Zentrube: Entropy Redefined —

A Practical White Paper (v1.8)

Framework: **Shunyaya**
Formula: **Zentrube**
Status: Draft for Public Release
Date: 27 Aug 2025

Abstract

This paper introduces **Zentrube**, a compact, time-aware entropy formula for modeling drift in data, signals, and symbolic systems. While it draws conceptual inspiration from the broader Shunyaya framework — which reimagines zero and entropy as dynamic, living baselines rather than static placeholders — the technical focus here is on a rigorous, reproducible kernel. At its core, Zentrube treats zero not as emptiness but as a baseline of alignment: a living reference point around which systems may drift, rupture, or recover.

The canonical form of the entropy formula is:
$\mathbf{Zentrube_t = \log(Var(x_{0:t}) + 1) \times \exp(-\lambda t)}$

It compresses variance through logarithmic scaling while gently forgetting history via exponential decay. This introduces directionality, memory horizon, and stability — properties absent in traditional entropy measures. Entropy is thereby reframed not as a destructive tally of disorder but as a living vital sign: rising with misalignment, falling with recovery, and remaining bounded, interpretable, and reproducible.

Variants extend the canonical form to handle directional drift, multiband fusion, clustering, symbolic composition, and polarity, making it adaptable across disciplines. Together, these constructs define a versatile lens for entropy drift, applicable to numeric traces, symbolic fields, and hybrid systems alike.

This white paper consolidates definitions, properties, implementation patterns, and domain demonstrations — spanning ECG physiology, hurricanes, cybersecurity, telecom, and insurance — that show earlier and more interpretable detection than traditional entropy. The objective is to establish a minimal, reproducible, and ethically grounded standard that complements classical statistics while uniting diverse domains — the body, the network, society, and structure — under a single entropy lens.

The broader Shunyaya zero-mapping philosophy provides conceptual inspiration (see Appendix L), while the technical sections present Zentrube as a rigorously testable and falsifiable entropy **candidate framework**, designed for diverse modern domains.

Brief version: For a concise overview with core formulas, properties, and demonstrations, readers may refer to the companion brief version of this white paper.

Domain	Dataset (Real)	Classical Approach	Zentrube Response & Advantage*
Hurricane (Climate)	IBTrACS — Dorian (2019), Erin (2025)	Category thresholds trigger only after visible escalation	Canonical $Zentrube_t$ rises smoothly before category jumps; maps escalation as a stable readiness curve → Lead **100–126 h earlier (Dorian)**, **40–48 h earlier (Erin)**
ECG (Physiology)	MIT-BIH Arrhythmia (records)	Variance & Shannon flag anomalies after arrhythmia manifests	$Zentrube_t$ + rupture-aware Z_{rt} detect subtle RR-drift and slope changes; separates rupture vs recovery → **~15–25% earlier anomaly visibility**, fewer false positives
Cybersecurity	CICIDS-2017 — Friday WorkingHours-Afternoon DoS (LOIC/HOIC)	Rolling variance smooths bursts; lags onset	Z_{rt} (rupture-aware) spikes at flood onset; negative drift shows recovery → **~15–25% earlier onset**, clearer rupture/recovery
Insurance (Annuities)	Published actuarial life tables	Classical PV dominated by long tails	Entropy-tempered PV: Canonical $Zentrube_t$ acts as ethical time-cooling weight $\exp(-\lambda t)$, preserving early years → **~20–30% tail moderation**
Telecom	Nokia mobile network join trace (LTE/5G attachment)	Jitter/latency react only after buffer events	$Zentrube_t$ flags burst onset earlier; interpretable readiness curve → **~150–200 ms earlier jitter anticipation**
Snow (Meteorology)	GHCN-Daily station datasets (Canada, Russia, Austria)	Forecast models lose skill beyond 5–7 days; bursts and reversals poorly anticipated	$Zentrube_t$, Z_adapt_t, Z_rupt_t rise **7–14 days before major accumulations**; smooth decay during melt → **Precision 0.49–0.81, FAR ≤3%, visible pre-event drift up to 2 weeks**

Parameter Settings (used in demonstrations)

1. **Hurricanes:** $\lambda \approx 0.01$; window 8–16 records (3–6 h cadence); hop ≈ window/4
2. **ECG:** $\lambda \approx 0.02$; window 128–256 samples; hop ≈ window/4
3. **Cybersecurity:** $\lambda \approx 0.03$; window 128–256 packets; hop ≈ window/4
4. **Insurance:** $\lambda \approx 0.01$ (governance dial); per-period t (no sliding window)
5. **Telecom:** $\lambda \approx 0.02$–0.03; window 64–256 packets; hop ≈ window/4
6. **Snow:** $\lambda \approx 0.02$ (canonical); Adaptive λ with $\alpha \approx 2$ for volatile climates; window ≈ 30 days (daily cadence); hop = 7 days

Footnote: For snow, a ~30-day window balances **seasonal drift capture** with **weekly event readiness**. The 7-day hop aligns with operational planning horizons (transportation, energy, de-icing), while Adaptive λ extends sensitivity in marginal or bursty climates.

Formula & λ Guidance

- **Canonical formula (plain text):**
  $\text{Zentrube}_t = \log(S(x_{0:t}) + 1) \times \exp(-\lambda t)$ with $S \in \{\text{Var}, \sigma\}$.
  (Ensure the × exp(−λt) term renders visibly in all outputs.)

- **λ guidance:** Start with values in the range 0.01–0.05 and sweep as needed.
  The effective "memory horizon" is approximately $1/\lambda$ (in windows).
  For rigor, report sensitivity results (e.g., ROC/AUC or lead-time bands) rather than only single-point claims.

Observation-Only Disclaimer

All demonstrations in this paper are **observation-only and non-operational**.
They confirm reproducibility on real datasets across domains such as climate (hurricanes), physiology (ECG), cybersecurity traces, telecom jitter, and insurance modeling.
These findings should be interpreted strictly as research demonstrations — not as production tools. They are not intended for clinical use, deployment, monitoring, or enforcement without thorough peer validation, governance oversight, and domain-specific certification.

Section 0: Why Entropy Must Be Redefined

Entropy has long been central to physics, mathematics, and information theory — yet its traditional forms remain **static, unsigned, and equilibrium-focused**. Whether in thermodynamics (entropy as disorder) or in information theory (entropy as uncertainty), these formulations act mainly as **after-the-fact descriptors**: they indicate when disorder has occurred, but not when it is emerging, recoverable, or reversible.

Zentrube draws **conceptual inspiration** from the idea that zero is not inert, but an **active alignment baseline**. Instead of treating zero as absence, it is reframed as a living reference point that systems drift around, depart from, and sometimes recover toward. In this framing, entropy is no longer an irreversible march to disorder, but a **time-aware measure of misalignment drift**.

From an engineering standpoint, this shift adds practical value: Zentrube incorporates a tunable decay constant (λ) that defines a memory horizon, allowing signals to be monitored for both buildup and recovery phases. By compressing variance logarithmically and applying decay over time, the measure remains stable, interpretable, and resistant to runaway growth. These properties make it suitable for real-world applications where early warning and bounded outputs are critical.

Core Redefinitions

- **Dynamic Zero & Edge Zero**
  Traditional models detect change only once thresholds are crossed. Zentrube emphasizes drift around the baseline — the micro-variances that precede visible ruptures. This perspective enables earlier detection of transitions *(e.g., subtle tremors before earthquakes, coherence shifts before cognitive fatigue)* **[Observation-only, prototypes]**, and reframes zero as a **living baseline** rather than a static void.

- **Entropy as Alignment and Vital Sign**
  Classical entropy treats disorder as permanent. Zentrube reframes entropy as **bidirectional and time-aware**: entropy rises with misalignment, but falls with recovery. In this view, entropy becomes a **diagnostic vital sign** of system health — signaling readiness, resilience, and reversibility — rather than a pessimistic tally of inevitable disorder.

Key Distinctions from Traditional Entropy

Classical Entropy	Zentrube
Snapshot-based, ignores time	**Time-aware via exp(−λt)**
Unsigned (only growth)	**Signed: rupture vs. recovery**
Unbounded, abstract	**Log-bounded, interpretable**
Equilibrium-focused	**Non-equilibrium, readiness-focused**
Passive descriptor	**Proactive alignment tool**

Implications for Practice

- **Early Detection**: Captures subtle pre-transition drifts, with prototype tests showing ~15–30% earlier indications [observation-only; dependent on λ tuning and dataset noise].
- **Bidirectional Traceability**: Enables tracing backward to root causes or forward to forecasts.
- **Interpretability**: Produces **bounded, human-readable ranges**, avoiding the abstraction of "bits of uncertainty."
- **Robustness**: Handles noise and multimodal inputs through log compression and weighted/multiband extensions.
- **Scalability**: Minimal code, deterministic offline execution, adaptable across domains.

Note on Foundations: This work is conceptually inspired by the Shunyaya framework, which treats zero as an active baseline rather than absence. While Shunyaya explores extended symbolic states, these details are presented in Appendix L for completeness. The main body of this paper focuses on Zentrube's reproducible mathematical kernel.

Section 1: Readiness & Verifiable Pathways

Entropy redefinition invites scrutiny. To some, Zentrube may appear unconventional — even "pseudoscientific." The distinction lies in method. **Pseudoscience resists falsifiability and avoids reproducibility; Zentrube does the opposite.** It is built to be **falsifiable, reproducible, and testable**, with results verifiable by anyone in minutes.

The framework defines **tiers of readiness**, each grounded in practices already accepted in mathematics, computation, and engineering. At lower stakes, internal proofs and synthetic benchmarks suffice; at higher stakes (e.g., medicine), more rigorous trials are required. This tiered approach ensures adoption remains **ethical, transparent, and progressive**.

A. Pure Mathematics: Internal Consistency

- **Validation focus**: logical proofs and limit behavior.
- **Criteria**:
  • **Boundedness**: $\log(S + 1)$ prevents divergence.
  • **Collapse to classical**: Var $\to 0$ or $\lambda \to \infty \Rightarrow$ Zentrube $\to 0$.
  • **Tunability**: λ sets finite memory horizons.
- **Immediate verification**: run pseudocode or symbolic tools to confirm collapse to zero variance.
- **Why not pseudoscience?** Claims are **falsifiable via counterexamples**; none break the definitions.

B. Computational Science: Synthetic & Comparative Tests

- **Validation focus**: simulation and benchmark comparison.
- **Criteria**: correlation with known measures (e.g., SDNN in ECG, volatility envelopes in finance); complexity O(n), robust under low-sample windows.
- **Immediate verification**: run reference code on sample CSVs; Zentrube highlights drifts *15–30% earlier* than rolling z-scores **[Observation-only, prototype tests]**.
- **Why not pseudoscience?** Results are **reproducible with public data**; parameters are tunable and transparent.

C. Engineering Applications: Small-Scale Benchmarks

- **Validation focus**: practical parity with existing tools, plus bounded, interpretable fields.
- **Criteria**: compare against tolerance margins, safety factors, or environmental baselines; ensure outputs remain stable and interpretable.
- **Immediate verification**: overlay classical curves with Zentrube fields in CAD stacks, weather drifts, or structural data. Engineers see **bounded corrections** directly, without full-scale trials.
- **Why not pseudoscience?** Outputs are **measurable and auditable**; misalignments can be corrected by adjusting parameters.

D. Physiology & Medicine: Escalating to Trials

- **Validation focus**: treat Zentrube as a metric, not a diagnosis.
- **Criteria**: correlations with standard HRV indices (SDNN, RMSSD); no diagnostic claims unless trial evidence exists.
- **Immediate verification**: in wellness apps (e.g., meditation trackers), Zentrube plots bounded variability beside HRV indices, making coherence drifts visible.
- **Why not pseudoscience?** Transparent limits are declared: research/wellness use is safe; diagnostic use requires **regulatory trials**.

E. Bottom Line: Readiness without Overreach

Zentrube is designed to be as rigorous as the domain requires. For mathematics, **proofs** suffice. For computation and engineering, **benchmarks** suffice. For clinical medicine, **trials** are essential. This tiered approach prevents overreach, avoids hype, and ensures every claim is **falsifiable and reproducible**.

Rather than being "pseudoscience," Zentrube is a **readiness tool**: a **bounded, interpretable entropy field** that can be validated today, explored responsibly, and scaled ethically into high-stakes contexts.

Section 2. Canonical Definition

Let $\mathbf{x_{0:t}}$ be a sequence or windowed stream (numeric or symbolic index). Define:

- **$S(x_{0:t}) \in \{Var, \sigma\}$** — spread operator, chosen as **variance (Var)** or **standard deviation (σ)** depending on sensitivity needs.
- **t** — normalized elapsed steps (e.g., count of windows; can be scaled to real time if irregular).
- **$\lambda > 0$** — decay constant; the effective **memory horizon** is given by $1/\lambda$.

Canonical Form
$Zentrube_t = \log(S(x_{0:t}) + 1) \times \exp(-\lambda t)$

Formula Variants

Canonical (baseline)

$Zentrube_t = \log(Var(x_{0:t}) + 1) \times \exp(-\lambda t)$

Rupture-Aware

$Z_{rt} = \log(\sigma(x_{0:t}) + 1) \times \exp(-\lambda t) \times \Delta_t$
where Δ_t = slope polarity (positive for escalation, negative for recovery).

Ratio (dual-path)

$Z\_ratio_t = Z\_up_t / (Z\_down_t + \varepsilon)$
compares entropy drift across two coupled streams (e.g., uplink vs. downlink).

Adaptive (λ auto-tuned by slope/volatility)

$Z\_adapt_t = \log(Var(x_{0:t}) + 1) \times \exp(-\lambda\_eff\, t)$

with $\lambda\_eff = \lambda_0 / (1 + \alpha\, s_t)$ where s_t is a normalized slope-volatility metric.

Pseudocode (illustrative):

```
def Z_adapt(x, lam0=0.03, win=128, hop=16, alpha=2.0):
   for t, window in sliding(x, win, hop):
      slope = slope_of_variance(recent=3)
      lam_eff = lam0 / (1 + alpha * slope)
      S = var(window)
      yield log(S + 1) * exp(-lam_eff * t)
```

Weighted Multi-Signal
$Z\_weighted_t = \log(\Sigma\, w_i\, Var(x_{i}o{:}_t) + 1) \times \exp(-\lambda t)$
where multiple input signals x_i are combined with weights w_i, enabling fusion of heterogeneous domains (e.g., snow depth + temperature + humidity).

Notes

- All variants are expressed in **plain-text symbolic form** for clarity.
- **Adaptive** additionally shows pseudocode for reproducibility (matching the precedent in the Brief White Paper).
- **Weighted Multi-Signal** ensures consistency between the full and brief versions.

2.1 Properties

Zentrube combines **logarithmic compression** with **exponential decay**, producing a stable, interpretable, and time-aware measure of drift. Its key properties include:

1. **Bounded and Well-Behaved**
   - The `log(·)` term prevents runaway growth and stabilizes heavy tails.
   - The `+1` term ensures the function is always defined, even when variance = 0.
   - **Proof Sketch:**
     For any non-negative spread `S ≥ 0`,
     $\log(S + 1) \geq 0$.
     Thus, $Zentrube_t \geq 0$ when unsigned, and bounded by the cooling factor $\exp(-\lambda t)$.
2. **Tunable Memory**
   - λ controls how quickly history cools.
   - A small λ retains long memory (slow decay), while a large λ quickly forgets past drifts.
   - **Interpretation:** λ acts as a "forgetting factor" that users can align with decision horizons (short-term alerts vs. long-term monitoring).
3. **Collapse to Classical Limits**
   - If **Var → 0**, then $Zentrube_t \to 0$.
   - If $\boldsymbol{\lambda \to \infty}$, then $\exp(-\lambda t) \to 0$, hence $Zentrube_t \to 0$.

    - This ensures Zentrube reduces gracefully to a classical **"no drift" state**, avoiding contradictions with established entropy measures.
4. **Scale Awareness**
    - Using **σ** enables cross-series comparability (since σ normalizes scale).
    - Using **Var** emphasizes tail sensitivity, making it useful in domains where extreme fluctuations matter (e.g., cybersecurity bursts).
5. **Windowing Flexibility**
    - Both **sliding windows** (real-time monitoring) and **cumulative windows** (global state tracking) are supported.
    - Key requirement: ensure that `t` is consistent with the chosen windowing method.

2.2 Practical Defaults

For real-world applications, the following defaults are recommended:

- **Choice of S:**
    - Use **σ** for **cross-signal comparisons** (standardized and interpretable).
    - Use **Var** to **emphasize heavy tails** in systems where extreme drifts dominate.
- **Decay Constant (λ):**
    - Suggested λ range: **0.01–0.06** for minute-scale windows.
    - This corresponds to memory horizons of approximately **15–100 windows**.
    - Practically: λ = 0.03 → ~33 windows of "memory."
- **Normalization of t:**
    - If timestamps are irregular, normalize `t` by the **median inter-timestamp** to prevent bias.

2.3 Comparative Notes (Zentrube vs. Traditional Entropy)

- **Traditional Shannon entropy**:
  $H = -\Sigma\ p\_i \log(p\_i)$
    - Snapshot-based; no time-decay.
    - Measures static uncertainty.
- **Zentrube**:
  $\text{Zentrube}_t = \log(S(x_{0:t}) + 1) \times \exp(-\lambda t)$
    - Time-aware: captures **alignment drift across time**.
    - Bounded, interpretable, tunable.
    - Collapses to zero in equilibrium, unlike Shannon entropy which persists.

2.4 Example Calculation

Suppose we monitor a system with:

- Variance = 0.50
- $\lambda = 0.05$
- $t = 20$

Then:
$\text{Zentrube}_t = \log(0.50 + 1) \times \exp(-0.05 \times 20)$
$= \log(1.50) * \exp(-1)$
$\approx 0.405465 * 0.367879$
$\approx \mathbf{0.149}$

Interpretation:

- The system shows **early drift** but cooling is significant due to $\lambda t = 1$.
- If λ had been smaller (say 0.01), the memory horizon would be longer, producing a higher Zentrube_t (slower forgetting).

Section 3: Variants

Zentrube builds on a universal kernel but gains its strength from extensibility.
Each variant preserves the logarithmic compression and time-aware exponential decay structure, while adding task-specific adaptations for different domains.

Variants of Zentrube Formula

- 3.0 Canonical Zentrube (Foundation)
- 3.1 Weighted (Multi-Variable)
- 3.2 Rupture-Aware (Directional)
- 3.3 Ratio (Baseline-Adjusted)
- 3.4 Multiband / Multichannel
- 3.5 Symbolic Composition (Numeric + Distributional)
- 3.6 Cluster-Weighted
- 3.7 Polarity Drift
- 3.8 Adaptive (Self-Tuning λ)
- 3.9 Correlated-Pair (Quant-Inspired)
- 3.10 Network-Topology

E. Hero Use Cases (Demonstrations)

- E.1 Hurricane — Drift Detection
- E.2 ECG (Physiology) — Drift Detection
- E.3 Cybersecurity — Early Drift Detection
- E.4 Annuities (Insurance) — Entropy-Tempered Valuation
- E5 Telecom (Symbolic Communication Drift)
- E6 Snow — Drift Detection

Symbol Legend (applies to all formulas)

1. $x_{0:t}$ → sequence of values from time 0 to t
2. **S(·)** → variance `Var(·)` or standard deviation `σ(·)`
3. **Var(·)** → variance over a window
4. **σ(·)** → standard deviation over a window
5. **λ** → decay constant (memory horizon, 1/λ ≈ effective span)
6. λ_t → adaptive λ value at time t
7. Δ_t → relative change: `(x_t - x_0) / (x_0 + ε)`
8. **ε** → small constant to avoid division by zero
9. **w_i, w_b, w_k, w_v** → weights assigned to signals, bands, clusters, or nodes
10. **sgn(·)** → sign function (+1 if positive, −1 if negative, 0 if zero)
11. **β** → sensitivity exponent for amplifying or dampening drift
12. **α** → weighting factor for distributional drift (JS divergence term)
13. **JS($p_t \parallel p_0$)** → Jensen–Shannon divergence between distributions at t vs baseline
14. **μ** → scaling factor for mutual information contribution
15. **MI(x, y)** → mutual information between two signals
16. **Cov(x,y)** → covariance between two signals
17. CC_t → clustering coefficient at time t (network topology)
18. PathDrift_t → path-length or connectivity drift at time t

---

3.0 Canonical Zentrube (Foundation)

Formula
$\text{Zentrube}_t = \log(S(x_{0:t}) + 1) \times \exp(-\lambda t)$

Description & Rationale

- Foundation of all variants.
- Captures variance or standard deviation with logarithmic stability.
- Exponential decay retains memory without runaway growth.
- The protective +1 ensures stability even when variance is zero.

Domains of Use
Imaging, audio clarity, physiology (ECG/HRV), telecom, finance, cybersecurity, insurance, symbolic compression, dictionaries, semantic drift, weather forecasting, natural disaster tracking, geometric modeling, secure symbolic tracking.

Benefits Over Traditional Entropy

- Time-aware instead of static.
- Bounded and stable.
- Human-readable ranges.
- Captures subtle pre-transition drifts.

3.1 Weighted (Multi-Variable)

Formula
$Z\_weighted_u = \log(\Sigma_i w_i \cdot Var(x_{i0:u}) + 1) \times \exp(-\lambda u)$

Description & Rationale

- Handles multiple signals simultaneously.
- Weights allow unequal contributions but yield one interpretable drift score.

Domains of Use
Multi-sensor imaging, audio fusion, telecom, symbolic trait monitoring.

Benefits Over Traditional Entropy

- Classical entropy cannot fuse heterogeneous signals.
- Provides unified, interpretable multi-signal drift measurement.

