WSTS 2026

Timing Integrity

for AI / Agent Pipelines

When timing faults masquerade as model regressions

Teams debug models first — and suspect time last.

Anand Ozarkar

Why AI Systems Make This Uniquely Dangerous

Traditional System with Clock Skew

- 📋 Clock skew → wrong timestamps
- 🚨 Explicit error or alert fires
- 🔍 Team investigates time layer
- ✅ Root cause found quickly

VS

AI Pipeline with Clock Skew

- 🕐 Clock skew → corrupted features
- 🤖 Model interprets corruption as signal
- 📉 Behavior drifts — no alert fires
- 🔬 **Team debugs model for 1+ days**

AI systems don't report timing failures. They absorb them — and call them something else.

Where Time Enters an AI Pipeline

Ingest	Features	Serving	Decisions	Humans
Event stamping	Window joins	Cache staleness	Rate-limit buckets	Approval deadlines
Out-of-order arrivals	TTL freshness	Retrieval freshness	Policy expiry	Audit timestamps
Source clock drift	Lag features	Context age	Retry ordering	SLA windows

⟵ Time confidence must flow as a first-class signal through every stage ⟶

Two failure surfaces we'll examine: bot-detection pipeline · recommendation pipeline

A Taxonomy of Timing Failures in AI Systems

Failure Type	What the Team Observes	AI / System Risk
Telemetry holes → bursts	Windows under-count, then over-count. Dashboards look noisy.	Corrupt feature distributions
Sequence violations	Out-of-order events degrade order-dependent features.	Idempotency and causal order broken
Rate-limit mis-bucketing	Skew moves requests across minute/second boundaries.	False throttles or attacker bypass
Expiry mis-enforcement	Policy TTLs and cache freshness inconsistent across services.	Stale consent / expired policy active
Retry Heisenbugs	Retries collide with delayed originals — duplicate side effects.	Double-actions, double-charges
Silent AI drift	Behavior shifts gradually—no obvious fault or alert.	**Team debugs model for days; time is root cause**

The Incident: A Story You'll Recognize

We had a bot-detection pipeline.

One day, false positives started climbing. Quietly. No alert fired.

We did what any team would do: **we blamed the model.**

"Model regression?"

Check last deploy
Diff feature weights

"Feature drift?"

Audit data pipeline
Review distributions

"Threshold tuning?"

Re-evaluate cutoffs
Re-run validation

More than a day of debugging. Wrong layer. Entire time.

What Was Actually Happening

1 **One telemetry host drifts**

Clock lags behind fleet time

→

2 **Wrong timestamps written**

Requests stamped with stale time

→

3 **Windows have holes → bursts**

Feature aggregates silently broken

→

4 **Bot-detection FP rate rises**

Legit traffic mis-classified at scale

→

5 **Team debugs the model**

>1 day lost on wrong layer

⚠ **Impact Moment**

During the entire debug window, bot traffic that should have been blocked was passing undetected. Every hour spent tuning model thresholds was another hour the system was making wrong decisions at scale — compounding silently, with no alert.

Skew didn't trigger a timing alert. It triggered a model regression investigation.

Root cause: a single telemetry host with a lagging clock. Fixed in minutes once found.

The Same Failure — Different Domain

🎾 Tennis String Recommendation

1 User buys new tennis racket

2. Preference update enters event stream

3. Update arrives 45 min late — outside window

4. Ranking uses stale context → recommends old strings

5. Team debugs retrieval logic and cache TTLs

6. **Root cause: event-time skew in preference pipeline**

The Pattern Repeats Across Domains

🛡 **Fraud / Bot Detection**

Skew corrupts behavioral windows

📊 **Incident Detection**

Delayed telemetry creates alert lag

🎯 **Recommendation**

Stale context degrades relevance

🖱 **Ads / Click Integrity**

Mis-bucketed events shift attribution

🔑 **Auth Rate-Limiting**

Skew enables TTL bypass

The structural fault is identical. The AI surface makes it look different every time.

