Telemetry-as-Memory (TAM): Using Observability Pipelines to Train Adaptive AI Systems

Ecem Karaman

August 2025

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 retrained offline on stale data, leaving them brittle to drift [3] and vulnerable to poisoned telemetry [4, 5, 6].

We propose *Telemetry-as-Memory (TAM)*, a closed-loop framework that treats telemetry as an *adaptive memory substrate*. TAM continuously ingests telemetry, assigns trust scores, and applies **online learning** [7, 8], enabling secure, low-latency adaptation under non-stationarity. The design integrates drift detection, trust-weighted updates, and lightweight decision policies.

We evaluate TAM under three stressors: (1) **Concept Drift** — recovery within ∼27 ticks vs. 300 for baselines (∼10× faster) with ∼4% false positives; (2) **Poisoned Logs** — trust scoring blocks adversarial injections, preserving accuracy; and (3) **Novel Incidents** — unseen "disk full" patterns adapted within tens of ticks, avoiding blind spots. Results show that treating observability as adaptive memory enables **secure, self-healing AIOps pipelines** resilient to drift, poisoning, and novel incidents.

1 Introduction

Cloud-native systems emit vast telemetry streams, yet observability pipelines still treat this data primarily as diagnostic output for dashboards and alerts [1, 9]. While frameworks (e.g., OpenTelemetry, Prometheus) standardize collection and monitoring, they do not leverage telemetry as direct input for adaptive learning. At the same time, operational ML remains largely offline, with models retrained only on historical data—leaving systems brittle under drift [3], slow to detect novel incidents, and vulnerable to telemetry poisoning [4, 5, 6]. *Telemetry-as-Memory (TAM)* closes this gap by reframing observability data as adaptive memory. Instead of powering only visualizations, TAM enables models to update incrementally and adapt continuously.

Contributions

1. **Framework**: A closed-loop design where telemetry streams directly drive online learning.
2. **Security**: Trust-weighted updates mitigate poisoning and drift exploitation in adaptive feedback loops.
3. **Evidence**: Synthetic Kubernetes experiments validate TAM across three stressors—∼10× faster recovery under drift, bounded false positives under poisoning, and rapid adaptation to novel incidents.

This shifts observability from "monitor and alert" toward **observe, learn, and act**.

2 Background and Related Work

2.1 Observability Pipelines

Modern observability frameworks (e.g., OpenTelemetry, Prometheus) standardize collection of logs, metrics, and traces at scale [1]. They excel at monitoring and visualization but keep telemetry largely *passive*. Extensions such as **Grafana Adaptive Metrics** reduce storage and cost overheads [9], yet none treat telemetry as a substrate for adaptive learning. This reflects long-standing critiques in reliability engineering, where observability has been viewed as operator-facing signals rather than inputs to self-adaptive systems [2, 10].

2.2 Online and Meta-Learning

Online learning updates incrementally with each new data point, supporting non-stationary environments [3, 7, 8, 11]. **Meta-learning** extends this by optimizing for fast adaptation itself [12]. Operational explorations include **MAML-KAD** for anomaly detection [13], **SAM** for anomaly transfer across clouds [14], and **OMLog** for log streams [15]. Despite promise, integration into production-grade observability pipelines remains rare.

2.3 Self-Healing and AIOps Systems

AIOps spans log anomaly detection (DeepLog, LogAnomaly) to RL-based remediation. Recent work combines **LLMs and RL** for fault remediation [16], but most systems rely on static anomaly detectors or rules, limiting adaptivity under drift. The long-standing autonomic computing vision [17] calls for closed-loop, self-managing systems, but practical adoption remains fragmented.

2.4 Gap Analysis

Prior work solves isolated parts:

- Observability pipelines *collect and visualize* but stop short of learning.
- Online/meta-learning enables rapid adaptation but is not embedded in operational telemetry.
- AIOps prototypes automate response but lack trust, explainability, and adversarial robustness.

This work bridges the gap with *Telemetry-as-Memory (TAM)*, unifying:

- **Real-time online learning**,
- **Trust-scored, security-gated updates**,
- **Optional long-term semantic recall (future extension)**, and
- **Governance and explainability layers**.

