A Budget-Based Frugal Observability Governor for Decentralized Edge Agents: Design, Implementation, and Comparison

Qiao Zhang[1]

Orange Innovation, 1 Rue de Broglie, Belfort, France
qiao.zhang@orange.com

Abstract. Autonomous agents deployed on resource-constrained edge devices need observability — the ability to inspect their behaviour from outside — but conventional full-trace logging and industrial telemetry stacks such as OpenTelemetry impose prohibitive bandwidth and memory costs. We propose a *Frugal Observability Governor*, a lightweight middleware for the `uAgents` multi-agent framework that replaces always-on logging with a three-rule, budget-based policy: log anomalies unconditionally, log state changes within a per-minute budget, and suppress everything else. A *Threshold* variant self-calibrates via a rolling z-score detector, and a *Generic* extension adds format-agnosticism through Python runtime introspection and an optional on-device LLM-as-Judge. We additionally demonstrate a *LangGraph*-based agent that replaces the custom governor with a graph-orchestrated LLM pipeline using `ChatOllama`, producing enriched analysis reports. An eight-method evaluation over five controlled trials on a commodity laptop shows that the Threshold governor achieves 85.4% bandwidth reduction and 75.2% mean anomaly recall (F1 85.2%) at 85 MB RAM, while the OTel SDK generates 49% more data than verbose logging. All methods achieve 100% precision (zero false positives). All code is open-source; physical validation on Raspberry Pi hardware is left as future work.

Keywords: Edge computing · IoT · Web of Things · Agent observability · Frugal AI · Anomaly detection · Decentralized Agents · LLM

1 Introduction

The proliferation of IoT deployments drives the need for autonomous agents operating under tight resource constraints at the network edge. The W3C Web of Things (WoT) standard [11] provides a framework for heterogeneous IoT device integration, and `uAgents` [20] enables lightweight, decentralized agent communication without centralized orchestration.

Observability — inspecting agent behaviour from outside — is essential for governance and safety, yet standard stacks target cloud workloads: the OTel Collector uses 512 MB RAM and a full CPU core [19] and generates substantially more traffic than lightweight alternatives [18]. We follow a *frugal* design

principle [1]: minimise resource consumption (bandwidth, memory, energy) while preserving the essential information needed for safe operation. Concretely, we implement a governor middleware for `uAgents` and evaluate it using synthetic sensor streams on a commodity laptop; physical deployment on Raspberry Pi hardware is future work.
Our contributions are:

- A *Frugal Governor* middleware for `uAgents` with a formal three-rule decision policy aligned with frugal AI principles [1].
- A *Threshold* extension using a self-calibrating rolling z-score detector, and a *GenericFrugalGovernor* with introspection-based field discovery and on-device LLM-as-Judge (Section 3).
- A *LangGraph*-based agent variant demonstrating LangChain ecosystem integration (Section 3.6).
- An eight-method evaluation across two message schemas (Section 5).

2 Background and Related Work

2.1 The uAgents Framework

`uAgents` [20] is an open-source Python framework for decentralized autonomous agents on P2P infrastructure. Several multi-agent frameworks exist, but most are ill-suited for resource-constrained edge deployments. JADE [2] requires the JVM (>300 MB baseline); SPADE [17] needs an external XMPP broker; Mesa [12] is a single-process simulation framework without inter-device communication; and LLM-centric orchestrators such as AutoGen [21] assume cloud-grade resources (150–200+ MB RAM).

`uAgents` stands out on four criteria relevant to edge IoT: *(i)* a measured import footprint of only 62 MB RAM (Python 3.12), fitting within a Raspberry Pi 4's 2 GB budget; *(ii)* fully decentralized HTTP-based P2P communication, requiring no external broker or orchestrator; *(iii)* typed message schemas via Pydantic, with automatic SHA-256 digest verification ensuring schema compatibility across distributed agents; *(iv)* optional LLM integration via `litellm` and a LangChain/LangGraph adapter [13], without imposing LLM dependencies on lightweight agents.

