

Forum:[Hack Forums](#)

Thread: [Enterprise Probabilistic Graph Analysis Engine](#)

Dystortion - January 28, 2026, 3:06 pm

Enterprise Probabilistic Graph Analysis Engine Statistical Framework (ZINB / HMM / Change-Points / Bootstrap + Correlation-Aware Fusion)

TL;DR

This engine turns sparse, messy, time-dependent event data into calibrated signals with uncertainty, and then fuses + propagates them across a dependency graph without failing under noise, regime changes, or numerical edge cases.

Context

I've been working alongside individuals in the hopes of building a production probabilistic analysis engine. It's a statistical inference pipeline sitting underneath a dependency graph decision system aimed to hit AWS.

I can't share the exact final commercial scoring functional code (NDA + obvious reasons of it being a multi million dollar project). What I can share is the statistical architecture I created.

Why most scoring systems fail in production

Most scoring systems fail for the same boring reasons: They treat zeros as “nothing” instead of asking whether the system was inactive, blind, gated, or truly absent.

They assume stationarity (fixed parameters) even though real systems flip behaviour under load, deployments, outages, seasonality, and human intervention.

They output a point score and pretend it's truth, with no confidence intervals, no calibration, no uncertainty measures.

They fuse signals linearly and ignore correlation (double-counting the same evidence through multiple features).

They collapse on edge cases: small sample sizes, extreme sparsity, regime changes, and messy ingestion.

0) Setup

We model a directed graph $G=(V,E)$. Each edge $e=(u \rightarrow v)$ has an observed time series of event counts, and we infer latent edge parameters / states and produce a calibrated score with uncertainty.

Four estimation layers generate signals + uncertainty bounds, then a correlation-aware fusion stage combines them, then results are propagated across the graph with stability constraints.

What each layer contributes

Layer 1 (ZIP/ZINB / sparse counts):

Stops you hallucinating certainty from data that's mostly zeros.

Layer 2 (HMM regime detection):

Captures repeating behaviour modes (normal / stressed / degraded / noisy / post-change stabilisation). Pooling across regimes smears parameters and breaks calibration.

Layer 3 (Change-points):

Detects structural breaks where the generating process permanently changed (deployments, policies, topology changes).

Layer 4 (Uncertainty):

This is where downstream decision-making becomes rational instead of up in the air.

Layer 1) Sparse + Overdispersed Count Modeling (ZIP / ZINB family)

Most people assume a clean Poisson process. Real operational counts show: excess zeros (structural inactivity, gating, downtime, or true absence)
overdispersion (variance > mean)

A standard baseline we use is Zero-Inflated Negative Binomial (ZINB). The point is not "fancy", it's honest uncertainty when data is messy.

One common Negative Binomial parameterization (mean μ , dispersion ϕ) is:

Likelihood decomposition (handles the structural-zero mass correctly):

Estimation

We fit via EM-style latent decomposition where Z_t indicates whether a zero is structural. The key responsibility term (for $x_t=0$) is:

Then the structural-zero probability update is:

For (μ, ϕ) the M-step is typically numerical (bounded transforms + curvature checks + fallbacks). In production, you can't pretend optimisers always behave.

Layer 2) Temporal Regime Detection (HMM with Layer-1 emissions)

Static parameters break when behaviour shifts. We use an HMM with emissions matched to Layer 1 (ZIP/ZINB/NB depending on edge characteristics).

Forward-backward recursion (computed in log-space in practice):

State count selection is done via penalized likelihood criteria (BIC-style) plus sanity checks on occupancy/stability.

Numerical safety

Everything is implemented with log-domain primitives to prevent probability underflow:

Layer 3) Change-Point Detection (structural breaks)

HMMs capture recurring regimes. Some changes are structural. We detect breakpoints using an ensemble (offline optimal segmentation + online detection), then fuse candidates via local likelihood gain + agreement.