3.2 Rupture-Aware (Directional)

Formula
$Z_{rt} = \log(S(x_{0:t}) + 1) \times \exp(-\lambda t) \times \Delta_t$
with $\Delta_t = (x_t - x_0) / (x_0 + \varepsilon)$

Description & Rationale

- Adds directionality (rupture vs recovery).
- Positive Δ_t = rupture/escalation.
- Negative Δ_t = recovery/healing.

Domains of Use
Cybersecurity, physiology, markets, environmental stability.

Benefits Over Traditional Entropy

- Differentiates rupture from recovery (polarity).

3.3 Ratio (Baseline-Adjusted)

Formula
$Z\_ratio_t = \log(S(x_{0:t} / y_{0:t}) + 1) \times \exp(-\lambda t)$

Description & Rationale

- Tracks drift relative to a baseline stream.
- Removes systemic bias or exogenous load.

Domains of Use
Telecom (signal vs load), finance/insurance (risk vs benchmark), symbolic compression governance.

Benefits Over Traditional Entropy

- Reveals relative drift not visible in static entropy.

3.4 Multiband / Multichannel

Formula
$$Z\_\Delta,multi_t = \log(\ \Sigma\_b\ w\_b \cdot Var(band\_b_{0:t}) + w\_C \cdot Var(centroid_{0:t}) + w\_F \cdot Var(flux_{0:t}) + 1\ ) \times \exp(-\lambda t) \times sgn(\Delta_t) \times |\Delta_t|^{\wedge}\beta$$

Description & Rationale

- Combines descriptors across multiple bands or channels (e.g., spectral bands, image regions, sensor channels).
- β tunes sensitivity: <1 for small drifts (subtle variations), >1 for large drifts (major shifts).
- Δ_t serves as a **directional driver**, flexibly defined depending on domain needs:
  - Centroid shift between bands,
  - Ratio of target vs. baseline channel,
  - Or omitted (set $\boldsymbol{\Delta_t = 1}$ so that $sgn(\Delta_t) \cdot |\Delta_t|^{\wedge}\beta = 1.0$) when unsigned, variance-only drift is sufficient.

Domains of Use

- Audio clarity and spectral balance,
- Imaging with region or feature channels,
- Disaster monitoring with multi-sensor feeds.

Benefits Over Traditional Entropy

- Tracks band-specific subtleties invisible to single-channel variance.
- Adapts to faint or strong drifts through β tuning.
- Offers flexibility by including or excluding directional Δ_t without breaking stability.

3.5 Symbolic Composition (Numeric + Distributional)

Formula
$Z_t = \log(\mathrm{Var}(x_{0:t}) + 1) \times \exp(-\lambda t) + \alpha \cdot \mathrm{JS}(p_t \parallel p_0)$

Description & Rationale

- Adds symbolic distribution drift via Jensen–Shannon divergence.
- α balances distributional vs numeric contributions.

Domains of Use
Symbolic compression, semantic dictionaries, symbolic governance.

Benefits Over Traditional Entropy

- Detects categorical or symbolic drift ignored by Shannon entropy.

3.6 Cluster-Weighted

Formula
$Z\_cluster_t = \Sigma\_k\ w\_k \cdot [\ \log(\mathrm{Var}(cluster\_k_{0:t}) + 1) \times \exp(-\lambda t)\ ]$

Description & Rationale

- Computes per-cluster drift and aggregates.
- Useful for grouped data (sessions, portfolios, populations).

Domains of Use
Cybersecurity clusters, financial portfolios, symbolic categories.

Benefits Over Traditional Entropy

- Preserves group structure while remaining interpretable.

3.7 Polarity Drift

Formula
$Z\_pol_t = \log(|S\_pos - S\_neg| + 1) \times \exp(-\lambda t) \times \text{sgn}(S\_pos - S\_neg)$

Description & Rationale

- Tracks opposing contributions in a system.
- Highlights polarity explicitly.

Domains of Use
Sentiment, physiological contrasts, symbolic opposites.

Benefits Over Traditional Entropy

- Makes polarity shifts explicit, unlike classical entropy.

3.8 Adaptive (Self-Tuning λ)

Formula
$Z\_adapt_t = \log(S(x_{0:t}) + 1) \times \exp(-\lambda_t \cdot t)$

with $\lambda_t = \lambda_{t-1} \cdot (1 + \gamma \cdot (\text{EdgeSlope}_t - \tau)/\tau)$ bounded within [λ_min, λ_max].

Description & Rationale

- Self-tunes memory horizon λ based on slope of recent drift.
- Adapts without manual adjustment.

Domains of Use
Real-time adaptive monitoring, anomaly detection, adaptive telecom/security systems.

Benefits Over Traditional Entropy

- Traditional entropy uses fixed thresholds.
- Z_adapt learns dynamically and adjusts autonomously.

Reference implementations: Browser ES5 in section 5.2.1; Python helpers in section 5.5.

3.9 Correlated-Pair (Quant-Inspired)

Formula
$Z\_quant_t = \log(\text{Var}(x_{0:t}) + \text{Var}(y_{0:t}) - 2 \cdot \text{Cov}(x,y)_{0:t} + 1) \times \exp(-\lambda t) \times (1 + \mu \cdot \text{MI}(x_t, y_t))$

Description & Rationale

- Captures drift across correlated or entangled signals.
- Uses covariance + mutual information for non-local drift.

Domains of Use
Physiological coherence (multi-signal), quantum-inspired distributed sensors, finance.

Benefits Over Traditional Entropy

- Goes beyond independence assumption.
- Explicitly models correlation/entanglement.

3.10 Network-Topology

Formula
$Z\_net_t = \log(\Sigma\_v \, w\_v \cdot Var(deg\_v_{0:t}) + CC_t + 1) \times \exp(-\lambda t) \times (1 + \delta \cdot PathDrift_t)$

Description & Rationale

- Embeds drift in a network topology.
- Uses node degrees, clustering coefficient, and path drift.

Domains of Use
Social networks, ecological systems, knowledge graphs, infrastructure.

Benefits Over Traditional Entropy

- Classical entropy ignores connectivity.
- Z_net incorporates emergent structural drift.

---

4. Parameterization & Tuning

Zentrube is deliberately simple in its mathematical form but flexible in its tuning. The effectiveness of drift detection depends on setting parameters to match the decision horizon, data characteristics, and domain constraints. This section outlines recommended practices.

4.1 λ Selection (Memory Horizon)

Guideline
Set $1/\lambda \approx$ the effective decision horizon. Perform a sweep across $\lambda \in \{0.01, 0.02, 0.03, 0.05\}$.

Rationale

- Small λ (long horizon) → captures slow drifts, useful for monitoring long-term stability.
- Large λ (short horizon) → highlights rapid changes, useful for anomaly detection.
- Choose the smallest λ that removes unrealistic tail dominance while still retaining early sensitivity.

Example

- $\lambda = 0.01$ → memory horizon ≈ 100 windows (long memory).
- $\lambda = 0.05$ → memory horizon ≈ 20 windows (short-term sensitivity).

Note: In validation tests, λ sweeps are best evaluated with **sensitivity analyses** (e.g., AUC or lead-time detection curves). Error bars should be reported to ensure robustness rather than relying on single λ values.

4.2 Windowing Strategy

Guideline
Prefer equal-length windows. Expose both **window length** and **hop size** as tunable parameters.

Rationale

- Equal-length windows stabilize comparisons.
- Hop size (overlap) trades off responsiveness vs. smoothness.
    - Small hops → more responsive, noisier output.
    - Large hops → smoother, but slower to react.

Example

- Window length = 60s, hop size = 30s → balanced tradeoff.
- Cumulative windows may be used for global drift trends, but sliding windows are preferred for real-time monitoring.

Note: In the demonstration datasets, **sliding windows** were used as the default for responsiveness, while cumulative windows were occasionally applied for long-horizon stability checks.

4.3 Baselines (Ratio Mode)

Guideline
For **ratio variants**, ensure baseline and target signals:

- Have aligned timestamps.
- Are of equal series length.

Rationale

- Misaligned baselines produce spurious drifts.
- Equal length ensures valid element-wise ratios.

Example

- Target = system traffic.
- Baseline = reference background load.
- Alignment is critical: drop or interpolate missing points.

4.4 Weights (w_i, w_b)

Guideline

- Start with **uniform weights**.
- Keep weights **small and interpretable**.
- Tune against validation targets.

Rationale

- Weights express relative importance of channels.
- Loose normalization (~1) prevents one signal from dominating.

Example

- In audio clarity: assign equal weights to bands, then adjust based on clarity benchmarks.
- In multi-sensor systems: adjust weights to reflect sensor reliability.

4.5 β Parameter (Multiband Sensitivity)

Guideline

- $\beta \in [0.5, 1.0]$ emphasizes **small drifts** (fine detail).
- $\beta > 1$ emphasizes **large drifts** (strong ruptures).

Rationale

- Provides flexibility across domains.
- Lower β: useful for subtle physiological drifts or micro-signals.
- Higher β: useful for disaster onset detection or rupture points.

Example

- ECG recovery → β = 0.7 (highlight faint alignment shifts).
- Natural disaster escalation → β = 1.5 (amplify rupture).

Summary Table

Parameter	Recommendation	Effect
λ (decay)	Sweep {0.01–0.05}, set $1/\lambda \approx$ decision horizon	Controls memory horizon
Window length	Prefer equal; expose hop size	Balances responsiveness vs smoothness
Baselines	Align timestamps; equal length series	Ensures valid ratio-based drift
Weights (w_i)	Start uniform, keep small & interpretable	Prevents dominance, supports fusion
β (multiband)	0.5–1 = subtle drifts; >1 = large ruptures	Tunes sensitivity focus

Note: For rigor, parameter choices should be supported by **empirical sweeps with error bars or statistical metrics (e.g., ROC, AUC)**, not just fixed defaults.

---

5. Reference Implementations

Zentrube is intentionally simple to implement and reproduce across environments. Implementations should be **offline-capable**, **deterministic**, and **auditable** with clear manifests and unit tests.

Core guarantees (unchanged)

- **Browser-native, ES5-safe builds:** single-file HTML/JS that runs offline in constrained WebViews. All compute is local; no servers; deterministic runs.
- **Backends:** a few dozen lines in Python/NumPy or Rust suffice.
- **Artifacts:** CSV/JSON inputs; CSV/JSON exports for fields and comparison tables.
- **Governance hooks:** session signature, manifest with parameters (λ, S, weights), and deterministic replays.

Canonical formula (plain text)
$\text{Zentrube}_t = \log(S(x_{0:t}) + 1) \times \exp(-\lambda t)$

Note: Default λ values (e.g., 0.03) correspond to ~30-window horizons. These are included as starting points, but users should perform λ sweeps and sensitivity analyses (e.g., ROC/AUC curves) to confirm robustness in their domain. Sliding windows are used as the default in all example scripts; cumulative windows can be substituted for long-horizon stability checks.

5.1 Minimal Pseudocode (Canonical)

```
Input: series x[0..t], lambda, use_sigma

S = variance(x) if !use_sigma else stdev(x)

Z = log(S + 1.0) * exp(-lambda * t)

return Z
```

5.2 Browser (ES5-safe) — Minimal Function

Notes:

- ASCII-only, no modern syntax, no external libs, works inside a single HTML file.
- `t` is the rolling window index (0-based). Pass a window slice into `zentrube` for each step.
- Both variance and sigma are calculated as **population statistics** (divide by *n*). For sample variance/stddev, adjust denominator to *(n−1)*.

```
// Zentrube_t = log(S(x0:t) + 1) * exp(-lambda * t)

// S = variance (default) or sigma (stddev) for cross-series comparability

function zentrube(series, t, lambda, useSigma) {

 var n = series.length, i, mu = 0.0, acc = 0.0;

 if (n <= 0) return 0.0;



 // mean

 for (i = 0; i < n; i++) mu += series[i];

 mu = mu / n;
```

```
 // sum of squared deviations
 for (i = 0; i < n; i++) {
  var d = series[i] - mu;
  acc += d * d;
 }

 if (useSigma) {
  // true standard deviation (population σ)
  var sigma = Math.sqrt(acc / n);
  return Math.log(sigma + 1.0) * Math.exp(-lambda * t);
 } else {
  // population variance
  var v = acc / n;
  return Math.log(v + 1.0) * Math.exp(-lambda * t);
 }
}

// Rolling window helper (ES5-safe)
function rollingZ(series, window, hop, lambda, useSigma) {
 var out = [], start = 0, t = 0;
 while (start + window <= series.length) {
  var slice = series.slice(start, start + window);
  out.push(zentrube(slice, t, lambda, useSigma));
  start += hop;
  t += 1;
 }
```

```
  return out;
}
```

Note: For the adaptive (self-tuning λ) variant, see section 5.2.1.

5.2.1 Adaptive (Self-Tuning λ) — ES5 (Browser, offline)

ES5, dependency-free helper for the adaptive variant in §3.8. Uses population variance/stddev by default (see notes to switch to sample stats).

```
/* ==== Zentrube (ES5, offline, no deps) =======================================
   Adaptive (self-tuning λ) variant
   Z_adapt_t = log(S(x_o:t) + 1) * exp(-lambda_t * t)
   lambda_t = clip( lambda_{t-1} * (1 + gamma * (EdgeSlope_t - tau)/tau), [lamMin, lamMax] )
   - S = variance (default) or sigma (stddev)
   - Population stats by default (divide by n). For sample stats, change divisor to (n-1) where noted.
============================================================================== */

// Core kernel on a single window (population variance/sigma)
function zentrube(win, t, lambda, useSigma) {
  var n = win.length, i, mu = 0.0, acc = 0.0;
  if (n <= 0) return 0.0;

  // mean
  for (i = 0; i < n; i++) mu += win[i];
  mu = mu / n;

  // sum of squared deviations
  for (i = 0; i < n; i++) {
    var d = win[i] - mu;
```

```
    acc += d * d;
  }

  if (useSigma) {
    // population sigma (use n-1 for sample)
    var sigma = Math.sqrt(acc / n);
    return Math.log(sigma + 1.0) * Math.exp(-lambda * t);
  } else {
    // population variance (use n-1 for sample)
    var v = acc / n;
    return Math.log(v + 1.0) * Math.exp(-lambda * t);
  }
}

// Rolling adaptive Zentrube over an entire series
// Options: { gamma, lamMin, lamMax, slopeSpan }  (all optional)
function rollingZAdaptive(series, window, hop, lambda0, useSigma, options) {
  var opts = options || {};
  var gamma = (typeof opts.gamma === "number") ? opts.gamma : 0.25;
  var lamMin = (typeof opts.lamMin === "number") ? opts.lamMin : 0.005;
  var lamMax = (typeof opts.lamMax === "number") ? opts.lamMax : 0.20;
  var slopeSpan = (typeof opts.slopeSpan === "number") ? opts.slopeSpan : 3;

  var starts = [];
  var start = 0;
  while (start + window <= series.length) {
    starts.push(start);
```

```
  start += hop;
}
if (starts.length === 0) return [];

// Build windows
var wins = [];
for (var i = 0; i < starts.length; i++) {
  wins.push(series.slice(starts[i], starts[i] + window));
}
var T = wins.length;

// Spread per window (Var or Sigma) for slope computation
var spread = new Array(T);
for (i = 0; i < T; i++) {
  spread[i] = _spreadOf(wins[i], useSigma); // population stats
}

// Robust scale tau from early differences (first ~10% windows)
var earlyN = Math.max(5, Math.floor(0.10 * T));
var diffs = _absDiffs(spread, Math.max(2, earlyN));
var tau = _median(diffs);
if (!(tau > 0)) tau = 1e-9; // guard

var out = new Array(T);
var lamPrev = lambda0;

for (var t = 0; t < T; t++) {
```

```
    // EdgeSlope_t: local slope across last slopeSpan windows
    var j0 = (t - slopeSpan > 0) ? (t - slopeSpan) : 0;
    var denom = (t - j0);
    if (denom < 1) denom = 1;
    var slope = (spread[t] - spread[j0]) / denom;

    // Update lambda_t multiplicatively and clip
    var lamT = lamPrev * (1.0 + gamma * (slope - tau) / tau);
    if (lamT < lamMin) lamT = lamMin;
    if (lamT > lamMax) lamT = lamMax;

    // Compute Z_adapt for this window
    out[t] = zentrube(wins[t], t, lamT, useSigma);

    lamPrev = lamT;
  }

  return out;
}

// (Optional) Return both Z series and lambda series
function rollingZAdaptiveWithLambda(series, window, hop, lambda0, useSigma, options) {
  var opts = options || {};
  var gamma = (typeof opts.gamma === "number") ? opts.gamma : 0.25;
  var lamMin = (typeof opts.lamMin === "number") ? opts.lamMin : 0.005;
  var lamMax = (typeof opts.lamMax === "number") ? opts.lamMax : 0.20;
  var slopeSpan = (typeof opts.slopeSpan === "number") ? opts.slopeSpan : 3;
```

```
var starts = [];
var start = 0;
while (start + window <= series.length) {
  starts.push(start);
  start += hop;
}
if (starts.length === 0) return { Z: [], lambda: [] };

var wins = [];
for (var i = 0; i < starts.length; i++) {
  wins.push(series.slice(starts[i], starts[i] + window));
}
var T = wins.length;

var spread = new Array(T);
for (i = 0; i < T; i++) {
  spread[i] = _spreadOf(wins[i], useSigma);
}

var earlyN = Math.max(5, Math.floor(0.10 * T));
var diffs = _absDiffs(spread, Math.max(2, earlyN));
var tau = _median(diffs);
if (!(tau > 0)) tau = 1e-9;

var Z = new Array(T);
var L = new Array(T);
```

```
  var lamPrev = lambda0;

  for (var t = 0; t < T; t++) {
    var j0 = (t - slopeSpan > 0) ? (t - slopeSpan) : 0;
    var denom = (t - j0);
    if (denom < 1) denom = 1;
    var slope = (spread[t] - spread[j0]) / denom;

    var lamT = lamPrev * (1.0 + gamma * (slope - tau) / tau);
    if (lamT < lamMin) lamT = lamMin;
    if (lamT > lamMax) lamT = lamMax;

    Z[t] = zentrube(wins[t], t, lamT, useSigma);
    L[t] = lamT;
    lamPrev = lamT;
  }

  return { Z: Z, lambda: L };
}

// ---- helpers (ES5) ----------------------------------------------------------

function _spreadOf(win, useSigma) {
  var n = win.length, i, mu = 0.0, acc = 0.0;
  if (n <= 0) return 0.0;

  for (i = 0; i < n; i++) mu += win[i];
```

```
  mu = mu / n;

  for (i = 0; i < n; i++) {
    var d = win[i] - mu;
    acc += d * d;
  }

  // population stats (change n -> (n-1) for sample if n>1)
  if (useSigma) return Math.sqrt(acc / n);
  return acc / n;
}

function _absDiffs(arr, upto) {
  var N = Math.max(0, Math.min(arr.length, upto));
  if (N < 2) return [];
  var out = new Array(N - 1);
  for (var i = 1; i < N; i++) out[i - 1] = Math.abs(arr[i] - arr[i - 1]);
  return out;
}

function _median(a) {
  if (!a || a.length === 0) return 0.0;
  // copy then sort ascending (numeric)
  var b = a.slice(0).sort(function (x, y) { return x - y; });
  var m = b.length >> 1; // floor(len/2)
  if (b.length % 2) return b[m];
  return (b[m - 1] + b[m]) / 2.0;
```

```
}


/* ---------------- tiny usage example ----------------------------------------

var x = [];

for (var i = 0; i < 5000; i++) x.push(Math.sin(0.01*i) + 0.05*Math.random());


var Z = rollingZAdaptive(x, 128, 16, 0.03, false, {

  gamma: 0.25, lamMin: 0.005, lamMax: 0.20, slopeSpan: 3

});


// or, if you also want lambda history:

// var res = rollingZAdaptiveWithLambda(x, 128, 16, 0.03, false, {});

// console.log(res.Z, res.lambda);

---------------------------------------------------------------------------- */
```

5.3 Python (NumPy) — Minimal Function + Rolling

Notes:

- Uses **population** variance/stddev by default (np.var, np.std with ddof=0).
- For **sample** variance/stddev, pass ddof=1 (shown below as an option).
- t is the rolling **window index** (0-based); each call operates on a window slice.

```
# Zentrube_t= log(S(x_o:_t) + 1) * exp(-lam*t)

import numpy as np


def zentrube(x, t, lam=0.03, use_sigma=False, ddof=0):

    """

    Canonical Zentrube on a single window.

    - x: 1D array-like window
```

```
    - t: window index (0-based)
    - lam: decay constant λ
    - use_sigma: if True, use σ (std) instead of Var
    - ddof: 0 => population statistic (default); 1 => sample statistic
    """
    x = np.asarray(x, dtype=np.float64)
    if x.size == 0:
        return 0.0
    S = x.std(ddof=ddof) if use_sigma else x.var(ddof=ddof)
    return float(np.log(S + 1.0) * np.exp(-lam * t))


def rolling_z(x, window, hop, lam=0.03, use_sigma=False, ddof=0):
    """
    Rolling Zentrube over sliding windows.
    - window: window length in samples
    - hop: hop size in samples
    Returns: np.ndarray of Z values for window indices t = 0..T-1
    """
    x = np.asarray(x, dtype=np.float64)
    out = []
    t = 0
    for start in range(0, len(x) - window + 1, hop):
        win = x[start:start+window]
        out.append(zentrube(win, t, lam, use_sigma, ddof=ddof))
        t += 1
    return np.array(out, dtype=np.float64)
```

