Harbormaster: Evidence-Gated, Replay-Safe Maritime Anomaly Detection on AWS

Arun Sharma
arun08sharma@gmail.com

Abstract

Ships broadcast their positions through the Automatic Identification System (AIS), and those reports can be false or missing. An operator who acts on an anomaly alert needs that alert to be attributable, reviewable, and recoverable after a failure. This paper describes Harbormaster, a production-shaped system on AWS that follows three rules. Physics checks run on every report before any learned model runs. The DynamoDB read store is an idempotent projection of PostgreSQL, and a guard on the log sequence number (LSN) of each change protects every write to it. A candidate model must pass a holdout gate and a shadow comparison before a canary gives it traffic, and a burn-rate check guards each canary step. The paper proves that replaying any prefix or suffix of the change log leaves each projected key at the value of its highest applied LSN. It also names the effects that this result does not cover, such as repeated cache invalidations and repeated audit rows. In a bounded AWS window, a one-hour soak returned 35,999 HTTP 200 responses to 36,000 requests, with a 95th-percentile (p95) client latency of 142.751 ms. In a separate bounded AWS run of 900 s, the stream path received a burst of 400 records/s, and its consumer lag later drained to zero. Every number in the paper carries a label that says where it was measured, and the paper lists the parts of the design that were never built.

1 Introduction

Ships report their identity, position, speed, and course through the Automatic Identification System (AIS) [32]. A vessel can switch its transponder off [41], and a report can also carry a position that no vessel could reach. Two historical cases show what these anomalies look like. In one track, two consecutive reports are 207.846 km and 815 s apart, which implies a speed of 255.026 m/s. In another track, one vessel stays silent for 14 days, 18 hours, and 59 minutes. Both cases come from a read-only case analysis, and Harbormaster did not score either of them.

An anomaly alert is useful only when a person can act on it safely. The alert must be attributable, so the operator can see which rule or model produced it and from which inputs. It must be reviewable, so an analyst can record a judgment without changing the automated score. It must also be recoverable, so the derived state stays correct after a crash, a redelivered message, or a replay. Harbormaster is built around one design question. How can a cloud pipeline turn untrusted AIS reports into alerts that stay attributable, reviewable, and recoverable when messages repeat and when models change?

The system makes three design choices. A fast physics check runs before the heavier model, and the deterministic scorer needs no learned model to return a result. PostgreSQL holds the authoritative registry, and its DynamoDB read store is an idempotent projection of the change log. A candidate model must pass gates that produce evidence before it receives traffic. The system is production-shaped, and its AWS parts ran only in attended, time-boxed windows. That environment is no longer running.

Every number in the paper carries a scope label. The label names a bounded AWS window, a local run, a synthetic drill on real project code, or the public repository. A reader can rerun only the public test suite, the local change data capture (CDC) tests, and the public drill scripts, and Appendix A states that boundary. The paper makes five contributions.

- It describes a physics-first scoring path in which a required-speed check and a stream-layer physics gate run before any learned model, and the gap detector uses space-time prisms to merge candidate gaps (Section 4).
- It gives a replay-safe CDC path. Its DynamoDB projection accepts a change only when that change's LSN is newer, and the paper proves that replay leaves each key at its highest applied LSN under stated assumptions (Section 5).
- It specifies an evidence-gated promotion ladder of holdout, reward-hacking, shadow, and canary checks, and it tests the holdout, shadow, and canary steps in synthetic drills (Section 6).
- It documents a scale-to-zero design for an asynchronous model endpoint, together with two metric-dimension bugs that each disabled one scaling path without an error (Section 7).
- It reports a scoped evaluation in which every result names where it ran, and it lists the parts of the design that were never built (Sections 8 and 9).

2 Related Work

Maritime anomaly detection. Riveiro et al. [32] review maritime anomaly detection work on data, methods, systems, and user aspects. TREAD [28] learns shipping routes from AIS and flags vessels that leave them. Welch et al. [41] map where fishing vessels disable their AIS transponders, which shows that a reporting gap can itself be the behavior of interest. Harbormaster's corridor detector follows the route-learning idea, and its lane builder uses the Douglas-Peucker algorithm [15] and HDBSCAN [10].

Space-time prisms and trajectory gaps. Time geography describes the set of places that a moving object can reach between two observed points [21], and Miller [27] brought the space-time prism into geographic information systems. Sharma and Shekhar [35] used prisms to bound the region where two vessels could have met during a trajectory gap. Sharma et al. [36] detected abnormal trajectory gaps with physics-based bounds on vessel motion, through the STAGD gap detector and a dynamic region merge. Those two papers share an author with this one. Harbormaster uses the prism test and the gap detection of that work inside a serving system, and it proposes no new detection method.

Stream processing and change data capture. Apache Flink [11] runs stateful stream programs as pipelined dataflows, and its snapshots keep keyed state consistent after a failure [12]. MillWheel [1] and the Dataflow model [2] define event time and watermarks for late data. Kafka [24] carries the change events that Debezium [14] reads through PostgreSQL logical replication [39]. DBLog [6] uses watermarks to interleave table snapshots with log events. Harbormaster handles the overlap between Debezium's snapshot rows and streamed changes with a simpler rule, which applies every snapshot row at LSN zero. Kleppmann [23] treats derived data as a view of an ordered log. This paper states that view as a proposition for one sink built on DynamoDB conditional writes [16].

Machine learning operations and gating. Sculley et al. [33] describe the maintenance debt that builds up around machine learning code. The ML Test Score [9] turns deployment readiness into a rubric of tests, and TFX [7] places data and model validation inside the training pipeline. Engineers interviewed by Shankar et al. [34] describe a shadow stage, in which a new model makes live predictions that users do not see. That stage comes before a rollout to a small share of live users. Canary analysis [13] compares a new version with the old one before a full rollout. Gandalf [25] correlates fault signals with ongoing rollouts and stops a rollout that it judges unsafe. Harbormaster's canary uses the multiwindow burn-rate alerts of Thurgood et al. [40], which build on the service level objectives of site reliability engineering [8]. Its holdout gate uses AUC [17] and the continuous ranked probability score (CRPS) [20, 19], and its endpoint follows the serverless goal of holding no idle capacity [22].

Drift and reward hacking. Dataset shift [29] and concept drift [18] explain why a model that passed its holdout can degrade after deployment. Rabanser et al. [30] compare shift detectors and find that two-sample tests work best on the outputs of a pretrained classifier. Harbormaster's input check pairs the Population Stability Index [42] with the Kolmogorov-Smirnov statistic [26]. Amodei et al. [5] name reward hacking as a concrete safety problem, and Skalse et al. [37] give it a formal definition. Harbormaster's probe checks one narrow form of it, in which a preference-tuned candidate raises its reward while it respects physics less.

3 System Overview

Harbormaster splits its work into two planes (Figure 1). The training plane belongs on an off-cloud research GPU cluster, and this paper treats it as design only. The serving plane runs on AWS, and its parts ran in bounded windows. No trained checkpoint was promoted into AWS serving, so the model endpoint served a labeled demo stand-in image.

Five workflows. The design has five workflows, and each one has its own trigger. Workflow 1 checks each incoming vessel report, and Workflow 2 saves an analyst's review label. Workflow 3 carries committed registry changes into the serving copies, and Workflow 4 builds shipping-lane context from historical batches. In the design, Workflow 5 submits a separate model job to an asynchronous endpoint, and its job service, dispatch worker, and result collector are not built. The ingestor replays only a synthetic replay fixture of 2,709 events for 8 vessels over 6 hours, and no real-time AIS source was connected.

Two brokers and the stream job. Position reports and database changes travel on separate brokers. Provisioned Amazon Kinesis [3] carries position reports, and Managed Flink 1.20 [4] consumes them with state keyed by the vessel's MMSI (Maritime Mobile Service Identity). Kafka carries Debezium change events. The AWS windows used Amazon MSK Serverless, and the local stack runs a single-node KRaft Kafka that Strimzi manages on kind. The Flink job keeps the last five fixes for each vessel, and it updates that history before it applies the physics gate. For each report, one keyed operator computes features and applies the gate. For each report that passes, it writes a feature item to DynamoDB and posts a signed request to the scorer.

The scoring front door. The scorer is the entry point for every scoring request, and the project calls it the front door. It is a FastAPI service on ECS Fargate, and the Terraform at the reviewed commit runs it on Fargate Spot capacity. Requests reach it through an API Gateway HTTP API with IAM authorization and a VPC Link that resolves the service through Cloud Map, so the path has no standing load balancer. The deterministic scorer can call an asynchronous SageMaker endpoint for the data term P_{data}. The ECS task on AWS set no endpoint, so P_{data} came from a numpy surrogate there. In the design, review cases go to a PostgreSQL table, and the Streamlit console that reads them ran only locally. On AWS, the review queue fell back to process memory, and Section 9 explains why.

