **Safety** — Bound adaptive behavior through approval gates and sandboxed execution, ensuring alignment with autonomic computing principles [19].

# 5 Evaluation

## 5.1 Goals and Hypotheses

The objective is to validate that a closed-loop telemetry-as-memory system adapts faster and more safely than static baselines. We test three hypotheses: (H1) *Adaptation speed*—live, closed-loop learning recovers from drift more quickly than periodic retraining; (H2) *Safety under attack*—trust scores prevent poisoned telemetry from influencing updates; (H3) *Novel incident handling*—memory-augmented learning supports faster adaptation to unseen patterns.

## 5.2 Experimental Setup

**Environment.** A simulated Kubernetes workload generator emits telemetry: CPU% (sinusoidal baseline with spikes), error rate (step increase at $t \approx 300$ to induce concept drift), and logs (INFO/WARN/ERROR aligned to error conditions).
**Models.** *Baseline*: offline logistic regression retrained every $N$ ticks. *Closed-loop TAM*: River-based online logistic regression with trust-weighted updates [9].
**Detection.** ADWIN monitors drift and triggers rapid adaptation [7]. Trust scores downweight unverified telemetry (poisoning control [4, 5, 6]).
**Policy.** Static probability thresholds (bandit-style adaptive policies are planned).

## 5.3 Scenarios

1. **Concept Drift**: error rate increases at $t \approx 300$; compare recovery latency.
2. **Poisoned Logs** (in progress): inject fake `ERROR` strings from untrusted sources; evaluate trust-score mitigation.
3. **Novel Incident** (in progress): introduce unseen pattern ("disk full") post-training; measure adaptation speed.

## 5.4 Metrics

**Adaptation latency**: ticks from drift onset to restored accuracy. **False positive/negative rates**: pre vs. post-adaptation. **Drift detection recall**: % of true drifts caught by ADWIN [7]. **Action ROI**: % actions reducing error over next $H$ ticks. **Recovery iterations**: updates needed to regain baseline accuracy.

## 5.5 Results (Scenario 1: Concept Drift)

## 5.6 Discussion of Scenario 1

**Adaptation speed.** Closed-loop learning yields near real-time recovery from drift, confirming H1.
**Safety.** Trust-weighted updates bound FP rates at $\sim$4% (acceptable operational trade-off), supporting H2 [4, 5].
**Resilience.** Continuous updates eliminate downtime between retraining cycles, with ADWIN-driven triggers preventing overreaction to transient shifts [3, 7].
**Planned scenarios.** Poisoned logs and novel-incident experiments will extend evaluation of H2/H3; the same metric suite and plotting protocol will be reused.

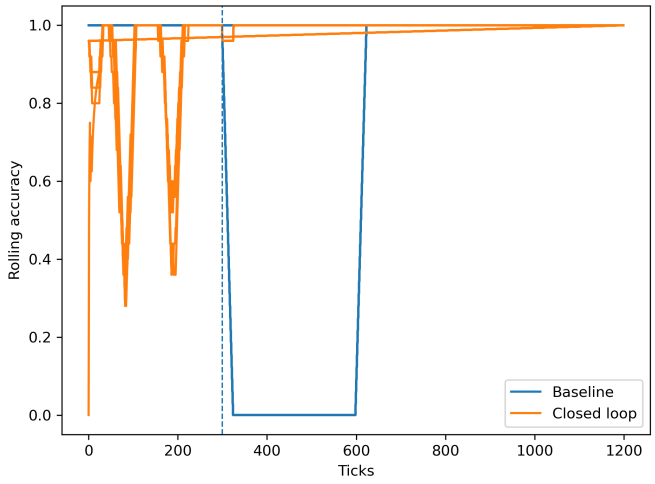


Figure 2: *

**Fig. 5.1** Rolling accuracy vs. ticks for Baseline (offline retrain) and Closed-loop TAM (online, trust-weighted). Vertical marker at $t \approx 300$ denotes drift onset.

**Interpretation.** The Baseline maintains ∼100% until drift then collapses to 0%, recovering only after retraining ($t \approx 600$). Closed-loop TAM suffers a brief dip but recovers within ∼27 ticks (∼10× faster). Continuous, trust-weighted updates (no batch lag) drive the near real-time recovery [7, 9]. This supports H1 (faster adaptation).