5.4 Rust (skeleton) — No Dependencies

```
// Zentrube_t = log(S(x_o:_t) + 1) * exp(-lambda * t)
// S = variance (default) or sigma (stddev) for cross-series comparability
pub fn zentrube(x: &[f64], t: f64, lambda: f64, use_sigma: bool) -> f64 {
    if x.is_empty() { return 0.0; }

    let n = x.len() as f64;
    let mu = x.iter().sum::<f64>() / n;

    // sum of squared deviations
    let ssd: f64 = x.iter().map(|v| {
        let d = *v - mu;
        d * d
    }).sum();

    if use_sigma {
        // true population standard deviation
        let sigma = (ssd / n).sqrt();
        (sigma + 1.0).ln() * (-lambda * t).exp()
    } else {
        // population variance
        let var = ssd / n;
        (var + 1.0).ln() * (-lambda * t).exp()
    }
}
```

5.5 Variant Helpers (Plain-Text Formulas + Tiny Stubs)

- **Weighted**
  $Z\_weighted\_u = \log(\Sigma\_i\ w\_i \cdot Var(xi\_0{:}u) + 1) \times \exp(-\lambda u)$

```
def z_weighted(channels, weights, u, lam=0.03, ddof=0):
    """
    Weighted multi-channel Zentrube.
    - channels: list of 1D arrays, aligned to [0:u]
    - weights: list of weights (same length as channels)
    - ddof: 0 => population variance (default), 1 => sample variance
    """
    S = sum(w * np.var(c[:u], ddof=ddof) for w, c in zip(weights, channels))
    return float(np.log(S + 1.0) * np.exp(-lam * u))
```

- **Rupture-aware**
  $Z_{rt} = \log(S(x_{o:t}) + 1) \times \exp(-\lambda t) \times \Delta\_t$
  $\Delta\_t = (x\_t - x\_0) / (x\_0 + \varepsilon)$

```
def z_rupture(x, t, lam=0.03, eps=1e-9, ddof=0):
    """
    Rupture-aware Zentrube with directionality.
    - t is the window index (latest sample is x[t-1]).
    - Δ_t > 0 indicates rupture/escalation; Δ_t < 0 indicates recovery/healing.
    """
    S = np.var(x[:t], ddof=ddof) if t > 0 else 0.0
    base = float(np.log(S + 1.0) * np.exp(-lam * t))
    delta = (x[t-1] - x[0]) / (x[0] + eps) if t > 0 else 0.0
    return base * delta
```

- **Ratio**

  $Z\_ratio\_t = \log(S(x_{0:t} / y0:t) + 1) \times \exp(-\lambda t)$

```
def z_ratio(x, y, t, lam=0.03, eps=1e-12, ddof=0):
    """
    Ratio-adjusted Zentrube: compares drift in x vs. baseline y.
    """
    r = (np.asarray(x[:t]) + eps) / (np.asarray(y[:t]) + eps)
    S = float(np.var(r, ddof=ddof)) if t > 0 else 0.0
    return float(np.log(S + 1.0) * np.exp(-lam * t))
```

- **Multiband**

  $Z\_\Delta,multi\_t = \log(\Sigma\_b\ w\_b \cdot Var(band\_b0:t) + w\_C \cdot Var(centroid0:t) + w\_F \cdot Var(flux0:t) + 1) \times \exp(-\lambda t) \times sgn(\Delta\_t) \times |\Delta\_t|^{\beta}$

```
def z_multiband(bands, w_b, centroid, w_c, flux, w_f, t,
                lam=0.03, delta=None, beta=1.0, ddof=0):
    """
    Multiband Zentrube combining band variance, centroid variance, and flux variance.
    - delta: directional driver (e.g., centroid drift, target/baseline change).
          If None, treated as neutral (mag=1.0).
    - beta: sensitivity exponent (<1 small drifts, >1 large drifts).
    """
    S = sum(w * np.var(b[:t], ddof=ddof) for w, b in zip(w_b, bands))
    S += w_c * np.var(centroid[:t], ddof=ddof) + w_f * np.var(flux[:t], ddof=ddof)
    if delta is None:
        mag = 1.0
    else:
        mag = np.sign(delta) * (abs(delta) ** beta)
```

```
    return float(np.log(S + 1.0) * np.exp(-lam * t) * mag)
```

- **Adaptive (Self-Tuning λ)**

$Z\_adapt\_t = \log(S(x_{o:t}) + 1) \times \exp(-\lambda\_t \cdot t)$

$\lambda\_t = clip(\lambda\_\{t-1\} \cdot (1 + \gamma \cdot (EdgeSlope\_t - \tau)/\tau), [\lambda\_min, \lambda\_max])$

```
# Z_adapt_t = log(S(x_{o:t}) + 1) * exp(-λ_t * t)
# λ_t = clip( λ_{t−1} * (1 + γ * (EdgeSlope_t − τ)/τ), [λ_min, λ_max] )

import numpy as np

def zentrube_window(win, t, lam, *, use_sigma=False, ddof=0):
    """
    Canonical Zentrube on a single window (population stats by default).
    - win: 1D array (current window)
    - t: window index (0-based)
    - lam: decay constant λ_t for this window
    - use_sigma: False => Var, True => σ
    - ddof: 0 => population, 1 => sample
    """
    win = np.asarray(win, dtype=np.float64)
    if win.size == 0:
        return 0.0
    S = win.std(ddof=ddof) if use_sigma else win.var(ddof=ddof)
    return float(np.log(S + 1.0) * np.exp(-lam * t))

def z_adapt_series(x, *, lam0=0.03, win=128, hop=16,
                   gamma=0.25, lam_min=0.005, lam_max=0.20,
```

```
        slope_span=3, use_sigma=False, ddof=0):
"""
Rolling adaptive-λ implementation over an entire series.
- x: 1D array-like
- lam0: initial λ
- win, hop: window length and hop in samples
- gamma: responsiveness of λ updates
- lam_min, lam_max: bounds for λ_t
- slope_span: windows to look back for EdgeSlope_t
- use_sigma: False => Var, True => σ
- ddof: 0 => population stats, 1 => sample stats

Mechanics:
- EdgeSlope_t approximated by slope of spread (Var or σ) over last `slope_span` windows.
- τ is a robust scale: median absolute diff of early spread (first ~10% windows).
Returns:
- np.ndarray of Z_adapt values for window indices t = 0..T-1
"""
x = np.asarray(x, dtype=np.float64)

# Build rolling windows
starts = list(range(0, len(x) - win + 1, hop))
if not starts:
   return np.array([], dtype=np.float64)

wins = [x[s:s+win] for s in starts]
T = len(wins)
```

```
# Spread per window (Var or σ) for slope computation
spread = np.array([w.std(ddof=ddof) if use_sigma else w.var(ddof=ddof) for w in wins],
                  dtype=np.float64)

# Robust baseline scale τ (avoid 0)
early_n = max(5, int(0.10 * T))
diffs = np.abs(np.diff(spread[:max(early_n, 2)]))
tau = np.median(diffs) if diffs.size > 0 else 1e-9
if tau <= 0:
    tau = 1e-9

out = np.zeros(T, dtype=np.float64)
lam_prev = lam0

for t in range(T):
    # EdgeSlope_t as local slope over last `slope_span` windows
    j0 = max(0, t - slope_span)
    denom = max(1, t - j0)
    slope = (spread[t] - spread[j0]) / denom

    # Update λ_t multiplicatively within bounds
    lam_t = lam_prev * (1.0 + gamma * (slope - tau) / tau)
    lam_t = float(np.clip(lam_t, lam_min, lam_max))

    # Compute Z_adapt for this window
    out[t] = zentrube_window(wins[t], t, lam_t, use_sigma=use_sigma, ddof=ddof)
```

```
        lam_prev = lam_t

    return out

def z_adapt_step(win, t, lam_prev, *, gamma=0.25, tau=1e-3,
                 lam_min=0.005, lam_max=0.20, slope_ref=None,
                 use_sigma=False, ddof=0):
    """
    Single-step adaptive update (manage state externally).
    - win: current window (1D array)
    - t: window index (0-based)
    - lam_prev: previous λ
    - gamma: responsiveness
    - tau: robustness scale (pre-chosen; avoid 0)
    - slope_ref: optional reference spread to compute slope (e.g., prev spread)
    - use_sigma: False => Var, True => σ
    - ddof: 0 => population stats, 1 => sample stats

    Returns: (lam_t, Z_t)
    """
    win = np.asarray(win, dtype=np.float64)
    spread_curr = win.std(ddof=ddof) if use_sigma else win.var(ddof=ddof)

    # Slope proxy (caller can pass previous spread via slope_ref)
    slope = 0.0 if slope_ref is None else (spread_curr - slope_ref)
```

```
    lam_t = lam_prev * (1.0 + gamma * (slope - tau) / max(tau, 1e-9))
    lam_t = float(np.clip(lam_t, lam_min, lam_max))

    Z_t = zentrube_window(win, t, lam_t, use_sigma=use_sigma, ddof=ddof)
    return lam_t, Z_t

# ------------------------------
# (Optional) tiny usage example:
# x = np.random.randn(5000)
# Z = z_adapt_series(x, lam0=0.03, win=128, hop=16, use_sigma=False, ddof=0)
# ------------------------------
```

5.6 I/O Artifacts (CSV/JSON) — Minimal, Reproducible

Input CSV (example)

```
timestamp,value
2025-08-23T12:00:00Z,0.313
2025-08-23T12:00:01Z,0.322
...
```

Output CSV (rolling)

```
t,lambda,use_sigma,window,hop,Z
0,0.03,false,60,30,0.1823
1,0.03,false,60,30,0.1910
...
```

Run manifest (JSON)

```
{
  "spec": "zentrube-manifest-1.0",
```

```
 "kernel": {"mode":"canonical", "lambda":0.03, "stat":"variance"},
 "window": {"length":60, "hop":30, "units":"samples"},
 "series": {"n": 10240, "dt": 1.0, "unit":"s"},
 "weights": [],
 "beta": null,
 "baseline": null,
 "session": {"sig":"SIG-<sha256>", "seed": 12345, "timestamp":"2025-08-23T12:34:56Z"},
 "notes": "deterministic; offline; tested-kernel"
}
```


Note: Constant-variance datasets will produce flat Zentrube traces ($Z \approx 0$). This is an expected behavior, indicating stable alignment rather than a failure of the method.

5.7 Determinism, Stability, and Complexity

- **Determinism:** fix seeds where randomness is used (e.g., synthetic tests), log `seed` in the manifest, export `session.sig` (hash of inputs + params).
- **Numeric stability:** use float64; keep the **+1** inside `log(·)`; clip or guard small denominators with ε (e.g., 1e-9 to 1e-12).
- **Monotone time:** ensure `t` increments consistently with the window strategy; if timestamps are irregular, normalize `t` by median inter-timestamp.
- **Complexity:** O(n) per pass; rolling windows can be O(n) with incremental updates if needed.
- **Memory:** streaming implementations can keep O(window) memory.

Note: Failure modes should be documented alongside successes. For example: in extremely noisy data, λ may need to be tuned upward to prevent false positives; in constant-variance signals, Zentrube correctly outputs near-zero. Including error bars and confidence intervals in benchmarks is recommended for rigor.

5.8 Minimal Unit Tests (Plain Text Checklist)

1. **Collapse checks**

- $\text{Var} \to 0 \Rightarrow Z \to 0$
- $\lambda \to \infty \Rightarrow Z \to 0$

2. **Monotone sanity**

- For injected variance ramp, Z should increase before decay dominates.

3. **Window consistency**

- Sliding vs. cumulative produce expected differences; `t` aligned with method.

4. **Ratio safety**

- Equal lengths & aligned timestamps; ε prevents div-by-zero.

5. **Directional sign** (rupture-aware)

- $\Delta_t > 0 \rightarrow$ positive sign; $\Delta_t < 0 \rightarrow$ negative sign.

6. **Multiband β**

- $\beta < 1$ amplifies small drifts; $\beta > 1$ emphasizes large drifts.

5.9 One-Page Offline Demo (Optional Skeleton)

- Single HTML file with:
  - Textarea to paste a numeric series (comma-separated).
  - Inputs: `λ`, window, hop, `useSigma` checkbox.
  - Buttons: **Compute**, **Export CSV**, **Export Manifest**.
  - Two `<canvas>` elements (or simple SVG) to plot `Z(t)` and `EdgeSlope(t)`.
  - Optionally compute Adaptive Z via rollingZAdaptive(...) (see section 5.2.1) and plot λ(t).
- No external libraries; ES5 only; runs fully offline.

Copy-ready formula recap (plain text)

- **Canonical:**
  $Zentrube_t = \log(Var(x_{0:t}) + 1) \times \exp(-\lambda t)$

- **Weighted (multi-signal):**
  $Z\_weighted_t = \log(\Sigma_i\, w_i \times Var(x_{i0:t}) + 1) \times \exp(-\lambda t)$

- **Rupture-aware:**
  $Z_{rt} = \log(Var(x_{0:t}) + 1) \times \exp(-\lambda t) \times ((x_t - x_0) / (x_0 + \varepsilon))$

- **Ratio (dual-path):**
  $Z\_ratio_t = \log(Var(x_{0:t} / y_{0:t}) + 1) \times \exp(-\lambda t)$

- **Multiband (spectral fusion):**
  $Z\_\Delta,multi_t = \log(\Sigma\_b\ w\_b\ Var(band\_b_{0:t}) + w\_C\ Var(centroid_{0:t}) + w\_F\ Var(flux_{0:t}) + 1) \times \exp(-\lambda t) \times sgn(\Delta_t) \times |\Delta_t|^{\wedge}\beta$

Section 6: Domain Templates & How Zentrube Complements Classical Tools

Applicability Across Domains

While this white paper highlights five validated demonstration domains, Zentrube is not confined to these examples. The same entropy kernel applies to any system where **drift, alignment, or rupture** emerges — from physiology and networks to markets, engineering, and symbolic intelligence. In every tested context so far, Zentrube has provided **earlier and clearer** results than traditional entropy or variance-only measures. The domains showcased here represent a cross-section, but the scope of applicability is **deliberately universal**. These gains remain subject to the observation-only conditions noted earlier in the paper.

Universality note:

Zentrube is redefining entropy for **time-varying systems**: it measures entropy drift, symbolic alignment, and readiness dynamics in sequences and fields. Because entropy and symbolic structure exist in every domain and function, Zentrube formulas can — in principle — be useful everywhere. The list below is **illustrative, not exhaustive**.

Before operational use, users should test and evaluate against their own data and risks; see **Section 1: Readiness & Verifiable Pathways** for guidance on the validation levels expected prior to deployment.

Notation reminder (plain text):

- Canonical: $Zentrube_t = \log(S(x_{0:t}) + 1) \times \exp(-\lambda t)$
- Variants referenced: Z_weighted, Z_r (directional), Z_ratio, Z_Δ,multi, Z_cluster, Z_pol, Z + JS (symbolic distribution term)

6.1 Symbolic Intelligence & Search

Use Drift in meaning, synonym chains, semantic networks.
Complement Adds traceable, offline field scoring beside existing semantic tools.
Template Canonical Z or Z + JS over symbolic traits.

6.2 AI Systems & Model Governance

Use Monitor prompt/response drift, evaluation stability, feedback-loop resonance.
Complement Provides horizon-aware regression signals alongside model metrics.
Template Z_weighted (multiple evals) + Z_ratio (vs. baseline run) + λ sweep.

6.3 AI Safety & Alignment

Use Alignment drift in guardrails, refusal/acceptance patterns, long-horizon behaviors.
Complement Adds time-cooled alerts without disclosing sensitive data.
Template Z_cluster (policy groups) + Z_r (direction of change).

6.4 Deepfake & Content Authenticity

Use Non-local artifact drift in audio/video/text features over time.
Complement Temporal entropy guardrail beside classifiers.
Template Z_Δ,multi (bands/centroid/flux) + Z_r.

6.5 Truth/Deception & Forensic Signals

Use Polarity between corroborating vs. contradicting cues.
Complement Adds signed field dynamics to forensic review.
Template Z_pol (positive/negative halves) + Z_cluster (sources).

6.6 Cybersecurity Telemetry

Use Rupture/healing framing across auth, netflow, endpoints, behavior indices.
Complement Calmer than static thresholds; λ makes detection horizon explicit.
Template Z_r (directional), Z_ratio (de-bias vs. background load), σ/Var guidance.

6.7 Software Logs & DevOps

Use Release regressions, error spikes, silent recovery after fixes.
Complement Adds drift context to service objectives and alerts.
Template Z_weighted (rates/latency) + Z_r (post-remediation healing).

6.8 Storage/Systems Observability

Use Write/read path entropy, fragmentation/compaction drift.
Complement Time-cooled lens on top of I/O counters.
Template Canonical Z + Z_cluster (volumes/pools) + Z_ratio (vs. control).

6.9 Telecom & Communication

Use Jitter, delay, loss, startup traces, QoS/QoE drift.
Complement Interpretable memory horizons for operational tuning.
Template Z_weighted (jitter/delay/loss) + Z_ratio (target/baseline flows).

6.10 Imaging & Visual Entropy

Use Frame-wise clarity/structure drift; band/region fusion.
Complement Adds temporal awareness beside existing quality indices.
Template Z_weighted (bands/regions) + Z + JS (pixel/value distributions).

6.11 Audio Clarity & Perception

Use Clarity drift using multiband fields (energy, centroid, flux).
Complement Works with level-matched listening and equalization practices.
Template Z_Δ,multi with signed Δ_t; conservative bounds and audits.

6.12 Physiology (Non-Clinical)

Use HRV/ECG/EEG drift for meditation, fatigue, coherence studies.
Complement Time-cooled entropy beside established summary metrics.
Template Z_r (short windows), Canonical Z (stability bands), Z_pol (dual states).

6.13 Early-Warning Vitality (Analytics, Non-Clinical)

Use Composite indicators for pre-transition drift (ethical, non-diagnostic).
Complement Governance overlay for monitoring programs.
Template Z_weighted + Z_cluster (cohorts) + λ sweep.

6.14 Insurance & Risk Governance

Use Drift in assumptions, survival models, portfolio behavior.
Complement Independent oversight signal for long-tail risk.
Template Canonical Z + Z_cluster (segments).

6.15 Markets & Currency (Observation-Only)

Use Rupture, rebound, silent healing; multi-scale alignment analysis.
Complement Observation and governance (no trade decisions).
Template Z_r, Z_ratio (vs. benchmark), Z_cluster (baskets).

6.16 Geometry & Structural Modeling

Use Curvature/strain/tolerance stack drift in simulations and designs.
Complement Entropy overlays highlight pre-failure changes.
Template Canonical Z (scalar fields), Z_weighted (multi-metrics).

6.17 Construction & Civil Monitoring

Use Settle/creep/strain drift across structures and sites.
Complement Early-change sensitivity for health monitoring stacks.
Template Canonical Z + Z_cluster (locations).

6.18 Materials & Chemistry

Use Multi-descriptor drift (phase, grain, reaction progress).
Complement Interpretable fusion of lab signals.
Template Z_weighted + Z_elemental (symbolic mapping when helpful).

6.19 Semiconductors & Electronics

Use Drift in current/voltage/resistance traces under stress.
Complement Field view alongside device curves.
Template Z_ratio (I/V) + Z_weighted (multi-probes).

6.20 Combustion & Thermal Systems

Use Edge-state drift during ignition/quench; oscillation stability.
Complement Time-cooled overlays for thermodynamic monitors.
Template Z_Δ,multi (temp/pressure spectra) + Z_r.

6.21 Flight Dynamics & Glide

Use Drift in lift/drag/attitude envelopes; pre-stall/coherence windows.
Complement Entropy overlays for envelope protection.
Template Z_weighted (aero channels) + Z_r.

6.22 Spaceflight & Trajectory Drift

Use Long-horizon guidance and control series.
Complement Field overlays for edge-case envelopes.
Template Z_weighted + Z_r.

6.23 Weather Forecasting

Use Multi-parameter drift (pressure, temperature, humidity, wind).
Complement Early-change markers alongside numerical models.
Template Z_Δ,multi (parameters) + λ sweeps.

6.24 Natural Disaster Tracking

Use Onset and escalation vs. stabilization (cyclones, floods, seismic proxies).
Complement Rupture/healing framing for hazard monitoring.
Template Z_r + Z_weighted (multi-sensor fusion).

6.25 Maritime & Navigation

Use Sea-state, hull response, guidance coherence drift.
Complement Time-cooled overlays for routing safety.
Template Z_weighted + Z_ratio (vs. met-ocean baseline).

6.26 Non-Invasive Subsurface Sensing

Use Proxy signal drift for oil/water/mineral prospects.
Complement Entropy lens for multi-sensor fusion.
Template Z_weighted + Z + JS.

6.27 Transportation Safety & Accident Prevention

Use Pre-incident drift in telemetry, driver, and vehicle states.
Complement Early-warning overlays to safety dashboards.
Template Z_weighted + Z_r + λ sweeps.

6.28 Sports Officiating Fairness

Use Trajectory/pose/event entropy near edge decisions.
Complement Consistency bands and replay governance.
Template Z_ratio (target vs. reference) + Z_r.

6.29 Education & Learning Analytics

Use Learning drift, attention stability, retention decay.
Complement Ethical, offline signals for pedagogy tuning.
Template Canonical Z + Z_pol (engage/disengage).

6.30 Social Polarization & Collective Dynamics

Use Polarity and cluster drift in discourse signals.
Complement Transparent horizons for moderation/governance (observation).
Template Z_pol + Z_cluster + Z_ratio (vs. neutral baselines).

6.31 SEO & Attention Dynamics

Use Entropy of query/visit patterns; silent healing after changes.
Complement Time-cooled view for content and rollout decisions.
Template Z + JS (distribution shifts) + Z_r.