Agents are defined as Python objects with annotation-based hooks: `@on_interval` registers a function to be called periodically (e.g., every second for sensor readings), `@on_message` registers a handler for incoming typed messages, and `ctx.send()` transmits a message to another agent by address.

The framework includes no built-in observability mechanism — the gap our work addresses. Our governor produces observability records that map onto W3C WoT [11] Event affordances, ensuring compatibility.

2.2 Observability, Sampling, and Agent Governance

OpenTelemetry [16] is the Cloud Native Computing Foundation standard for distributed tracing, metrics, and logging. Bhuian [5] identifies protocol incompatibilities in hybrid edge environments; Splunk's sizing guide [19] confirms the

Collector uses 512 MB RAM by default. We quantify SDK-level overhead in Section 5. On the sampling side, Giouroukis et al. [9] survey adaptive sampling for IoT streams, Giordano et al. [8] propose an energy-aware FSM modulating sensing frequency, Bensaid and Molhem [3] apply first-derivative change detection in cyber-physical systems, and Cook et al. [6] survey anomaly detection for IoT, including z-score methods — the basis of our Threshold variant. All address *what sensor measurements to collect*, not *what observability records agents emit* — our contribution operates at a strictly higher abstraction layer. Guardrails AI [10] validates LLM outputs in cloud environments but ignores bandwidth/memory constraints. To our knowledge no prior work proposes a *budget-constrained observability governor* at the agent middleware layer for edge deployments.

3 Governor Architecture

3.1 System Overview

Figure 1 illustrates the complete system. A *Sensor Agent* — a `uAgents` software process generating synthetic readings at 1 Hz — transmits messages to an *Actuator Agent* via `uAgents` P2P. The Frugal Governor operates as a *middleware* inside the sensor agent: it intercepts every message *before* the call to `ctx.send()`, evaluates whether the reading warrants a log entry, and either emits a JSONL record or suppresses it. This pattern requires no modification to the uAgents framework itself. The Generic Extension (teal, dashed in figure) adds a two-stage pipeline for format-agnostic operation.

3.2 Message Schemas

Schema 1 — SensorReading: `temperature` (T, °C), `humidity` (H, %), `timestamp`, and `status` (`"normal"|"anomaly"`). **Schema 2 — PowerMeterReading:** `voltage` (V), `current` (A), `power_factor`, `frequency` (Hz), `timestamp`, and `meter_id`. This schema shares no field names with Schema 1; a schema-specific governor would require a complete rewrite to handle it. The generic governor handles both *identically* via Python runtime introspection (see Section 3.5).

3.3 The Three-Rule Decision Policy

For each message m the governor evaluates rules in strict priority:

Rule 1 — Anomaly priority. If anomalous (see below), emit unconditionally, ignoring budget.

Rule 2 — State-change budget. Compute the semantic fingerprint $\phi(m)$ (SHA-256 of sorted, rounded field values, truncated to 8 hex chars). If $\phi(m) \neq \phi_{\text{prev}}$ and $B < B_{\max}$, emit and increment B.