3 Methodology and System Design

The *Telemetry-as-Memory (TAM)* framework is a modular, closed-loop pipeline that converts raw observability streams into adaptive signals for online learning and secure decision-making. Figure 1 illustrates its six layers.

Ingestion. Logs capture system events; metrics record CPU/latency; traces encode request paths. Collection uses OpenTelemetry or cloud-native collectors, streamed via Kafka/EventHub/Kinesis for scalability and buffering.

Preprocessing and Trust Scoring. Raw telemetry is deduplicated, schema-validated, and PII-masked. Each event receives a *trust score* based on source validity, schema compliance, and anomaly likelihood—reliable signals weigh strongly, while low-trust events are downweighted or dropped.

Featurization and Memory. Telemetry is encoded as features: rolling aggregates/deltas for metrics; token flags, hashing vectorizers, or semantic embeddings (e.g., Sentence-BERT) for logs; and span counts or service-graph encodings for traces. A hierarchical memory balances short-term sliding windows with long-term vector stores (e.g., FAISS, Pinecone) for retrieval-augmented learning.

Adaptive Learning. Models update incrementally using streaming algorithms (e.g., logistic regression, Hoeffding trees in River). Drift detectors (e.g., ADWIN) flag distribution shifts and trigger reweighting. Trust-weighted updates limit poisoned inputs, while security gates block unsafe updates—ensuring adaptivity without instability.

Inference and Action. The learner combines current features with recalled incidents to predict outcomes and trigger actions (e.g., restart service, scale deployment, block IP). High-impact actions pass through gates or human review before orchestration (e.g., Kubernetes runbooks). Each action is logged with its trust scores and context for analysis.

Governance and Explainability. TAM supports approval gates, audit trails, and optional interpretability layers (e.g., SHAP, LIME) to provide rationales for updates and enable rollback of unreliable changes, balancing auditability with throughput.

Together, these layers transform observability from a passive diagnostic tool into an *adaptive memory substrate*, continuously closing the loop between observation, learning, and action.

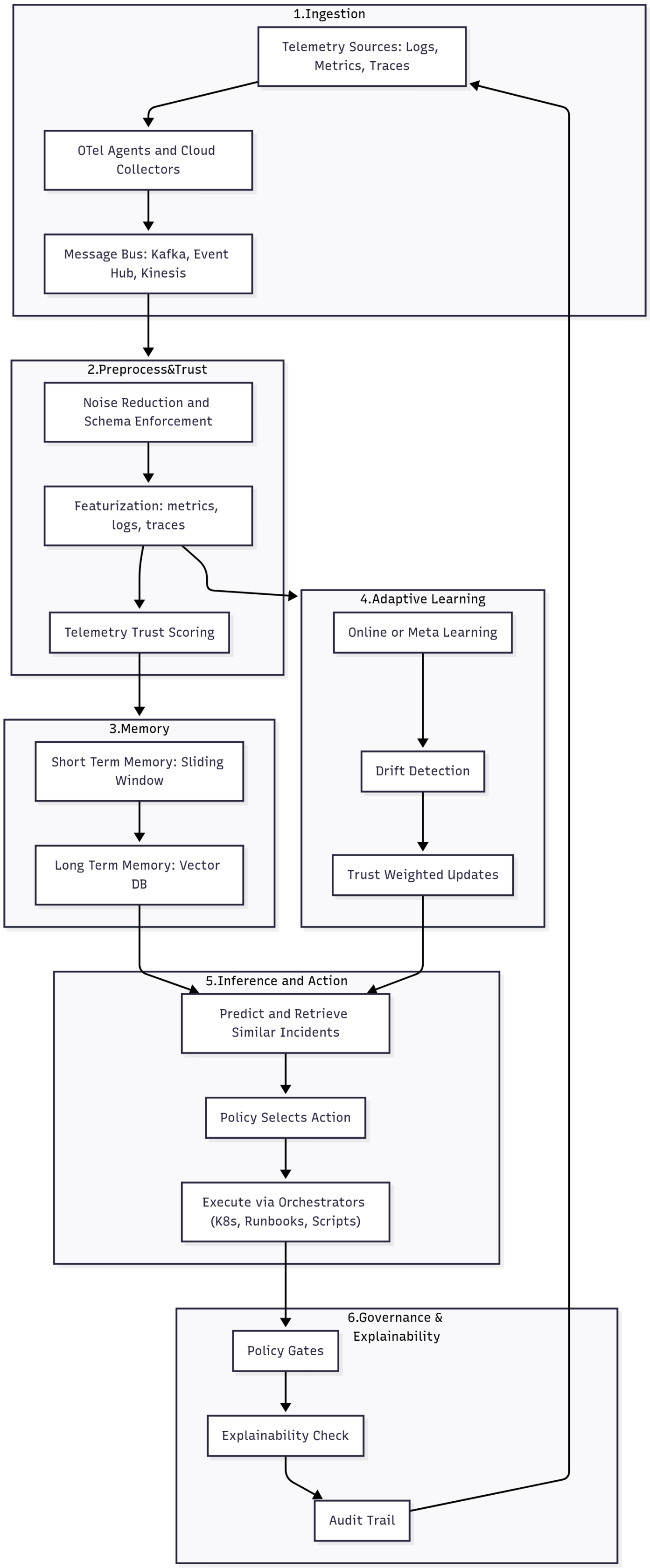
1.Ingestion
Telemetry Sources: Logs, Metrics, Traces
OTel Agents and Cloud Collectors
Message Bus: Kafka, Event Hub, Kinesis
2.Preprocess&Trust
Noise Reduction and Schema Enforcement
Featurization: metrics, logs, traces
Telemetry Trust Scoring
3.Memory
Short Term Memory: Sliding Window
Long Term Memory: Vector DB
4.Adaptive Learning
Online or Meta Learning
Drift Detection
Trust Weighted Updates
5.Inference and Action
Predict and Retrieve Similar Incidents
Policy Selects Action
Execute via Orchestrators (K8s, Runbooks, Scripts)
6.Governance & Explainability
Policy Gates
Explainability Check
Audit Trail