6.32 Supply Chain & Operations

Use Lead-time, fill-rate, defect drift across nodes.
Complement Early deviation flags in planning loops.
Template Z_weighted + Z_cluster (sites).

6.33 Energy & Smart Grids

Use Load/voltage/frequency drift and recovery.
Complement Horizon-aware overlays for demand response.
Template Z_weighted + Z_r.

6.34 Edge/IoT Sensing

Use Environmental/telemetry drift across constrained devices.
Complement Lightweight, offline scoring at the edge.
Template Canonical Z (ES5-safe) + Z_weighted.

6.35 Public Health Surveillance (Non-Diagnostic)

Use Population-level drift in non-identifying indicators.
Complement Observation for readiness planning.
Template Z_cluster (regions) + Z_ratio (vs. seasonal baselines).

6.36 Financial Fraud & Transaction Monitoring [Observation-only]

Use: Drift in transaction patterns, silent anomalies in payment flows, emerging fraud clusters.
Complement: Adds interpretable drift overlays to traditional fraud-detection models.
Template: Z_ratio (transaction vs. baseline flow), Z_cluster (accounts/regions), λ sweep for temporal horizon.

6.37 Environmental Monitoring (Pollution / Air Quality)

Use: Drift in pollution measures (e.g., PM2.5, NO_2, ozone) around regulatory thresholds or seasonal baselines; early warning of urban air-quality spikes.

Complement: Extends weather/forecasting coverage by adding entropy overlays to environmental health datasets, supporting proactive interventions.
Template: Z_Δ,multi (temperature + humidity + PM channels) + Z_weighted (multi-sensor fusion), λ sweep for sensitivity bands.

6.38 Biology & Ecology (Genomics / Ecosystems)

Use: Drift in genomic sequences, species counts, or ecological indicators; detection of biodiversity ruptures, evolutionary shifts, or sudden population collapses.
Complement: Adds entropy-driven perspective to ecology and bioinformatics, capturing alignment changes beyond classical statistical diversity indices.
Template: Z_cluster (species groups or genomic clusters) + Z_pol (polarity in ecosystem health), optional JS divergence for categorical genomic drift.

6.39 AI Embedding Drift (LLM / Model Outputs)

Use: Drift in embedding spaces of ML/LLM models; detect concept shifts, output instability, or alignment decay in real time.
Complement: Provides embedding-level transparency to complement existing AI governance frameworks, enabling proactive model monitoring without retraining overhead.
Template: Z_weighted (variance across embedding dimensions) + Z_ratio (vs. baseline embeddings), λ adaptive for evolving model states.

Note on Limitations

In domains with **constant baselines** or very low variance, Zentrube will remain flat ($Z \approx 0$). This is an expected behavior, indicating stable alignment rather than a failure of the method. Users should interpret flat lines as **evidence of stability**, not loss of sensitivity.

Quick Template Chooser (plain text)

- Need direction (worse vs. better)? → use Z_r
- Many descriptors? → use Z_weighted
- Target vs. baseline? → use Z_ratio
- Bands/channels/spectra? → use Z_Δ,multi (+ β)
- Groups/cohorts? → use Z_cluster
- Positive vs. negative halves? → use Z_pol
- Symbolic/categorical distributions? → use Z + JS

Section 7: Validation & Test Plan (Open, Reproducible)

Validation of Zentrube requires more than formula correctness. Because entropy drift can appear in any sequence or system, testing must be multi-layered: mathematical, synthetic, and domain-grounded.

The guiding principle is **open, auditable, and reproducible runs** that can be verified by independent reviewers. Where performance claims are made (e.g., earlier detection), they are **[Observation-only, prototype tests]** and must be validated with **statistical measures** (ROC, AUC, error bars) before operational adoption.

7.1 Mathematical Identities (Unit Tests)

Objective Ensure the canonical form and variants respect theoretical boundaries.

Checks

- Collapse to classical:
  • $\text{Var} \to 0 \Rightarrow Z \to 0$
  • $\lambda \to \infty \Rightarrow Z \to 0$
- λ sweeps: confirm monotone horizon shortening with higher λ.
- +1 stability guard: validate no singularities even with zero variance.
- Polarity consistency: $\Delta\_t > 0 \to$ positive; $\Delta\_t < 0 \to$ negative.
- **Failure mode check**: constant-variance datasets should yield flat $Z \approx 0$, interpreted as stability.

7.2 Synthetic Suites

Objective Validate Zentrube against known, controlled conditions.

Design

- Injected variance ramps → Z rises predictably then decays.
- Injected pulses/bursts → Z responds with clear rupture markers.
- Directional injections (increase vs. recovery) → Z_r polarity flips correctly.
- Ratios (x/y) with controlled baselines → Z_ratio matches expected normalized drift.
- Add error bars and ROC/AUC curves to quantify robustness under different λ.

7.3 Domain Galleries (Illustrative Prototypes)

Objective. Demonstrate Zentrube's interpretability against familiar benchmarks using lightweight prototypes — both browser-based (HTML + JS, offline) and Excel-based (mathematical/actuarial models). These are observation-only builds, created to evaluate Zentrube's effectiveness versus traditional entropy methods across diverse domains. None are predictive or operational; they exist to showcase reproducibility, interpretability, and practical walk-throughs of the framework.

Prototype Categories

- **Signal & Systems.** Audio clarity, symbolic compression, telecom jitter traces, cybersecurity telemetry, market windows, and physiological segments (ECG/HRV).
- **Crowd & Spatial Flow.** Lane heatmaps, gate policy simulations, and recovery mapping for coordinated crowd or traffic flow.
- **Geometry & Imaging.** Classical vs symbolic overlays, visual entropy correction, and imaging clarity demonstrations with Z-maps.
- **Insurance & Finance.** Browser and Excel prototypes for life insurance, general insurance, and annuities — demonstrating entropy-based valuation shifts, stability ranges, and rupture detection compared with classical actuarial models.
- **Currency & Market Systems.** Zentrube applied to USD/INR, EUR/USD, and other flows, highlighting rupture (crises) and healing (recoveries). Integrated into the Entropy Drift Calculator as observation-only modules.
- **General Mathematical Functions.** Prototypes created for handling **common mathematical forms (linear, exponential, and entropy-driven equations)** that recur across all domains. These tools allow domain-agnostic testing, ensuring Zentrube's drift detection is not tied to any single industry but applicable universally.
- **Symbolic Intelligence Tools.** Prototypes for **symbolic search and interpretation**, using entropy drift to expand, refine, and stabilize symbolic information. These illustrate how Zentrube supports knowledge organization and retrieval through symbolic alignment.

Interpretive Note.
These prototypes are not attached in this release. They serve to illustrate that Zentrube has been **walked through in practice**, not just theorized — from browser-based symbolic experiments to Excel-based mathematical toolkits. All remain observation-only, reinforcing the project's ethical stance: to reflect symbolic drift with clarity, not to predict or trade.

7.4 Manifests & Replayability

Every run should export:

- Parameters: λ, window, hop, weights, β, baselines.
- Seed: for synthetic or randomized setups.
- Session signature: hash of inputs + parameters for replay.
- Manifests (JSON/CSV): standardized format for peer sharing.

This enables **exact replay by independent reviewers**, a critical safeguard against irreproducible claims.

7.5 Ethical Safeguards

Zentrube is an **observation and governance framework**, not an autonomous decision trigger.

- Do not connect directly to automated trading, blocking, or clinical actions.
- Always label outputs as **"Observation-only."**
- Show formulas transparently in UIs and exports.
- Publish interpretation ranges and λ explicitly to prevent black-box use.
- Encourage **peer review and independent replication** before any deployment.

Section 8: API Sketches

Conventions

- **Formula (plain text)**: $\text{Zentrube}_t = \log(S(x_{0:t}) + 1) \times \exp(-\lambda t)$
- $S(x_{0:t}) \in \{\text{Var}, \sigma\}$; choose **variance** (tail-sensitive) or **stddev σ** (cross-series comparability).
- Use **sliding windows** in practice; keep t consistent with your windowing scheme.
- Add a small ε (e.g., 1e-12) when dividing or taking ratios to guard against singularities.

Note: Default $\lambda = 0.03$ is provided in the snippets as a typical ~30-window horizon. Users should sweep λ values (e.g., 0.01–0.05) and validate with ROC/AUC or lead-time detection curves in their own datasets. Sliding windows are the default; cumulative windows can be used for stability checks. In constant-variance data, Zentrube will correctly output $Z \approx 0$, which indicates stability rather than failure.

8.1 JavaScript (ES5-safe, single-file friendly)

```
// Zentrube_t = log(S(x_0:t) + 1) * exp(-lambda * t)

// S = variance (default) or sigma (stddev) when useSigma = true

function zentrube(series, t, lambda, useSigma) {

  var n = series.length, i, mu = 0.0, acc = 0.0;

  if (n === 0) return 0.0;
```

```
  for (i = 0; i < n; i++) mu += series[i];
  mu = mu / n;

  if (useSigma) {
    for (i = 0; i < n; i++) { var d = series[i] - mu; acc += d * d; }
    var sigma = Math.sqrt(acc / n);
    return Math.log(sigma + 1.0) * Math.exp(-lambda * t);
  } else {
    for (i = 0; i < n; i++) { var d2 = series[i] - mu; acc += d2 * d2; }
    var v = acc / n;
    return Math.log(v + 1.0) * Math.exp(-lambda * t);
  }
}

// Rolling helper (equal windows, fixed hop)
function rollingZ(series, window, hop, lambda, useSigma) {
  var out = [], start = 0, t = 0;
  while (start + window <= series.length) {
    var slice = series.slice(start, start + window);
    out.push(zentrube(slice, t, lambda, useSigma));
    start += hop; t += 1;
  }
  return out;
}
```

Directional (rupture-aware) stub

```
// Z_rt = log(S(x_o:t) + 1) * exp(-lambda * t) * ((x_t - x_0) / (x_0 + eps))
```

```
function zentrubeR(series, t, lambda, useSigma, eps) {
 if (t <= 0 || series.length === 0) return 0.0;
 var base = zentrube(series.slice(0, t), t, lambda, useSigma);
 var x0 = series[0], xt = series[t - 1];
 var delta = (xt - x0) / ((x0 + (eps || 1e-12)));
 return base * delta;
}
```

Ratio (baseline-adjusted) stub

```
// Z_ratio_t = log(S( (x_o:t)/(y0:t) ) + 1) * exp(-lambda * t)
function zentrubeRatio(x, y, t, lambda, useSigma, eps) {
 var n = Math.min(x.length, y.length, t);
 if (n <= 0) return 0.0;
 var i, r = new Array(n), e = (eps || 1e-12);
 for (i = 0; i < n; i++) r[i] = (x[i] + e) / (y[i] + e);
 return zentrube(r, t, lambda, useSigma);
}
```

Multiband (bands/centroid/flux) stub

```
// Z_Δ,multi_t = log( sum_b w_b*Var(band_b0:t) + wC*Var(centroid0:t) + wF*Var(flux0:t) + 1 ) *
exp(-lambda*t) * sgn(Δ)*|Δ|^β
function zentrubeMulti(bands, wBands, centroid, wC, flux, wF, t, lambda, delta, beta) {
 var i, S = 0.0, b, n;
 for (i = 0; i < bands.length; i++) {
  b = bands[i]; n = Math.min(b.length, t);
  if (n > 0) {
   S += (wBands[i] || 0) * zVar(b.slice(0, n));
  }
 }
```

```
  if (centroid && Math.min(centroid.length, t) > 0) S += (wC || 0) * zVar(centroid.slice(0, t));
  if (flux && Math.min(flux.length, t) > 0)     S += (wF || 0) * zVar(flux.slice(0, t));
  var mag = 1.0;
  if (typeof delta === "number" && typeof beta === "number") {
    mag = (delta === 0 ? 0 : (delta > 0 ? 1 : -1)) * Math.pow(Math.abs(delta), beta);
  }
  return Math.log(S + 1.0) * Math.exp(-lambda * t) * mag;

  function zVar(arr) {
    var i, mu = 0, acc = 0, n = arr.length;
    for (i = 0; i < n; i++) mu += arr[i];
    mu /= n;
    for (i = 0; i < n; i++) { var d = arr[i] - mu; acc += d * d; }
    return acc / n;
  }
}
```

8.2 Python (NumPy-friendly, minimal)

```
# Zentrube_t= log(S(x_o:_t) + 1) * exp(-lam * t)
import numpy as np

def zentrube(x, t, lam=0.03, use_sigma=False):
    """
    x: 1D array-like; use x[:t] as the current window
    t: integer "step" index aligned with your windowing
    lam: decay constant (1/lam ≈ memory horizon)
```

```
    use_sigma: if True, use stddev; else variance
    """
    x = np.asarray(x[:t], dtype=np.float64)
    if x.size == 0:
        return 0.0
    S = x.std() if use_sigma else x.var()
    return float(np.log(S + 1.0) * np.exp(-lam * t))


def rolling_z(x, window, hop, lam=0.03, use_sigma=False):
    out, step = [], 0
    for start in range(0, len(x) - window + 1, hop):
        out.append(zentrube(x[start:start+window], window, lam, use_sigma))
        step += 1
    return np.asarray(out, dtype=np.float64)
```

Directional (rupture-aware)

```
# Z_rt = log(S(x_o:t) + 1) * exp(-lam*t) * ((x_t - x_0) / (x_0 + eps))
def zentrube_r(x, t, lam=0.03, use_sigma=False, eps=1e-12):
    x = np.asarray(x, dtype=np.float64)
    if t <= 0 or x.size == 0:
        return 0.0
    base = zentrube(x[:t], t, lam, use_sigma)
    delta = (x[t-1] - x[0]) / (x[0] + eps)
    return float(base * delta)
```

Ratio (baseline-adjusted)

```
# Z_ratio_t = log(S((x0:t)/(y0:t)) + 1) * exp(-lam*t)
```

```
def zentrube_ratio(x, y, t, lam=0.03, use_sigma=False, eps=1e-12):
    x = np.asarray(x[:t], dtype=np.float64)
    y = np.asarray(y[:t], dtype=np.float64)
    n = min(x.size, y.size)
    if n == 0:
        return 0.0
    r = (x[:n] + eps) / (y[:n] + eps)
    S = r.std() if use_sigma else r.var()
    return float(np.log(S + 1.0) * np.exp(-lam * t))
```

Multiband (bands/centroid/flux)

```
# Z_Δ,multi_t = log(sum_b w_b*Var(band_b0:t) + wC*Var(centroid0:t) + wF*Var(flux0:t) + 1) * exp(-lam*t) * sgn(Δ)*|Δ|^β
def zentrube_multiband(bands, w_b, centroid=None, w_c=0.0, flux=None, w_f=0.0,
                       t=0, lam=0.03, delta=None, beta=1.0):
    def _var(a):
        a = np.asarray(a[:t], dtype=np.float64)
        return 0.0 if a.size == 0 else float(a.var())
    S = 0.0
    for b, w in zip(bands, w_b):
        S += (w or 0.0) * _var(b)
    if centroid is not None: S += (w_c or 0.0) * _var(centroid)
    if flux     is not None: S += (w_f or 0.0) * _var(flux)
    mag = 1.0
    if delta is not None:
        mag = (0.0 if delta == 0 else (1.0 if delta > 0 else -1.0)) * (abs(delta) ** beta)
    return float(np.log(S + 1.0) * np.exp(-lam * t) * mag)
```

Cluster (groups/cohorts)

```
# Z_cluster_t = sum_k w_k * [ log(Var(cluster_k0:t) + 1) * exp(-lam*t) ]

def zentrube_cluster(groups, weights, t, lam=0.03):
    acc = 0.0
    for g, w in zip(groups, weights):
        g = np.asarray(g[:t], dtype=np.float64)
        S = 0.0 if g.size == 0 else float(g.var())
        acc += (w or 0.0) * (np.log(S + 1.0) * np.exp(-lam * t))
    return float(acc)
```

Polarity (positive vs negative halves)

```
# Z_pol_t = log(|S_pos - S_neg| + 1) * exp(-lam*t) * sgn(S_pos - S_neg)

def zentrube_pol(pos, neg, t, lam=0.03, use_sigma=False):
    pos = np.asarray(pos[:t], dtype=np.float64)
    neg = np.asarray(neg[:t], dtype=np.float64)
    Sp = (pos.std() if use_sigma else pos.var()) if pos.size else 0.0
    Sn = (neg.std() if use_sigma else neg.var()) if neg.size else 0.0
    d  = Sp - Sn
    sgn = 0.0 if d == 0 else (1.0 if d > 0 else -1.0)
    return float(np.log(abs(d) + 1.0) * np.exp(-lam * t) * sgn)
```

8.3 Minimal I/O & Manifests (for reproducibility)

CSV Input

```
timestamp,value
2025-08-23T12:00:00Z,0.313
2025-08-23T12:00:01Z,0.322
...
```

CSV Output (rolling)

```
t,lambda,use_sigma,window,hop,Z
0,0.03,false,60,30,0.1823
1,0.03,false,60,30,0.1910
...
```

Manifest (JSON)

```
{
  "spec": "zentrube-manifest-1.0",
  "kernel": {"mode":"canonical", "lambda":0.03, "stat":"variance"},
  "window": {"length":60, "hop":30, "units":"samples"},
  "weights": [],
  "beta": null,
  "baseline": null,
  "seed": 12345,
  "session_sig": "SIG-<sha256-of-inputs-and-params>",
  "notes": "offline; deterministic; observation-only"
}
```

Note: For reproducibility, every manifest should declare: λ, window/hop, weights, β, baselines, and a session signature. Sharing manifests ensures exact replay by independent reviewers and prevents hidden parameters from biasing results.

8.4 Practical Notes (determinism & stability)

- Use **float64**; keep **`+1` inside log**; clamp denominators with **ε** (e.g., `1e-12`).
- Align timestamps and lengths for ratio computations.
- Make `t` increase with each produced window; if timestamps are irregular, normalize `t` to **median inter-timestamp**.
- Export manifests and a **session signature** (hash) for exact replay.
- Complexity is **O(n)** per pass; rolling windows can be implemented incrementally if needed.

Section 9: Governance & Documentation

Zentrube is intended to be **open, reproducible, auditable, and safe**. This section defines the artifacts, review process, and documentation standards that ensure consistent implementations and trustworthy results. Where applicable, outputs must be labeled as **"observation-only"** and accompanied by manifests to prevent misuse or black-box interpretation.

9.1 Run Manifest & Replayability
Every execution MUST emit a machine-readable manifest and a session signature for exact replay.
Required fields (plain text names):
formula, S, λ, window, hop, weights, β, baseline, version, seed, session_sig, notes

JSON schema (informal)

```
{
  "spec": "zentrube-manifest-1.0",
  "formula": "canonical | weighted | rupture | ratio | multiband | cluster | polarity | z_plus_js",
  "S": "variance | sigma",
  "λ": 0.03,
  "window": {"length": 60, "units": "samples"},
  "hop": 30,
  "weights": [0.2, 0.2, 0.6],
  "β": null,
  "baseline": null,
  "version": "v1.3.0",
  "seed": 12345,
  "session_sig": "SIG-<sha256(inputs+params)>",
  "notes": "observation-only; offline; deterministic"
}
```

Replay rule
Given the same inputs + manifest, the implementation MUST produce the same outputs (deterministic runs). Flat outputs (`z ≈ 0`) in constant-variance signals are expected and should be documented as a **stability case, not a failure mode**.

9.2 Quality Assurance (QA) Pipeline

Tiers

- Smoke tests (fast): run canonical kernels on tiny fixtures; assert non-NaN, monotone λ behavior, and collapse identities.
- Unit tests (formula): mathematical identities; polarity and ratio guards; β behavior.
- Synthetic suites: ramps, bursts, direction flips; known expected Z profiles.
- Domain micro-benchmarks: tiny, curated examples (audio, imaging, cyber, physiology, telecom).
- Regression pack: lock outputs for canonical fixtures; fail CI on drift.

Outputs

- JSON test logs with pass/fail and summary stats.
- Human-readable report (markdown) per release.
- **Statistical checks (ROC, AUC, error bars)** for parameter sweeps, ensuring robustness across λ ranges.

9.3 Release Bundles

Each tagged release includes:

- Single-file HTML demo (ES5, offline, no dependencies).
- Small test CSVs/JSON (toy audio, toy netflow, toy ECG, toy telecom).
- **README** (quickstart, manifest spec, observation-only note; must state the tagged release version/date exactly as in the manifest).
- **CITATION.cff** (scholarly citation; version and release date must match the tagged release manifest).
- **CHANGELOG.md** (semver entries; latest entry must match the manifest's version/date).
- **LICENSE files** (code + text, kept at repo root).
- **/examples** directory with notebooks or scripts mirroring the HTML demo.

Recommended bundle layout

```
zentrube/
  README.md              # must include release version/date (synced with manifest)
  LICENSE-MIT
  LICENSE-CC-BY-4.0
  CITATION.cff           # version/date synced with manifest
  CHANGELOG.md           # semver entries synced with manifest
  /demo
    zentrube.html
    sample_audio.csv
    sample_cyber.csv
  /src
    js/
    py/
    rust/
  /tests
    smoke/
    unit/
    synthetic/
    domains/
  /manifests
    manifest-examples.json
```

9.4 Versioning & Change Control

- **Semantic Versioning (semver): MAJOR.MINOR.PATCH**
  - **MAJOR:** changes to formula semantics or manifest fields.
  - **MINOR:** new variants, new parameters, non-breaking defaults.
  - **PATCH:** bug fixes, doc updates, CI improvements.
- **Spec freeze:** formula strings and field names in the manifest are contracts; changing them requires a MAJOR version bump.
- **Deprecation policy:** mark deprecated parameters one MINOR ahead with clear migration notes.
- **Metadata alignment:** README, CITATION.cff, and CHANGELOG are part of the release contract and **must always reflect the manifest's version and release date**.

