Threat prioritization under uncertainty

A decision-support prototype. Simulated data throughout; every constant is a synthetic placeholder and represents no real system.

The problem it addresses

A decision-maker facing more tracks than they can attend to, with information that is incomplete, unequally trustworthy, and ageing at different rates, has two distinct questions:

What do I act on now? and **what might I be about to get wrong?**

These have different answers. A track that is dangerous *and* well characterised belongs at the top of the first list and the bottom of the second — there is nothing left to learn about it. A track that might be nothing or might be everything can sit mid-table on consequence while being the single most valuable thing to look at again.

Systems that produce one ranked list answer the first question and silently discard the second. This prototype produces both, from one belief state, side by side.

How uncertainty is represented

Every field of a track is a distribution, not a number. There is no separate "confidence score" bolted on beside the data; confidence *is* the width of the distributions.

Classification is a **Dirichlet** over categories. Its concentration parameters are literally an evidence count: a sensor report that says "looks ballistic" increments one of them. This yields the class probabilities, the confidence in those probabilities, and a sampler, from one object with no special-casing. A track with no evidence has maximum entropy and minimum concentration, and looks that way on screen without any code path for "unknown."

Kinematics are **Gaussian**, which is what makes the staleness model honest.

Missing fields are replaced by wide **population priors**, not dropped and not scored as zero. A track with nothing known about it is still rankable; it simply ranks as *unknown, and therefore worth looking at*, which falls out of the model rather than being coded as an exception.

Three questions a skeptical engineer asks

1. Why doesn't uncertainty raise the score?

Because it is already represented, in the width. Adding an uncertainty bonus to the mean would count it twice.

Ranking is done by drawing 2,000 samples per track from its belief state, computing a score in each sampled world, and ranking **per world** before aggregating. A high-threat, high-uncertainty track therefore gets a wide rank band — not a promotion. What it *does* get is a high attention score, which is the correct place for it.

When risk aversion is wanted, the **risk functional** changes, not the score: rank by the CVaR of the worst decile instead of the mean. That is a one-line change and it is the principled way

to express "be paranoid," with a knob that can be turned in front of someone rather than a constant buried in a formula.

A subtlety this makes visible: because scores are sampled, a track can rank above its neighbour on the mean while losing to it head-to-head in most sampled worlds — which happens when it scores enormously in a minority of worlds and zero in the rest. The interface reports that explicitly rather than hiding it behind a rank number.

2. Why is staleness modelled as variance growth rather than a penalty?

A freshness penalty (`score -= k · age`) is arbitrary, and it says something false. When data goes stale the mean is not wrong — it is the last thing we knew. What degrades is certainty. So:

$$\sigma^2(t) = \sigma_0^2 + q\cdot\Delta t + (\mathrm{rel}\cdot\Delta t)^2$$

This is the covariance propagation step of a filter with the measurement update removed, which is exactly the situation being modelled. The quadratic term is the one that matters: if the closing-rate estimate is uncertain, then not looking for Δt seconds produces an arrival-time error proportional to Δt, so the variance grows with Δt^2. A linear-only model was the first implementation, and it understated the effect badly enough that a track nobody had observed for a minute was treated as almost as well known as one under continuous observation.

Consequences that fall out with no additional code: a stale dangerous track *stays* dangerous and holds its action rank, while climbing the attention rank because it is now worth re-observing. Classification ages more slowly than kinematics, because a ballistic object does not become a UAS while we look away.

3. What is the attention column actually computing?

Not entropy. Sorting by uncertainty is the lazy version and it is wrong — it puts a completely unknown piece of debris above everything else. Uncertainty only matters where it could change what we do.

The attention score is the **expected regret attributable to a track**. For each sampled world, compare the committed top-K against the top-K that world would have chosen; the tracks that differ form pairs of "wrongly excluded" and "wrongly included," and the score gap of each pair is credited to both members, because resolving either one's uncertainty would have prevented that swap. Averaged over samples, this gives per-track value of information.

The behaviours this produces are the reason to prefer it:

- A dangerous track we are **certain** about scores **exactly zero**. It is never mis-ranked, so another look buys nothing.
- A track with enormous variance that is negligible in every sampled world also scores zero. Unknown and harmless stays unknown.
- A track that lands inside the committed set in some worlds and outside it in others scores high. That is precisely where more information changes the decision.

These are pinned by tests with hand-computable answers rather than by snapshots. The load-bearing one: track A deterministic at score 5, track B a fair coin over 0 and 10. Identical

means, entirely different information value. The analytic answer is 2.5 for both, and the test asserts it to a tolerance of 1e-12 — because the input is exact, so the output must be, and a loose tolerance is where a subtly broken Monte Carlo hides.

Two design decisions worth naming

Urgency is not actionability. The score includes *slack* — time to impact minus the time required to respond to a track of that class. When slack goes negative the action value is zero, because nothing can be done. A track eight seconds out against a thirty-second response requirement therefore leaves the actionable list, explicitly labelled, rather than sitting at the top of it. This looks wrong for about four seconds and then looks obviously right. Because slack is sampled, a track near the boundary is actionable in some worlds and not others, which the interface reports as a fraction rather than rounding to a boolean.

Ranks are shown as ties when they are ties. The engine reports how often each track actually outscores the one below it. When that is near a coin flip, the displayed ordering is not something the samples support, and the interface says so with the figure rather than presenting two confident ranks. On a typical board, action positions 2 and 3 are genuinely tied and stay tied at 8,000 draws — a real property of the board, not a sampling artefact.

The operator is a first-class constraint

A person can pin a track into the committed set, suppress one ("not acting on this now"), or assert a consequence multiplier. Three properties make this more than a UI affordance:

Constraints never touch the belief state. An override is a separate object on the track, never merged into `assetValue` or `classEvidence`. Someone saying "this matters more than you think" is not evidence, and writing it into a distribution would launder a preference into an observation — after which nothing downstream could tell them apart. The display always shows what the model concluded and what a person imposed, side by side.

A constraint changes what we are committed to, not what we believe. That is the whole mechanism, and it is why the feature required no new mathematics. The expected-regret machinery already prices whatever set we are actually committed to; changing that set by hand means the attention column immediately reports what the change costs. Suppress a genuinely top-ranked track and it becomes the top attention priority, carrying the exact expected regret its exclusion introduced. The system does not argue with the operator — it carries out the instruction and reports the price, in the same units as everything else.

Every constraint is replayable. Overrides are routed through the same event reducer as sensor reports and scenario beats, so an operator decision appears in the frame dump, survives a reset, and replays identically. An override that could not be replayed would be the one non-reproducible thing in an otherwise reproducible system.

What it is not

No multi-source fusion. No asset or interceptor assignment. No geospatial view. No learned models — every number is inspectable and every weight lives in one documented file, because the goal is a prioritization core that can be argued with, not one that hides its reasoning.

Reproducibility

Seeded throughout. The full 90-second scenario runs headless with no browser, and two runs produce byte-identical output. Scenario beats are asserted by tests rather than described in prose, so a change that stops a behaviour from occurring fails the suite instead of quietly becoming a different demo.

69 tests. The engine directory imports nothing outside itself and can be run, tested, and challenged on its own.