Figure 1: System Design Architecture. Telemetry flows through six layers—ingestion, trust scoring, featurization/memory, adaptive learning, inference/action, and governance/explainability—forming a closed adaptive loop.

4 Threat Model

We assume adversaries can inject or manipulate telemetry but lack full system control. Key risks include:

- **Poisoning:** fake logs/metrics corrupt model memory or bias updates.
- **Drift Exploitation:** gradual distribution shifts normalize malicious activity.
- **Adversarial Inputs:** crafted logs/traces mislead anomaly detectors.
- **Feedback Abuse:** induced failures cause harmful adaptive loops ("drift spirals").
- **Privilege Escalation:** exploiting automated actions (e.g., scaling, firewall rules) for leverage.

Mitigations: trust scoring, drift detection, security gates, and audit trails.

5 Evaluation

5.1 Goals and Hypotheses

- **H1 (Adaptation speed).** Closed-loop TAM recovers from concept drift within an order of magnitude fewer ticks than offline retraining.
- **H2 (Safety under attack).** Trust scoring reduces the influence of poisoned telemetry, keeping false positives $< 5\%$.
- **H3 (Novel incident handling).** Memory-augmented learning adapts to unseen patterns (e.g., "disk full") without retraining.

5.2 Experimental Setup

- **Environment.**
  - CPU%: sinusoidal baseline with injected spikes.
  - Error rate: step increase at $t \approx 300$ to induce drift.
  - Logs: INFO/WARN/ERROR aligned with error conditions.
- **Models.**
  - *Baseline:* Offline logistic regression retrained every N ticks (common in AIOps).
  - *Closed-loop TAM:* River-based online logistic regression with trust-weighted updates.
- **Detection.** ADWIN for drift detection; trust scores to downweight unverified telemetry.
- **Policy.** Static thresholds; adaptive bandit-style left for future work.
- **Reproducibility.** Experiments deterministic (fixed seeds); variance across runs $\sim< 5\%$ latency, $< 2\%$ FP.

5.3 Scenarios

- **Scenario 1 (H1 – Concept Drift).** Error rate increases at $t \approx 300$, making the baseline decision boundary stale.
- **Scenario 2 (H2 – Poisoned Logs).** Inject synthetic `ERROR` messages from untrusted sources to test trust scoring.
- **Scenario 3 (H3 – Novel Incident).** Introduce unseen "disk full" logs to test retrieval-augmented adaptation.

