Streaming, Drift-Aware Log Anomaly Detection with Sliding Conformal Calibration
(Reproducible, CI-tested implementation with strict provenance)

Felipe Arche[1]

[1]felipearche@users.noreply.github.com

Abstract

We present a practical pipeline for *streaming* log anomaly detection that couples a compact base detector with *Sliding Conformal* calibration to target a fixed false-positive rate, and *ADWIN* to reset calibration on detected drift. The system emits canonical experiment rows into a 24-column CSV and enforces audit-grade provenance (sizes + SHA-256 hashes) for artifacts. On a synthetic labeled dataset (`synth_tokens`) and an unlabeled micro-dataset (`mini_tokens`), the calibrated pipeline attains strong TPR@1%FPR with low latency; a tiny transformer variant improves throughput further. Code: github.com/felipearche/log-project.

1 Introduction

Context. Modern services emit millions of log lines per hour. Offline detectors are brittle under nonstationarity; operators need online methods with *calibrated false positive control*, predictable latency, and explicit drift handling. Our design combines a lightweight scorer (TF–IDF+IsolationForest or a compact transformer) with *sliding conformal calibration* and *ADWIN* resets to keep alerts aligned with a target FPR.

Contributions. (i) A practical, CPU-friendly streaming detector with quantile-based calibration and drift resets. (ii) A strict, audit-ready provenance pipeline that writes one canonical row per run to `experiments/summary.csv` and mirrors it in `docs/PROVENANCE.txt`. (iii) A small, reproducible benchmark with latency p95/p99 and throughput (events/s), plus TPR@1%FPR when labels exist. (iv) A release and CI setup pinned by SHA for long-term reproducibility.

2 Related Work and Background

Log anomaly detection. Classical log anomaly detection pipelines tokenize lines, apply a sparse representation (e.g., TF–IDF) and score with unsupervised detectors like Isolation Forest or one-class SVMs. This baseline remains attractive in streaming because it is light-weight, has small memory footprints, and degrades gracefully under drift when paired with calibration. In practice, TF–IDF + Isolation Forest is often competitive if you (i) re-fit the vectorizer online to avoid vocabulary staleness and (ii) maintain a small sliding buffer for threshold calibration.

Conformal prediction (CP). CP turns arbitrary nonconformity scores into calibrated decisions by maintaining a recent window of scores and choosing a quantile corresponding to a target false-positive rate (FPR). Under standard exchangeability assumptions, conformal guarantees validity in finite samples. In streaming, we use a *sliding, inductive* variant: keep the last W scores, take the $(1-\alpha)$-quantile as the threshold, and flag any new score above it as an anomaly. This makes the FPR self-correcting as conditions change.

Drift detection (ADWIN). Adaptive Windowing (ADWIN) monitors a data stream and detects statistically significant changes by maintaining two subwindows and testing for mean differences with confidence δ. When drift is signaled, we reset the conformal buffer (and optionally the model) so that calibration reflects the new distribution quickly.

Why this combination? CP handles day-to-day nonstationarity, while ADWIN handles regime shifts. Together, they let a simple detector sustain a constant target FPR without constant retuning.

System logs are ubiquitous, high-rate, semi-structured text streams. Detecting anomalies in real time is crucial for reliability and security but complicated by distribution shift and scarce labels. Deep sequence models such as *DeepLog* [2] and *LogBERT* [3] advance the state of the art, yet their training and compute budgets can hinder fast deployment. We instead focus on a compact, reproducible pipeline: TF–IDF n-grams feeding *Isolation Forest* [4], wrapped with conformal calibration [7, 6] to control false alarms online and *ADWIN* [1] to adapt under drift. The result is a practical baseline you can run and audit end-to-end.

3 Problem Setup and Notation

Formalization

We observe a sequence of log lines $\{\ell_t\}_{t=1}^{\infty}$ and compute nonconformity scores $s_t = f_\theta(\ell_t)$ with either a sparse

baseline (TF–IDF + IsolationForest) or a compact transformer encoder. Let B_t be the calibration buffer at time t (multiset of the last W scores). We predict *anomaly* when $s_t > q_{1-\alpha}(B_{t-1})$, where $q_{1-\alpha}$ is the empirical $(1-\alpha)$-quantile. ADWIN runs in parallel on either raw features or scores; upon a change signal, we clear B_t and optionally reinitialize θ.

Metrics are computed online: p95/p99 latency (milliseconds), throughput (events/s), and, when labels are available, TPR@1%FPR. Each experimental run appends a canonical row to a 24-column CSV (see repository) to ensure strict provenance and auditability.