Batch lake and control. In the design, Workflow 4 runs PySpark on EMR Serverless and writes lane tables to Apache Iceberg [38]. EMR Serverless stops itself after 15 idle minutes. In a bounded AWS window, a valid backfill job succeeded, and a job with invalid data failed before any write. The lane builder uses a Douglas-Peucker tolerance of 200 m and an HDBSCAN minimum cluster size of 2. The publisher from lane tables to a serving reference is assumed integration, so the scorer reads a small synthetic lane reference. Terraform manages the infrastructure as code [31], and every phase sits behind a toggle that an apply turns off for teardown.

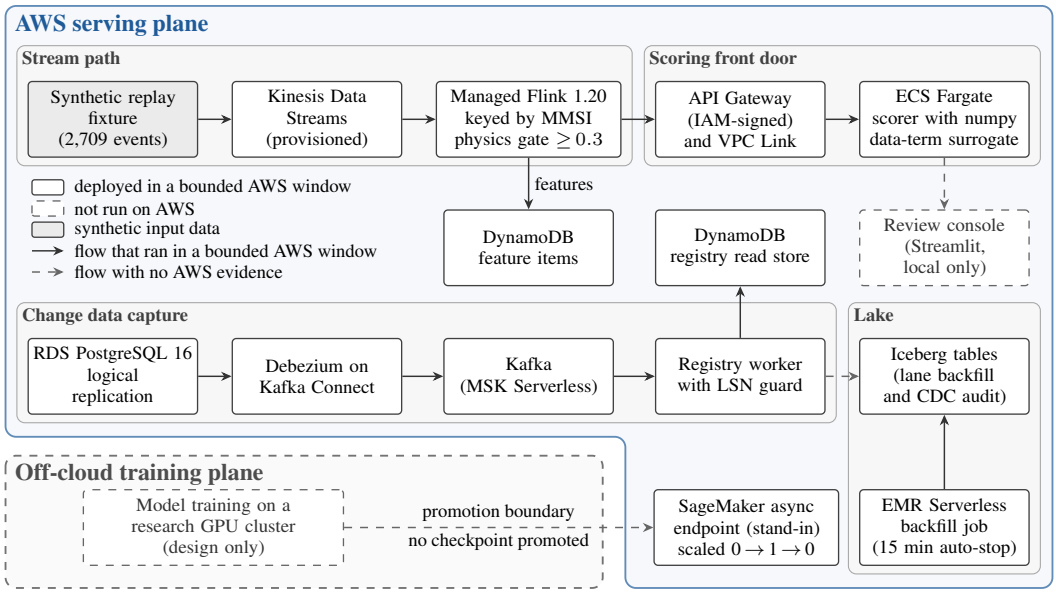


Figure 1: Harbormaster separates an off-cloud training plane from an AWS serving plane. Training on a research GPU cluster is design only, and no trained checkpoint crossed the promotion boundary. The Streamlit review console runs locally. Solid boxes are components that were deployed in a bounded AWS window, and solid arrows are flows that ran in a bounded AWS window. Dashed boxes did not run on AWS, and dashed arrows mark flows with no AWS evidence, such as the append to the CDC audit table. The Terraform at the reviewed commit runs the scorer on Fargate Spot capacity. The SageMaker asynchronous endpoint served a labeled stand-in image and scaled from zero to one instance and back. The ECS task sets no endpoint, so the scorer took its data term P_{data} from a numpy surrogate on AWS. The figure therefore draws no arrow between the scorer and the endpoint.

4 Physics-First Scoring

The scorer uses physics as its first source of evidence. Each rule in this section is a closed-form function of the reported fixes and the reference data, so an analyst can recompute any score from the constants in the public code.

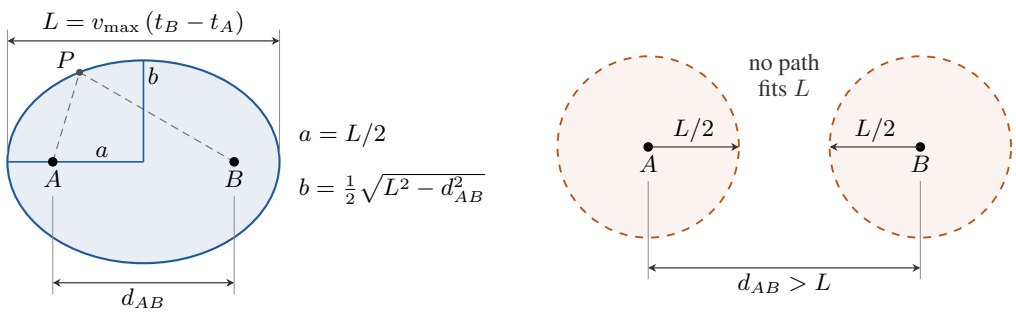
Required speed and the stream gate. A fix i has latitude ϕ_i, longitude λ_i, and time t_i. For consecutive fixes A and B, the system computes the haversine distance d_{AB} on a sphere of radius R and the speed v_{req} that the move requires.

$$d_{AB} = 2R \arcsin \sqrt{\sin^2 \tfrac{\phi_B - \phi_A}{2} + \cos\phi_A \cos\phi_B \, \sin^2 \tfrac{\lambda_B - \lambda_A}{2}}, \tag{1}$$

$$v_{\text{req}} = \frac{d_{AB}}{\max(t_B - t_A, \epsilon)}, \qquad p_{\text{phys}} = \min\left(1, \frac{v_{\max}}{\max(v_{\text{req}}, \epsilon)}\right). \tag{2}$$

The speed cap is $v_{\max} = 25$ knots, which is 12.861 m/s. The constant $\epsilon = 10^{-6}$ is in seconds inside the first maximum and in m/s inside the second. The plausibility p_{phys} equals 1 when a vessel at the cap could make the move, and it falls as the required speed rises. The Flink job forwards a report to the scorer only when $p_{\text{phys}} \geq 0.3$.

This gate delays the scoring of fast jumps, and it can also drop them. It drops every report whose required speed from the previous stored fix exceeds $v_{\max}/0.3$, which is about 83 knots. The scorer therefore does not see that report when it arrives. The Flink job still adds the dropped fix to the vessel's five-fix history. The next report that passes the gate carries that history to the scorer, and the speed detector uses the fastest segment in it. A jump between about 83 and 300 knots is therefore scored late, when a later report passes the gate. The jump goes unscored when the vessel's next four reports also fail the gate or never arrive, because the history then no longer holds both fixes of the jump. A jump above $12\, v_{\max}$, which is 300 knots, makes every later request that still carries both of its fixes fail with HTTP 422.



(a) The feasible case has $d_{AB} \le L$.

(b) The infeasible case has $d_{AB} > L$.

Figure 2: The space-time prism tests whether one vessel could have moved between two fixes at the speed cap. Fixes A and B lie a distance d_{AB} apart, and the reach length is $L = v_{\max}(t_B - t_A)$, where the code sets $v_{\max}$ to 25 knots. (a) When $d_{AB} \le L$, the feasible positions fill an ellipse with foci A and B, semi-major axis $a = L/2$, and semi-minor axis $b = \frac{1}{2}\sqrt{L^2 - d_{AB}^2}$. A point P on the boundary satisfies $|AP| + |PB| = L$. (b) When $d_{AB} > L$, the reach disks of radius $L/2$ around A and B do not meet at the midpoint time, so no path fits within L.

Space-time prism feasibility. The space-time prism [21, 27] is the set of places that a vessel could occupy between two fixes if it never exceeded $v_{\max}$. Its footprint on the map is an ellipse with foci at A and B, which Figure 2 draws. The reach length L sets the size of that ellipse.

$$L = v_{\max}(t_B - t_A), \qquad \text{feasible} \iff d_{AB} - \delta \le L, \tag{3}$$

$$a = \tfrac{1}{2}L, \qquad b = \tfrac{1}{2}\sqrt{\max\bigl(0,\, L^2 - d_{AB}^2\bigr)}. \tag{4}$$

The terms a and b are the semi-major and semi-minor axes when the pair is feasible, and the code allows a tolerance of $\delta = 10^{-6}$ m. An infeasible pair means that no vessel under the cap could have made the move. On the single-vessel scoring path, the scorer computes this prism for the latest pair of fixes, and the fused score does not read the result. The feasibility test for that pair matches the speed test $v_{\text{req}} \le v_{\max}$ up to the tolerance δ, so the speed detector already scores the same condition. The prism geometry does matter inside the gap detector, which merges candidate gaps whose prism bounding rectangles overlap.