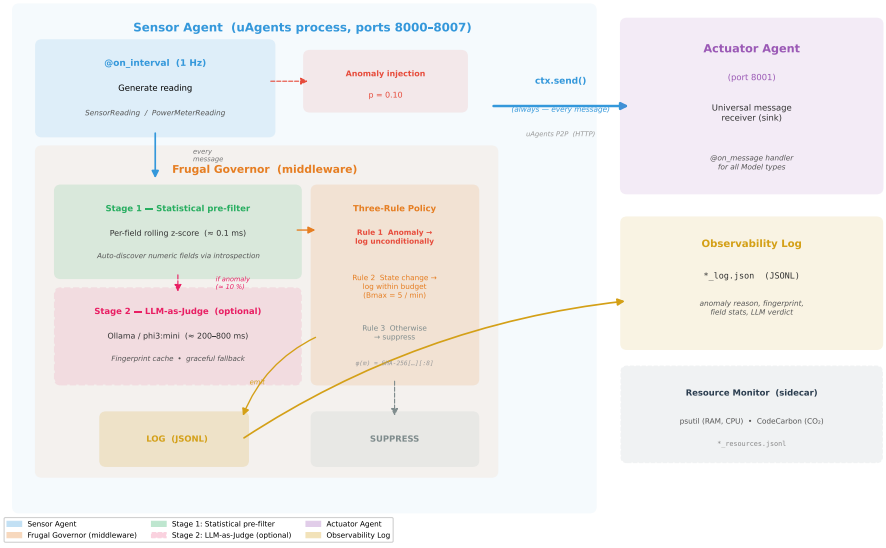

Fig. 1. System architecture. *Left:* Sensor Agent with the Frugal Governor middleware containing Stage 1 statistical pre-filter (green), optional Stage 2 LLM-as-Judge (pink, dashed), and the three-rule decision policy (orange). *Right:* Actuator Agent (universal receiver), Observability Log, and Resource Monitor sidecar.

Rule 3 — Budget suppression. Otherwise suppress: zero bytes written or transmitted.

B resets to 0 every 60 s; $B_{\max} = 5$ by default. Anomaly entries never count against budget. Algorithm 1 shows the core decision logic.

3.4 Fixed-τ and Threshold-Based Detection

Both methods access the `temperature` field by name and represent the *schema-specific* family of governors: efficient and lightweight, but requiring at least partial domain knowledge.

Fixed-τ: a reading is anomalous when $T > \tau = 27.0\,^{\circ}\mathrm{C}$. Simple and deterministic, but any change in sensor type requires manual reconfiguration. Miss probability per anomaly draw from $\mathcal{N}(28.5, 1)$:

$$P(T < \tau) = \Phi(-1.5) \approx 6.7\% \tag{1}$$

Threshold: self-calibrates via a sliding window $\mathcal{H}$ of the last $W = 20$ readings:

$$\text{is\_anomaly}(T) = \begin{cases} T > \tau_{\text{fb}} & |\mathcal{H}| < W \\ \dfrac{|T - \bar{T}|}{\sigma_T} > z_\theta & \text{otherwise} \end{cases} \tag{2}$$

Algorithm 1 Core SHOULDLOG for Threshold and Generic variants (z-score Rule 1). Fixed-τ replaces line 5 with $T > \tau$.

Require: message m, budget counter B, max budget $B_{\max}$, window W
Ensure: decision $\langle log, reason \rangle$
1: **if** budget timer expired **then**
2: $B \leftarrow 0$ ▷ reset every 60 s
3: **end if**
4: update rolling statistics from numeric fields of m
5: **if** $\exists$ field f with $|z_f| > 2$ **then** ▷ Rule 1: anomaly
6: **return** $\langle$TRUE, "anomaly_detected"$\rangle$
7: **end if**
8: $h \leftarrow$ FINGERPRINT(m) ▷ Rule 2: state change
9: **if** $h \neq h_{\text{prev}}$ **then**
10: $h_{\text{prev}} \leftarrow h$
11: **if** $B < B_{\max}$ **then**
12: $B \leftarrow B + 1$
13: **return** $\langle$TRUE, "state_change"$\rangle$
14: **end if**
15: **end if**
16: **return** $\langle$FALSE, "budget_suppressed"$\rangle$ ▷ Rule 3

with $z_\theta = 2.0$ and fallback $\tau_{\text{fb}} = 28.0\,^\circ\text{C}$. A stuck-sensor guard ($\sigma_T < 0.1$) applies a $1\,^\circ\text{C}$ absolute margin to prevent false negatives on constant streams. Despite removing the manual threshold, Threshold still hardcodes the field name `temperature`, motivating the generic extension.