We observe a stream of log lines $\{\ell_t\}_{t=1}^{\infty}$. A featurizer $\phi(\cdot)$ maps each ℓ_t into $\mathbb{R}^d$ after tokenization and masking of dynamic substrings (numbers, identifiers, hex). A base detector $f : \mathbb{R}^d \to \mathbb{R}$ outputs a *nonconformity score* $s_t = f(\phi(\ell_t))$. Let B_t be a FIFO buffer of recent scores considered representative of "normal" behavior. For a target false-positive rate $\alpha \in (0, 1)$, Sliding Conformal sets the decision threshold

$$\tau_t \;=\; \text{Quantile}_{1-\alpha}(B_t), \tag{1}$$

and flags ℓ_t as anomalous if $s_t > \tau_t$. An online drift detector D (ADWIN) monitors $\{s_t\}$; on alarm, B_t is cleared (and optionally f reinitialized). We target α=0.01 and report p95/p99 latencies (ms) and throughput (events/s).

4 Method

Calibration details

We maintain a fixed-size ring buffer storing the last W scores. At each time step t, we (i) insert s_t into the buffer, (ii) update an approximate quantile data structure, and (iii) set $\tau_t = q_{1-\alpha}(B_t)$. The decision rule is $s_{t+1} > \tau_t$. On an ADWIN signal, we empty the buffer and warm-start it over the next W steps to avoid using stale thresholds.

Transformer option

The transformer encoder is intentionally tiny (few heads, few layers) and trained once offline on generic logs to learn token co-occurrence structure. In streaming mode we freeze weights and only update the calibrator. This keeps memory and CPU predictable while capturing patterns (e.g., templated vs. free-form lines) that sparse methods miss.

Feature extraction. Each line is tokenized; dynamic substrings (e.g., numbers, UUID-like, hex) are masked to reduce spurious novelty. We build TF–IDF n-gram features [5].

Base detectors. (i) TF–IDF + Isolation Forest (unsupervised, robust, low memory); (ii) a tiny transformer scorer that drops into the same calibration/drift hooks. Scores from either act as nonconformity.

Sliding Conformal. Maintain a buffer of size W with scores assumed exchangeable over short windows. After warmup, threshold by the $(1 - \alpha)$ empirical quantile.

Drift handling. ADWIN maintains an adaptive window and signals a change when subwindows differ significantly [1]. On alarm, clear the calibration buffer (and optionally reinit f) to quickly realign thresholds.

5 Algorithm (Streaming Loop)

1. Acquire ℓ_t; tokenize and mask dynamic fields to obtain $\phi(\ell_t)$.
2. Compute score $s_t = f(\phi(\ell_t))$.
3. If $|B_t| < W$ (warmup): append s_t to B_t and continue.
4. Set $\tau_t := \text{Quantile}_{1-\alpha}(B_t)$; emit anomaly if $s_t > \tau_t$.
5. Feed s_t to ADWIN; on alarm: clear B_t (and optionally reset f).
6. Slide B_t with s_t in FIFO fashion.

6 Implementation and Reproducibility

Exact commands (from README)

Quickstart (Windows/PowerShell).

```
python -m venv .venv; .\.venv\Scripts\Activate.ps1
pip install -r env/dev-requirements.lock
pytest -q
```

Docker (no local Python).

```
docker run --rm -v "${PWD}:/app" -w /app python:3.11.9-
    slim /bin/bash -lc \
  "pip␣install␣-r␣env/dev-requirements.lock␣&&␣pytest␣-q"
```

Reproduce a calibrated run & figures.

```
$env:COMMIT = (git rev-parse --short HEAD).Trim()
docker build -t log-project:latest .
docker run --rm -v "${PWD}:/app" -e COMMIT=$env:COMMIT log
    -project:latest
docker run --rm -v "${PWD}:/app" -e COMMIT=$env:COMMIT log
    -project:latest ‘
  python scripts/make_plots.py --summary experiments/
    summary.csv
docker run --rm -v "${PWD}:/app" -e COMMIT=$env:COMMIT log
    -project:latest ‘
  python scripts/make_readme_table.py --csv experiments/
    summary.csv --out README_TABLE.txt
```

These commands mirror the CI and guarantee parity across machines. Each run appends exactly one row to the canonical CSV and one block to the provenance log.

CLI. The script `src/stream.py` exposes `-mode`, `-data`, `-labels` (optional), and `-no-calib` for ablations. Each run appends *one* canonical row to `experiments/summary.csv` and emits a matching block in `docs/PROVENANCE.txt`.

Provenance. `data/HASHES.txt` records artifact sizes and uppercased SHA-256. Protected JSONs (`synth_tokens`, `synth_labels`, `mini_tokens`) are byte-exact and end without a final LF. Text files are UTF-8 (no BOM), LF-only, enforced by hooks.

QA gates. CI runs pre-commit (format + hygiene), mypy, CSV schema/format guard, and tests. Windows uses Python 3.11; actions/images are pinned.