Speed severity. The speed detector uses the fastest segment in the supplied history. Let $v^\star$ be the largest v_{req} over consecutive fixes.

$$s_{\text{speed}} = \begin{cases} 0, & v^\star \le v_{\max}, \\ \min\bigl(1,\ (v^\star/v_{\max} - 1)/3\bigr), & v^\star > v_{\max}. \end{cases} \tag{5}$$

The severity saturates at $4\,v_{\max}$. The scorer rejects a request with HTTP 422 when any segment exceeds $12\,v_{\max}$, and that request receives no score.

Gap severity. A silence longer than the 600 s coverage threshold opens a candidate gap. The gap detector follows the abnormal gap detection of Sharma et al. [36], whose abnormal gap measure bounds a vessel's possible movement during a gap with a space-time prism. Harbormaster keeps the name abnormal gap measure (AGM) for its own formula, which blends kinematic implausibility with a data term.

$$\text{AGM} = w\,(1 - p_{\text{phys}}) + (1 - w)\,P_{\text{data}}, \qquad w = 0.6, \tag{6}$$

$$P_{\text{data}} = \text{logistic}\left(\frac{\ln(1 + d) + 0.001\,\Delta t - 12}{3}\right), \tag{7}$$

$$s_{\text{gap}} = \text{clip}\bigl(0.5\,f_{\text{dur}} + 0.5\,\text{AGM}\bigr), \qquad f_{\text{dur}} = \text{clip}\left(\frac{\Delta t - 600}{7200 - 600}\right). \tag{8}$$

Here d is the gap distance in meters, Δt is the gap duration in seconds, and clip limits a value to $[0, 1]$. The term p_{phys} applies the plausibility rule of Equation (2) to the two fixes that bound the gap. The detector merges candidate gaps whose prisms overlap and scores the earliest gap in each merged

group. The scorer then keeps the group with the largest AGM. Equation (7) is a numpy surrogate. The target design replaces it with a diffusion model trained on the off-cloud cluster, and on AWS the P_{data} term came from the surrogate.

Corridor severity and fusion. The corridor detector measures the distance d_c in meters from the current fix to the nearest lane segment. It can also add an unexpected-turn reason when the history is long enough and the vessel turns sharply far from any waypoint. A watchlist lookup also runs on every request outside the plan, and a watchlist hit or a sanctions hit each adds a reason with a fixed severity. The scorer fuses the severities s_i of all reasons with a noisy-OR rule.

$$s_{\text{corr}} = \text{clip}\left(\frac{d_c - 2000}{6000 - 2000}\right), \tag{9}$$

$$S = 1 - \prod_i (1 - s_i), \qquad c = 1 - 2\min(S,\, 1 - S), \tag{10}$$

$$\text{review} \iff c < 0.7 \;\vee\; S \geq 0.6. \tag{11}$$

The noisy-OR treats the reasons as separate pieces of evidence, so one strong reason can raise the score close to 1 by itself. The confidence c is 1 at the ends of the score range and 0 at $S = 0.5$. The confidence c depends only on S, so the clause $c < 0.7$ holds exactly when $0.15 < S < 0.85$. Equation (11) therefore sends a case to review exactly when $S > 0.15$, and the clause $S \geq 0.6$ changes the outcome only when $S \geq 0.85$. The analyst's label is stored next to the case, and it never overwrites the automated score.

The deterministic planner. The planner chooses detectors from the length of the history, and it uses no language model. It always runs the corridor detector. It adds the prism, speed, and validate steps when the history holds at least one prior fix. It adds the gap detector when the history holds at least three. Algorithm 1 in Appendix C lists the full rule. The validate step runs a kinematic region gate that rejects anchors that imply more than $1.05\, v_{\max}$. The single-vessel scoring path passes it no anchors, so the step has no effect on that path.

5 Replay-Safe Change Data Capture

PostgreSQL 16 on RDS holds the authoritative vessel, watchlist, and sanctions tables, and every other copy of that data is derived (Figure 3). Debezium runs inside Kafka Connect and reads committed changes through logical replication [39, 14]. Kafka delivers each change event at least once, so a consumer restart or a replay can deliver the same event again. In the code, a registry worker takes three actions for each event. It attempts a conditional write of a whole item to the DynamoDB read store, removes the stale Redis cache entry, and appends a row to an Iceberg audit table.

The commit protocol. The worker applies every event in a batch, flushes every sink, and only then commits its Kafka offsets. When a sink fails, the offsets stay uncommitted, and Kafka redelivers the batch. The worker skips Kafka tombstones, which are compaction metadata. A source delete becomes a soft-delete item that the worker writes at the delete's own LSN.

The LSN guard. Each item in the read store keeps the LSN of the change that last wrote it. The worker writes with a DynamoDB conditional put [16]. The condition compares the stored LSN $\ell^\star(k)$ of key k with the incoming LSN ℓ.

$$\textsc{apply}(k, \ell, v) \iff \ell^\star(k) = \perp \;\vee\; \ell^\star(k) < \ell. \tag{12}$$

Here $\perp$ means that the item has no stored LSN yet. A rejected write signals a stale or duplicate delivery, so the worker records the event as not applied and raises no error. The worker applies rows from Debezium's initial snapshot at $\ell = 0$, and the audit row keeps the snapshot's own LSN. A snapshot row therefore applies only to an empty key, and every streamed change outranks it.

Proposition 1 (Replay convergence) *Consider a change log of events $e_j = (k_j, \ell_j, v_j)$ and a projection that starts empty. Assume that (i) for each key, an LSN identifies at most one value, (ii) the check and the write in Equation (12) happen atomically for each item, (iii) each accepted write replaces the whole item, and (iv) every delivered event maps to a valid item key. Let $\mathcal{D}$ be any finite*

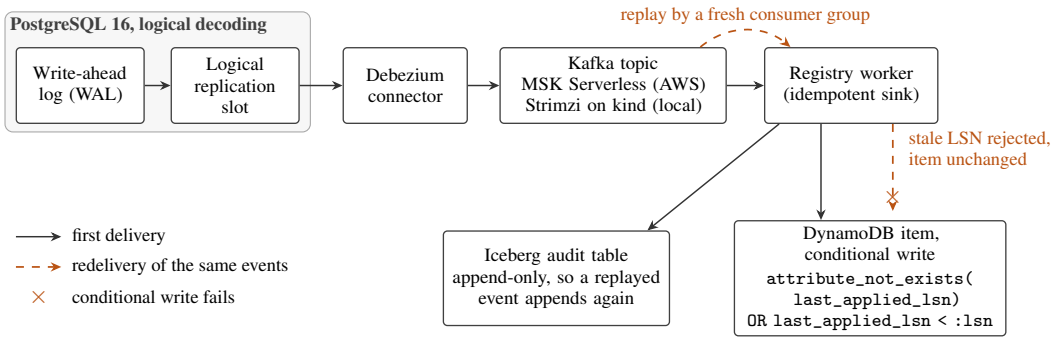


Figure 3: The change data capture (CDC) path copies PostgreSQL 16 changes into DynamoDB. Debezium reads the write-ahead log through a logical replication slot and publishes change events to Kafka. Kafka ran as Amazon MSK Serverless in the bounded AWS window and as Strimzi on kind locally. The registry worker writes each item with the conditional write shown, so the write succeeds only when the item has no applied LSN or holds an older one. A replayed event with a stale LSN fails that condition and leaves the item unchanged. The Iceberg audit table appends every delivered event, so its rows can repeat after a redelivery. No AWS record shows the audit append, so Figure 1 draws that edge dashed.

sequence of deliveries drawn from the log, in any order and with any repetition. After the worker processes $\mathcal{D}$, each key k that appears in $\mathcal{D}$ holds the value of the event with the highest LSN for k in $\mathcal{D}$, and every other key stays empty. In particular, replaying any prefix or suffix of the log after a full pass leaves each projected key at the value of its highest applied LSN.

Appendix C.1 proves the proposition by induction on the length of $\mathcal{D}$. The invariant is that each key seen so far stores the largest LSN among its delivered events, together with that event's value.

The commit protocol redelivers a batch after any transient sink failure. The delivered set therefore eventually covers every event that maps to a valid key. This holds as long as the worker keeps making progress and Kafka keeps each event until the worker commits it. Each key then holds the value of its highest-LSN event, which is its latest committed row or a soft-delete marker. This result also needs a log that holds every committed change, and that condition fails if the replication slot is dropped or invalidated. A re-snapshot after such a loss cannot repair a key that already holds a streamed LSN, because the worker applies snapshot rows at $\ell = 0$ and the guard rejects them. A re-snapshot that carries a different value at LSN zero also breaks assumption (i). The guard then keeps the first snapshot row, and any streamed change still outranks it.

What the guard does not cover. The proposition covers one sink and one key at a time, and the paper claims nothing stronger. Redis invalidations can repeat after a redelivery. The Iceberg audit table appends every delivered event, so a redelivered batch appends its rows again. The worker also buffers audit rows until the batch flush. A crash in the middle of a batch can lose those rows, and the redelivery then records events that were already applied as not applied. The worker also skips a content error, which is an event that no key mapping can apply. It audits that event as not applied and commits past it, so redelivery never repairs that key. A reader can also see one key updated before another, because the guard orders changes per key only. The paper therefore describes this path as an idempotent, LSN-guarded projection. It does not claim that each change takes effect once across every destination.

