Why i built an audit engine for hidden instability

From: **Joseph.Maxwell**
Joseph.Maxwell02@proton.me

To: **Joseph.Maxwell02@proton.me**
Joseph.Maxwell02@proton.me

On: **Thursday, 18 June 2026 at 20:59:26**

Most engineering systems are not difficult to analyse because they fail.
They are difficult because they continue working while the structure supporting that output is changing.
A model can retain a good test score while its feature rankings become unstable.
A material can carry load while fatigue history has narrowed its remaining admissible futures.
An institution can maintain throughput while recovery capacity is being consumed.
A language model can produce fluent text while instruction hierarchy and context orientation degrade.
The common failure is not visible output collapse.
The common failure is mistaking visible output for structural viability.
This article describes the engineering route I have been building around that distinction: a constrained state-space grammar, a CTAP-based perturbation audit, and a validation path for testing whether hidden instability can be measured before ordinary performance metrics make the failure obvious.

1. The modelling problem

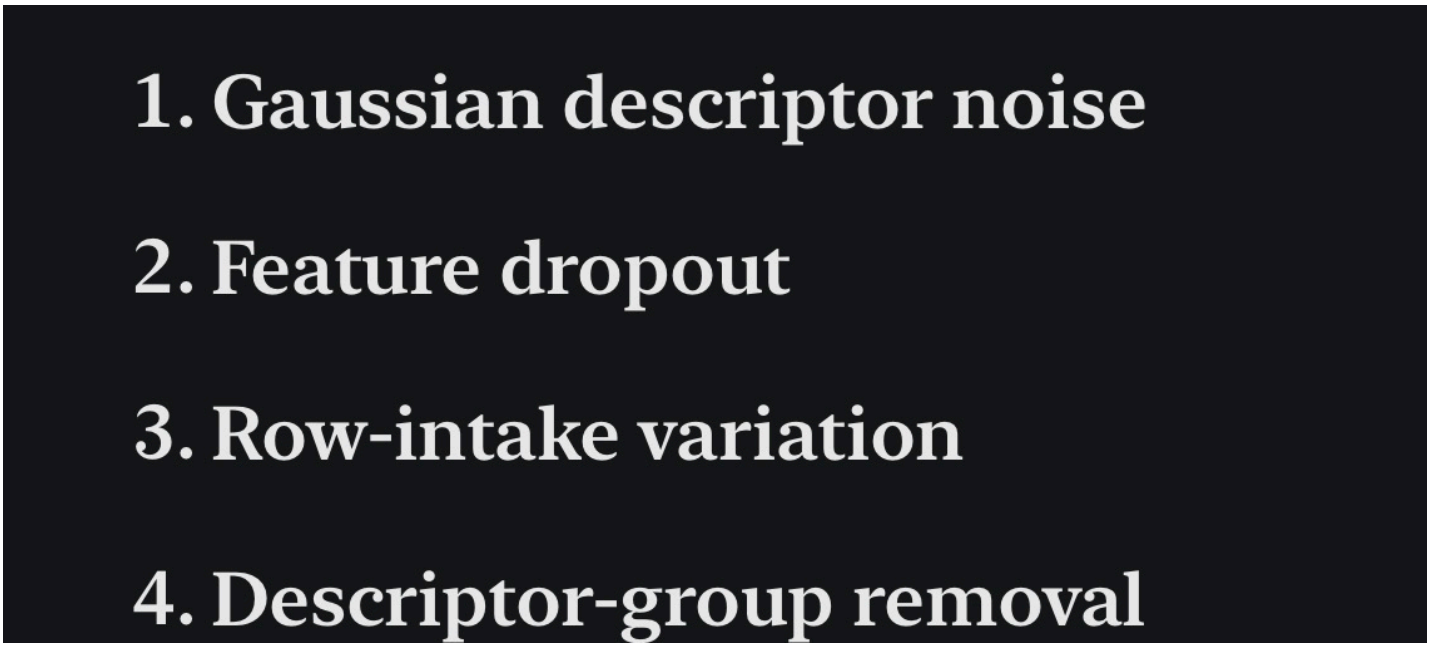
The usual description says:
"The system failed over time."
But elapsed time is not the mechanism.
Two systems can experience the same duration and end in different structural states.
The difference is not the interval itself. The difference is what accumulated, what dissipated, what remained recoverable, and what future states became inaccessible.
So the modelling question becomes:
How do we represent accumulated unresolved constraint, altered reachability, temporal coherence, and recovery capacity separately?
The minimal grammar I am using is:

$$x(t+1) = T[x(t), C(t), P(t), \Pi(t), \rho(t), N(t)]$$

Where:
$x(t)$ = current system state
$C(t)$ = active constraint field
$P(t)$ = persistence or retained structural influence
$\Pi(t)$ = CTAP, the accumulated unresolved constraint
$\rho(t)$ = recovery or dissipation capacity
$N(t)$ = noise
T = transition operator
The point of this expression is not to claim a universal physical law.
It is a modelling grammar. It forces the analyst to declare what is often compressed into phrases like "over time", "under stress", "memory", "recovery", or "drift".

2. CTAP: the accumulation variable
CTAP stands for:
Constrained Temporal Accumulation Parameter.
In the abstract grammar, I denote it by Π.
The basic discrete update is:
$\Pi(t+1) = \Pi(t) + U(t) - \rho(t)$
Where:
$U(t)$ = unresolved load or constraint accumulation
$\rho(t)$ = recovery or dissipation capacity
A continuous version would read:
$\Pi(t)$ = accumulated sum of $[U(t) - \rho(t)]$ over the relevant transition history
The important claim is simple:
Equal elapsed duration does not imply equal CTAP.
That is the shift.
A system does not become fragile because time passed.
It becomes fragile because unresolved constraint accumulated faster than it could be dissipated, or because the basin that previously made recovery possible deformed.
This makes CTAP different from instantaneous stress.
Stress = current load.
CTAP = unresolved accumulation over transition history relative to recovery capacity.
3. Memory as altered reachability
The second object is persistence.
I am not treating memory as stored record alone. I am treating memory as altered future accessibility.
Let:
$R(x)$ = reachable set from state x
H = prior history
Then memory exists when:
$R(x \mid H) \neq R(x \mid \text{reset})$
A persistence quantity can be written as:
$P_H(x) = d_R [R(x \mid H), R(x \mid \text{reset})]$
Where:
$P_H(x)$ = persistence under history H
d_R = distance or difference between reachable-state sets
This is a prospective definition.
It does not ask only:
“What record of the past exists?”
It asks:
“What futures has the past made easier, harder, impossible, or newly accessible?”
That matters in engineering because many systems remember structurally without storing a clean history.
A material remembers loading through fatigue state.
A model remembers data and perturbation history through feature stability and prediction geometry.
An institution remembers policy through changed operational affordances.
A cognitive system remembers load history through thresholds, access, and recovery cost.

The observable test is future divergence.
If two systems look similar now but have different reachable futures because of different histories, the framework says memory is present.
4. Temporal coherence: orientation within retained information
The third object is temporal coherence.
A system can retain information and still lose orientation within it.
That distinction matters especially in AI, institutions, and cognition.
Let:
$\hat{x}(t)$ = system's internal estimate of its state
$x(t)$ = operational state
Then localisation error can be written as:
$E_L(t) = d_X [\hat{x}(t), x(t)]$
Where:
$E_L(t)$ = localisation error
d_X = distance or mismatch in the declared state space
A simple temporal coherence score is:
$TC(t) = 1 / [1 + E_L(t)]$
More generally:
$TC(t) = f [E_L(t), O(t), A(t)]$
Where:
$O(t)$ = ordering confidence
$A(t)$ = anchor stability
This separates storage from orientation.
A language model can still contain the relevant tokens while losing instruction hierarchy.
An organisation can keep documents while losing the context that made those documents meaningful.
A person can remember fragments while losing sequence, priority, or current placement.
In this grammar:
Persistence defines how history changes future accessibility.
Temporal coherence defines whether the system remains oriented within that history.
This is one of the most important separations in the whole framework.
5. From theory to audit
The practical engineering question is:
Can these objects be measured under controlled perturbation?
That is where the perturbation audit comes in.
Instead of only asking whether a model performs well at baseline, the audit asks how the model deforms when controlled pressure is applied.
Perturbation families include:



The baseline model score is treated as visible output.
The audit layer tracks structural reliability.
The audit asks:
Do predictions drift?
Do descriptor rankings remain stable?
Does variance inflate?
Does performance degrade?
Do model-family rankings change?
Does the system return to baseline after perturbation release?
Does instability appear before ordinary error metrics make it obvious?
This converts the theoretical grammar into an engineering procedure.
6. Operational CTAP
The first operational scalar was:
CTAP(k) = 1/4 × [prediction_drift(k) + ranking_instability(k) + variance_inflation(k) + performance_loss(k)]
Where:
k = perturbation level
prediction_drift(k) = change in predictions relative to baseline
ranking_instability(k) = instability of descriptor or feature rankings
variance_inflation(k) = increased variance across runs, seeds, or perturbations
performance_loss(k) = degradation in ordinary metrics such as MAE, RMSE, or R^2
But this scalar is not the theory.
The theory object is:
Π = accumulated unresolved constraint
The measured estimator is:
CTAP_audit = operational instability estimate under perturbation
The mature form is a component vector:
CTAP_vector(k) = [
prediction_drift(k),
ranking_instability(k),
variance_inflation(k),
performance_loss(k)

]
A scalar score is only a projection:
CTAP_scalar(k) = w · CTAP_vector(k)
Where:
w = declared weighting vector
That distinction matters.
If the scalar is treated as canonical too early, the audit becomes another hidden compression. If the vector is reported, the analyst can see what kind of instability is actually carrying the signal.
7. The non-performance CTAP test
The most obvious criticism is:
“If CTAP includes performance loss, maybe it is just error degradation renamed.”
So the next estimator removes performance loss:
CTAP_no_perf(k) = 1/3 × [prediction_drift(k) + ranking_instability(k) + variance_inflation(k)]
This asks a sharper question:
Does instability remain visible when direct performance loss is excluded?
If yes, CTAP is not reducible to ordinary error metrics.
If no, the estimator collapses back into performance loss and the claim weakens.
That is the correct falsification pressure.
8. The materials audit
The first operational demonstration used a materials-modelling task.
The target was shear modulus.
The cleaned dataset had 1,939 rows and seven numeric descriptors:
formation_energy_per_atom
density
band_gap
total_magnetization
bulk
elastic_anisotropy
poisson
The strongest baseline family was HistGradientBoosting, with approximately:
$R^2 = 0.897$
MAE = 4.239
RMSE = 11.328
On a normal modelling report, that looks like a strong result.
But the audit asks a different question:
Does the model remain structurally reliable under perturbation?
The first shear audit found:
CTAP: 0.000 → 0.346
TC: 1.000 → 0.798
The largest CTAP-minus-performance gap was modest:
0.052
Mean final recovery error was:
0.051
So the result was useful, but not overclaimed.
It did not prove universal early warning.

It showed that audit variables can reveal structure not exhausted by baseline performance.

9. Descriptor persistence

The descriptor persistence result is conceptually important because it operationalises memory-as-reachability in model space.

A descriptor that remains recurrent and rank-stable under perturbation continues to constrain reachable model behaviours.

A descriptor that appears useful at baseline but loses recurrence has weaker persistence, even if it contributed to ordinary performance.

In the shear audit, persistent descriptors included:

bulk = 0.939

density = 0.903

poisson = 0.878

Less persistent descriptors included:

band_gap = 0.563

formation_energy_per_atom = 0.669

This does not prove material causality.

It shows how retained influence can be audited under perturbation.

That is the engineering value.

10. Robustness and estimator hygiene

The next robustness pass extended the audit to bulk and band-gap targets.

The question was no longer:

Can CTAP be computed?