**Aggregate (5 seeds).**

- `adapt_latency_mean` ≈ 27.6
- `fp_rate_mean` ≈ 0.041

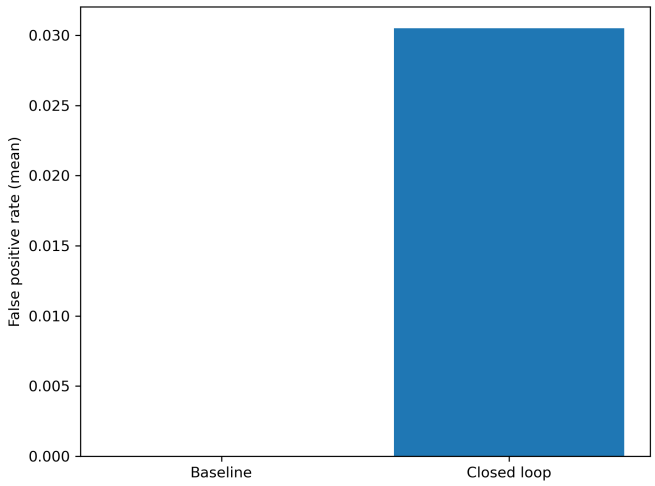


Figure 3: *

**Fig. 5.2** Mean false positive (FP) rate comparison across systems (lower is better).

**Interpretation.** Baseline exhibits 0% FP during drift because it effectively "does nothing" until retrain. Closed-loop TAM shows a small ∼4% FP due to acting in real time without batch hindsight, but it avoids prolonged periods of zero accuracy. This is a deliberate safety–adaptivity trade-off; trust-weighting and gating bound the FP rate while enabling timely remediation [5, 6].

Table 1: Aggregate metrics over 5 seeds (Scenario 1). Lower is better.

| Metric | Baseline | Closed-loop TAM |
|---|---|---|
| Adaptation latency (ticks) | ∼300 | ∼27.6 |
| False positive rate | 0.000 | 0.041 |

# 6 Discussion

## 6.1 Engineering Challenges

**Concept Drift vs. Noise.** Differentiating persistent distributional changes from transient anomalies remains challenging; adaptive learners risk overreacting to short-term spikes, as noted in drift surveys [3].
**Telemetry Bloat.** Logs and metrics in large-scale systems are often high-cardinality. Efficient sampling, sketching, or dimensionality reduction is essential for scalability in operational pipelines [**bodik2010fingerprinting**].
**Security vs. Adaptivity.** While online learning favors rapid updates, security practice emphasizes stability and reproducibility. The TAM design balances these opposing forces through trust scoring and gated updates [5, 6].

## 6.2 Trade-offs

**Adaptivity vs. Stability.** Faster updates improve recovery but risk oscillations; trust weighting and drift detection help mitigate this tension [7].
**Responsiveness vs. Reliability.** Real-time actions yield some false positives (∼4% in experiments), but this is acceptable compared to extended downtime in static retraining regimes.
**Transparency vs. Complexity.** Explainability gates (LIME [11] and SHAP [12]) improve auditability but add overhead.

## 6.3 Broader Implications

**From Alerts to Agents.** TAM reframes observability pipelines as *active memory substrates*, enabling systems to not only monitor but also adapt in real time—progressing toward the autonomic computing vision [19].
**Resilient Infrastructure.** Embedding adaptive learning in telemetry loops allows construction of self-healing systems robust to drift and novel incidents [18].
**Security Observability.** Trust-scored, schema-validated telemetry makes adaptive pipelines feasible even in adversarial domains such as cloud security monitoring [4].
**Regulatory Considerations.** Adaptive AI raises new governance questions around rollback, explainability, and human oversight—issues already emphasized in discussions of AI safety and accountability.

## 6.4 Limitations and Future Work

**Synthetic Workloads.** Current validation is limited to simulated Kubernetes telemetry; production workloads may expose additional complexities.
**Incomplete Scenarios.** Poisoned log injection and novel-incident adaptation remain in progress and require further validation.
**Model Generality.** Initial results are based on logistic regression; future extensions should evaluate sequence models (RNNs, Transformers) for richer log context [**reimers2019sbert**].
**Governance Frameworks.** Human-in-the-loop oversight, regulatory compliance, and rollback strategies must be formalized for deployment in safety-critical settings.