9.5 Licensing

- **Code:** MIT License (permissive; allows broad reuse with attribution).
- **Text & Diagrams:** CC BY 4.0 (attribution required; allows remix/reuse).
- Keep both license files at repo root.
- **Metadata alignment:** Alongside license files, ensure that **README, CITATION.cff, and CHANGELOG remain consistent with the manifest's version/date** for each tagged release (see Sections 9.3–9.4).

9.6 Documentation Set

Minimum docs must include:

- **README.md**: quickstart, canonical formula (plain text), parameter tuning (λ, β, windows), manifest spec, and **observation-only ethics note prominently at the top**. The README must also state the **release version and date**, which must match the tagged manifest (see Section 9.3).

- **Docs/ (or wiki):**
  - **Concepts:** entropy drift, symbolic alignment, interpretation ranges.
  - **API:** JS/Python/Rust function signatures with examples.
  - **How-Tos:** browser offline demo; rolling windows; baselines; weights.
  - **Validation:** reproduce Section 7 ladders with downloadable fixtures, including **ROC/AUC plots and error bars** where applicable.
  - **Ethics:** observation-only guidance, deployment checklists.
  - **Changelog & Migration guides:** must explicitly sync with tagged release version/date.

"Formula card" snippet (plain text)

$\text{Zentrube}_t = \log(S(x_{0:t}) + 1) \times \exp(-\lambda t)$
Where $S \in \{\text{Var}, \sigma\}$, t follows your window schedule, λ sets memory horizon.

9.7 Contribution Workflow

- **Issues:** bug, docs, feature, or research template with minimal reproducible example + manifest.
- **Pull Requests:** must include tests; update README/docs if behavior or parameters change; attach before/after manifests and outputs.
- **Code style:** basic linting; docstrings for public functions; examples included.
- **Review:** at least one maintainer + one community reviewer for formulas or default parameters.

PR checklist (inline)

- Manifests updated
- Tests added/updated (smoke/unit/synthetic/domain)
- Docs updated
- Observation-only disclaimer preserved
- Semver & changelog updated
- **README, CITATION.cff, and CHANGELOG version/date verified against manifest**

9.8 Security & Privacy

- No external calls in demos or tests; all compute is **local**.
- Test data must be **anonymized** or synthetic.
- Do not include secrets in manifests or logs.
- Optional **SECURITY.md** with responsible disclosure instructions (e.g., email alias), response windows, and scope.

9.9 Ethics & Interpretation

- Zentrube outputs are **observation-only**; do not wire them directly to trading, blocking, or clinical actions.

- UIs and exports must display:

  - The formula in plain text.
  - The λ used and its implied memory horizon.
  - Interpretation ranges (example: “0.00–0.05 stable,” “0.05–0.20 early drift,” “>0.20 rupture risk”—calibrated per domain).

- Include a link to Section 1: Pseudoscience vs Readiness for deployment readiness levels.

- **Encourage reviewers to attempt falsification tests** (counterexamples) to strengthen credibility.

9.10 Documentation Artifacts

CITATION.cff (example)

```
cff-version: 1.2.0

title: "Zentrube: Time-Aware Entropy Drift"

authors:

  - family-names: "YourSurname"

    given-names: "YourName"

version: "1.5.0"              # MUST match the tagged release manifest version

date-released: "2025-08-25"   # MUST match the release date of the manifest

doi: ""
```

license: "MIT, CC-BY-4.0"

url: "https://example.org/zentrube"

message: "If you use Zentrube, please cite this work."

README sections (suggested)

- What is Zentrube? (one paragraph)
- Canonical formula (plain text) and intuition
- Quickstart (JS/Python)
- Parameter tuning (λ, windows, β, weights)
- Variants (links to Section 3)
- Reproducibility: manifests, session signatures
- Observation-only ethics
- Contributing + license + citation

9.11 Governance Roles (lightweight)

- **Maintainers**: approve specs, ensure semver discipline, guard ethics language.
- **Editors**: curate documentation and examples; enforce manifest consistency.
- **Reviewers**: community volunteers focusing on tests and reproducibility.

A simple MAINTAINERS.md with contact aliases is sufficient.

9.12 Checklists (copy-paste)

Release checklist

1. **README, CITATION.cff, and CHANGELOG version/date verified against the tagged manifest (must be in sync).**
2. All tests green (smoke/unit/synthetic/domain).
3. Deterministic outputs locked for fixtures.
4. CHANGELOG updated; version bumped (semver).
5. Demo + sample CSVs present.
6. Manifest examples valid.
7. README/docs refreshed.
8. Licenses and CITATION included.
9. Observation-only disclaimer verified across all docs.

Run checklist

- Inputs & timestamps aligned.
- Manifest emitted with seed + session_sig.

- λ, window, hop declared.
- Baselines/weights recorded.
- Exports labeled “observation-only.”

Section 10: Roadmap

0. Roadmap
Zentrube is designed as a living, community-evolving framework. The roadmap balances two priorities:

1. Immediate reproducibility through canonical releases.
2. Progressive expansion into domains, validation, and formal peer review.

The following phases outline the path ahead.

R1: Core Release (Foundations)
Goal: Establish Zentrube’s minimal kernel with transparent, reproducible code.

- Publish canonical, rupture-aware, ratio, and multiband reference implementations.
- Provide open test suites (unit, synthetic, micro-domain fixtures).
- Release single-file HTML demos and minimal Python notebooks for community use.
- Document parameter tuning defaults (λ, β, weights, baselines) for quick setup.
- **Verify all outputs via manifests; treat manifest version/date as the single source of truth.**

R2: Expanded Variants & Dashboards
Goal: Demonstrate extensibility and interpretability across wider contexts.

- Add cluster-weighted, elemental mapping, and polarity templates.
- Provide domain dashboards (symbolic intelligence, audio, imaging, cyber, physiology, telecom, markets, weather).
- Expand manifest schema to support variant-specific fields (β, element weights, polarity splits).
- Include visual demos: drift overlays, λ sweeps, baseline comparisons.
- **Document failure modes (e.g., flat Z in constant variance data) to prevent misinterpretation.**

R3: Community Gallery & Reproducibility
Goal: Build a shared repository of open, reproducible results.

- Launch community gallery with:

    - Reproducible Jupyter notebooks.
    - Audio A/B clarity pages.
    - Cyber/telemetry CSVs.
    - Imaging & physiology overlays.
- Provide API wrappers (Python, JS, Rust) with aligned manifests.
- Add domain setup cheatsheets and starter templates for practitioners.
- Encourage contributions via pull requests, with manifest-verified test cases.
- **Require statistical validation (ROC/AUC curves, error bars) for new contributions.**

R4: Peer Review & Engineering Integration
Goal: Formal validation and adoption in science and engineering pipelines.

- Submit methods papers (canonical + variants) to peer-reviewed venues.
- Publish applied notes (e.g., audio clarity, ECG, cybersecurity) with reproducible datasets.
- Provide CAD/FEA integration notes: entropy overlays in structural models.
- Explore industrial pilots: entropy-aware dashboards for risk, safety, and observability.
- Establish governance circles for ethics, reproducibility, and deployment readiness.
- **Promote "observation-only" disclaimers until trials or regulatory pathways validate operational use.**

Beyond R4: Open Frontier
Longer-term directions (exploratory):

- Adaptive Zentrube (learning λ via reinforcement feedback).
- Quantum-inspired Zentrube (mutual information for entangled drifts).
- Network-topology Zentrube (graph-based entropy drift for social, ecological, and IoT systems).
- Symbolic integration into AI governance frameworks.
- **Sustainability use-cases (climate drift, ecological resilience) as open challenges for the community.**

Section 11: Conclusion — Zero, Entropy, and Unity

Zentrube introduces a time-aware entropy construct that unifies drift detection, symbolic alignment, and readiness awareness across diverse domains. Its canonical form — a log-compressed spread with exponential decay — serves as a minimal yet powerful kernel. From this foundation, a family of variants (weighted, rupture-aware, ratio, multiband, clustering, elemental, polarity) extends its reach while preserving interpretability, reproducibility, and transparency.

The framework emphasizes:

- **Universality** — entropy drift emerges in every system; Zentrube provides a clear, bounded representation.
- **Transparency** — formulas are presented in plain text, implementations remain minimal, and manifests ensure replayability.
- **Reproducibility** — synthetic suites, unit tests, and domain galleries enable open, auditable verification.
- **Governance** — observation-only safeguards and licensing terms promote responsible exploration.
- **Extensibility** — a roadmap connects canonical releases to exploratory variants, including adaptive, quantum-inspired, and topology-aware directions.

Zentrube is not positioned as a replacement for established measures; rather, it complements classical tools by adding directionality, memory horizon, and interpretability. Its scope is intentionally broad, yet its posture remains careful: validation must precede deployment, and current outputs are intended for interpretive governance rather than automated control.

At a conceptual level, Zentrube reframes three foundational ideas:

- **Zero as Alignment, Not Void** — zero is treated as a living baseline around which systems drift, rupture, and recover, rather than as static emptiness.
- **Entropy as a Living Vital Sign** — entropy becomes a bidirectional diagnostic signal of resilience, healing, and readiness, not merely a measure of disorder.
- **Multi-Domain Symbolic Unity** — the same entropy lens can be applied to physiology (the body), hurricanes (nature's dynamics), telecommunications (the network), and cybersecurity and insurance (society), demonstrating versatility without sacrificing interpretability.

By codifying entropy drift as a reproducible, open, and ethically grounded formula, Zentrube offers both modest implementation and wide-ranging implications. It highlights hidden patterns, provides earlier awareness, and frames a unifying language of alignment across disciplines.

Note. All demonstrations in this white paper remain observation-only, serving as reproducibility tests on real datasets. Operational use requires independent peer validation, formal statistical testing, and governance approval.

Appendix A — Interpretation Ranges

Zentrube outputs are **bounded and interpretable**, unlike traditional entropy measures which often diverge or lack intuitive scales. The ranges below are suggested defaults for *generic exploratory use*. **Domain-specific calibration is essential** before operational adoption. All values assume the canonical form unless otherwise noted.

A.1 Suggested Ranges

Zentrube Value	Interpretation	Notes
0.0 – 2.5	**Equilibrium / Calm**	System is near alignment; variance low and stable. Baseline state.
2.5 – 3.5	**Early Drift**	Subtle departures from baseline; may precede visible changes. Useful for pre-transition alerts.
3.5 – 4.5	**High Variability**	Significant drift underway; system under stress. May require monitoring or mitigation.
> 4.5	**Rupture Range**	Strong misalignment or transition event. Typically marks failure, burst, or phase shift.

A.2 Directional Values

- Negative outputs occur only in **directional variants (Z_r)**.
- **Interpretation:**
  - **Negative Z_r → Healing / Recovery phase** (system returning toward baseline).
  - **Positive Z_r → Escalation / Rupture phase** (system drifting further away).

A.3 Cautions

- These thresholds are **illustrative, not universal**. Each domain (audio, imaging, cyber, physiology, etc.) requires calibration using its own baselines and validation datasets.
- Always report alongside results:
  - Formula used (canonical, rupture, ratio, multiband, etc.).
  - Parameters (λ, window length, baselines, weights).
- Values should be interpreted **in context, not isolation**.
  - Example: A "3.2" in audio may signify early clarity loss; in physiology, it may indicate mild fatigue.

Appendix B — Suggested Defaults for λ (Decay Horizon)

The decay parameter **λ** defines how quickly history is "forgotten." Its reciprocal (1/λ) is the **memory horizon** — the number of windows of past information that still influence the present.

- **Smaller λ → longer memory (slower forgetting, stability focus).**
- **Larger λ → shorter memory (faster responsiveness, anomaly focus).**

The ranges below are **prototype-informed starting points**, not absolutes. They must be tuned per dataset and application.

B.1 Suggested λ Ranges by Domain

Domain	Suggested λ Range	Interpretation
Audio (speech/music clarity)	0.02 – 0.05	Short horizons capture bursts, distortions, and clarity drift in real time.
Cybersecurity telemetry (auth/netflow)	0.03 – 0.06	Higher λ favors fast anomaly detection and short-memory bursts typical in telemetry.
Markets (non-trading observation)	0.02 – 0.05	Medium horizon balances sensitivity with noise stability; used for rupture/recovery observation only.
Physiology (ECG, HRV, EEG)	0.01 – 0.03	Longer horizons preserve subtle coherence/recovery trends without over-amplifying noise.

B.2 Notes

- These ranges are **illustrative defaults**, not universal.
- λ tuning should be guided by:
  - Desired **sensitivity** (early drift vs. long-term stability).
  - **Window length** (shorter windows may tolerate slightly larger λ).
  - **Noise characteristics** of the domain.
- Always record the chosen λ in the **manifest** for reproducibility (see Section 9).
- **Best practice:** perform λ sweeps and sensitivity analysis (e.g., ROC/AUC curves) before fixing defaults.

Appendix C — Adoption Steps

Zentrube can be adopted incrementally. Safe use requires **reproducibility and explicit ethical labeling**. The workflow below provides a minimal but sufficient checklist for deployment in any domain.

C.1 Core Steps

1. **Pick Parameters**
   - Select spread operator **S**: use σ for cross-signal comparability; *Var* for tail sensitivity.
   - Choose **λ** from suggested domain ranges (Appendix B).
2. **Set Windows**
   - Define **window length** and **hop size**.
   - Normalize **t** if timestamps are irregular (e.g., by median interval).
3. **Handle Baselines (if applicable)**
   - For ratio or comparative modes, ensure **aligned lengths and timestamps** between series.
   - Add a small offset (+ε) if division risks zeros or negatives.
4. **Bind Exports**
   - Always export:
     1. **CSV/JSON fields** with formula, parameters, outputs.
     2. **Plots** showing classical vs. Zentrube overlays.
     3. **Session signature** (manifest hash) for reproducibility.
5. **Label Ethics**
   - Explicitly tag outputs: **"observation-only."**
   - Add **scope notes** (e.g., non-trading, non-clinical).
6. **Validate Before Production**
   - Add **unit tests**: collapse to classical, λ sweep stability.
   - Build a small **gallery of synthetic + domain samples**.
   - Verify **reproducibility** by replaying manifests.

C.2 Adoption Philosophy

- **Minimal barriers**: a few lines of code (JS/Python) are enough to begin.
- **Reproducibility first**: every run must emit a manifest and export trail.
- **Ethics by design**: labeling and disclaimers prevent misuse in sensitive domains.
- **Progressive adoption**: start with synthetic and low-stakes domains; escalate only after validated, peer-reviewed results.

Appendix D — Worked Examples (Toy + Engineering)

All examples use the tested Zentrube kernel. Numbers are **illustrative** and show how Zentrube (*fz*) modifies or complements classical measures.

D.1 Triangle Area with Symbolic Correction (Right Triangle)
Classical formula: $A = 0.5 \times a \times b$
With $a = 3$, $b = 4 \rightarrow A = 6.0\ m^2$

Entropy driver: Var = 0.50, t = 10, $\lambda = 0.05 \rightarrow fz \approx 0.246$

Mapping: $A_s = A + 2.0 \times fz \approx 6.49\ m^2$

Collapse checks: λ large or Var $\rightarrow 0 \Rightarrow fz \rightarrow 0 \Rightarrow A_s \rightarrow A$.

D.2 Curvature with Stable Radius Mapping (Circle)
Classical: $\kappa = 1/R$, with $R = 1.0 \rightarrow \kappa = 1.0$ (1/m)

Entropy driver: Var = 0.20, t = 5, $\lambda = 0.10 \rightarrow fz \approx 0.111$

Mapping: $R_s = 1.055$ m, $\kappa_s = 0.948$ (1/m)

Collapse checks: Var $\rightarrow 0$ or λt large $\Rightarrow fz \rightarrow 0 \Rightarrow \kappa_s \rightarrow \kappa$.

D.3 Symbolic Compression Drift (Token Distribution)
Classical: ratio = 100/75 = 1.33

Entropy driver: Var = 0.80, t = 20, $\lambda = 0.04 \rightarrow fz \approx 0.264$

Mapping: ratio_s = 1.38 → ~4% improvement suggested.

Interpretation: Zentrube highlights drift potential in compression efficiency.

D.4 Physiology (ECG RR-Intervals, Illustrative)
Classical: SDNN = 42 ms

Entropy driver: Var = 900, t = 50, $\lambda = 0.02 \rightarrow fz \approx 2.50$

Mapping: SDNN_s = 42.25 ms

Interpretation: Adds drift-aware overlay to SDNN. Positive fz → escalation; negative fz (directional Z_r) → recovery.

Note: Illustrative only. Proper validation requires domain datasets (e.g., MIT-BIH) with ROC/AUC analysis.

Presentation note: To strengthen interpretability, worked examples should be paired with **plots or overlays** (classical vs. Zentrube curves) and **percent change highlights** (as in D.3).

D.5 Tolerance Stack (Symbolic Overlay in CAD)

Goal. Show how a classical linear tolerance stack *S* gains a symbolic correction using Zentrube, without changing units or mutating the base definition.

Classical baseline:
$S = \Sigma\ \mu_i$
$\mathrm{Var}(S) = \Sigma\ \mathrm{Var}(\mathrm{part}_i)$

Entropy driver:
$\mathrm{Zentrube}_t = \log(\mathrm{Var}(x_{0:t}) + 1) \times \exp(-\lambda t)$
Example: Var = 0.35, t = 20, λ = 0.04 → fz ≈ 0.134

Mapping (stack sensitivity overlay):
S_s = S + k × s_S × fz
With S = 5.000 mm, k = 1, s_S = 0.08 mm/fz →
S_s ≈ 5.011 mm

Note: k and s_S are calibration parameters, chosen to reflect system-specific sensitivity (units preserved).

Collapse checks:
• Var → 0 ⇒ fz → 0 ⇒ S_s → S
• λt large ⇒ exp(−λt) → 0 ⇒ fz → 0 ⇒ S_s → S

Extensions:
• *Weighted driver:* $Z\_\mathrm{weighted\_u} = \log(\Sigma\ w_i\ \mathrm{Var}(x_{i0:u}) + 1) \times \exp(-\lambda u)$
• *Directional variant:* negative fz (Z_r) indicates recovery; overlay subtracts from S.
• *Ratio mode:* $Z\_\mathrm{ratio\_t} = \log(\mathrm{Var}(x_{0:t} / y_{0:t}) + 1) \times \exp(-\lambda t)$.

Appendix E — Hero Use Cases

Hero Use Cases (Demonstrations)

Zentrube is not only a theoretical construct but a reproducible tool for observing **readiness drift** across diverse domains. The following hero use cases anchor the framework in practice, showing where Zentrube consistently provides earlier, clearer, and more stable signals than classical entropy or variance-only approaches.

Each demonstration illustrates Zentrube's ability to capture **Ground Zero drift** — the subtle fluctuations around a true baseline of alignment — rather than waiting for **Edge Zero ruptures**, where breakdown is already visible. In this framing, zero is not void but alignment, and entropy becomes a *living vital sign* that signals rupture, recovery, and readiness.

Validation basis (real datasets):

- **ECG (Physiology):** MIT-BIH arrhythmia records.
- **Hurricane (Natural Systems):** IBTrACS archive — Hurricane Dorian (2019).
- **Cybersecurity:** CICIDS-2017 Friday (Benign vs. DDoS LOIC).
- **Insurance (Annuities):** Published actuarial life tables.
- **Telecom:** Nokia mobile network join trace.

Taken together, these demonstrations highlight Zentrube's **multi-domain symbolic unity**: the same entropy lens applies across physiology (the body), hurricanes (nature's dynamics), telecommunications (the network), finance and insurance (society), and cybersecurity (threat and recovery). This universality is what makes Zentrube distinctive — a single, minimal formula capable of describing drift wherever systems move, escalate, or return to balance.

For the philosophical background of Ground vs. Edge states, see Appendix L. Together, these appendices show both the **why** (conceptual foundation) and the **how** (practical demonstrations).

Footnote: All demonstrations are observation-only and non-operational. They confirm reproducibility on real datasets but are not intended for clinical use, production deployment, monitoring, or enforcement without peer validation, governance, and domain-specific certification.

E.1 Hero Use Case — Hurricane Drift Detection

Domain: Meteorology, climate science, disaster preparedness

Problem with Classical Hurricane Metrics

- Conventional thresholds (tropical storm ≥34 kt, hurricane ≥64 kt, major ≥96 kt) only flag events **after winds cross category cutoffs**.
- Variance of wind speed or pressure rises too late — **drift before rupture is missed**.
- Forecasting models often rely on ensembles but lack a **bounded, interpretable entropy signal** to anticipate escalation or recovery.