Slot-lag monitoring. A logical replication slot makes PostgreSQL retain write-ahead log (WAL) until the consumer confirms it, so a stalled consumer makes the database hold disk. The monitor measures slot lag in bytes with `pg_wal_lsn_diff`, and a CloudWatch alarm fires at 200 MiB.

$$\Delta_{\text{slot}}(t) = \text{LSN}_{\text{current}}(t) - \text{LSN}_{\text{confirmed}}(t), \tag{13}$$

$$\text{alarm at minute } t \iff \min_{j \in \{0,1,2\}} M_{t-j} \geq \theta_{\text{slot}}. \tag{14}$$

Here M_t is the largest slot lag that the monitor publishes in one-minute period t, so the alarm needs three consecutive periods at or above the threshold. The threshold θ_{slot} equals $200 \times 1024 \times 1024$ bytes, which is 209,715,200 bytes. The monitor measures lag from the slot's restart position when the slot has no confirmed position. A slot can also fall behind while its captured tables are quiet, because

the connector confirms a position only when it receives a change. A Debezium heartbeat query that upserts one row handles that case, and Section 8.3 reports its effect.

6 Evidence-Gated Promotion

A candidate model reaches traffic only through the ladder in Figure 4. The ladder has up to four steps, which are a holdout gate, a reward-hacking probe, a shadow comparison, and a weighted canary. A failure at any step stops the candidate. A candidate cannot skip the holdout gate, the shadow comparison, or the canary. The probe runs only when the caller supplies a probe result, so supervised retraining candidates skip it.

Holdout gate. The holdout gate requires an AUC of at least 0.85, a mean CRPS of at most 1.0, and a calibration ratio inside $[0.8, 1.2]$. The AUC compares binary anomaly labels with the candidate's anomaly scores. The model also predicts a Gaussian $\mathcal{N}(\mu, \sigma^2)$ for each observation y, and CRPS has a closed form for that case [20].

$$\mathrm{CRPS}\big(\mathcal{N}(\mu,\sigma^2),\, y\big) = \sigma\left[z\left(2\Phi(z)-1\right) + 2\varphi(z) - \frac{1}{\sqrt{\pi}}\right], \qquad z = \frac{y-\mu}{\sigma}, \tag{15}$$

$$\rho = \frac{\overline{e^2}}{\overline{\sigma^2}}, \qquad e = y - \mu. \tag{16}$$

Here Φ and φ are the standard normal distribution and density functions, and a bar denotes a mean over the holdout set. The CRPS threshold is in the units of y. A ratio ρ above 1 means that the errors are larger than the predicted variance, and a ratio below 1 means that they are smaller. Every threshold must pass, so a strong AUC cannot make up for poor calibration.

Reward-hacking probe. The probe compares a preference-tuned candidate with the baseline on a held-out set. Let $\bar{R}$ be the mean total reward, u the rate of structural-constraint violations, and $\bar{s}$ the mean structural term. A sample violates the structural constraint when its structural term falls below zero.

$$\text{block} \iff \bar{R}_{\text{cand}} > \bar{R}_{\text{base}} \;\wedge\; \big(u_{\text{cand}} > u_{\text{base}} \;\vee\; \bar{s}_{\text{cand}} < \bar{s}_{\text{base}}\big). \tag{17}$$

The rule blocks a candidate whose reward rises while its physical consistency gets worse. The second clause catches a candidate that keeps the same number of violations and makes each one more severe.

Shadow comparison. The shadow step scores the same N inputs with the champion and the candidate. It passes when the mean absolute score difference is at most a limit δ_{sh} that the caller sets, and the drills set it to 0.05.

$$\frac{1}{N}\sum_{j=1}^{N}\left|s_j^{\text{champ}} - s_j^{\text{cand}}\right| \le \delta_{\text{sh}}. \tag{18}$$

Canary and burn rate. The canary moves the candidate through the weights 5, 25, 50, and 100. At each weight, a burn-rate check reads the score-success service level objective (SLO). Consider a window W with bad_W failures among total_W requests and an SLO target T. The burn rate divides the observed bad ratio by the allowed error fraction (1 minus the SLO target) [40].

$$B_W = \frac{\mathrm{bad}_W/\mathrm{total}_W}{1-T}, \qquad \text{tier fires} \iff B_{W_{\text{long}}} > \theta \;\wedge\; B_{W_{\text{short}}} > \theta. \tag{19}$$

A window with no requests has $B_W = 0$. The system uses three tiers. A burn rate above 14.4 over 1 h and 5 min pages, and a burn rate above 6 over 6 h and 30 min also pages. A burn rate above 3 over 1 day and 2 h raises a warning. The two paging tiers match the recommended tiers of Thurgood et al. [40], and the warning tier is the project's own choice. A page at any canary weight reverts the full weight to the champion in one action, and a warning does not revert.

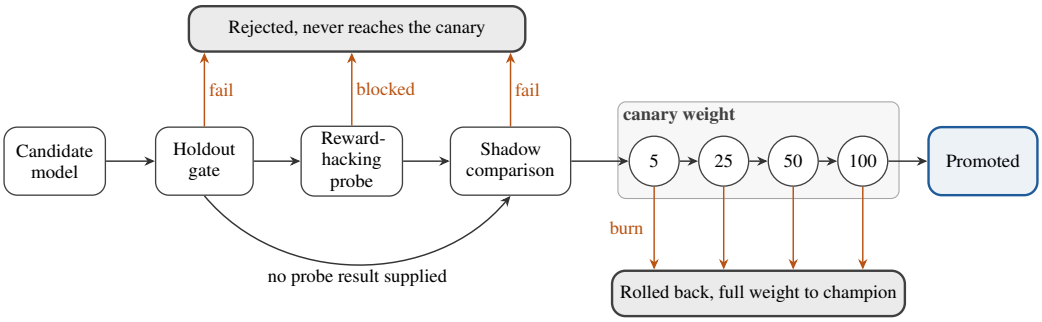


Figure 4: The promotion ladder moves a candidate model through up to four checks. Black edges move the candidate forward, and orange edges end the promotion. The candidate must pass the holdout gate, then the reward-hacking probe when a probe result is supplied, and then the shadow comparison. Every gate has a reject edge, and a rejected candidate never reaches the canary. The canary advances through weights 5, 25, 50, and 100 while the burn rate stays healthy, and a burn at any step returns the full weight to the champion. The gate results reported in this paper come from synthetic drills. The live asynchronous endpoint has one variant, so the canary moved no traffic on AWS.

Drift and the feedback loop. The code includes three detectors that watch for drift after promotion. They cover input shift, calibration, and concept drift, and the concept-drift detector works only through a proxy signal. The input check compares a reference sample with a current sample for each numeric feature, and Appendix C.2 gives its test and thresholds.

The feedback loop ends at preference data. A builder turns analyst labels of `incorrect` into offline preference records, and no code path retrains a model from them. No reinforcement-learning fine-tune ran, so no real candidate has reached the probe. The live endpoint allows one variant, so the canary is traffic-inert there. The shadow and canary results in Section 8.4 therefore come from synthetic drills.

7 Scale-to-Zero Serving

In the design, the heavier model runs behind a SageMaker asynchronous inference endpoint. The endpoint holds a request queue, so it can accept a request while it has no instance and serve that request after it wakes. Its autoscaling target allows between zero and one instance, and two policies share that target. Both policies use `EndpointName` as their only metric dimension.

$$\frac{\text{backlog}(t)}{n_{\text{inst}}(t)} \to 5 \quad (n_{\text{inst}} \geq 1), \qquad n_{\text{inst}} \leftarrow 1 \iff \min\left(\bar{H}_{k-1}, \bar{H}_k\right) \geq 1 \quad (n_{\text{inst}} = 0). \tag{20}$$

- **Track.** A target-tracking policy steers the metric `ApproximateBacklogSizePerInstance` toward a setpoint of 5. When the backlog empties, it scales the endpoint in to zero instances.
- **Wake.** Target tracking cannot see the move from zero instances, so a separate step policy handles it. The step policy sets the capacity to exactly 1 when the average $\bar{H}_k$ of `HasBacklogWithoutCapacity` is at least 1 in two consecutive 60 s periods, and its cooldown is 60 s.

The target caps capacity at one instance, so the tracking policy in practice only scales the endpoint in to zero.