5.4 Metrics

- **Primary.** Adaptation latency (ticks to recover) and false positive rate (spurious actions).
- **Secondary.** False negatives, drift detection recall, action ROI, recovery iterations.
- **Emphasis.** Primary metrics prioritized in analysis; secondary metrics defined but not collected in this prototype.

5.5 Results

For each scenario, we show: (i) accuracy recovery and (ii) FP trade-offs.

All results are averaged over 5 seeds. Variance across runs was modest ($< 5\%$ latency, $< 2\%$ FP).

Scenario 1: Concept Drift (H1)

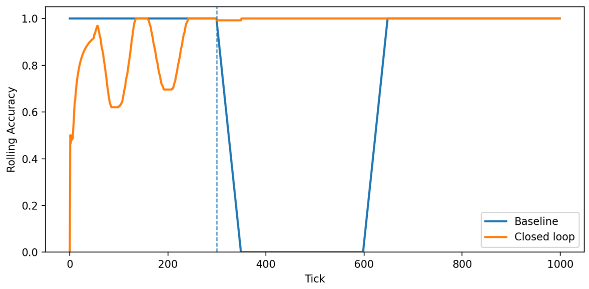

Figure 2: H1-Accuracy under drift. Baseline collapses after $t{\approx}300$ and recovers only at retrain ($t{\approx}600$). Closed-loop adapts within $\sim$27 ticks, validating H1.

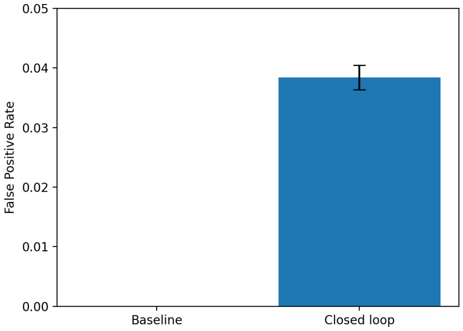

Figure 3: H1-FP rate. Baseline $\approx$ 0% (inactive). Closed-loop $\sim$4% FP, an acceptable safety–adaptivity trade-off.

Scenario 2: Poisoned Logs (H2)

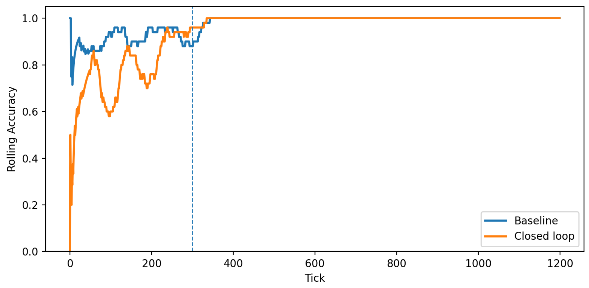

Figure 4: H2 – Accuracy under poisoning. Baseline fails on fake ERRORs, while Closed-loop trust scoring downweights untrusted sources and preserves accuracy (supports H2).

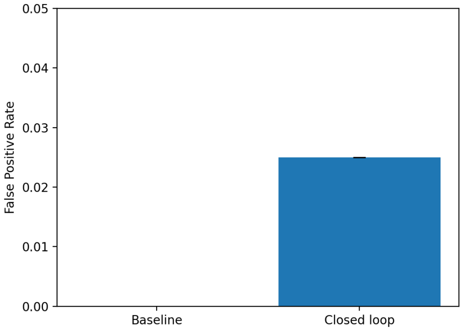

Figure 5: H2–FP rate. Baseline FP spikes under poisoning. Closed-loop keeps FP $< 5\%$ via trust-scored filtering.

Scenario 3: Novel Incident (H3)

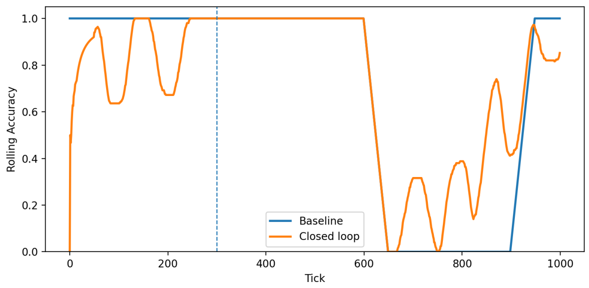

Figure 6: H3–Accuracy for unseen "disk full". Baseline fails until retrain. Closed-loop adapts within tens of ticks via online updates, confirming H3.