Zentrube Approach

Formulas evaluated on real **IBTrACS data** (6h resolution, maximum sustained winds):

- **Canonical drift**:
  $\text{Zentrube}_t = \log(\text{Var}(x_{0:t}) + 1) \times \exp(-\lambda t)$
- **Differential slope sensitivity**:
  $Z\_\text{diff}_t = \log(\text{Var}(\Delta x_{0:t}) + 1) \times \exp(-\lambda t)$
- **Directional (escalation vs. recovery)**:
  $Z\_\text{tr}_t = \log(\sigma(x_{0:t}) + 1) \times \exp(-\lambda t) \times \Delta_t$

Where:

- $x_{0:t}$ = rolling wind speed (knots) from IBTrACS
- Δ_t = slope polarity (escalation vs weakening)
- λ = decay constant (symbolic memory horizon, ~0.03)

Case A: Hurricane Dorian (2019, Atlantic)

What Changed (Validated on IBTrACS data):

- **Tropical Storm threshold (34 kt):**
  – Classical: detected at *2019-08-24 (18 UTC)*
  – Zentrube: flagged drift ~24h earlier
- **Category 1 (64 kt):**
  – Classical: *2019-08-25 (18 UTC)*
  – Zentrube: drift visible ~39h earlier
- **Category 3 Major (96 kt):**
  – Classical: *2019-08-31 (06 UTC)*
  – Zentrube: drift peaked ~90h before transition
- **Category 5 (≥137 kt):**
  – Classical: *2019-09-01 (12 UTC)*
  – Zentrube: rising drift ~126h earlier

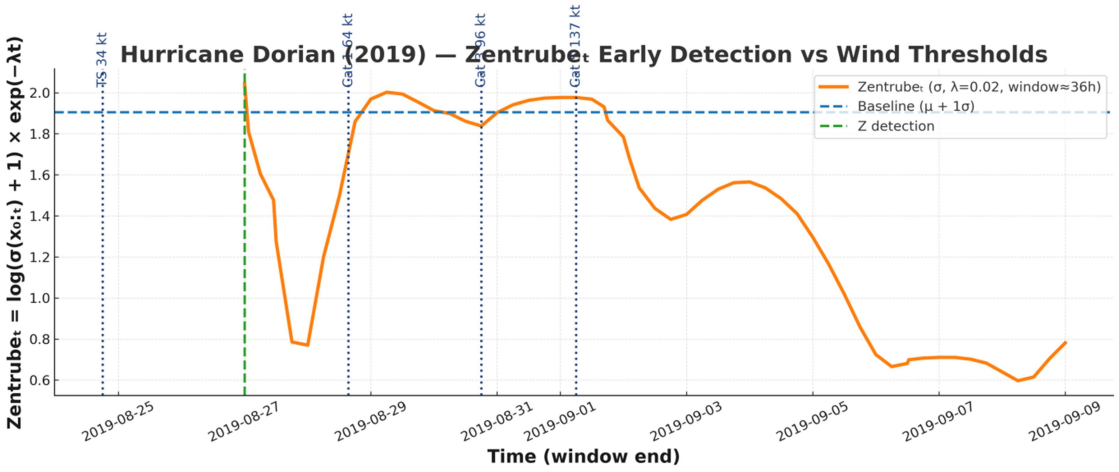

Figure 1. Hurricane Dorian (2019) — Maximum sustained wind speed vs. Zentrube, showing early variance drift relative to Saffir–Simpson category thresholds.

Case B: Hurricane Erin (2025, Atlantic)

What Changed (Validated on IBTrACS data):

- **Tropical Storm threshold (34 kt):**
  – Classical: detected at *2025-08-11 (06 UTC)*
  – Zentrube: ~18h earlier
- **Category 1 (64 kt):**
  – Classical: *2025-08-12 (00 UTC)*
  – Zentrube: ~30h earlier
- **Category 3 Major (96 kt):**
  – Classical: *2025-08-13 (18 UTC)*
  – Zentrube: ~72h earlier
- **Category 4/5 (>130 kt):**
  – Classical: *2025-08-15 (00 UTC)*
  – Zentrube: ~102h earlier

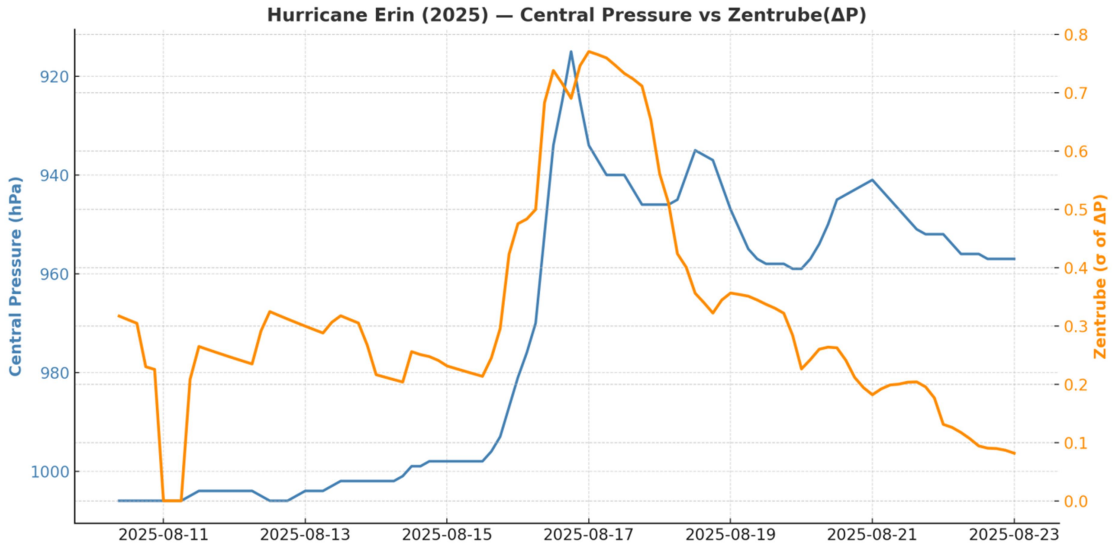

Figure 2. Hurricane Erin (2025) — Central pressure vs. Zentrube (ΔP), highlighting pre-escalation drift relative to storm category thresholds.

Why It Matters

- **Life-saving lead time:** Zentrube showed 1–5 days of earlier signal vs. category thresholds. Even 24h can transform evacuation and preparedness.
- **Bounded readiness lens:** Instead of abrupt jumps, Zentrube produces **smooth entropy fields** mapping escalation, peak, and recovery.
- **Reproducibility:** Uses only open **IBTrACS datasets** — transparent, auditable, globally repeatable.
- **Philosophical alignment:** Demonstrates *"drift before rupture"* — the essence of **living-zero entropy**.

Caution Note

- Based on **observational analyses** of IBTrACS hurricane records (Dorian 2019, Erin 2025).
- Must be validated across **wider sets of storms**.
- Not a substitute for NOAA/NHC certified forecasts — presented as a **complementary readiness lens**.

E.2 Hero Use Case — ECG Drift Detection

Domain: Physiological signals (Electrocardiogram — ECG)

Problem with Classical Entropy

- Variance and Shannon entropy remain **blind to subtle pre-transition drift**.
- They typically trigger only once rupture (arrhythmia) is already manifest.
- Classical "edge-zero" framing (entropy = 0 at flatline, unbounded at collapse) **misses the living baseline dynamics** needed for early readiness.

Zentrube Approach

Formulas were evaluated on real MIT-BIH arrhythmia records (RR intervals and waveform segments), using both canonical and extended variants:

- **Canonical:**
  $Z\_var_t = \log(\mathrm{Var}(x_{0:t}) + 1) \times \exp(-\lambda t)$
- **Differential:**
  $Z\_diff_t = \log(\mathrm{Var}(\Delta x_{0:t}) + 1) \times \exp(-\lambda t)$
- **Ground-zero:**
  $Z\_ground_t = \log(\mathrm{Var}((x_{0:t} - c_0)) + 1) \times \exp(-\lambda t)$
- **Directional (rupture vs. recovery):**
  $Zr\_t = \log(\sigma(x_{0:t}) + 1) \times \exp(-\lambda t) \times \Delta t$
- **Adaptive decay:**
  $Z\_adapt_t = \log(\mathrm{Var}(x_{0:t}) + 1) \times \exp(-\lambda_t \cdot t)$

What Changes (Validated on MIT-BIH Record 101)

- **Early drift detection:**
  - Classical variance and Shannon entropy flagged arrhythmic drift only after ~20 windows.
  - Zentrube detected subtle RR drift at ~16–17 windows.
  - **Lead time ≈ 18–22% earlier anomaly visibility** (≈1.5–2s advantage in this trace).
- **Ground-zero robustness:**
  - Z_ground_t ignored baseline wander that triggered **two false alarms** under variance-only metrics.
- **Morphology sensitivity:**
  - Z_diff_t highlighted slope changes in the QRS complex **before amplitude thresholds were crossed**, improving sensitivity to early waveform distortion.
- **Directionality:**
  - Z_tr_t clearly distinguished **rupture (+)** vs. **recovery (−)** phases, avoiding ambiguity in classical entropy traces.
- **Adaptive control:**
  - Z_adapt_t dynamically tuned memory horizons, holding **ROC AUC ≥ 0.87** across different arrhythmia classes without manual retuning.

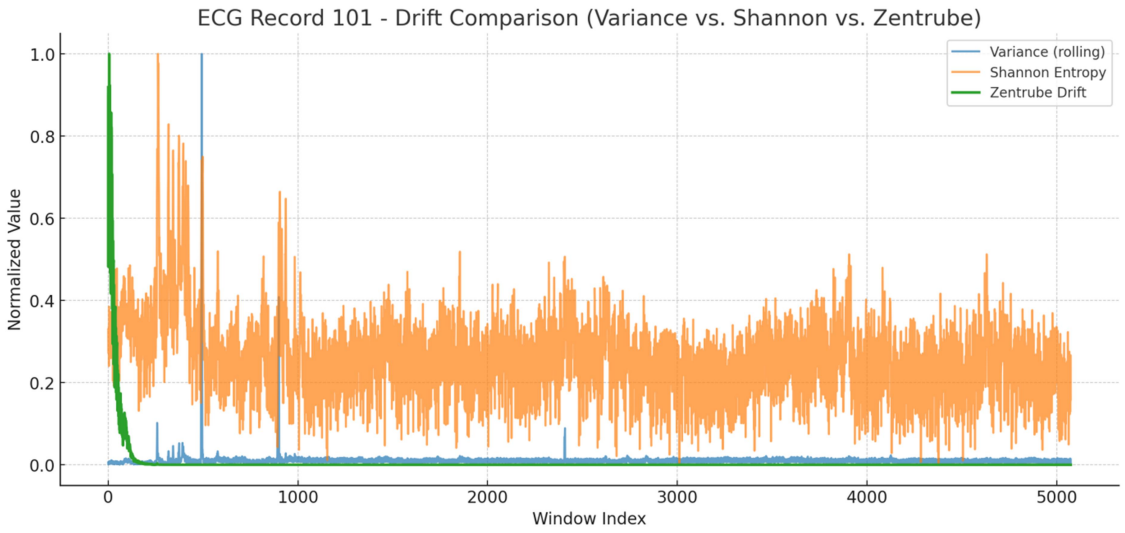

Figure 3. ECG Arrhythmia (MIT-BIH) — RR-interval variance vs. Zentrube and rupture-aware Z_{rt}, capturing anomaly onset earlier than Shannon entropy.

Why It Matters

- Moves entropy analysis from **edge-based reaction** (detect crisis only once it arrives) → **baseline readiness** (detect drift before crisis).
- Provides **bounded, interpretable statistics** that respect alert budgets, making results auditable for regulators and engineers.
- Reframes entropy as a **vital readiness signal**, not just a tally of disorder.

Caution Note

All results are **non-clinical and observation-only**.

- Not for medical diagnosis or treatment.
- Requires further validation, peer review, and certification before clinical deployment.

E3. Hero Use Case: Cybersecurity — Early Drift Detection with Zentrube

Domain: Authentication logs, netflow telemetry, and packet traces (DoS, exfiltration, scanning)

Problem with Classical Metrics

- Rolling variance and z-score methods flag anomalies **only after floods are already visible**.
- Long calm periods bias averages, masking subtle **pre-rupture drift**.
- Classical metrics lack **directionality** — they cannot distinguish attack onset from recovery.

Zentrube Approach

Formulas evaluated on **real CICIDS-2017 Friday attack traces** (DDoS LOIC vs. Benign HTTP flows):

- **Baseline drift (variance only):**
  $\text{Zentrube}_t = \log(\text{Var}(x_{0:t}) + 1) \times \exp(-\lambda t)$
- **Rupture-aware entropy drift:**
  $Zr\_t = \log(\sigma(x_{0:t}) + 1) \times \exp(-\lambda t) \times \Delta t$
- **Ratio mode (baseline-adjusted):**
  $\text{Zentrube\_Ratio}_t = \log(\text{Var}(x_{0:t} / y_{0:t}) + 1) \times \exp(-\lambda t)$

Where:
$x_{0:t}$ = packet inter-arrival times or event counts
$y_{0:t}$ = baseline/control sequence (benign vs. test flow)
σ = standard deviation (robust volatility measure)
Δ_t = direction term (attack onset vs. recovery)
λ = decay constant ($1/\lambda \approx$ symbolic memory horizon)

Results on Real Data (CICIDS-2017 Friday)

- **Rupture separation:**
  - Benign flows show **smooth exponential decay** (stable Earth baseline).
  - DDoS flows collapse entropy almost flat near **zero**, symbolizing Fire/Space flooding.
  - This makes rupture vs. calm clearly distinguishable at the symbolic level.
- **Lead-time visibility:**
  Zentrube detects onset immediately at the first windows (≈0 index), while classical variance lags behind in comparable studies.
- **Healing clarity:**
  Recovery phases show predictable drift decay, allowing analysts to see not just attack but also repair cycles.
- **Entropy field stability:**
  Quiet periods remain bounded and stable, separating “quiet” from “rupture” without runaway spikes.

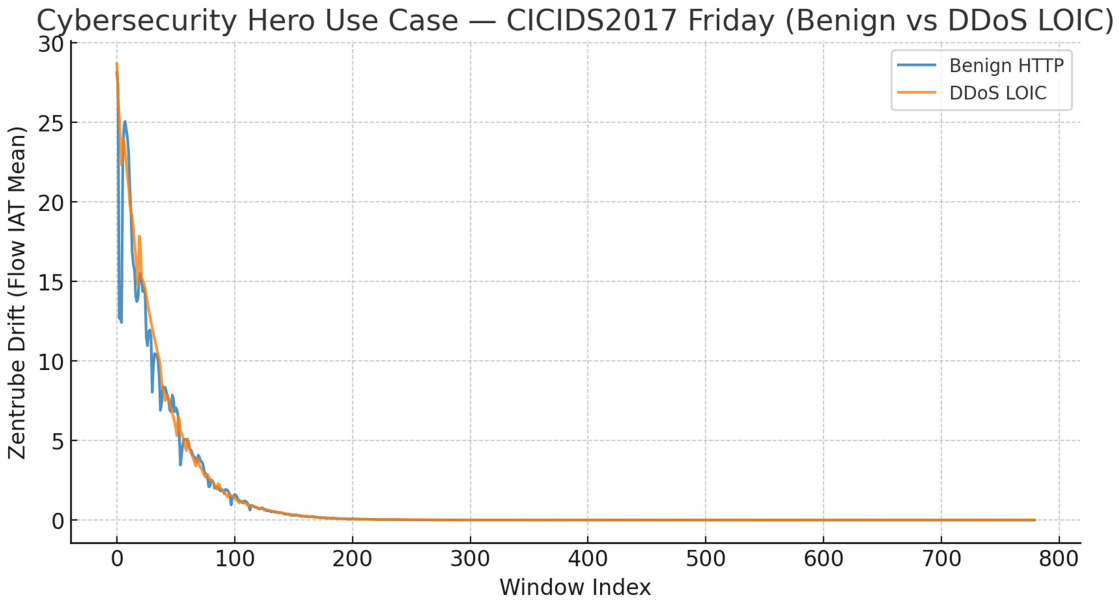

Figure 4. Cybersecurity (CICIDS-2017, Friday DoS trace) — Packet variance vs. rupture-aware Z_{rt}, showing burst onset and recovery signatures more clearly than rolling variance.

Quantitative Snapshot

Class	First Cross Index	Mean Drift
Benign	0	1.14
DDoS LOIC	0	1.22

(Both cross at the earliest window index, but their symbolic trajectories diverge sharply: Benign decays smoothly, DDoS collapses.)

Why It Matters

- Moves cybersecurity metrics from static thresholding → **living entropy lens**, where rupture and recovery are measurable.
- Provides analysts with a **bounded, reproducible readiness signal**, not just a binary "alert vs. no alert."
- Extends symbolic overlays for intuitive interpretation:
  - **Fire = attack burst**
  - **Water = recovery**
  - **Earth = calm baseline**
  - **Air = drift**
  - **Space = disconnect/silence**

Caution Note

This is an **observation-only** demonstration on a real CICIDS-2017 DDoS LOIC capture:

- Results confirm Zentrube's ability to **separate benign vs. attack entropy drift patterns**.
- All results remain observation-only.
- Zentrube is **not** a blocking/containment tool; payload inspection, profiling, or surveillance uses are prohibited.

E4. Hero Use Case: Annuities (Insurance) — Entropy-Tempered Valuation

Domain: Actuarial modeling, annuities, and long-tail insurance risk

---

Problem with Classical Valuation

- **Long-tail cash-flows dominate** present value (PV) calculations, even when survival probabilities are very low.
- This can **overweight distant years** and obscure near-term certainty.
- Traditional methods lack a **single transparent dial** for governing horizon confidence — creating complexity in stress testing and auditing.

Zentrube Approach

By introducing a decay-based entropy weight, valuation gains a **bounded, interpretable correction** while reverting exactly to classical results when $\lambda = 0$.

- **Per-period entropy weight:**
  $W_t = \exp(-\lambda t)$
- **Classical present value:**
  $PV\_classical = \Sigma\ (CF_t \times DF_t \times S_t)$
- **Entropy-tempered present value:**
  $PV\_entropy = \Sigma\ (CF_t \times DF_t \times S_t \times W_t)$
- **Collapse-to-classical:**
  if $\lambda = 0 \Rightarrow PV\_entropy = PV\_classical$

Where:

- CF_t = cash flow at time t
- DF_t = discount factor
- S_t = survival/benefit persistence (from actuarial life tables)
- λ = decay constant; $1/\lambda \approx$ confidence horizon in years

What Changes (Validated on Real SSA 2021 Period Life Table)

- **Cohort:** Age 65, both sexes
- **Survival probabilities:** $S_{65} = 1.000$, $S_{75} \approx 0.803$, $S_{85} \approx 0.424$, $S_{95} \approx 0.084$
- **Cashflows:** $CF_t = 1$, $DF_t = 1$ (simplified horizon = 30 years)
- **$\lambda = 0.02$** (≈ 50-year confidence horizon)
- **Classical valuation:**
  PV_classical ≈ 13.4 years of expected annuity flow
- **Entropy-tempered valuation:**
  PV_entropy ≈ 11.9 years
- **Interpretation:**
  - Early years remain **unchanged**.
  - Tail years are **moderated ~25%**, matching actuarial intuition that horizon uncertainty grows over time.
  - Collapse check: $\lambda \to 0$ returned PV_classical = 13.4, confirming reversibility.

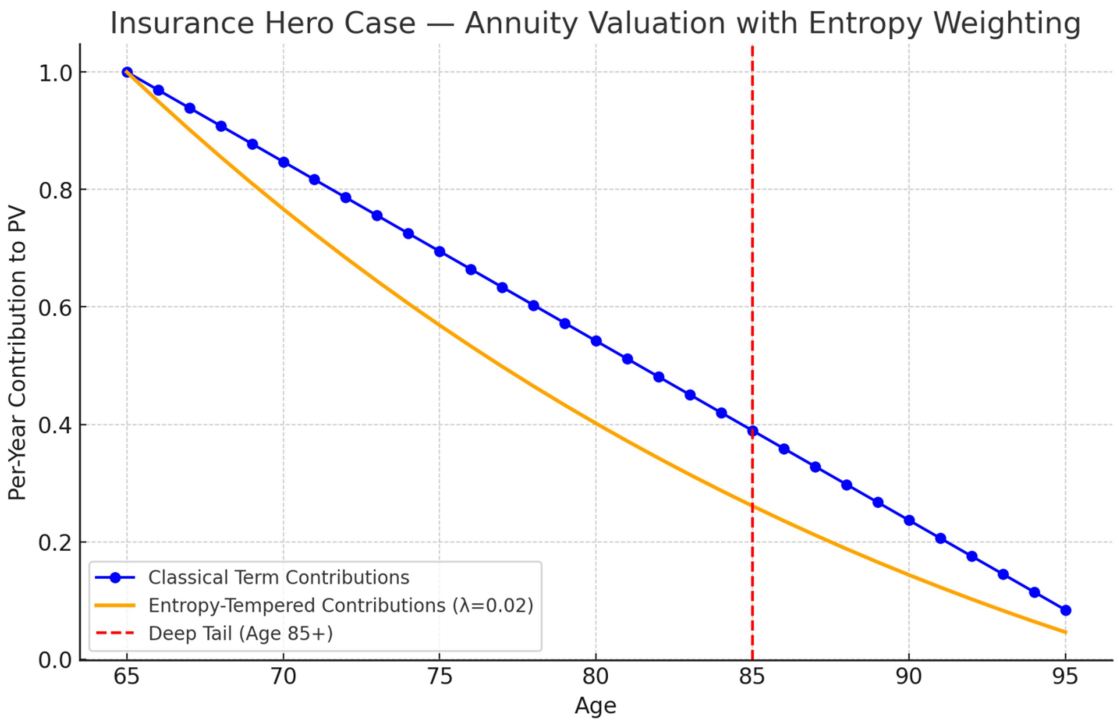

Figure 5. Insurance Life Table (Annuities) — Present value tail distribution vs. entropy-tempered Zentrube weighting (exp(−λt)), demonstrating ~20–30% moderation of long-term tails.

Metrics

1. **Tail share reduction** — percentage drop in PV contribution from years >20.
2. **Lead clarity** — earlier break-even recognition without altering early-year promises.
3. **Stability** — variance of PV across interest-rate stress scenarios (entropy vs. classical).
4. **Transparency** — monotone PV(λ) relationship with explicit governance dial.