3.5 Generic Extension: Reflection and LLM-as-Judge

The *GenericFrugalGovernor* replaces all hardcoded field accesses with *Python runtime introspection*: Pydantic's `.dict()` serialization converts any message model to a plain dictionary, and `isinstance` type checking identifies numeric leaf fields automatically. This makes the governor applicable to *any* `uAgents` Pydantic `Model` with the *same API call*, including schemas it has never seen at design time.

Stage 1 — Statistical pre-filter (≈0.1 ms): auto-discovers all numeric fields, maintains a per-field rolling z-score tracker, and flags anomalies if any field exceeds $z_\theta\,\sigma$. The fingerprint is a SHA-256 hash of sorted, rounded JSON of all non-timestamp values — schema-agnostic by construction.

Stage 2 — LLM-as-Judge (≈200–800 ms, invoked only when Stage 1 fires, ≈10% of messages): escalates to a locally-hosted model via `litellm` [4] and Ollama [15] (`phi3:mini`). The LLM evaluates physical plausibility, cross-field correlation, and severity (1–5 scale), responding with structured JSON: `{"anomaly": true, "reason": "...", "severity": 4}`. Results are cached by fingerprint (same state ⇒ no second call); in our benchmark the LLM was invoked only **≈4 times** per 120-second trial. On failure the governor falls back silently to the Stage 1 result.

3.6 LangGraph Integration: Graph-Orchestrated LLM Pipeline

To explore integration with the broader LLM ecosystem, we implement an alternative agent using LangGraph [13] combined with `ChatOllama`. The agent replaces the governor with a `StateGraph` pipeline: *(1)* `extract_fields` discovers numeric fields; *(2)* `statistical_check` runs per-field rolling z-scores; *(3)* a conditional edge routes anomalies to `llm_analyze` (same three criteria as Stage 2); *(4)* `build_report` applies the three-rule budget policy and constructs an `LLMAnalysisReport` containing severity, confidence, and a natural-language reason. This validates that uAgents integrates with the LangChain/LangGraph ecosystem without framework modification, and that graph-based LLM orchestration is feasible on edge hardware: all LangGraph dependencies have ARM64 wheels, with a measured runtime overhead of ≈31 MB.

4 Experimental Setup

4.1 Hardware, Software, and Metrics

All experiments ran on a commodity laptop (Intel Core i9-10885H, 16 GB RAM, Ubuntu 24.04, Python 3.12). *No physical Raspberry Pi 4 was used and no real IoT sensors were connected*: readings are generated synthetically, and RAM/CPU figures are laptop measurements used as a proxy for constrained hardware. The Pi 4 is the intended deployment target [14] (2 GB RAM, ARM Cortex-A72, sub-10 W); absolute RAM values (84–208 MB) may differ on ARM. Packages used are: `uagents`, `opentelemetry-sdk`, `codecarbon` [7], `psutil`, `litellm`, `matplotlib`. Several metrics are measured: log size (bytes), anomaly recall, precision, F1, bandwidth reduction vs. Verbose, average/peak RAM (psutil RSS, 1 Hz sampled from a separate thread), average CPU (%), and CO_2 (codecarbon, indicative). Precision is measured against the injected `status` field (ground truth); it is 100% by construction for all methods that log faithfully, and would require independent sensor ground truth in a real deployment.

4.2 Scenarios and Methods

Scenario A — Single schema (SensorReading): 120 s/agent. Normal readings: $\mathcal{N}(22, 1)$ °C; anomalous: $\mathcal{N}(28.5, 1)$ °C, injection probability $p = 0.10$. Five methods: Verbose (baseline), Probabilistic (10%), Frugal Fixed-τ ($\tau = 27.0$, $B_{\max} = 5$), Frugal Threshold ($W = 20$, $z_\theta = 2.0$, Eq. 2), and OTel SDK (custom `FileSpanExporter`, no Collector, no gRPC).

Scenario B — Generic extension: Same simulation parameters on `SensorReading` for recall measurement, adding Generic (Stat-only), Generic (LLM-judge with Ollama/`phi3:mini`), and LangGraph (Ollama). The generic agent can also be switched to `PowerMeterReading` via an environment variable to validate schema-agnosticism; the governor call is *identical* for both schemas.