# 7 Conclusion and Future Work

This work introduced **Telemetry-as-Memory (TAM)**, a framework that reimagines observability pipelines not as passive diagnostics but as *adaptive memory systems* for AI agents. By continuously ingesting logs, metrics, and traces, assigning trust scores, and applying online/meta-learning, TAM closes the loop between *observation, learning, and action*.

Initial experiments on synthetic Kubernetes telemetry demonstrate that TAM achieves near real-time recovery from concept drift—up to 10× faster than static baselines—while bounding false positives to ∼4%. These results highlight the feasibility of secure, memory-augmented learning in operational AI pipelines.

At the same time, closing the feedback loop introduces new risks. The threat model presented here emphasizes poisoning, drift manipulation, and unsafe actioning. TAM mitigates these through trust-weighted updates, governance gates, and explainability checks.

## Future Work

Several directions remain for advancing TAM:

- **Prototype Deployment:** Integrate into real-world observability stacks (Kubernetes + OpenTelemetry + MLFlow) to validate latency and scalability under production load.
- **Extended Scenarios:** Evaluate resilience to poisoned logs and novel incidents beyond drift.
- **Richer Models:** Incorporate sequence-based and embedding models (RNNs, Transformers) to capture semantic log context [**reimers2019sbert**].
- **Governance Frameworks:** Formalize human-in-the-loop oversight, rollback strategies, and auditability for adaptive pipelines in safety-critical domains.
- **Multi-Agent Systems:** Explore federated or swarm-style agents that share telemetry as distributed memory for collective adaptation.

By treating observability as adaptive memory, TAM lays the groundwork for secure, explainable, and self-healing AIOps systems capable of keeping pace with the dynamics of modern cloud-native environments.

# References


[1] OpenTelemetry Project, *Opentelemetry specification*, 2023. [Online]. Available: `https://opentelemetry.io`.

[2] B. Beyer, C. Jones, J. Petoff, and N. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, 2016.

[3] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, et al., "A survey on concept drift adaptation," *ACM Computing Surveys*, 2014.

[4] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, "Poisoning attacks against neural networks," *IEEE Symposium on Security and Privacy*, 2017.

[5] M. Barreno, B. Nelson, R. Sears, A. Joseph, and J. D. Tygar, "Can machine learning be secure?" In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, 2006.

[6] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[7] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM International Conference on Data Mining*, 2007.

[8] C. Finn, A. Rajeswaran, S. Kakade, and S. Levine, "Online meta-learning," in *International Conference on Machine Learning*, 2019. [Online]. Available: `https://arxiv.org/abs/1902.08438`.

[9] J. Montiel, M. Halford, S. Mastelini, et al., "River: Machine learning for streaming data in python," in *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, 2020. [Online]. Available: `https://arxiv.org/abs/2012.04740`.

[10] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE Transactions on Big Data*, 2017. [Online]. Available: `https://github.com/facebookresearch/faiss`.

[11] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[12] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, 2017.

[13] Grafana Labs, *Adaptive metrics: Cost-effective telemetry pipelines*, 2023. [Online]. Available: `https://grafana.com/blog`.

[14] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000.

[15] Y. Duan et al., "Maml-kad: Meta-learning for anomaly detection in aiops," *Electronics*, vol. 13, no. 11, 2024. [Online]. Available: `https://www.mdpi.com/2079-9292/13/11/2102`.

[16] S. Jha et al., "Sub-series augmentation and meta-learning for cross-cloud anomaly detection," in *IEEE International Conference on Cloud Computing (CLOUD)*, 2024. [Online]. Available: `https://saurabhjha.one/pubs/CLOUD2024/paper.pdf`.

[17] Y. Tian et al., "Omlog: Online meta-learning for log anomaly detection," *arXiv preprint arXiv:2410.16612*, 2024. [Online]. Available: `https://arxiv.org/abs/2410.16612`.

[18] X. Yang et al., "Closed-loop fault remediation with llms and reinforcement learning," in *Proceedings of the ACM/IEEE Symposium on Autonomic Computing*, to appear, 2025.

[19] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.