The Agentic AI Problem Is Worse

Classical pipelines make one decision. Agents make chains — timing errors compound at every step.

RAG / Retrieval Staleness

Context retrieved at T0 is stale by T1 when the agent acts. Long-running agents are most exposed — retrieved facts expire mid-chain.

Tool-Call Retry Duplicates

A timed-out call retries but the original completes late. Agent sees duplicate results or triggers duplicate side effects.

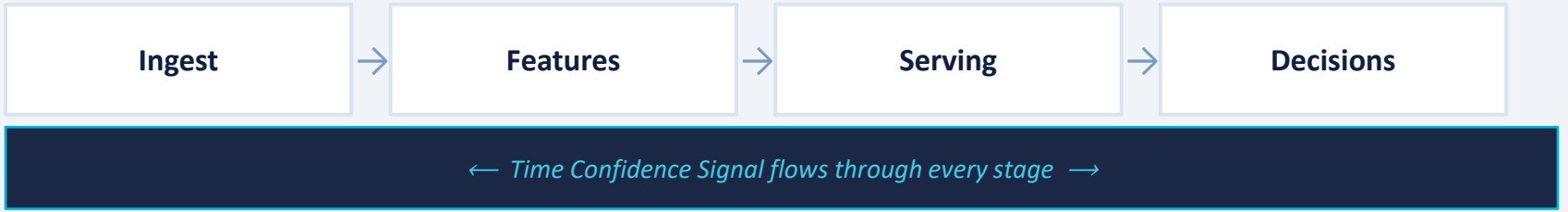
Multi-Agent Ordering Breaks

In orchestrator/worker topologies, skew between agents breaks causal ordering. Agent B acts on state Agent A has not produced yet.

The same TrustedTime + tiered-action patterns apply — and matter even more when agents act autonomously.

The Fix: Make Time Quality Explicit

Four patterns that give AI pipelines time awareness — without rebuilding from scratch.



01

TrustedTime API

App code gets now() + confidence + source
Never call system clock directly for decisions

02

Monotonic Epochs

Rate-limit counters and windows keyed to
monotonic epochs — immune to wall-clock drift

03

Event-Time Windows

Watermarks + late-data compensation
Drift-tolerant streaming aggregates

04

Tiered Actions

Final (≥0.9) / Provisional (≥0.7) / Read-only
System degrades safely when time degrades

Pattern 01: TrustedTime API

```
# Interface (language-agnostic)

class TrustedTime:
  def now() -> datetime     # current time
  def monotonic() -> float        # non-decreasing
  def confidence() -> float        # 0.0 to 1.0
  def source() -> str          # 'ptp','nts','gps'

# Confidence weights
conf = (0.40 * c_offset
      + 0.25 * c_jitter
      + 0.20 * c_freshness
      + 0.15 * c_diversity)

# Use it
t = TrustedTime()
if t.confidence() >= 0.9:
    take_final_action()
```

What This Gives You

✅ **Time quality is explicit**

Confidence is a first-class value in code—not hidden in infra

✅ **Testable**

Mock TrustedTime in tests. Inject confidence values. No magic.

✅ **Source-aware**

Know if you're on GPS, NTS, PTP, or holdover

✅ **Tiering ready**

Confidence value drives provisional / final decisions directly

Don't call System.currentTimeMillis() for decisions. Call TrustedTime().confidence() first.

Hot Path vs Recovery Path

HOT PATH — online, low-latency

Monotonic epoch buckets

Rate-limits and counters never move backward

Causal sequence features

Build on (epoch, local_counter++) — not wall clock

Confidence check at entry

Block or tier-down if conf < threshold

Provisional flag on writes

Low-confidence events quarantined, not discarded

⇄

RECOVERY PATH — async, compensating

Event-time windows + watermarks

Late arrivals trigger compensating updates

Re-score when confidence recovers

Finalization daemon promotes provisionals

Backfill quarantined records

Recompute aggregates over repaired window

Rollback on timeout + high risk

Safe default: undo if deferral too long

Clock Health is an Operational Contract

Clock-Health SLO Targets

Metric	Target	Gate
offset_p99	≤ 2 ms	⛔ Deploy
offset stability 5min	≤ 1 ms change	⛔ Deploy
wander_p95	≤ 0.5 ms/min	⛔ Deploy
confidence duty cycle	≥ 99.9% @ 0.8	⛔ Deploy + Runtime
sync_loss_rate	≤ 0.01%/min/region	⚠ Alert