Two dimension bugs. Both policies had a bug in their CloudWatch metric dimensions. An AI audit session compared the SageMaker Terraform with AWS's own canonical examples and returned 19 numbered findings. The session recommended changes and edited nothing itself. Its first finding, which the public repository's incident log records as P25, was that the target-tracking metric specification lacked the `EndpointName` dimension. SageMaker publishes the metric under that dimension, so the policy's alarms queried a series that does not exist. Those alarms never received data, and scale-in to zero could never fire. The audit found the gap from the code alone, before any live symptom appeared, and the fix added the dimension.

In a bounded AWS window, a live check of that fix exposed a second bug, P26. The wake alarm carried an extra `VariantName` dimension, and SageMaker does not publish `HasBacklogWithoutCapacity` under it. The alarm received no datapoints for more than 15 minutes while a real backlog

waited, so the endpoint would never have woken from zero. P26 is not one of the 19 audit findings, and its fix removed the extra dimension. Both bugs left every resource present and every apply clean. Neither bug raised an error, and only a live check of the scaling behavior exposed P26.

Evidence on the endpoint. In a bounded AWS window, a controlled drill woke the endpoint from zero instances. The backlog alarm fired, capacity moved to one, and the same request returned a score. The endpoint then returned to zero instances. The record gives no wake time, so this paper reports none. The endpoint served a labeled demo stand-in image, and AWS rejected a second variant on the asynchronous endpoint. The endpoint therefore stays single-variant.

The Kubernetes path. A KEDA ScaledObject on EKS gives a second scale-from-zero path for a serving deployment. This path ran once in a bounded AWS window and was then torn down, and ECS stayed the front door. Appendix D.3 reports that test.

8 Evaluation

This section reports what was measured and where it ran. Table 1 lists each result with its scope label and the kind of artifact that holds it. BOUNDED-AWS-WINDOW marks a measurement on real AWS resources during one attended, time-boxed window, and its evidence is private. LOCAL marks a run on a local machine or a local stack. SYNTHETIC-DRILL marks a deterministic drill that runs real project code on synthetic inputs without any infrastructure. PUBLIC-REPO marks a result that the public repository states or reproduces. No experiment in this section measures detection accuracy. Appendix D.2 reports the scoring-kernel benchmark, the audit tally, and the test suites from Table 1.

8.1 Stream run under load

The stream run tested the pipeline under a burst in a bounded AWS window (Figure 5). A producer sent 34,123 records into provisioned Kinesis over a 900 s run. The load held 5 records/s, except for a burst of 400 records/s from 60 s to 135 s. An observer polled the Flink backpressure ratio every 2 s and kept 270 samples. Each sample waited for the Flink dashboard to answer, so the samples arrived less often than the poll interval.

Backpressure on the source and key-by vertex first rose above 0.1 at 79.5 s, inside the burst window. Inside that window, its maximum was 0.927 over 22 samples. The ratio kept rising after the burst ended, and it first reached 0.998 about 160 s after the load started. That value is the maximum for the whole run. The ratio cleared below 0.1 at about 694 s, and the final three samples read 0.0. The downstream vertex reported 0.0 throughout. That vertex chains the keyed operator with a print sink, and the keyed operator runs feature computation, the physics gate, a DynamoDB write, and a signed scorer call inline.

CloudWatch reports lag in one-minute buckets, so the lag figures here are one-minute bucket maxima. Kinesis iterator age peaked at 139,000 ms in a single bucket, and that bucket's average was 2,278.7 ms, so the peak was brief. Flink's `millisBehindLatest` also peaked at 139,000 ms, and it held that maximum in four consecutive buckets. Both series then drained to 0. In the same window, KEDA on EKS took 1.89 s from the scale request to the first HTTP 200, and Appendix D.3 explains why this measures a warm cluster.

The run measured the pipeline under load, and it did not measure detection quality. The evidence for this run does not record whether the inline scorer calls succeeded, so the paper makes no claim about them.

8.2 One-hour soak through the ECS front door

The soak tested serving capacity in a bounded AWS window. After a warmup of 60 s and 600 requests, one 3,600 s trial offered 10 requests per second with open-loop arrivals and no retries. Each request passed through the IAM-signed API Gateway route and the VPC Link to the ECS service. The trial returned 35,999 HTTP 200 responses and one HTTP 503, for a success ratio of 0.99997 against a 0.999 gate. Goodput counts a request only when it succeeds within 1,000 ms of its scheduled start, and 35,996 requests met that bound.

Table 1: Each measured result appears with its scope. PUBLIC-REPO means that the public repository states or reproduces the result. A reader can rerun the public test suite, the local CDC tests, and the public scripts for the skew, canary, and slot drills. The soak row carries PUBLIC-REPO because the public README restates it, and it cannot be rerun. The AWS rows come from bounded windows whose evidence is private. The row groups follow Section 7, Sections 8.1 to 8.4, and Appendix D.2.

Result	Value	Scope	Artifact
SageMaker wake from zero	0 to 1 to 0 instances, no wake time recorded	BOUNDED-AWS-WINDOW	acceptance record
Stream run load	34,123 records in 900 s, 5 records/s with a 400 records/s burst from 60 to 135 s	BOUNDED-AWS-WINDOW	load capture
Backpressure, source and key-by vertex	0.927 in the burst window, 0.998 after it, below 0.1 at about 694 s	BOUNDED-AWS-WINDOW	Flink capture
Lag, one-minute maxima	139 s for Kinesis and for Flink, then 0	BOUNDED-AWS-WINDOW	CloudWatch export
Scale request to first HTTP 200, KEDA on EKS	1.89 s	BOUNDED-AWS-WINDOW	scale timeline
One-hour soak outcomes	35,999 of 36,000 HTTP 200, one 503	BOUNDED-AWS-WINDOW, PUBLIC-REPO	run summary, README
Soak latency, p95 and p99	client 142.751 and 240.283 ms, server 7.048 and 9.348 ms	BOUNDED-AWS-WINDOW	run summary
CDC acceptance on MSK Serverless	drills A to E passed, replay hash unchanged	BOUNDED-AWS-WINDOW	acceptance record
Local CDC end-to-end tests	5 passed in 36.41 s on a fresh stack	LOCAL, PUBLIC-REPO	drill record, README
Insert to read store	0.57 s against a 5 s target	LOCAL	drill record
Slot growth and drain	74,035,000 bytes, then 0	LOCAL	drill record
Stale event after a newer one	guarded severity held at 0.95	SYNTHETIC-DRILL	drill record
Candidate with serving skew	holdout passed, shadow 0.6056 against 0.05	SYNTHETIC-DRILL	drill record
Canary with planted regression	reverted at weight 25	SYNTHETIC-DRILL	drill record
Scoring kernel latency	p95 0.6109 ms on one thread	LOCAL	benchmark record
AI audit of SageMaker Terraform	19 findings in four groups	LOCAL	audit record
Public snapshot test suite	821 passed, 12 skipped, 83.72% coverage	PUBLIC-REPO	README
Private regression suite	1,633 passed, 21 skipped	LOCAL	gate record

Over the 35,999 successes, client latency had a p50 of 101.418 ms, a p95 of 142.751 ms, and a p99 of 240.283 ms, with a maximum of 1,474.254 ms. Server latency had a p50 of 5.219 ms, a p95 of 7.048 ms, and a p99 of 9.348 ms, with a maximum of 45.451 ms. The client called the public API Gateway endpoint, so client latency includes the network path, and server latency is the time inside the service. The largest start lag of the client was 60.583 ms, below its 100 ms limit, so the load generator kept its schedule.

The soak repeated one fixed request whose history held one fix. The planner therefore ran the corridor, prism, speed, and validate steps, and it never ran the gap detector or the P_{data} term. The soak also started prewarmed after a successful preflight, so it supports no cold-start claim. It measures serving capacity, and it measures no detection quality.