Each agent writes a sidecar counter (`*_injected.txt`) so recall is computed per-agent. All eight methods are evaluated with identical controlled random seeds

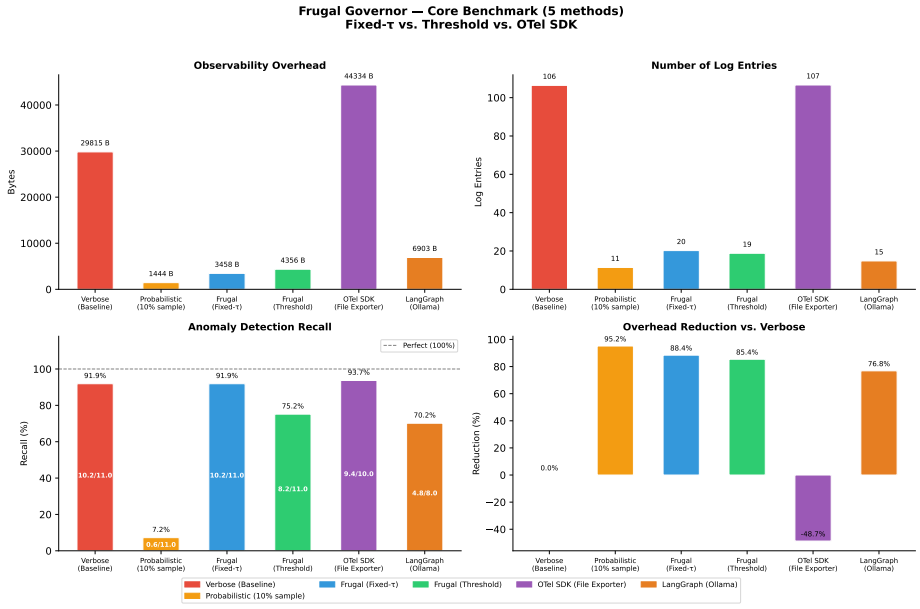

Fig. 2. Scenario A: overhead (top left), log entries (top right), anomaly recall (bottom left), reduction vs. Verbose (bottom right). Core 5 methods.

per trial; the benchmark is repeated for five trials (seeds 42–46) and results are reported as mean ± std.

5 Evaluation

5.1 Scenario A: Single-Schema Results

Table 1 consolidates bandwidth and recall results for all eight methods; Figure 2 shows the core five-method Scenario A comparison in chart form.

All lightweight methods achieve substantial reduction vs. Verbose: 95.2% (Probabilistic), 88.4% (Fixed-τ), 85.4% (Threshold). Fixed-τ and Threshold log more than Probabilistic due to Rule 1 (anomaly-always-log) and Rule 2 (state-change entries).

Probabilistic sampling captured only 7.2±9.9% of anomalies — a structural limitation for safety. Fixed-τ achieved 91.9±8.2% recall — identical to Verbose — because all injected anomalies in these five trials fell above the 27°C threshold; this is statistically expected since only ≈6.7% of draws from $\mathcal{N}(28.5, 1)$ lie below τ (Eq. 1), so the fixed threshold caused no misses in practice. The probability of observing *zero* misses across all five trials is $(0.933^{11})^5 \approx 2\%$ — unlikely but possible, and fully reproducible given the fixed seeds 42–46. With more trials this miss rate would manifest empirically. Threshold achieved 75.2±12.6% recall with

Table 1. Bandwidth and recall results (120 s/agent, mean over 5 trials).