When SLOs Go Red

🚫 **Block risky deploys**

CI/CD gate fails — don't ship to degraded fleet

⬇ **Degrade to provisional**

Runtime auto-tiers from Final → Provisional

📋 **Queue for human review**

High-risk decisions held for manual approval

📣 **Alert before KPIs collapse**

Time SLO fires before business metrics move

When time health degrades, the system must not continue pretending everything is fine.

Test Skew Like Latency and Failover

If a pipeline is time-sensitive, it deserves time-fault tests.

Constant

+10ms offset
held steady

`inject_skew(p)`

Step

Sudden +20 ms
jump at T+30m

`inject_skew(p)`

Sawtooth

0→25 ms cycling
every 5 min

`inject_skew(p)`

Asymmetric

One-way 8 ms
path skew

`inject_skew(p)`

Assert in CI:

🛡 **Fraud / Abuse**

assert: Precision ≥ 0.95 · Recall ≥ 0.96

⏱ **Rate Limits**

assert: No false throttles · No attacker bypass

📣 **Recommendation**

assert: Ranking drift < threshold · Freshness intact

⚙ **Gate Behavior**

assert: Provisional tier fires at correct confidence threshold

Same harness reuses across: fraud · incidents · recommendations · ads · identity

Operations Playbook: What to Do When It Happens

DETECT

- Alert on confidence duty-cycle dips
- Watch for window 'holes then bursts'
- Track false-positive trends vs time-quality metrics
- Monitor sync_loss_rate per region

CONTAIN

- Quarantine low-confidence writers
- Switch read paths to signed/trusted sources
- Drop to provisional tier for risky actions
- Drain the lagging host from traffic

RECOVER

- Backfill or compensate late-arriving events
- Extend late-data horizon temporarily
- Re-score or re-run decisions over repaired window
- Gradually promote quarantined records

PREVENT

- Enforce SLO gates in CI/CD
- Add skew tests to staging pipeline
- Add clock-health dashboards and runbooks
- Schema: add time_confidence + epoch fields

Schema migration: add time_confidence (float) · time_source (enum) · epoch (uint64) to every event record

Where to Start: Three Entry Points

You don't need all four patterns on day one.

Start Here

Skew-Injection Harness

Inject constant and step skew into staging. Measure your key pipeline metrics. This alone tells you whether your system is more time-sensitive than you think — usually in one day.

Effort: Low · Value: Immediate diagnosis

Next

TrustedTime API + Confidence Metadata

Wrap your clock calls. Add time_confidence and time_source to event records. This gives you observability and unlocks tiered decision logic without changing the hot path.

Effort: Medium · Value: Observability + tiering

Then

Clock-Health SLOs + CI/CD Gates

Define offset_p99, wander_p95, and confidence duty cycle as first-class SLOs. Wire them into your deploy gate and runtime degradation policy. Now timing is an operational contract.

Effort: Medium · Value: Prevention + auto-degradation

Five Things to Take Away

1. **Ask not only 'did the model regress?' but also 'did time quality change?'**
2. AI systems absorb timing failures—they don't report them. That's what makes them dangerous.
3. Expose time confidence in code and data. Make it a first-class observable signal.
4. Gate risky automation on Clock-Health SLOs — before business metrics move.
5. Start with the skew-injection harness. One day of testing beats one day of wrong debugging.

This is not really a talk about clocks. **It is a talk about keeping AI systems trustworthy when time lies.**

Questions

Anand Ozarkar · WSTS 2026