7 Datasets and Metrics

Datasets. We use two repository-hosted tokenized sets. `mini_tokens` is unlabeled and sized for smoketests and performance measurements. `synth_tokens` contains synthetic labels to estimate recall at a fixed FPR. Both are small by design so that results are reproducible on commodity laptops and in CI.

Metrics. We report (a) **latency** p95 and p99 in milliseconds measured end-to-end per line; (b) **throughput** in processed events per second (*eps*); and (c) **TPR@1%FPR** when labels are available. Calibration uses a sliding window of recent scores of size W and the empirical $(1-\alpha)$ quantile as threshold; unless otherwise stated we target α=0.01. Drift is monitored with ADWIN; upon a change signal the calibration buffer is reset so thresholds adapt quickly without manual retuning.

Provenance. Each run appends a single row (24 fields) to `experiments/summary.csv` (commit, config, W, α, ADWIN δ, latency stats, eps, etc.) and a matching block to `docs/PROVENANCE.txt`. Plots and README tables are generated directly from the CSV to avoid manual transcription.

8 Evaluation Protocol

Datasets. We report on two internal tokenized datasets distributed in the repository: `mini_tokens` (unlabeled) and `synth_tokens` with labels for TPR estimation. The unlabeled set exercises latency/throughput; the labeled set exercises calibration quality.

Procedure. For each configuration (baseline vs. transformer; calibrated vs. no-calib), we stream the full dataset once. We record (i) summary throughput and p95/p99 latency; (ii) if labels are present, TPR@1%FPR using the conformal threshold. All measurements are taken on an otherwise idle host; we recommend reporting medians over 3 runs.

Reporting. Results are emitted to a single CSV with a 24-column schema (commit, config, window W, target α, ADWIN δ, latency stats, throughput, energy if available, etc.). A matching block is appended to `docs/PROVENANCE.txt`. Figures in this paper are generated directly from the CSV.

Table 1: Calibrated-only snapshot (target FPR = 1%).

dataset	mode	calibration	TPR@1%FPR	p95 (ms)	p99 (ms)	eps
synth_tokens	baseline	conformal	1.0000	3.2	3.4	328.5
synth_tokens	transformer	conformal	0.9833	1.6	1.9	731.6
mini_tokens	baseline	conformal	NA	7.4	7.4	119.0
mini_tokens	transformer	conformal	NA	1.5	1.5	454.6

Figure 1: Latency p95 (ms) across configurations.

`synth_tokens` (labeled) supports TPR@1%FPR, p95/p99 latency, and throughput. **`mini_tokens`** (unlabeled) supports latency/throughput only. We report TPR at 1% FPR when labels are available, and always report p95/p99 (ms) and events/s. Throughput is measured over steady-state windows; latency is per-line end-to-end.

9 Results

Summary of findings (qualitative). Calibrated variants maintain a stable alert rate close to the target α despite normal variation. The sparse baseline (TF–IDF+IsolationForest) yields the highest throughput with low CPU cost, making it a strong default. The compact transformer improves scoring quality on free-form lines but adds modest latency; in small-scale runs the difference remains within typical operational budgets.

Reproducibility. All figures in this paper are produced from `experiments/summary.csv`. Re-run the benchmark and regenerate plots/tables with the repository scripts; replace the PNGs referenced in the figures to update this document.

10 Ablations and Sensitivity

Window size W. Larger W stabilizes thresholds but slows adaptation; smaller W adapts faster but increases quantile variance.

Target FPR α. Lower α reduces false alarms but may reduce TPR; α=0.01 is a pragmatic default for operations.

ADWIN δ. Smaller δ is more conservative (fewer false alarms) at the cost of slower shift detection.

Figure 2: Latency p99 (ms) across configurations.

Figure 3: Throughput (events/s).

Transformer depth/width. Shallow configurations often match or exceed Isolation Forest throughput while maintaining quality after calibration.

11 Engineering Notes

Calibration buffer. We implement the sliding window as a fixed-size ring buffer and maintain the threshold with an approximate quantile structure (e.g., GK sketch) to avoid sorting at every step. On ADWIN signals we clear and warm-start the buffer to prevent stale thresholds.

Throughput and backpressure. We default to one-at-a-time updates for predictable p95/p99 latency. Batch scoring improves CPU throughput but increases tail-latency variance; for strict SLOs we recommend batch size = 1 with backpressure-aware ingestion.

Tokenization and hashing. A hashed TF–IDF vectorizer avoids vocabulary blow-ups under template churn and enables constant-memory updates. For the transformer path we cap token length, prefer byte/character pieces for robustness, and freeze weights in streaming mode.

Reproducibility engineering. CI is pinned by SHA, environments are lock-filed, and data integrity is tracked via size and SHA-256 in `data/HASHES.txt`. Scripts emit deterministic CSV rows and avoid wall-clock dependent randomness.