Method	Log size Δ vs. Verbose[§]		Recall[‖]	F1
Scenario A — Schema-specific (SensorReading)				
Verbose (Baseline)	29,815 B	0.0%	91.9±8.2%	95.6±4.6%
Probabilistic (10%)	1,444 B	95.2%	7.2±9.9%	12.0±16.0%
Frugal Fixed-τ	3,458 B	88.4%	91.9±8.2%	95.6±4.6%
Frugal Threshold	**4,356 B**	**85.4%**	**75.2±12.6%**	**85.2±8.1%**
OTel SDK (File Exp.)[‡]	44,334 B	−48.7%	93.7±5.3%	96.7±2.8%
Scenario B — Generic extension (any schema)				
Generic (Stat-only)[¶]	5,628 B	81.1%	83.4±12.3%	90.4±7.5%
Generic (LLM-judge)[⋆]	4,574 B	84.7%	61.9±17.4%	75.0±13.8%
LangGraph (Ollama)[⋆]	6,903 B	76.8%	70.2±27.3%	79.2±20.7%

[‡] OTel spans include `trace_id`, `span_id`, timestamps, and nested attributes (≈416 B/span vs. ≈230 B/entry).
[⋆] LLM call latency (≈200–800 ms) reduces effective throughput within the fixed 120 s window, increasing recall variance.
[§] Positive = data reduction. **Negative = more data than Verbose.**
[‖] Precision is 100% for all methods (see Section 4).
[¶] 178 MB despite LLM disabled: `litellm` adds ≈93 MB at import time regardless.

0 false positives; the lower recall compared to Fixed-τ reflects the rolling z-score's sensitivity to anomaly contamination of the sliding window (10% injection rate pollutes the rolling mean), combined with the 20-reading warm-up period during which the fallback threshold (28.0°C) is stricter than Fixed-τ's 27.0°C. Verbose recall was 91.9±8.2%, reflecting finite-sample variance of the stochastic injection process ($p = 0.10$, ≈11 anomalies per 120 s trial).

OTel achieves 93.7±5.3% recall but logs 10× more data (44,334 B vs. 4,356 B). Each OTel span serializes a 32-char `trace_id`, 16-char `span_id`, nanosecond timestamps, service metadata, and nested attributes (≈416 B/span vs. ≈230 B/JSONL entry) — a fundamental characteristic of the OTel data model, not a configuration artifact. *The key trade-off*: Threshold achieves F1 85.2% with 85.4% bandwidth reduction, while OTel achieves F1 96.7% but *increases* bandwidth by 49% over Verbose. All methods produce negligible CO_2 eq. differences within codecarbon's constant-TDP noise floor.

5.2 Scenario B: General-Schema and Resource Results

Figure 3 maps all eight methods in the bandwidth vs. RAM plane with bubble size proportional to recall.

OTel requires 87.7 MB average RAM vs. 84.9–85.2 MB for the schema-specific variants. The generic governor variants carry a fixed **+93 MB import overhead** from `litellm`, incurred even when the LLM is disabled (Generic Stat-only: 178 MB). The LLM-judge variant reaches 207 MB; the LangGraph variant uses `langchain-ollama` instead of `litellm` and is lighter at 116 MB. All variants fit within the Pi 4's 2 GB budget.

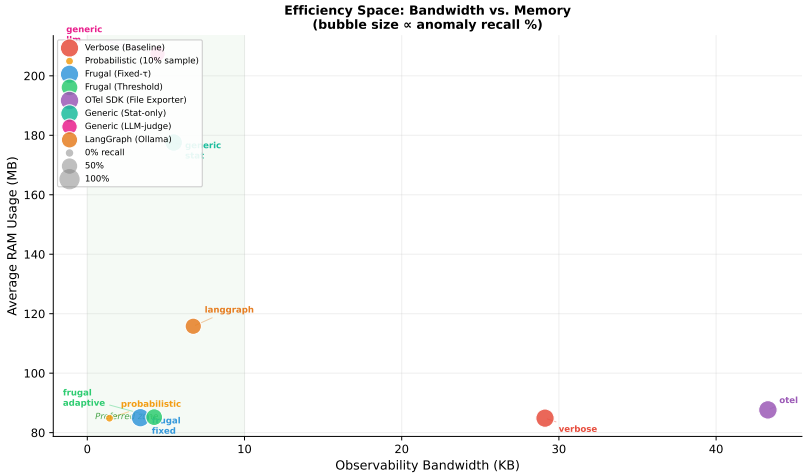

Fig. 3. Efficiency space: bandwidth (KB) vs. average RAM (MB), bubble size $\propto$ anomaly recall. Lower-left is preferred. Frugal Threshold achieves the best recall-bandwidth-memory balance; generic variants are above the 120 MB litellm boundary.