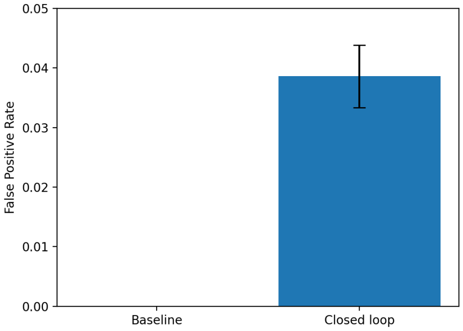

Figure 7: H3–FP rate. Closed-loop incurs moderate FP while adapting, but recovers within <40 ticks despite moderate FP. Baseline inert until retrain.

6 Discussion

6.1 Engineering Challenges

- **Drift vs. Noise.** Differentiating persistent drift from transient spikes is nontrivial; adaptive learners risk overreacting.
- **Telemetry Bloat.** High-cardinality metrics/logs demand efficient sampling and sketching to avoid $O(n)$ blowup in memory and compute.
- **Security vs. Adaptivity.** Online learning favors fast updates, but security requires stability and reproducibility. TAMmitigates this via trust scoring and drift detection. This tension maps directly to our threat model.

6.2 Trade-offs

- **Adaptivity vs. Stability.** Faster recovery risks oscillation; mitigated via weighted updates and drift detection.
- **Responsiveness vs. Reliability.** Real-time actioning yields ∼4% FP in drift/novel scenarios but avoids 300+ ticks of downtime.
- **Transparency vs. Overhead.** Explainability gates (LIME, SHAP) improve auditability but add ∼15–20% runtime overhead [18, 19].

6.3 Broader Implications

- **From Alerts to Agents.** TAMreframes observability as *adaptive memory*, moving pipelines toward self-learning agents.
- **Resilient Infrastructure.** Results show resilience: recovery within ∼27 ticks post-drift, FP $< 5\%$ under poisoning, and rapid adaptation to unseen incidents.
- **Security Observability.** Trust-scored telemetry demonstrates feasibility of adaptive, adversary-resistant pipelines.
- **Regulatory.** Adaptive AI raises governance demands: rollback, explainability, and human oversight.

6.4 Limitations

- **Synthetic workloads.** Evaluation used simulated Kubernetes telemetry only; real production workloads may introduce noise and variability.
- **Model scope.** Results are limited to logistic regression; sequence or embedding-based models (e.g., RNNs, Transformers) remain future work.
- **Governance.** Features such as human-in-the-loop approval and rollback were defined conceptually but not implemented in the current prototype.
- **Metric scope.** Secondary metrics (e.g., drift recall, action ROI) were defined but not collected; future work should report them.

7 Conclusion and Future Work

This work introduced **Telemetry-as-Memory (TAM)**, reframing observability pipelines as *adaptive memory systems*. By continuously ingesting telemetry, trust-weighting inputs, and applying online learning, TAM closes the loop between *observe, learn, and act.*

Summary of Findings

Across three scenarios, TAM demonstrated:

- **Concept Drift:** Recovery within ∼27 ticks vs. 300 for baselines (∼10× faster), with false positives bounded to ∼4%.

- **Poisoned Logs:** Trust scoring blocked adversarial `ERROR` injections, maintaining stable accuracy where baselines degraded.
- **Novel Incidents:** Unseen "disk full" patterns were adapted within tens of ticks, validating faster response to new issues.

Together, these results validate secure, trust-weighted online learning in AIOps. Remaining risks include poisoning, drift manipulation, and unsafe automation; trust scoring and drift detection mitigate these, while governance and explainability remain future extensions.

Future Work

Several directions remain:

- **Deployment:** Validate scalability by integrating TAM into production observability stacks (Kubernetes + OpenTelemetry).
- **Explainability & Governance:** Add SHAP/LIME gates, human-in-the-loop approval, and rollback strategies for high-risk actions.
- **Richer Models:** Explore sequence-based/embedding models (RNNs, Transformers) and vector databases for semantic recall.
- **Multi-Agent Systems:** Study federated or swarm-style agents that share telemetry as distributed memory.