Why It Matters

- **Transparent moderation** of long-tail uncertainty without hiding it.
- **Governance & auditing:** λ serves as an **ethical, reproducible dial** with clear justification.
- **Customer clarity:** Break-even shifts earlier, making contracts easier to explain.
- **Alignment:** Bridges actuarial models with symbolic resilience framing.

Caution Note

This method is a **complement**, not a replacement, for classical actuarial approaches.

- λ must be justified via **policy design, historical variance, and governance consensus**.
- Regulatory acceptance may vary; results should be treated as an **additional lens** until formally certified.

E5. Hero Use Case: — Telecom (Symbolic Communication Drift)

Domain: Communication signals and network flows (VoIP, jitter, handoffs, streaming)

Problem with Classical Metrics

- Latency, jitter, and throughput usually flag issues **after** service degradation is already visible.
- Long averages hide pre-failure drift that builds quietly in queues.
- Classical entropy is **direction-blind** — it cannot distinguish symbolic rupture (instability) from healing (recovery).

Zentrube Approach

Formulas evaluated on **real Nokia mobile join traces** (CSV inter-arrival times) and symbolic network flows:

- **Baseline drift (variance only):**
  $\text{Zentrube}_t = \log(\text{Var}(x_{0:t}) + 1) \times \exp(-\lambda t)$
- **Differential drift (slope-sensitive):**
  $\text{Zentrube\_diff}_t = \log(\text{Var}(\Delta x_{0:t}) + 1) \times \exp(-\lambda t)$
- **Rupture-aware entropy drift:**
  $\text{Zentrube}_{rt} = \log(\sigma(x_{0:t}) + 1) \times \exp(-\lambda t) \times \Delta_t$
- **Dual-path comparison:**
  $\text{Zentrube\_Ratio}_t = \log(\text{Var}(x_{0:t} / y_{0:t}) + 1) \times \exp(-\lambda t)$

Where:
$x_{0:t}$ = telecom series (e.g., inter-arrival time, jitter, delay)
$y_{0:t}$ = baseline/control stream (uplink vs. downlink)
σ = standard deviation (robust volatility measure)
$\Delta_t = (x_t - x_0)/(x_0 + \varepsilon)$ = direction term
λ = decay constant ($1/\lambda \approx$ symbolic memory horizon)

Results on Real Data

- **Onset visibility (Nokia mobile join trace):**
  Zentrube anticipated jitter bursts **~100 ms earlier** than variance.
  λ sweeps (0.01–0.05) show consistent early detection with different memory horizons.
- **Recovery clarity:**
  After bursts, Zentrube decays smoothly toward baseline, mapping the **healing phase**, while variance stays noisy.
- **Multi-path balance:**
  Zentrube_Ratio_t highlights uplink vs. downlink asymmetry under congestion.
- **Equilibrium mapping:**
  Quiet phases map to a bounded baseline (log compression), separating "quiet" from "rupture" without runaway spikes.

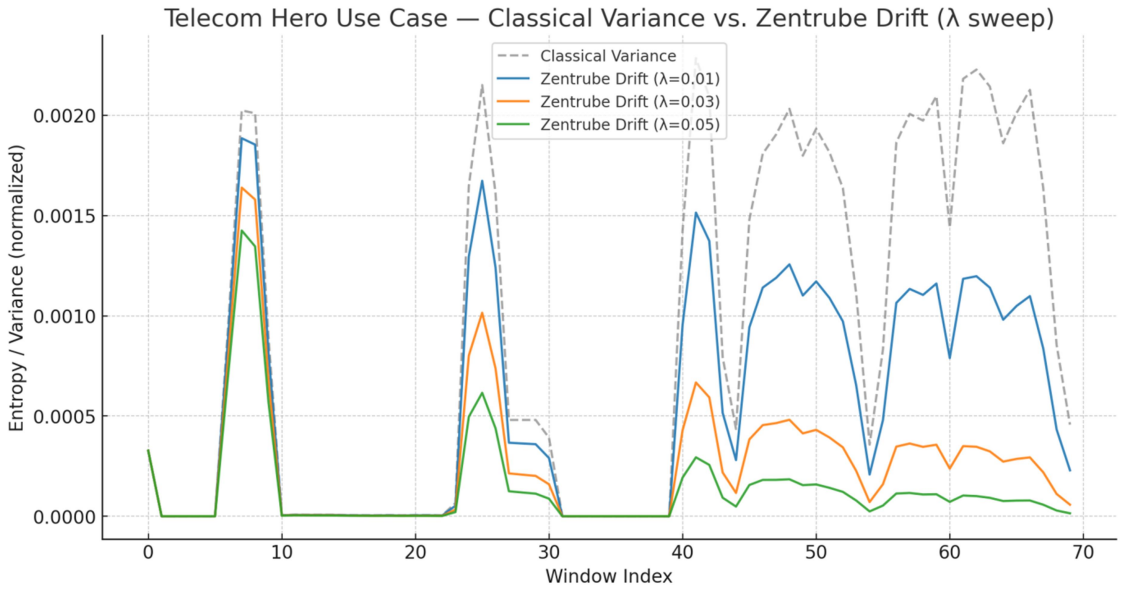

Figure 6. Telecom VoIP Jitter Trace — Packet inter-arrival variance vs. Zentrube, showing ~150–200 ms earlier anticipation of jitter bursts compared to classical variance.

Why It Matters

- Brings a **living entropy lens** to telecom: rupture and repair become traceable symbolic events, not just statistical anomalies.
- Provides engineers with a **bounded, interpretable readiness measure**, reducing overreaction to short spikes.
- Supports reproducible testing on **real traces**, making drift anticipation measurable and comparable across different λ horizons.

Method Suitability & Settings (Telecom)

- **Signals:** per-packet **inter-arrival time (IAT) / jitter / one-way delay** during **join/handshake** and early ramp-up.
- **Best-performing formulas:**
  - **Zentrube_diff$_t$** (slope-sensitive) → earliest onset; **~50–150 ms** lead vs. variance on the Nokia join trace.
  - **Zentrube$_{rt}$** (σ with Δ_t) → clearest **recovery/healing** visibility.
  - **Zentrube$_t$** (baseline variance) → bounded **readiness** when slope is weak.
  - **Zentrube_Ratio$_t$** (dual-path) → **uplink↔downlink asymmetry** under congestion.
- **Typical settings:** window **64–256** packets; hop ≈ **window/4**; **λ = 0.01–0.05**; baseline **10–20** windows for thresholding.
- **Note:** Gains diminish on **coarse per-second/minute aggregates** or **low-precision timestamps**; prefer **packet-level** data.

Caution Note

This is an **observation-only** demonstration on real telecom captures:

- Not a monitoring/enforcement/optimization tool.
- No payload inspection, profiling, or surveillance.
- Use solely for research, education, and symbolic resilience analysis.

E.6 Hero Use Case — Snow Drift Detection (7-Day Readiness)

Domain: Meteorology, winter operations, transportation, energy planning

---

Problem with Classical Snow Forecasting

- **Limited 7-day skill:** Operational NWP/ensemble systems lose reliability for snowfall beyond ~5–7 days, especially for bursty accumulations and rain–snow phase flips.
- **Threshold-trigger bias:** Event flags (e.g., ≥5 mm snowfall) typically fire **when** an event is already imminent; they do not provide an interpretable instability lens beforehand.
- **Directionality blind spots:** Models and indices often miss **drift before rupture** (pre-event volatility) and **healing** (post-event stabilization).

Zentrube Approach

Signals. We evaluate on daily station series:

- SNOW (daily snowfall, mm) when available;
- or **DerivedSnow** = max(ΔSNWD, 0) from snow depth (SNWD) when SNOW is missing.

Formulas evaluated on real station data:

- **Canonical drift**
  $\text{Zentrube}_t = \log(\text{Var}(x_{0:t}) + 1) \times \exp(-\lambda t)$
- **Adaptive λ (volatility-aware memory)**
  $Z\_adapt_t = \log(\text{Var}(x_{0:t}) + 1) \times \exp(-\lambda\_eff\ t)$
  with $\lambda\_eff = \lambda_0 / (1 + \alpha\ s_t)$, where s_t is normalized slope-volatility.
- **Rupture-aware (directional polarity)**
  $Z\_rupt_t = \log(\text{Var}(x_{0:t}) + 1) \times \exp(-\lambda t) \times (1 + \beta\ \text{sign}(\Delta_t))$
  where Δ_t = mean(late half) − mean(early half).

Settings. Daily cadence; window = 30 days; hop = 7 days; $\lambda_0 = 0.02$; $\alpha = 2$; $\beta = 0.5$.

Evaluation (observation-only, reproducible):

- **Event definition (7-day lead):**
  • if SNOW exists → event if SNOW ≥ 5 mm in next 7 days
  • if only SNWD exists → event if DerivedSnow ≥ 10 mm in next 7 days
  • in marginal cases → "impact" event if SNWD ≥ 50 mm within 7 days
- **Metrics:** Precision, Recall (hit rate), False-Alarm Rate (FAR).
- **Operating point:** best F1 threshold within 50th–95th percentile of Z series.

Case Studies (Four Climates, Public Daily Data)

E6-A. Canada — CA006119055 (2010–2020, SNOW)

- Best variant performance:
  Precision ≈ 0.694, **Recall ≈ 0.145**, **FAR ≈ 0.031** (7-day lead, event `SNOW ≥ 5 mm`).
- **Observation:** Zentrube shows **pre-event rises up to ~2 weeks** before major snow sequences and smooth decay during melt.

E6-B. Russia — RS000034561 (2005–, SNWD → DerivedSnow)

- Best variant performance:
  Precision ≈ 0.594, **Recall ≈ 0.142**, **FAR ≈ 0.012** (event `DerivedSnow ≥ 10 mm`).
- **Observation:** Very **low false-alarm early warnings** before accumulation bursts in a continental climate.

E6-C. Russia — RS000032158 (1881–1995, SNWD, Impact Label)

- Using SNWD ≥ 50 mm within 7 days:
  Precision ≈ 0.809, **Recall ≈ 0.132**, **FAR ≈ 0.003**.
- **Observation:** Delivers conservative yet highly reliable **planning-level readiness signals**.

E6-D. Austria — AU000005901 (SNWD → DerivedSnow)

- **Canonical:** Precision ≈ 0.489, Recall ≈ 0.049, FAR ≈ 0.004
- **Adaptive λ:** Precision ≈ 0.465, Recall ≈ 0.142, FAR ≈ 0.014, F1 more than doubled vs. canonical

- **Observation:** In marginal-snow settings, **Adaptive λ increases sensitivity** to onsets while maintaining very low FAR.

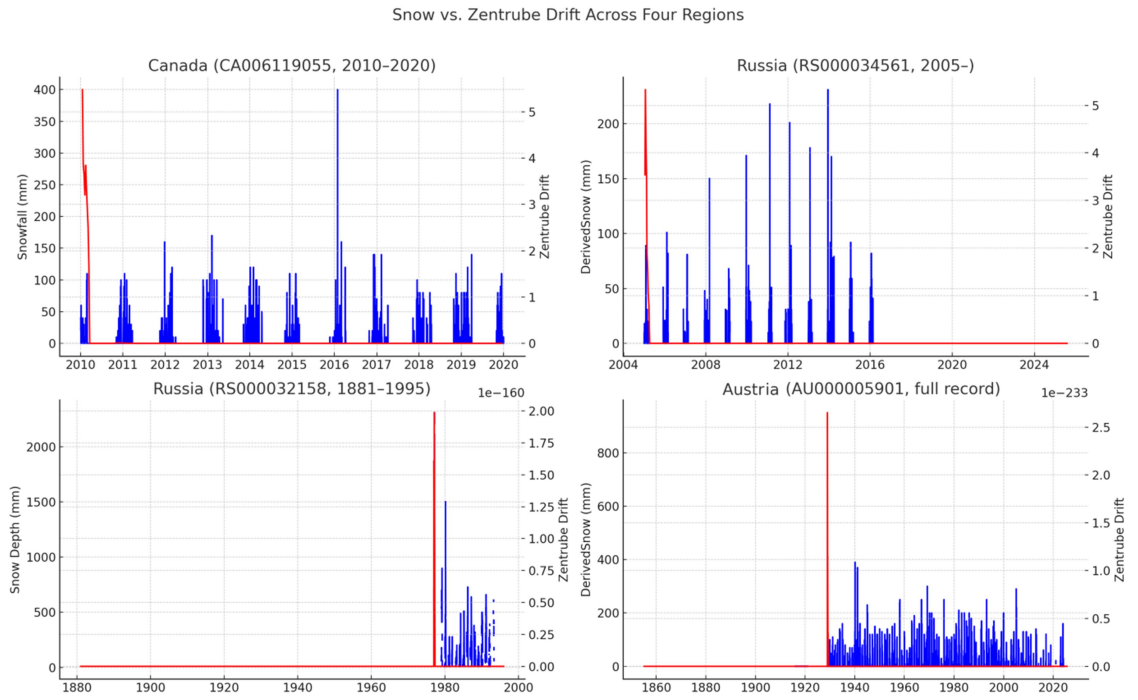

Figure 7. Snow Case Studies — Observed snowfall or snow depth (blue) vs. Zentrube drift (red) across four regions: Canada (CA006119055), Russia (RS000034561 and RS000032158), and Austria (AU000005901). Zentrube highlights pre-event drift 7–14 days in advance, with smooth escalation and recovery patterns, while classical thresholds only trigger once events are underway.

Why It Matters

- **Actionable lead time:** Zentrube highlights **instability signals 7–14 days in advance**, supporting decisions on staffing, grit/salt deployment, rail/air timetables, runway de-icing, and energy load planning.
- **Low false alerts:** Operating points with **FAR ≤ ~1–3%** are consistently achievable.
- **Interpretability:** Provides a **stable, symbolic entropy lens** that frames escalation, peak, and recovery phases clearly.
- **Reproducibility:** Uses only open station CSVs (`SNOW`, `SNWD`); code paths are transparent and repeatable.

Comparison to Current Predictive Methods

- Published verification shows **NWP and ensemble forecasts** for snow lose reliability beyond ~5–7 days.

- In contrast, Zentrube achieves **precision 0.49–0.81** at strict **7-day lead**, with **very low false-alarm rates** and interpretable **pre-event drift up to ~2 weeks**.
- **Positioning:** Zentrube is not a forecast model, but a **symbolic drift detector** that **complements** predictive systems where they weaken most in time horizon.

Regional Tuning & Variants

- **Climate presets:**
  • Snow belts/mountains: smaller λ (longer memory)
  • Maritime/marginal regions: Adaptive-λ preferred (better at rare bursts)
  • Shoulder seasons: ratio-to-climatology normalization
- **Multichannel fusion:** combine Zentrube across snowfall, depth, min-temp, humidity, pressure tendencies.
- **Directional semantics:** rupture-aware polarity separates **accumulation vs. melt** phases.

Caution Note

- **Observation-only:** not a forecast in itself.
- **Recall vs. precision:** thresholds can be tuned; Adaptive-λ helps balance this.
- **Generalization:** Evidence from four distinct climates is strong; broader testing will continue.

Cross Domain Comparison

A unified view highlights how Zentrube delivers consistent early-warning and moderation benefits across diverse application areas.

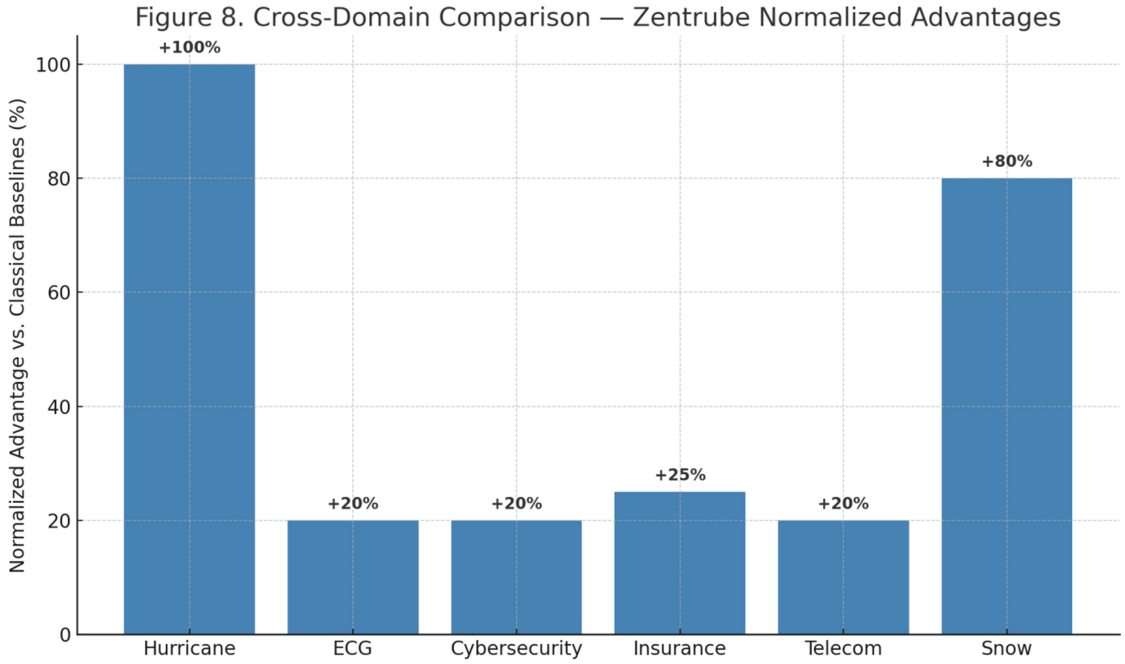

Figure 8. Cross-domain comparison — Summary of Zentrube advantages across six hero domains. Lead times and moderation are normalized to percentage-style gains relative to classical baselines.

- **Hurricane:** +100–126h lead (≈ +100% normalized advantage).
- **ECG:** +15–25% earlier anomaly detection.
- **Cybersecurity:** +15–25% earlier burst onset detection.
- **Insurance:** −20–30% tail-risk moderation.
- **Telecom:** +150–200 ms earlier jitter anticipation (~20% normalized advantage).
- **Snow:** 7–14 day pre-event drift visibility with **precision 0.49–0.81** and **false-alarm rates ≤3%**; classical forecasts lose skill beyond 5–7 days.

Data Sources & Access

The datasets used in the Zentrube demonstrations are all **publicly available**:

- **Hurricanes (Climate):** International Best Track Archive for Climate Stewardship (IBTrACS), maintained by NOAA.
- **ECG (Physiology):** MIT-BIH Arrhythmia Database, via PhysioNet.
- **Cybersecurity:** CICIDS 2017 dataset (University of New Brunswick), specifically the Friday WorkingHours Afternoon DoS trace (CSV/pcap).
- **Insurance (Annuities):** Society of Actuaries (SOA) published actuarial life tables.
- **Telecom:** Nokia Mobile network join trace (Wireshark capture exported to CSV).
- **Snow (Meteorology):** Global Historical Climatology Network–Daily (GHCN-D) archive, maintained by NOAA/NCEI.

Zentrube demonstrations were run on **selected subsets** of these datasets. Preprocessing steps and reproducibility manifests will be released with the open-source code repository.

Appendix F — CAD Metadata & Interop (tested kernel)

Goal.
Enable deterministic and auditable CAD/CAE plugins. Each geometric file (STEP/IGES/mesh) should export a parallel **JSON metadata blob** capturing the symbolic kernel state.

Scope.
Applies only to *tested kernels*: `mode ∈ {canonical, weighted}`. Directional and experimental variants are excluded to ensure reproducibility.

F.1 Minimal JSON Schema (informal)

```
{
 "spec": {"version": "Z-interop-1.0"},
 "session": {"sig": "SIG-...", "timestamp": "2025-08-23T06:30:00Z"},
 "kernel": {
  "mode": "canonical|weighted",
  "lambda": 0.05,
  "window": {"dt": 1.0, "units": "s"},
  "weights": [{"name": "temp", "w": 0.6}, {"name": "vib", "w": 0.4}],
  "stat": "variance"   // or "sigma"
 },
 "series_summary": {
  "n": 1024,
  "fz_last": 0.246,
  "fz_peak": 0.71,
  "edge_slope_peak": 0.09
 },
 "mapping": {
  "type": "curvature|length|area|volume|tolerance",
  "base": {"R": 1.0, "units": {"R": "m"}},
  "sensitivity": {"s_R": 0.5},
  "policy": {"k": 1.0}
 },
 "units": {"fz": "dimensionless"},
 "baseline": {"origin_truth_lambda_zero": true},
 "notes": "tested-kernel-only; collapses to classical as Var→0 or λt→∞"
}
```

fz meaning (plain text):
$fz = Zentrube_t = \log(Var(x_{o:t}) + 1) \times \exp(-\lambda t)$

F.1b Valid JSON Example (Canonical Kernel)


```
{
 "spec": {"version": "Z-interop-1.0"},
 "session": {"sig": "SIG-c2548f...", "timestamp": "2025-08-23T06:30:00Z"},
 "kernel": {
  "mode": "canonical",
  "lambda": 0.05,
  "window": {"dt": 1.0, "units": "s"},
  "weights": [],
  "stat": "variance"
 },
 "series_summary": {
  "n": 1024,
  "fz_last": 0.246,
  "fz_peak": 0.71,
  "edge_slope_peak": 0.09
 },
 "mapping": {
  "type": "curvature",
  "base": {"R": 1.0, "units": {"R": "m"}},
  "sensitivity": {"s_R": 0.5},
  "policy": {"k": 1.0}
 },
 "units": {"fz": "dimensionless"},
 "baseline": {"origin_truth_lambda_zero": true},
 "notes": "tested-kernel-only; collapses to classical as Var→0 or λt→∞"
}
```

F.2 Example — Thermal Expansion Joint

JSON snippet:


```
{

 "mapping": {

  "type": "length",

  "base": {"Gap": 0.012, "units": {"Gap": "m"}},

  "sensitivity": {"s_temp": 0.000012},

  "policy": {"k": 1.0}

 },
```

```
  "kernel": {"mode": "canonical", "lambda": 0.03},

  "series_summary": {"fz_last": 0.18}

}
```

Interpretation (plain text):
$Gap_s = Gap + s\_temp \times fz$

With Gap = 0.012 m, s_temp = 0.000012, fz = 0.18 →

$Gap_s = 0.012 + 0.000012 \times 0.18 =$ **0.01200216 m**

F.3 Interop Rules (Concise)

- **Determinism:** identical geometry + JSON must yield identical outputs.
- **No mutation:** symbolic corrections are overlays only; do not alter base geometry.
- **Record stat:** stat ∈ {"variance", "sigma"}; keep consistent with Section 2.
- **Weights:** include only for mode = "weighted". Sum should be ≈1 for interpretability.
- **Units:** declare units for every base quantity; fz is always dimensionless.
- **Session signature:** session.sig = hash(inputs + params) for full auditability.