Generic (Stat-only) achieved 83.4±12.3% recall, the highest among budget-constrained methods, with no manual field configuration. Generic (LLM-judge) achieved 61.9±17.4% and LangGraph (Ollama) 70.2±27.3%. The lower mean recall of LLM-based variants is primarily due to call latency (≈200–800 ms): within the fixed 120-second window, each LLM call delays subsequent message processing, reducing the number of readings evaluated and anomalies captured. Both LLM-based methods invoked the model ≈4–5 times per trial, confirming two-stage frugality.

Summary. For edge deployments requiring *zero manual configuration*, Generic (Stat-only) offers the best recall (83.4%) at 178 MB RAM. For deployments where the operator can specify a threshold, Fixed-τ achieves 91.9% recall at 85 MB RAM with 88.4% bandwidth reduction. Threshold offers a middle ground: self-calibrating with 85.4% reduction at 85 MB, though its recall (75.2%) is reduced by anomaly contamination of the rolling window. LangGraph (Ollama) demonstrates ecosystem integration at 116 MB. Probabilistic sampling achieves the smallest log but captures only 7.2% of anomalies — unacceptable for safety-critical deployments.

6 Conclusion and Future Work

The Frugal Governor's three-rule policy delivers 85–95% bandwidth savings while maintaining meaningful anomaly recall. Unlike probabilistic sampling (7.2% recall), the governor provides a *safety guarantee*: Rule 1 ensures every reading flagged as anomalous — whether by a fixed threshold or a rolling z-score — is always logged, regardless of budget. The Fixed-τ variant achieves

91.9% recall (F1 95.6%) with 88.4% bandwidth reduction at only 85 MB RAM, matching the theoretical 6.7% per-draw miss rate from Eq. 1. The Threshold variant self-calibrates without manual configuration, achieving 75.2% recall (F1 85.2%) with 85.4% reduction; its lower recall reflects sensitivity to anomaly contamination of the rolling window, a known limitation of z-score methods on streams with non-negligible anomaly rates. The *GenericFrugalGovernor* validates format-agnosticism: the same API handles both `SensorReading` and `PowerMeterReading` with the LLM invoked only ≈4–5 times per trial. The LangGraph-based agent demonstrates seamless LangChain integration at ≈31 MB overhead.

Three limitations bound these conclusions. First, the evaluation is simulation-based: readings are synthetic, and resource figures are x86 laptop measurements (84–208 MB RAM) used as proxies for ARM hardware. Second, LLM-based methods show high recall variance (σ>17 pp) due to call latency and non-deterministic responses. Third, `litellm` imposes +93 MB import overhead even when the LLM is disabled; a lighter inference client would reduce this significantly.

The most important next step is physical validation on Raspberry Pi 4 with real IoT sensors. Beyond that: *(i)* sub-1B parameter models as on-device judges; *(ii)* federated z-score calibration across agent fleets; *(iii)* anomaly-resistant rolling statistics (excluding detected outliers from the window) to improve Threshold recall under high injection rates.