It was:

Does CTAP remain informative when performance loss is removed?

The answer was mixed but useful.

Non-performance CTAP showed separation from raw performance loss in multiple targets, which strengthens the irreducibility claim.

But the estimator was also variance-dominated.

At the target-level summary, the non-performance CTAP component shares were approximately:

shear: 0.933 variance share

bulk: 0.958 variance share

band_gap: 0.861 variance share

This is not a failure.

It is estimator information.

It means the scalar score is not self-interpreting.

The scientifically mature reading is:

CTAP is currently strongest as a component profile or audit vector, not as a canonical scalar index.

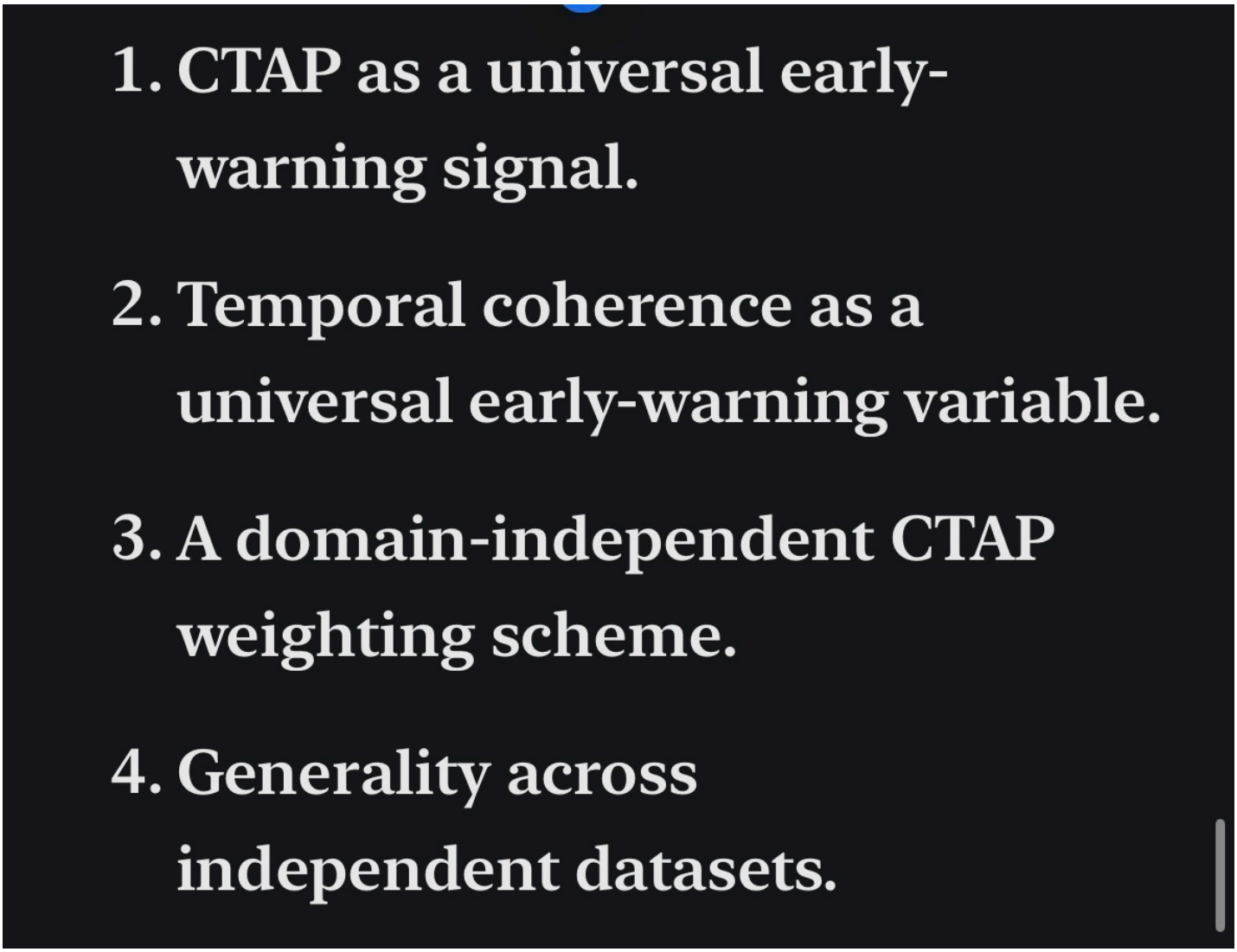
That is a better result than pretending the score is cleaner than it is.