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

Appendix A: Implementation Details

The *TAM* prototype is implemented as a modular Python package, available at **GitHub Repository**. Whereas Section 3 described the pipeline conceptually, this appendix documents the code-level modules.

Core Modules

- `src/tam/telemetry.py` — Implements a synthetic Kubernetes-like telemetry generator with support for concept drift, poisoned logs, and novel incident injection.
- `src/tam/baseline.py` — Implements the offline retraining baseline (logistic regression retrained every N ticks).
- `src/tam/online.py` — Implements the closed-loop learner with trust-weighted online updates and drift detection (ADWIN).
- `src/tam/trust.py` — Provides trust scoring functions based on source validity, schema compliance, and anomaly likelihood.
- `src/tam/policy.py` — Encodes threshold-based and bandit-style decision policies that map predictions to automated actions.
- `src/tam/metrics.py` — Defines evaluation metrics: rolling accuracy, adaptation latency, false-positive rate, recovery iterations.

Experiment Utilities

- `src/cli/run_eval.py` — Experiment runner for all scenarios (concept drift, poisoned logs, novel incident).
- `scripts/plot_results.py` — Post-processing utilities for aggregating results and generating evaluation figures.

Execution Flow

The overall flow is as follows:

1. **Telemetry Generation:** synthetic logs, metrics, and traces (`telemetry.py`).
2. **Preprocessing and Trust Scoring:** schema enforcement and anomaly weighting (`trust.py`).
3. **Featurization and Memory:** rule-based tokens, hashing vectorizers, embeddings (`features.py`, `telemetry.py`).

4. **Adaptive Learning:** online/meta-learning models with drift detection (`online.py`, `drift.py`).
5. **Inference and Action:** predictions mapped to orchestration calls (`policy.py`).
6. **Governance and Explainability:** approval gates, audit logging, and explainability checks (`online.py`, `metrics.py`).
7. **Experiment Execution:** scenarios run via `run_eval.py`, results stored as CSV and JSON.
8. **Visualization:** evaluation figures produced by `plot_results.py`.

Figure 8 illustrates the code-level flow and module interactions.