8.3 Change data capture

A bounded AWS window ran the CDC path on an MSK Serverless cluster, with Debezium Connect and the worker on Fargate. Five drills passed. This evidence is private, and the public repository cannot reproduce it. Drill A flagged a controlled watchlist entry, and the scorer returned a score of 1 with human review required. The entry was then deleted. Drill B replayed the topic with a fresh consumer group. The replay read 3 tombstones, which the worker skips, and the LSN guard rejected 3 change events. The 3 DynamoDB items and their content hash did not change. Drill C delivered

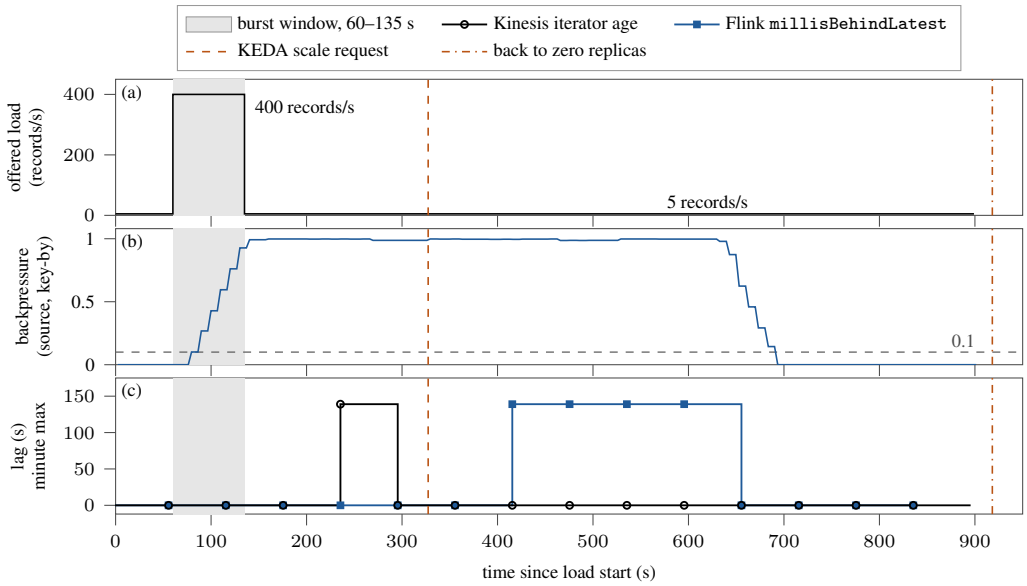


Figure 5: The stream run shows offered load, backpressure, and lag from one bounded AWS window. (a) The load generator cycled the synthetic replay fixture into provisioned Kinesis at 5 records/s, with a burst of 400 records/s from 60 to 135 s. Managed Flink consumed the stream. The run measured the pipeline under load and did not measure detection quality. (b) Backpressure on the source and key-by vertex reached 0.927 inside the gray burst window and 0.998 after it. It dropped to the dashed 0.1 line or below near 694 s. (c) CloudWatch reports lag at one-minute resolution, so each step holds a per-minute maximum. Both series peaked at a lag of 139 s and then drained to 0. The vertical lines mark a separate KEDA on EKS scale-from-zero test in the same window. The dashed line marks the scale request at 327.37 s. The first HTTP 200 came 1.89 s later, at 329.26 s, so the figure does not draw it as a separate line. The deployment returned to zero replicas at 918.24 s.

4 writes that spanned a forced redeployment of Kafka Connect. Drill E paused Connect, and 48 writes built a slot lag of 27,408 to 27,464 bytes. The alarm fired after 3 one-minute datapoints on a temporary 16 KiB threshold, and the normal threshold of 209,715,200 bytes was then restored. The drill tried other thresholds only to calibrate the alarm. The drill D cleanup later soft-deleted the 52 records that drills C and E wrote. After the drills, Terraform reconciliation reported no changes, and the connector and its task were running.

The same window exposed a slot that grew while the connector still reported RUNNING. Before the repair, slot lag was 1,006,637,152 bytes. A Debezium heartbeat query now upserts one row, so the slot advances when the source tables are quiet. After the fix, slot lag fell to 4,328 bytes. The same path also runs on a local stack, and Appendix D.1 reports those runs.

Two drills isolate the failures that the design guards against. In a local slot drill on a throwaway PostgreSQL 16 container, a logical slot with no consumer grew at every one of five write rounds, from 6,926,136 to 74,035,000 bytes. The lag alert fired at a drill threshold of 65,536 bytes, and draining the slot returned the lag to 0. A synthetic duplicate drill ran the guard against an in-memory sink. When a stale event at LSN 2000 arrived after LSN 3000, the unguarded sink regressed the severity from 0.95 to 0.9, and the guarded sink kept 0.95. After a crash between the sink acknowledgement and the offset commit, the guard rejected all five redeliveries, and the state hash matched the baseline.

8.4 Promotion gate drills

Two synthetic drills run the real gate code with a synthetic toy scorer and synthetic AIS-shaped inputs, with no AWS. In the skew drill, a candidate had a bug in its online feature encoding. The holdout gate passed it with an AUC of 1.0000, a CRPS of 0.0563, and a calibration ratio of 0.9955. The drill derives its labels by thresholding the same scores, and it draws its observations from the predicted Gaussian. These holdout values therefore follow from the construction and do not measure a model. The shadow comparison then failed the candidate, with a mean absolute difference of 0.6056 and a maximum of 0.9363 against the 0.05 limit. The holdout set could not see a bug that lived only in the serving path, and the shadow step caught it.

In the canary drill, a candidate passed the holdout gate with an AUC of 1.0000, a CRPS of 0.5807, and a calibration ratio of 1.0567. This drill also builds its holdout set so that the candidate passes. The candidate then passed the shadow step with a mean absolute difference of 0.0088. The canary advanced to weight 5 and met a planted regression at weight 25. The rollback step then returned the full weight to the champion, and the final status was `rolled_back`. The drill injects a burn check that reports a page at weight 25, so it tests the revert path and does not exercise Equation (19).

9 Lessons and Limitations

What is not built. The paper claims none of the following as running.

- Candidate V3, a model candidate, is not deployed. The AWS endpoint served only a labeled demo stand-in image, so no trained model produced a served score.
- No reinforcement-learning fine-tune ran, and the feedback loop ends at offline preference data.
- Distributed tracing was never built, and the paper reports no drift monitoring on AWS.
- CDC on AWS ran only in bounded acceptance windows. The canary is single-variant and traffic-inert on the live endpoint.
- The stream carried synthetic load, and the ingestor replays only a synthetic fixture. The Flink job has no event-time windows or watermarks, and its gate scores the fast jumps in Section 4 late or drops them.
- The lane release publisher is assumed integration, and the review console ran only locally.
- The Workflow 5 job service, dispatch worker, and result collector are not built.

Lessons from the evidence. A resource that exists and raises no error is weak evidence that its mechanism works. The P25 and P26 alarms were present and quiet while scale-in and wake-up could never fire. The useful check is a live test of the behavior that the resource provides.

An HTTP 200 does not show that the intended detector ran. In one AWS streaming window, Flink kept one fix per vessel, so the planner never added the gap detector while every request still returned 200. The fix keeps five fixes, which clears the three-fix threshold of the gaps step. A code review found a related silent fallback. The AWS task definition never injected the database credential, so the review queue had fallen back to process memory.

A replication slot holds disk until a consumer drains it. The local slot drill grew a slot at every write round, and the AWS window found a slot at 1,006,637,152 bytes while the connector reported RUNNING. Slot lag therefore needs its own alarm, and quiet source tables need a heartbeat.

A holdout gate cannot see a bug that lives only in the serving path. The skew drill showed this case directly. A summary can also misstate its artifact, so this paper takes every number from the raw artifact and labels its scope. For example, 0.927 is the burst-window maximum, and the raw capture shows the run maximum of 0.998 after the burst.

10 Conclusion

Harbormaster keeps each guarantee small and checkable, so that its alerts stay attributable, reviewable, and recoverable. Closed-form rules produce every score, and the paper states which jumps the stream gate scores late or drops. The CDC path gives a per-key replay guarantee with a short proof, and it names the effects that can repeat. The promotion ladder turns model trust into gates that produce evidence. In synthetic drills, the shadow step caught a skewed candidate that the holdout gate passed, and the canary reverted a planted regression. The measured results are small and scoped, and each one names where it ran. The next steps are a trained model on the endpoint, a canary that can move traffic, and CDC that runs outside attended windows.

References

[1] Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. MillWheel: Fault-tolerant stream processing at internet scale. *Proceedings of the VLDB Endowment*, 6(11):1033–1044, 2013. doi: 10.14778/2536222.2536229.

[2] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. The Dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8(12):1792–1803, 2015. doi: 10.14778/2824032.2824076.

[3] Amazon Kinesis. Amazon Kinesis Data Streams. https://aws.amazon.com/kinesis/data-streams/. Amazon Web Services.

[4] Amazon Managed Service. Amazon Managed Service for Apache Flink. https://aws.amazon.com/managed-service-apache-flink/. Amazon Web Services.

[5] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016. URL https://arxiv.org/abs/1606.06565.

[6] Andreas Andreakis and Ioannis Papapanagiotou. DBLog: A watermark based change-data-capture framework. *arXiv preprint arXiv:2010.12597*, 2020. URL https://arxiv.org/abs/2010.12597.

[7] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. TFX: A TensorFlow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395, 2017. doi: 10.1145/3097983.3098021.

[8] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy, editors. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, 2016. ISBN 978-1-491-92912-4. URL https://sre.google/sre-book/table-of-contents/.

[9] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley. The ML Test Score: A rubric for ML production readiness and technical debt reduction. In *Proceedings of the IEEE International Conference on Big Data*, pages 1123–1132, 2017. doi: 10.1109/BigData.2017.8258038.

[10] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates. In *Advances in Knowledge Discovery and Data Mining (PAKDD)*, Lecture Notes in Computer Science, pages 160–172. Springer, 2013. doi: 10.1007/978-3-642-37456-2_14.