11. What the current evidence supports

The current evidence supports:

1. CTAP-like quantities can be operationalised.
2. CTAP is not simply raw error renamed.
3. Descriptor persistence can operationalise memory-as-reachability in model space.
4. Recovery is not always perfect reset.
5. Scalar CTAP is not yet canonical.
6. Component-vector reporting is currently stronger than headline scoring.

The current evidence does not yet strongly support:



That distinction is important.
The work is stronger if the claim boundary is preserved.
12. Validation route A: independent materials replication
The first validation route is independent materials replication.
Run the same perturbation audit across independent datasets and target properties.
Test:
CTAP_no_perf
against ordinary performance loss.
Ask whether CTAP profiles replicate across:
elastic properties
formation energy
band gap
thermal properties
different descriptor sets
different model families
The key question is:
Does the audit profile reveal instability not visible in baseline performance?
13. Validation route B: long-context AI audit
The second route is a long-context AI audit.
This is probably the fairest test for temporal coherence.
Construct long contexts with:
layered instructions

corrections
examples
stale information
distractor blocks
conflicting priorities
provenance requirements
Then measure:
instruction hierarchy preservation
ordering accuracy
provenance tracking
prompt-order sensitivity
contradiction handling
recovery after summarisation
recovery after rollback
output fluency versus task orientation
The claim is supported if temporal coherence degrades before visible output quality collapses.
The key failure pattern would be:
The information remains present, but the system loses orientation within it.
That is temporal coherence failure.
14. Validation route C: cognitive-state simulation
A cognition-based simulation can test the same temporal-coherence problem in a more mechanistic way.
For example, Cognimatica Modelo Mechanisto defines cognitive state as:
x(t) = {R(t), Ap(t), SNR(t), C_exec(t), T(t), B(t)}
Where:
R = representational geometry
Ap = aperture or working access
SNR = signal-to-noise ratio
C_exec = executive stability
T = threat or evaluative pressure
B = energetic or temporal reserve
Effective bandwidth is:
B_eff(t) = Ap(t) × SNR(t) × C_exec(t)
Error morphology can then be modelled as:
E(t) = {E_omit, E_dist, E_seq, E_freeze}
With freeze-like collapse driven by stacked interaction:
freeze_stack = T × (1 - Ap) × (1 - B)
E_freeze ≈ sigmoid[alpha × (freeze_stack - threshold)] × R
This route would test whether temporal-coherence markers such as sequencing error, ordering instability, regime-localisation uncertainty, or reaction-time variance rise before visible output collapse.
That would be a direct simulation test of the CTAP-temporal-coherence pairing.
15. Why this is engineering, not metaphor
The difference between metaphor and engineering is exposure to failure.
A metaphor can always be reinterpreted.
An engineering model has to survive declared tests.

This framework weakens if:
CTAP_no_perf collapses into ordinary performance loss.
Descriptor persistence adds no information.
Temporal coherence cannot be operationalised.
Recovery is always symmetric.
Cross-domain examples cannot declare state variables, constraints, transitions, and failure thresholds.
That is the point.
The system is designed to be attacked.
16. The actual contribution
The contribution is not that every system is the same.
The contribution is a disciplined separation:
elapsed time versus accumulated unresolved constraint
output versus structural viability
storage versus orientation
memory versus reachability
recovery versus reset
scalar score versus component audit profile
That separation changes the engineering question.
Instead of asking only:
“Does the model still work?”
we ask:
“What is changing underneath the fact that it still works?”
That is the useful question.
Because by the time visible output collapses, the important transition may already have happened