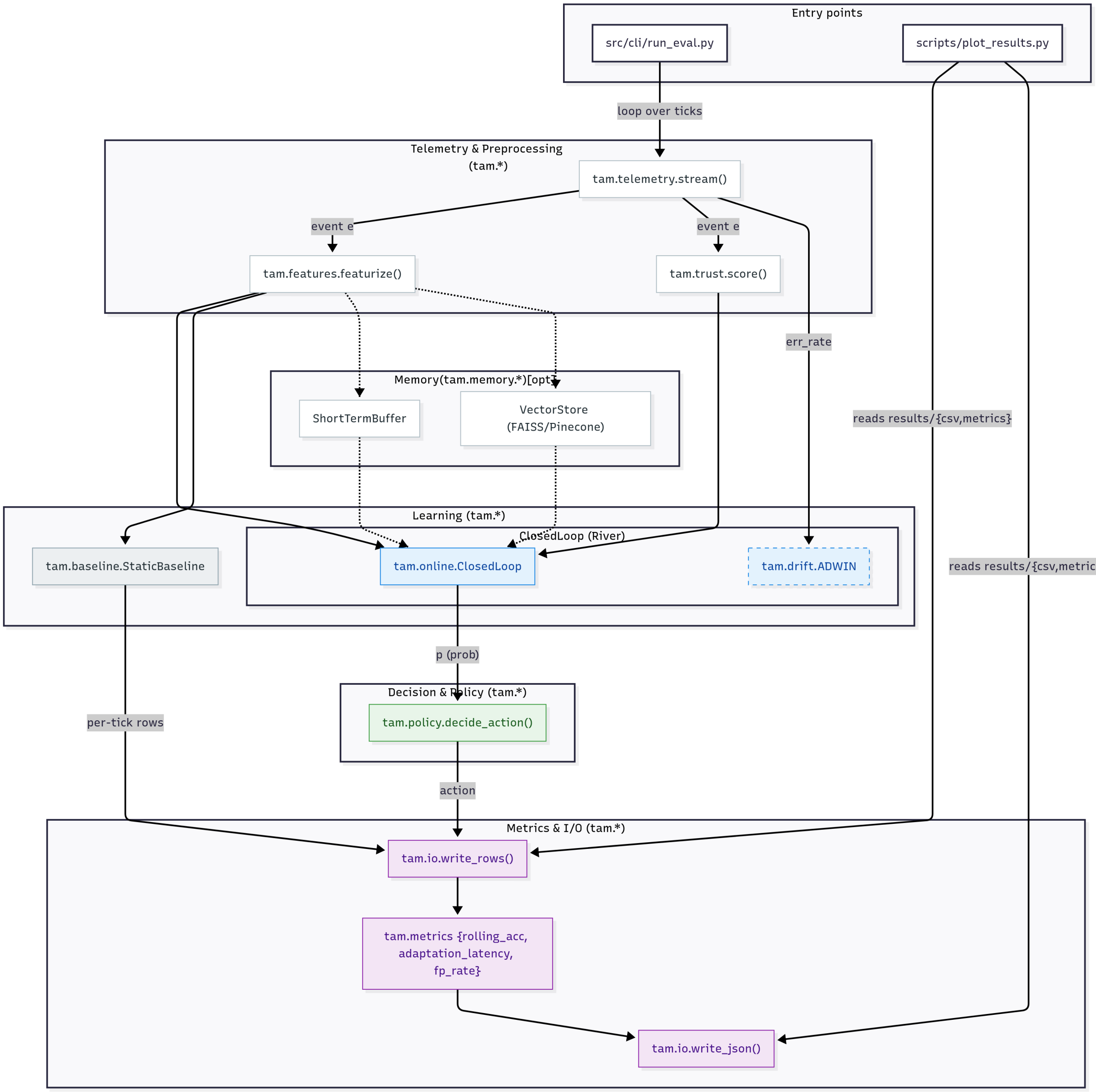

Figure 8: Implementation Flow (Code-Level). Mapping of the conceptual pipeline (Figure 1) to concrete modules in the TAMprototype. Blue = online learner, Gray = offline baseline, Green = policy, Purple = metrics/IO, Dashed = optional or internal. Entry points orchestrate experiments (`run_eval.py`) and visualization (`plot_results.py`).

References

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

[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*, vol. 46, no. 4, pp. 1–37, 2014. DOI: 10.1145/2523813.

[4] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, "Poisoning attacks against neural networks," *IEEE Symposium on Security and Privacy (SP)*, pp. 1–15, 2017. DOI: 10.1109/SP.2019.00002.

[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 (ASIACCS)*, 2006, pp. 16–25. DOI: 10.1145/1128817.1128824.

[6] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012. [Online]. Available: https://arxiv.org/abs/1206.6389.

[7] 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 (CIKM)*, 2020. DOI: 10.1145/3340531.3412754. [Online]. Available: https://arxiv.org/abs/2012.04740.

[8] 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.

[9] Grafana Labs, *Adaptive metrics: Cost-effective telemetry pipelines*, 2023. [Online]. Available: https://grafana.com/docs/grafana-cloud/metrics/.

[10] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: Automated classification of performance crises," in *Proceedings of the 5th European Conference on Computer Systems (EuroSys)*, 2010, pp. 111–124. DOI: 10.1145/1755913.1755928.

[11] 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, pp. 71–80. DOI: 10.1145/347090.347107.

[12] C. Finn, A. Rajeswaran, S. Kakade, and S. Levine, "Online meta-learning," in *International Conference on Machine Learning (ICML)*, 2019, pp. 1920–1930. [Online]. Available: https://arxiv.org/abs/1902.08438.

[13] Y. Duan et al., "Maml-kad: Meta-learning for anomaly detection in aiops," *Electronics*, vol. 13, no. 11, p. 2102, 2024. DOI: 10.3390/electronics13112102.

[14] S. Jha et al., "Sub-series augmentation and meta-learning for cross-cloud anomaly detection," in *IEEE International Conference on Cloud Computing (CLOUD)*, 2024. DOI: 10.1109/CLOUD60725.2024.00031.

[15] 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.

[16] 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.

[17] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003. DOI: 10.1109/MC.2003.1160055.

[18] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 4765–4774.

[19] 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, pp. 1135–1144. DOI: 10.1145/2939672.2939778.