12 Complexity and Memory

Baseline. With a hashed TF–IDF vectorizer of dimension d and IsolationForest of T trees, scoring is $O(d)$ per line and model memory is $O(T \cdot \text{depth})$ (small for shallow trees). The sliding conformal buffer adds $O(W)$ memory; we maintain the threshold via a streaming quantile structure (binary heap or GK-sketch) to avoid full sorts.

Transformer. A micro-transformer with L layers, hidden size h, and max token length n yields $O(L \cdot n \cdot h^2)$ compute without KV caching. In practice we cap n (truncate long lines) and prune width to stay within a few ms per line on commodity CPUs. Calibration and ADWIN overheads are negligible relative to the forward pass.

Throughput. End-to-end throughput is limited by featurization (tokenization + hashing), model scoring, and occasional calibrator updates. Batching can increase CPU efficiency but increases latency variance; we default to single-line updates for predictable p95/p99.

Latency measurement. Reported per-line timing includes tokenization, vectorization, scoring, conformal thresholding, and I/O.

Masking. Numeric fields and common identifiers are masked pre-vectorization to reduce spurious novelty from variable parts (e.g., PIDs, counters).

Threading. A lightweight pipeline can exploit batched featurization with micro-batching without changing conformal semantics.

13 Threat Model and Operations

Adversarial inputs. Attackers may inject rare templates to induce threshold drift. Conformal thresholds resist single outliers; ADWIN resets limit long-term damage.

Privacy. Logs may contain sensitive data even after masking. We recommend additional redaction and role-based access controls.

Human-in-the-loop. Anomalies should be triaged with feedback captured for future supervised refinement.

14 Reproducibility Checklist

- Deterministic seeds for featurization and model initialization.
- One canonical CSV row per run; 1:1 block in `docs/PROVENANCE.txt`.
- Artifact sizes and SHA-256 in `data/HASHES.txt` (uppercase).

- UTF-8 (no BOM), LF-only enforced by hooks; protected JSONs end without final LF.
- CI: pre-commit, mypy, CSV checks, tests on Windows and Ubuntu.

15 Limitations and Ethics

Limitations. (i) After a drift reset the calibration buffer is short; quantile estimates are noisy for $< W$ steps. (ii) Synthetic labels are useful for smoke tests but not a substitute for real, delayed, and noisy labels. (iii) The transformer path, while compact, still costs more CPU than the sparse baseline; very high-throughput workloads may prefer the baseline.

Ethics and operations. Log streams can contain sensitive information. This project does *not* train models on private data; published datasets are synthetic or sanitized. We expose calibration targets and thresholds to avoid "black-box" alerting and provide reproducible procedures so others can audit results. In production, alerts should flow through a human-in-the-loop triage process with feedback for false positives.

16 Responsible Release and Transparency

We release code with (i) CI pinned by SHA, (ii) environment pins for reproducibility, and (iii) explicit data integrity via file size + SHA-256 in `data/HASHES.txt`. We avoid training or publishing models on sensitive data, document limitations (e.g., temporary FPR drift under abrupt distribution shifts), and provide a precise runbook so others can replicate results without hidden steps.

Synthetic datasets are small; production diversity and rare security payloads may be underrepresented. Conformal validity assumes short-term exchangeability; severe non-stationarity may require larger buffers or hybrid detectors. We mask dynamic fields but recommend layered redaction given privacy risks in logs.

17 Conclusion and Future Work

Summary

A lightweight detector with sliding conformal calibration and ADWIN resets sustains a target FPR with predictable latency on commodity CPUs. In future work we will (i) add energy measurements when supported by hardware telemetry, (ii) explore character-level tokenization to reduce OOV effects, and (iii) provide a tiny distilled transformer variant with CPU-friendly kernels.

A compact, calibrated streaming pipeline with drift-aware resets provides competitive utility and strong reproducibility with an explicit audit trail. Future work: (i) template-aware tokenization, (ii) hybrid semi-supervision with weak labels, (iii) adaptive buffer sizing coupled to ADWIN confidence, and (iv) deployment studies on production traces.

References

[1] A. Bifet and R. Gavaldà. Learning from time-changing data with adaptive windowing. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, pages 443–448, 2007.

[2] M. Du, F. Li, G. Zheng, and V. Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1285–1298, 2017.

[3] H. Guo, Y. Yuan, J. Liu, T. Lv, et al. Logbert: Log anomaly detection via bert. *arXiv preprint arXiv:2103.04475*, 2021.

[4] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining (ICDM)*, pages 413–422, 2008.

[5] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[6] G. Shafer and V. Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9:371–421, 2008.

[7] V. Vovk, A. Gammerman, and G. Shafer. *Algorithmic Learning in a Random World*. Springer, 2005.