[11] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache Flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 38(4):28–38, 2015. URL http://sites.computer.org/debull/A15dec/p28.pdf.

[12] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. State management in Apache Flink: Consistent stateful distributed stream processing. *Proceedings of the VLDB Endowment*, 10(12):1718–1729, 2017. doi: 10.14778/3137765.3137777.

[13] Štěpán Davidovič and Betsy Beyer. Canary Analysis Service. *Communications of the ACM*, 61(5):54–62, 2018. doi: 10.1145/3190566.

[14] Debezium. Debezium Connector for PostgreSQL. https://debezium.io/documentation/reference/stable/connectors/postgresql.html.

[15] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, 1973. doi: 10.3138/FM57-6770-U75U-7727.

[16] Mostafa Elhemali, Niall Gallagher, Nick Gordon, Joseph Idziorek, Richard Krog, Colin Lazier, Erben Mo, Akhilesh Mritunjai, Somasundaram Perianayagam, Tim Rath, Swami Sivasubramanian, James Christopher Sorenson, III, Sroaj Sosothikul, Doug Terry, and Akshat Vig. Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*, pages 1037–1048. USENIX Association, 2022. URL https://www.usenix.org/conference/atc22/presentation/elhemali.

[17] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006. doi: 10.1016/j.patrec.2005.10.010.

[18] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):1–37, 2014. doi: 10.1145/2523813.

[19] Tilmann Gneiting and Adrian E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007. doi: 10.1198/016214506000001437.

[20] Tilmann Gneiting, Adrian E. Raftery, Anton H. Westveld, and Tom Goldman. Calibrated probabilistic forecasting using ensemble model output statistics and minimum CRPS estimation. *Monthly Weather Review*, 133(5):1098–1118, 2005. doi: 10.1175/MWR2904.1.

[21] Torsten Hägerstrand. What about people in regional science? *Papers of the Regional Science Association*, 24(1):6–21, 1970. doi: 10.1007/BF01936872.

[22] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. Cloud programming simplified: A Berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*, 2019. URL https://arxiv.org/abs/1902.03383.

[23] Martin Kleppmann. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, 2017. ISBN 978-1-449-37332-0.

[24] Jay Kreps, Neha Narkhede, and Jun Rao. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB Workshop*, Athens, Greece, 2011. URL https://www.microsoft.com/en-us/research/wp-content/uploads/2017/09/Kafka.pdf.

[25] Ze Li, Qian Cheng, Ken Hsieh, Yingnong Dang, Peng Huang, Pankaj Singh, Xinsheng Yang, Qingwei Lin, Youjiang Wu, Sebastien Levy, and Murali Chintalapati. Gandalf: An intelligent, end-to-end analytics service for safe deployment in large-scale cloud infrastructure. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 389–402. USENIX Association, 2020. URL https://www.usenix.org/conference/nsdi20/presentation/li.

[26] Frank J. Massey. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951. doi: 10.1080/01621459.1951.10500769.

[27] Harvey J. Miller. Modelling accessibility using space-time prism concepts within geographical information systems. *International Journal of Geographical Information Systems*, 5(3):287–301, 1991. doi: 10.1080/02693799108927856.

[28] Giuliana Pallotta, Michele Vespe, and Karna Bryan. Vessel pattern knowledge discovery from AIS data: A framework for anomaly detection and route prediction. *Entropy*, 15(6):2218–2245, 2013. doi: 10.3390/e15062218.

[29] Joaquin Quiñonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence, editors. *Dataset Shift in Machine Learning*. Neural Information Processing Series. The MIT Press, Cambridge, MA, 2009. ISBN 978-0-262-17005-5. doi: 10.7551/mitpress/9780262170055.001.0001.

[30] Stephan Rabanser, Stephan Günnemann, and Zachary C. Lipton. Failing loudly: An empirical study of methods for detecting dataset shift. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/846c260d715e5b854ffad5f70a516c88-Abstract.html.

[31] Akond Rahman, Rezvan Mahdavi-Hezaveh, and Laurie Williams. A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108:65–77, 2019. doi: 10.1016/j.infsof.2018.12.004.

[32] Maria Riveiro, Giuliana Pallotta, and Michele Vespe. Maritime anomaly detection: A review. *WIREs Data Mining and Knowledge Discovery*, 8(5):e1266, 2018. doi: 10.1002/widm.1266.

[33] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems*, volume 28, 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html.

[34] Shreya Shankar, Rolando Garcia, Joseph M. Hellerstein, and Aditya G. Parameswaran. Operationalizing machine learning: An interview study. *arXiv preprint arXiv:2209.09125*, 2022. URL https://arxiv.org/abs/2209.09125.

[35] Arun Sharma and Shashi Shekhar. Analyzing trajectory gaps to find possible rendezvous region. *ACM Transactions on Intelligent Systems and Technology*, 13(3):1–23, 2022. doi: 10.1145/3467977.

[36] Arun Sharma, Subhankar Ghosh, and Shashi Shekhar. Physics-based abnormal trajectory gap detection. *ACM Transactions on Intelligent Systems and Technology*, 15(5):1–31, 2024. doi: 10.1145/3673235.

[37] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems*, volume 35, pages 9460–9471, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/3d719fee332caa23d5038b8a90e81796-Abstract-Conference.html.

[38] The Apache Software Foundation. Apache Iceberg. https://iceberg.apache.org/.

[39] The PostgreSQL Global Development Group. PostgreSQL documentation: Logical replication. https://www.postgresql.org/docs/current/logical-replication.html.

[40] Steven Thurgood, Jess Frame, Anthony Lenton, Carmela Quinito, Anton Tolchanov, and Nejc Trdin. Alerting on SLOs. In Betsy Beyer, Niall Richard Murphy, David K. Rensin, Kent Kawahara, and Stephen Thorne, editors, *The Site Reliability Workbook: Practical Ways to Implement SRE*, chapter 5. O'Reilly Media, 2018. ISBN 978-1-492-02950-2. URL https://sre.google/workbook/alerting-on-slos/.

[41] Heather Welch, Tyler Clavelle, Timothy D. White, Megan A. Cimino, Jennifer Van Osdel, Timothy Hochberg, David Kroodsma, and Elliott L. Hazen. Hot spots of unseen fishing vessels. *Science Advances*, 8(44):eabq2109, 2022. doi: 10.1126/sciadv.abq2109.

[42] Bilal Yurdakul and Joshua Naranjo. Statistical properties of the population stability index. *The Journal of Risk Model Validation*, 14(4):89–100, 2020. doi: 10.21314/JRMV.2020.227.

A Reproducibility and Disclosure Boundary

What the public repository reproduces. The public repository is https://github.com/arunshar/harbormaster-maritime-ai, and it carries the MIT License. It holds the code, the tests, the Terraform modules, the synthetic replay fixture, the load-test scripts, and the scripts for the skew, canary, and slot drills. A reader can run its unit and integration suite without cloud credentials. At public commit fcac977, that suite reported 821 passed and 12 skipped tests, and later commits to the repository may change the count. A reader with Docker, kind, and kubectl can start the local CDC stack and run its five end-to-end checks. The coverage figure of 83.72% comes from a local run, because CI does not run coverage. The kernel benchmark script is public, and its measured numbers are not.

What stays private. The AWS evidence stays private. It includes the stream-run captures, the soak summary, the CDC acceptance record, the SageMaker acceptance record, and the teardown plan. These files contain account and resource identifiers, so the paper cites them by kind only. The public README states the soak outcome and the client latency. It also states that the repository claims no end-to-end CDC run on AWS. The AI audit record, the private regression suite, the drill transcripts, the duplicate-drill script, and the local CDC run records are private as well.

What the paper never claims.

- The paper claims no detection accuracy, no false-alert rate, and no result on real labeled cases.
- It never claims that a trained model served traffic on AWS.
- It claims no input other than synthetic load and the synthetic replay fixture.
- It never claims that each change takes effect once across every destination.
- It claims no SLO, no cold-start time, and no SageMaker wake time.
- It never claims that CDC ran on AWS outside bounded windows, and it reports no drift monitoring on AWS.

Build record. One git author made the 226 commits behind this paper. GitHub lists 49 pull requests for the private repository, and 48 were merged. Coding agents served as tooling. Git history holds 44 merge commits, and 38 of them carry the agent's branch prefix. Agents ran no Terraform apply and no push. The author ran every apply, push, login, and live window, and this record is self-reported.

B The Full System Board

Figure 6 places all five workflows on one page, together with the stores that they share. The board shows the integrated target design, and it does not show a deployed system.