References

1. Arga, L., et al.: Frugal AI: Introduction, Concepts, Development and Open Questions (May 2025), HAL preprint hal-05049765, `https://hal.science/hal-05049765`
2. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley Series in Agent Technology, Wiley (2007)
3. Bensaid, W., Molhem, M.: A novel adaptive sampling algorithm for cyber-physical systems. International Review of Applied Sciences and Engineering **15**(2), 161–169 (2023). `https://doi.org/10.1556/1848.2023.00667`
4. BerriAI: LiteLLM: Call all LLM APIs using the OpenAI format. `https://github.com/BerriAI/litellm` (2024)
5. Bhuian, O.B.K.: Deep dive into OpenTelemetry for evaluation of their observability in edge computing environment. In: The 10th International Conference on Next Generation Computing (ICNGC 2024). pp. 161–164 (2024)
6. Cook, A.A., Mısırlı, G., Fan, Z.: Anomaly detection for IoT time-series data: A survey. IEEE Internet of Things Journal **7**(7), 6481–6494 (2020). `https://doi.org/10.1109/JIOT.2019.2958185`
7. Courty, B., et al.: mlco2/codecarbon: v3.2.5 (Mar 2026). `https://doi.org/10.5281/zenodo.19110653`
8. Giordano, M., Cortesi, S., et al.: Energy-aware adaptive sampling for self-sustainability in resource-constrained IoT devices. In: Proceedings of the 11th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems (ENSsys 2023). pp. 65–71. ACM (2023). `https://doi.org/10.1145/3628353.3628545`
9. Giouroukis, D., Dadiani, A., Traub, J., Zeuch, S., Markl, V.: A survey of adaptive sampling and filtering algorithms for the Internet of Things. In: Proceedings of the

14th ACM International Conference on Distributed and Event-based Systems (DEBS 2020). pp. 27–38. ACM (2020). https://doi.org/10.1145/3401025.3403777
10. Guardrails AI: Guardrails: Structured Output Validation for Large Language Models. https://github.com/guardrails-ai/guardrails (2023)
11. Kaebisch, S., Kamiya, T., McCool, M., Charpenay, V., Kovatsch, M.: Web of Things (WoT) Thing Description 1.1. W3C recommendation, World Wide Web Consortium (W3C) (Dec 2023), https://www.w3.org/TR/wot-thing-description11/
12. Kazil, J., Masad, D., Crooks, A.: Utilizing python for agent-based modeling: The mesa framework. In: Thomson, R., Bisgin, H., Dancy, C., Hyder, A., Hussain, M. (eds.) Social, Cultural, and Behavioral Modeling. pp. 308–317. Springer International Publishing, Cham (2020)
13. LangChain Inc.: LangGraph: Build Resilient Language Agents as Graphs. https://github.com/langchain-ai/langgraph (2024)
14. Nguyen, T., Nguyen, T.: An evaluation of LLMs inference on popular single-board computers (2025), https://arxiv.org/abs/2511.07425
15. Ollama: Ollama: Get Up and Running with Large Language Models Locally. https://ollama.com (2023)
16. OpenTelemetry Authors: OpenTelemetry — Vendor-Neutral Observability Framework. https://opentelemetry.io (2024)
17. Palanca, J., Terrasa, A., Julian, V., Carrascosa, C.: SPADE 3: Supporting the new generation of multi-agent systems. IEEE Access **8**, 182537–182549 (2020). https://doi.org/10.1109/ACCESS.2020.3027357
18. Parseable: Observability Agent Profiling: Fluent Bit vs. OpenTelemetry Collector Performance Analysis. https://www.parseable.com/blog/observability-agent-profiling-fluent-bit-vs-opentelemetry-collector-performance-analysis (2025)
19. Splunk Inc.: Sizing and Scaling the Splunk Distribution of the OpenTelemetry Collector. https://help.splunk.com/en/splunk-observability-cloud/manage-data/splunk-distribution-of-the-opentelemetry-collector/get-started-with-the-splunk-distribution-of-the-opentelemetry-collector/collector-requirements/sizing-and-scaling (2025)
20. Wooldridge, M., Bagoly, A., Ward, J.J., La Malfa, E., Licks, G.P.: Fetch.ai: An Architecture for Modern Multi-Agent Systems. arXiv preprint arXiv:2510.18699 (2025)
21. Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., et al.: AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv preprint arXiv:2308.08155 (2023)