Calibration is the difference between math and dressing up as a fairy imo
Anyone can output a score in $[0,1]$. The hard part is making it mean something.

A calibrated system obeys statements like: If I say "0.80 confidence", it should be correct ~80% of the time (within error bands).

95% intervals should contain the true value ~95% of the time across simulation recovery tests.
When calibration drifts, the system should detect it and degrade gracefully.

Score Fusion (correlation-aware, calibration-first)

Each layer produces a signal and uncertainty. Combining them as a naive weighted average is mathematically lazy because it ignores dependence between signals.

Signal fusion without double-counting evidence

Two traps kill fused scoring: Correlation trap: multiple signals are derived from the same raw evidence, so a blend counts one observation multiple times.

Dominance trap: one noisy component dominates because it has scale advantage, not actual information.

So fusion has to be: scale-aware (map signals into comparable probability space),
correlation-aware (covariance structure matters),
calibration-first (better-calibrated often beats "sharper but wrong"),
stability constrained (weights shouldn't whip around due to minor input changes).

Baseline: inverse-variance weighting

Correlated case: GLS weights using estimated covariance

Fisher-information connection (intuition)

Final score is calibrated onto [0,1] and validated with reliability curves + proper scoring rules (not just "it looks right").

Production hardening

The difference between "works on my test data" and "survives production" is usually: numerical stability (log-space, clipping, guarding, overflow/underflow)
optimizer failure paths (fallback solvers, bounded transforms, restarts)
degenerate data (all zeros, single spike, missing windows, abrupt gaps)
monitoring (track drift + calibration, not just latency)
graceful degradation (bad input reduces confidence, not uptime)

Limitations

No statistical system gets a free pass: Sparse data has ceilings: if there's no signal, uncertainty stays high.

Model misspecification exists: mitigated via robustness tests + calibration monitoring.

Change-points are probabilistic: "true break" vs "temporary anomaly" isn't always clean.

Graphs amplify errors: propagation must be damped/capped and audited.

Validation Discipline

Simulation recovery: generate known DGPs, verify bias/MSE/coverage + regime/changepoint recovery.

Calibration: reliability curves, PIT checks (where applicable), coverage sanity.

Robustness: perturb priors/penalties, stress misspecification, degradation under corruption.

Production monitoring: drift detection, numeric warning telemetry, alerting on calibration degradation.

Thanks to @junooo@ for assisting with her kindness

newQuery - January 29, 2026, 2:34 am

calibrating confidence solving the "Boy who cried wolf" problem in monitoring like auto-scales or firewalls :)

thanks for your time spent and this golden share !

Dystortion - January 31, 2026, 9:35 am

(Jan 29, 2026 03:34 AM)newQuery Wrote: [\[CITE\]](#) calibrating confidence solving the "Boy who cried wolf" problem in monitoring like auto-scales or firewalls :)

thanks for your time spent and this golden share !

Excellent lmao, thank you for reading and enjoying. Not the easiest to inject

Hackers - February 10, 2026, 7:29 pm

Very nice, what use cases do you think this would be best for on HF?

I love algos and especially combining them. If you have enough data and enough algos, you can come up with some pretty interesting insights and with AI it's easy to quantify certain things but then have an LLM filter out the best for summaries & alerts.

Dystortion - February 11, 2026, 3:09 pm

(Feb 10, 2026 08:29 PM)nodes Wrote: [\[CITE\]](#) Very nice, what use cases do you think this would be

best for on HF?

I love algos and especially combining them. If you have enough data and enough algos, you can come up with some pretty interesting insights and with AI it's easy to quantify certain things but then have an LLM filter out the best for summaries & alerts.

I have to be careful with what I say in regards to what can be used of this. But for use cases on HF - it can be modified and scripted to build a multi-scoring bot detection system. It's just a general interference engine - Y went from regime A -> B, change-point detected, confidence rose, propagation capped - this allows removal of any guessing games that could be being played.