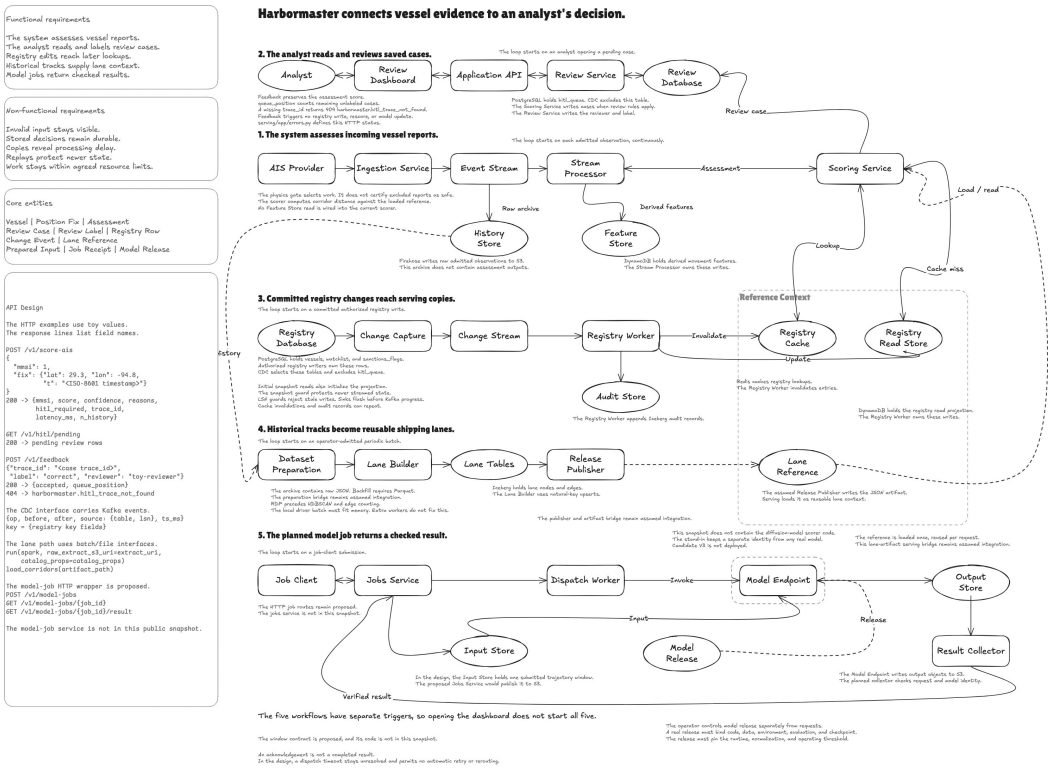


Figure 6: The five-workflow board, adapted from the public repository, shows the integrated target design of Harbormaster. It is a design map, not a picture of a deployed system. Several paths on the board are labeled as proposed or as assumed integration, and the board's own notes mark what the public snapshot does not contain. The AIS Provider box is a design placeholder, and only the synthetic replay fixture ran. The board draws the Workflow 5 Jobs Service, Dispatch Worker, and Result Collector with solid lines, and none of them is built. Zoom the PDF to read the notes, and the source SVG is in the public repository.

C Method Details

Algorithm 1 gives the rule that the scoring path in Section 4 follows for one report.

Algorithm 1 The deterministic scoring path handles one report.

Require: current fix x, history h of prior fixes
Sort (h, x) by time, drop fixes whose timestamp repeats, and let n be the number of remaining fixes minus one
if any segment of the sorted track needs a speed above $12\, v_{\max}$ **then**
    return HTTP 422 with no score
end if
$\mathcal{P} \leftarrow \{\text{corridor}\}$
if $n \geq 1$ **then**
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{\text{prism},\ \text{speed},\ \text{validate}\}$
end if
if $n \geq 3$ **then**
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{\text{gaps}\}$
end if
Run every detector in $\mathcal{P}$ and the watchlist lookup, and collect the severities s_i
Compute S and c with Equation (10)
Queue a review case when Equation (11) holds
return S, c, the reasons, and a trace identifier

C.1 Proof of the replay proposition

Proof. The proof uses induction on the length of $\mathcal{D}$ with one invariant. After any prefix of $\mathcal{D}$, each key seen so far stores the largest LSN $m(k)$ among its delivered events, together with that event's value, and every unseen key is empty. The invariant holds for the empty prefix. Suppose the next delivery is (k, ℓ, v). Assumption (iv) ensures that the delivery reaches the guard for key k. If k is empty, the condition holds, the write succeeds, and $m(k) = \ell$. If $\ell^\star(k) < \ell$, the write succeeds, and ℓ becomes the new maximum. If $\ell^\star(k) \geq \ell$, the condition fails and the item keeps its state. The maximum also stays the same, because $\ell \leq m(k)$. When $\ell = m(k)$, assumption (i) says that the stored value already equals v. Assumption (ii) ensures that no other write changes $\ell^\star(k)$ between the check and the put. Assumption (iii) ensures that the stored value comes from a single event and never mixes fields from two events. Each write touches one item, so every other key keeps the invariant. The final state therefore depends only on the set of delivered events. A replayed prefix or suffix adds no event outside that set, so the state stays the same. □

C.2 The input drift check

The input check in Section 6 compares a reference sample with a current sample for each numeric feature. It bins both samples into at most 10 equal-frequency bins whose edges are the distinct reference deciles. The outer edges are open, and each fraction has a floor of 10^{-6}.

$$\mathrm{PSI} = \sum_{b=1}^{B} \left(q_b^{\mathrm{cur}} - q_b^{\mathrm{ref}}\right) \ln \frac{q_b^{\mathrm{cur}}}{q_b^{\mathrm{ref}}}, \quad B \leq 10, \qquad D_{\mathrm{KS}} = \sup_x \left|F_{\mathrm{ref}}(x) - F_{\mathrm{cur}}(x)\right|, \tag{21}$$

$$p = Q_{\mathrm{KS}}\left(\left(\sqrt{n_e} + 0.12 + \frac{0.11}{\sqrt{n_e}}\right) D_{\mathrm{KS}}\right), \qquad n_e = \frac{nm}{n+m}. \tag{22}$$

Here q_b^{ref} and q_b^{cur} are the reference and current fractions in bin b, and F is an empirical distribution function. The function Q_{KS} is the survival function of the Kolmogorov distribution, and n and m are the two sample sizes. This p-value uses a standard asymptotic approximation to the Kolmogorov distribution with the effective sample size n_e. A feature drifts when $\mathrm{PSI} \geq 0.25$, or when $\mathrm{PSI} \geq 0.1$ and $p < 0.05$.

D Additional Measurements

D.1 Local change data capture runs

On the local stack, PostgreSQL 16 with logical decoding feeds Strimzi-managed Kafka on kind. In the first local run, a new watchlist row reached DynamoDB Local 0.57 s after the PostgreSQL insert, against a 5 s target, and all five end-to-end tests passed in 33.50 s. A second run on a fresh stack followed a change to the registry key schema. Its smoke test took 4.28 s, and the five tests passed in 36.41 s. Two attempts before this fresh-stack run recorded one slow result of 5.06 s and one warm-topic pass of 0.55 s.

D.2 Kernel latency, audit, and test suites

A local benchmark timed the single-event scoring kernel on one thread over 1,000 iterations after 200 warmup iterations. The request carried a 121-fix history, the watchlist was off, and no endpoint supplied P_{data}. The kernel had a p95 of 0.6109 ms and a p99 of 0.7483 ms, and its throughput was 1,760.2 scores/s. The public benchmark document describes the method, and it prints no measured numbers.

The AI audit session in Section 7 returned 19 findings in four groups. Two findings were marked as required fixes, seven were hardening items, six became decision records, and four needed no change. The second required fix was a SageMaker Model Package Group that nothing created, and a commit provisioned it.

The public snapshot's test command reported 821 passed and 12 skipped tests, with no failures. The 12 skipped tests need a live PostgreSQL database. Local line and branch coverage over the five packages was 83.72%, above an 80% floor, and CI does not run coverage. These are the only test figures that a reader can reproduce. The private regression gate passed 1,633 tests and skipped 21. A separate isolated suite of 57 first-party Lambda tests also passed, and the paper never adds the two counts.

D.3 The Kubernetes scaling test

In the design, a KEDA ScaledObject can also scale a serving deployment on EKS from a Kinesis trigger, with a minimum of zero replicas. This Kubernetes path was proven once in a bounded AWS window and then torn down. KEDA on EKS moved the deployment from zero replicas to one ready replica, and the time from the scale request to the first HTTP 200 was 1.89 s. That figure measures a warm cluster that scaled one pod, so it is not an SLO and not a cold-start result. The deployment later returned to zero replicas. A scheduled teardown guard then deleted the EKS node group and the cluster, and the window rolled back to ECS. The final Terraform plan showed 0 resources to add, 0 to change, and 0 to destroy. A decision record kept ECS as the front door, because ECS was already the rollback path and needs no cluster operations.