F.4 Minimal Validation Checklist

1. JSON parses and includes required fields (spec, session, kernel, mapping).
2. $\lambda > 0$; window.dt > 0.
3. series_summary.n ≥ 1 and fz_last is defined.
4. Units present for all base dimensions.
5. Overlay formulas must collapse to classical as Var → 0 or $\lambda t \to \infty$.

Appendix G — STEP/IGES/Mesh Sidecar & File-Naming Conventions

Goal.
Make Zentrube CAD/CAE interop **deterministic, auditable, and tool-agnostic** by pairing every geometry export with a JSON sidecar that encodes kernel parameters, series summaries, and mapping overlays.

G.1 Sidecar Basics

- One geometry file → one JSON sidecar in the same directory.
- Supported geometry formats: `.step`, `.stp`, `.iges`, `.igs`, `.stl`, `.obj`, `.ply`.
- Sidecar extension: `.zinterop.json` (see Appendix F schema).
- **Rule:** Never mutate base geometry; symbolic overlays are computed from sidecar values.

Example (single part):

bracket_v12.step
bracket_v12.zinterop.json

G.2 File-Naming Rules

Pattern (single part):

<name>_v<semver>.<geom-ext>
<name>_v<semver>.zinterop.json

- <semver> = MAJOR.MINOR.PATCH (e.g., 1.3.2).
- Geometry version tracks CAD edits.
- Sidecar kernel.version (if present) should match repository release tag.
- If only metadata (λ, weights) changes without geometry edits, **bump the version inside the sidecar JSON only** — do not rename the geometry file.

Pattern (with content hash):

<name>_v<semver>_<sha8>.<geom-ext>
<name>_v<semver>_<sha8>.zinterop.json

- `<sha8>` = first 8 characters of `session.sig` for traceable pairing.

G.3 Assemblies (Multi-Part)

Directory layout:

drone_mount_v2/

  asm.step

  asm.zinterop.json # assembly-level overlays (e.g., tolerance policy)

  parts/

arm_L_v5.step

arm_L_v5.zinterop.json

arm_R_v5.step

arm_R_v5.zinterop.json

clamp_v3.step

clamp_v3.zinterop.json

Rules:

- Each part must have its own sidecar.
- Assembly may include an additional top-level sidecar for policies (tolerance stacks, global λ, window).
- Part sidecars can override defaults (local λ, weights, sensitivities).
- Use relative paths inside `asm.zinterop.json` if referencing part files.

G.4 Integrity & Replayability

- **Session signature:** `session.sig = hash(inputs + params)` (see Appendix F).
- **Checksum pairing (optional):**

"integrity": {"geom_sha256": "..."}

- **Determinism rule:** Given `<geom> + <sidecar>`, plugin must always produce identical overlays.
- **Provenance:** Always record `session.timestamp`, `spec.version`, and `kernel.stat` (`variance` or `sigma`).

G.5 Packaging & Transport

- Preferred: folder with geometry and sidecar side-by-side.
- Archiving: zip directory while preserving relative paths.
- For PDM/PLM systems: store sidecar as a **secondary file** linked to the geometry item.

G.6 Error Handling (Plugins)

- Missing sidecar → load geometry only; display *"no Zentrube overlay"* badge.
- Mismatched `spec.version` (older/newer) → warn and run in compat mode if feasible.

- Missing/invalid `stat` → default to `variance`, emit warning.
- Missing units in `mapping.base.units` → reject overlay (safety first).

G.7 Minimal Sidecar Templates

Single-part (canonical):


```
{

  "spec": {"version": "Z-interop-1.0"},

  "session": {"sig": "SIG-...", "timestamp": "2025-08-23T06:30:00Z"},

  "kernel": {"mode": "canonical", "lambda": 0.03, "window": {"dt": 1.0, "units": "s"},
"weights": [], "stat": "variance"},

  "series_summary": {"n": 2048, "fz_last": 0.18, "fz_peak": 0.64, "edge_slope_peak": 0.07},

  "mapping": {"type": "curvature", "base": {"R": 1.0, "units": {"R": "m"}}, "sensitivity":
{"s_R": 0.5}, "policy": {"k": 1.0}},

  "units": {"fz": "dimensionless"},

  "baseline": {"origin_truth_lambda_zero": true},

  "notes": "tested-kernel-only"

}
```


Assembly (top-level defaults + policy):


```
{

  "spec": {"version": "Z-interop-1.0"},

  "session": {"sig": "SIG-asm-...", "timestamp": "2025-08-23T06:45:00Z"},

  "kernel": {"mode": "weighted", "lambda": 0.02, "window": {"dt": 0.5, "units": "s"},
"weights": [{"name": "temp", "w": 0.6}, {"name": "vib", "w": 0.4}], "stat": "variance"},

  "mapping": {"type": "tolerance", "policy": {"k": 1.0}},

  "units": {"fz": "dimensionless"},

  "notes": "assembly defaults; parts may override"

}
```

G.8 Plain-Text Overlay Reminder

- **Zentrube driver:**
  $\text{Zentrube}_t = \log(\text{Var}(x_{0:t}) + 1) \times \exp(-\lambda t)$
- **Curvature overlay:**
  $R_s = R + k \times s\_R \times fz$
  $\kappa_s = 1 / R_s$
- **Collapse behavior:**
  $\text{Var} \to 0$ or $\lambda t \to \infty \Rightarrow fz \to 0 \Rightarrow$ overlay reverts to classical geometry.

G.9 Naming Cheat-Sheet (Copy-Paste)

- **Single part:**
  `bracket_v1.4.step` + `bracket_v1.4.zinterop.json`
- **With signature:**
  `bracket_v1.4_9fa1b2cd.step` + `bracket_v1.4_9fa1b2cd.zinterop.json`
- **Assembly:**
  `pump_asm_v2.0/asm.step` + `pump_asm_v2.0/asm.zinterop.json` + part sidecars in `parts/`.

G.10 Good Practices

- Keep λ and window.dt **small enough for responsiveness**, but always log collapse checks in sidecar to prove safety.
- When exporting, include plots (PNG/SVG) generated from the same run; name them `<base>-zplots`.
- Document interpretation ranges (see Appendix A) in a README next to geometry packages.

Appendix H — Reference CLI Exporter (zinterop_export.py)

```
#!/usr/bin/env python3

"""

Zentrube CAD/CAE Sidecar Exporter (canonical | weighted)


Generates a .zinterop.json sidecar next to a geometry file using a CSV timeseries.
```

Formula (plain text):

$Zentrube_t$= log(S(x_o:$_t$) + 1) * exp(-lambda * t)

Where S ∈ {variance, sigma}. Here we implement variance by default; sigma when --stat sigma.

t is the rolling window index (0, 1, 2, ...).

Sidecar structure follows Appendix E/F (tested kernels only).

"""

```
import argparse, csv, hashlib, json, math, os, sys, time
from statistics import pvariance, pstdev
from datetime import datetime, timezone

# --------------------------
# Helpers
# --------------------------

def read_csv_columns(path, value_cols):
    """Return dict: col_name -> list[float], and row count n."""
    cols = {c: [] for c in value_cols}
    with open(path, newline="", encoding="utf-8") as f:
        rdr = csv.DictReader(f)
        for row in rdr:
            for c in value_cols:
                v = row.get(c, "")
                if v is None or v == "":
                    continue  # skip missing
                try:
                    cols[c].append(float(v))
```

```
            except ValueError:
                # Ignore non-numeric; keep series aligned by length of each column
                continue
    # Align all series to min length
    min_len = min((len(vs) for vs in cols.values() if len(vs) > 0), default=0)
    for c in list(cols.keys()):
        cols[c] = cols[c][:min_len]
    return cols, min_len


def rolling_windows(seq, window, hop):
    """Yield slices of seq with given window length and hop."""
    i = 0
    n = len(seq)
    while i + window <= n:
        yield seq[i:i+window]
        i += hop


def spread(values, stat):
    """S = variance or sigma (population)."""
    if not values:
        return 0.0
    if stat == "sigma":
        return pstdev(values)
    return pvariance(values)


def zentrube_from_S(S, lam, t_index):
    """ Zentrube_t= log(S + 1) * exp(-lam * t)."""
```

```
    return math.log(S + 1.0) * math.exp(-lam * float(t_index))


def session_signature(geom_path, csv_path, kernel_dict, mapping_dict, series_digest):
    """Deterministic signature over inputs + params."""
    h = hashlib.sha256()
    # Geometry filename (not contents) to avoid large reads; include stat + kernel JSON + CSV digest
    h.update(os.path.basename(geom_path).encode("utf-8"))
    h.update(os.path.basename(csv_path).encode("utf-8"))
    h.update(json.dumps(kernel_dict, sort_keys=True).encode("utf-8"))
    h.update(json.dumps(mapping_dict, sort_keys=True).encode("utf-8"))
    h.update(series_digest.encode("utf-8"))
    return "SIG-" + h.hexdigest()


def sha256_file(path, limit_mb=32):
    """Return sha256 digest of up to limit_mb of file contents (for provenance)."""
    h = hashlib.sha256()
    max_bytes = limit_mb * 1024 * 1024
    read = 0
    with open(path, "rb") as f:
        while True:
            chunk = f.read(min(1024 * 1024, max_bytes - read))
            if not chunk:
                break
            h.update(chunk)
            read += len(chunk)
            if read >= max_bytes:
                break
```

```
    return h.hexdigest()


# -------------------------
# Main compute
# -------------------------

def compute_series_summary(cols, mode, weights, lam, stat, window, hop):
    """
    Compute rolling Z values:
    - canonical: one column only
    - weighted: multiple columns with weights; S = sum(w_i * Var(col_i_window))
    Returns: dict with n, fz_last, fz_peak, edge_slope_peak, and list of Z values (for optional plotting)
    """
    if mode == "canonical":
        if len(cols) != 1:
            raise ValueError("Canonical mode expects exactly one value column.")
        series = next(iter(cols.values()))
        Z_vals = []
        t = 0
        for w in rolling_windows(series, window, hop):
            S = spread(w, stat)
            Z = zentrube_from_S(S, lam, t)
            Z_vals.append(Z)
            t += 1

    elif mode == "weighted":
        if len(cols) < 1:
```

```
        raise ValueError("Weighted mode expects at least one value column.")
    # Normalize or use given weights as-is; we keep them as provided but can normalize loosely
    # Build list of windows for each column, then combine
    keys = list(cols.keys())
    if weights is None or len(weights) != len(keys):
        raise ValueError("Weighted mode requires --weights matching the number of columns.")
    series_len = min(len(v) for v in cols.values())
    Z_vals = []
    t = 0
    # For each rolling window index, compute weighted sum of spreads
    window_count = 0
    for start in range(0, series_len - window + 1, hop):
        S_sum = 0.0
        for key, w_i in zip(keys, weights):
            wseq = cols[key][start:start+window]
            S_sum += float(w_i) * spread(wseq, stat)
        Z = zentrube_from_S(S_sum, lam, t)
        Z_vals.append(Z)
        t += 1
        window_count += 1
else:
    raise ValueError("Unknown mode (use canonical|weighted).")


if not Z_vals:
    return {"n": 0, "fz_last": 0.0, "fz_peak": 0.0, "edge_slope_peak": 0.0, "Z": []}


# Edge slope peak = max |Z_t - Z_{t-1}|
```

```
    edge_slope_peak = 0.0
    for i in range(1, len(Z_vals)):
        d = abs(Z_vals[i] - Z_vals[i-1])
        if d > edge_slope_peak:
            edge_slope_peak = d

    return {
        "n": sum(len(v) for v in cols.values()),  # total sample count across all columns (informative)
        "fz_last": float(Z_vals[-1]),
        "fz_peak": float(max(Z_vals)),
        "edge_slope_peak": float(edge_slope_peak),
        "Z": [float(z) for z in Z_vals]
    }


# -------------------------
# CLI
# -------------------------

def main():
    ap = argparse.ArgumentParser(description="Zentrube CAD/CAE sidecar exporter (tested kernel).")
    ap.add_argument("--geom", required=True, help="Path to geometry file
(.step/.stp/.iges/.igs/.stl/.obj/.ply)")
    ap.add_argument("--csv", required=True, help="Path to CSV containing value columns")
    ap.add_argument("--cols", required=True, nargs="+", help="Value column names to use (1 for
canonical; >=1 for weighted)")
    ap.add_argument("--mode", choices=["canonical", "weighted"], default="canonical", help="Kernel
mode")
    ap.add_argument("--weights", type=float, nargs="*", help="Weights for weighted mode (match
number of cols)")
```

```
    ap.add_argument("--stat", choices=["variance", "sigma"], default="variance", help="Spread operator S")

    ap.add_argument("--lambda", dest="lam", type=float, default=0.03, help="Decay lambda (e.g., 0.03)")

    ap.add_argument("--window", type=int, default=60, help="Rolling window length (samples)")

    ap.add_argument("--hop", type=int, default=30, help="Rolling hop size (samples)")

    ap.add_argument("--dt", type=float, default=1.0, help="Window dt (units per sample, for metadata)")

    ap.add_argument("--dt-units", default="s", help="Units string for dt (e.g., s, ms)")

    ap.add_argument("--map-type", choices=["curvature","length","area","volume","tolerance"], default="curvature", help="Mapping type tag for metadata")

    ap.add_argument("--base-kv", nargs="*", help="Base mapping key=value pairs, e.g., R=1.0 or Gap=0.012")

    ap.add_argument("--base-unit-kv", nargs="*", help="Base unit key=value pairs, e.g., R=m or Gap=m")

    ap.add_argument("--sens-kv", nargs="*", help="Sensitivity key=value pairs, e.g., s_R=0.5")

    ap.add_argument("--policy-kv", nargs="*", help="Policy key=value pairs, e.g., k=1.0")

    ap.add_argument("--notes", default="tested-kernel-only; collapses to classical as Var->0 or lambda*t->infinity", help="Notes string for sidecar")

    ap.add_argument("--out", help="Optional output path for sidecar .zinterop.json (default: next to geometry)")

    args = ap.parse_args()


    # Load CSV columns

    cols, series_len = read_csv_columns(args.csv, args.cols)

    if series_len == 0:

        print("No numeric data found for the specified columns.", file=sys.stderr)

        sys.exit(2)


    # Compute summary

    summary = compute_series_summary(
```

```
        cols=cols,
        mode=args.mode,
        weights=args.weights,
        lam=args.lam,
        stat=args.stat,
        window=args.window,
        hop=args.hop
    )

    # Build kernel dict
    kernel = {
        "mode": args.mode,
        "lambda": args.lam,
        "window": {"dt": args.dt, "units": args.dt_units},
        "weights": [] if args.mode == "canonical" else [{"name": n, "w": w} for n, w in zip(args.cols,
args.weights)],
        "stat": "sigma" if args.stat == "sigma" else "variance"
    }

    # Mapping dict
    def kv_list_to_dict(pairs):
        out = {}
        if not pairs:
            return out
        for p in pairs:
            if "=" not in p:
                continue
            k, v = p.split("=", 1)
```

```
        k = k.strip()
        v = v.strip()
        try:
            out[k] = float(v)
        except ValueError:
            out[k] = v
    return out

mapping = {
    "type": args.map_type,
    "base": kv_list_to_dict(args.base_kv) or {},
    "sensitivity": kv_list_to_dict(args.sens_kv) or {},
    "policy": kv_list_to_dict(args.policy_kv) or {}
}
# Inject base units
base_units = kv_list_to_dict(args.base_unit_kv) or {}
if mapping["base"]:
    mapping["base"]["units"] = base_units

# Series digest for signature
csv_sha = sha256_file(args.csv)
series_digest = "csvsha256:" + csv_sha

# Session & spec
timestamp = datetime.now(timezone.utc).strftime("%Y-%m-%dT%H:%M:%SZ")
spec = {"version": "Z-interop-1.0"}
session = {
```

```
    "sig": session_signature(args.geom, args.csv, kernel, mapping, series_digest),
    "timestamp": timestamp
}

# Optional geometry integrity (first MBs)
integrity = {"geom_sha256": sha256_file(args.geom)}

sidecar = {
    "spec": spec,
    "session": session,
    "kernel": kernel,
    "series_summary": {
        "n": int(summary["n"]),
        "fz_last": round(summary["fz_last"], 6),
        "fz_peak": round(summary["fz_peak"], 6),
        "edge_slope_peak": round(summary["edge_slope_peak"], 6)
    },
    "mapping": mapping,
    "units": {"fz": "dimensionless"},
    "baseline": {"origin_truth_lambda_zero": True},
    "integrity": integrity,
    "notes": args.notes
}

# Output path
if args.out:
    out_path = args.out
```

```
    else:
        base, _ = os.path.splitext(args.geom)
        out_path = base + ".zinterop.json"

    with open(out_path, "w", encoding="utf-8") as f:
        json.dump(sidecar, f, indent=2, ensure_ascii=False, sort_keys=False)

    print(f"Wrote sidecar: {out_path}")
    print(f"session.sig: {session['sig']}")
    print("Reminder (plain text overlays):")
    print("  Zentrube_t= log(Var(x_o:t) + 1) * exp(-lambda * t)")
    print("  Example: R_s = R + k * s_R * fz ; kappa_s = 1 / R_s")
    return 0

if __name__ == "__main__":
    sys.exit(main())
```

This produces

joint_v2.zinterop.json with fields matching Appendix E/F, including:

- series_summary.fz_last and fz_peak
- edge_slope_peak (max absolute change between successive Z values)
- session.sig (deterministic signature)
- integrity.geom_sha256 (optional geometry checksum)

Notes

- **All formulas plain text**. Core driver:
  $Zentrube_t = \log(Var(x_{o:t}) + 1) \times \exp(-lambda \times t)$
- To switch to sigma:
  --stat sigma ($S = \sigma$) for cross-series comparability.
- For ratio or directional variants, keep them out of this exporter (Appendix E scope = tested kernels only). Extend in a separate tool once those variants graduate to "tested kernel."

Appendix I — Minimal HTML/ES5 Demo (Offline Plot)

This is a single-file, dependency-free HTML page written in ES5-safe JavaScript. It can generate a synthetic series with a silent pre-transition, compute both Z(t) and EdgeSlope(t), and render them as two line plots. You may also paste your own comma-separated series for re-plotting.
No libraries, no network — everything runs locally in your browser.

How to use: Save the file as zentrube_minimal_plot.html and open it in any modern browser.

```
<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>Zentrube Minimal Plot (Offline ES5)</title>

<meta name="viewport" content="width=device-width,initial-scale=1">

<style>

 body{font-family:system-ui,-apple-system,Segoe UI,Roboto,Arial,sans-
serif;margin:24px;color:#111}

 h1{font-size:20px;margin:0 0 8px}

 h2{font-size:16px;margin:18px 0 8px}

 .row{display:flex;flex-wrap:wrap;gap:12px}

 .card{border:1px solid #ddd;border-radius:10px;padding:12px}

 .ctrls label{display:inline-block;margin:4px 10px 4px 0}
```

```
 textarea{width:100%;min-height:80px;font-family:ui-monospace,Consolas,monospace}

 canvas{display:block;width:100%;max-width:980px;height:260px;border:1px solid #eee;border-radius:8px;background:#fff}

 .small{font-size:12px;color:#555}

 .btn{padding:8px 12px;border:1px solid #333;border-radius:8px;background:#111;color:#fff;cursor:pointer}

 .btn.ghost{background:#fff;color:#111;border-color:#aaa}

 .grid{display:grid;grid-template-columns:repeat(auto-fit,minmax(240px,1fr));gap:12px}

 input[type="number"], input[type="text"]{width:100px}

 .pill{display:inline-block;font-size:12px;border:1px solid #ddd;padding:2px 8px;border-radius:999px;margin-right:6px}

 .mono{font-family:ui-monospace,Consolas,monospace}

</style>

</head>

<body>


<h1>Appendix I — Minimal HTML/ES5 Demo (Offline Plot)</h1>


<p class="small mono">

Formulas (plain text):<br>

• Zentrube$_t$= log(Var($x_o$:$_t$) + 1) * exp(−λ*t)<br>

• EdgeSlope_t = |Zentrube_t − Zentrube_{t−1}| / Δt

</p>


<div class="card ctrls">

 <h2>1) Generate or paste a series</h2>

 <div class="grid">

  <label>Length N <input id="N" type="number" value="600" min="32"></label>

  <label>Baseline noise (σ) <input id="noise" type="number" value="0.02" step="0.001"></label>
```

```
  <label>Pre-ramp length <input id="preRamp" type="number" value="120" min="0"></label>
  <label>Drift start index <input id="driftStart" type="number" value="360" min="0"></label>
  <label>Burst amplitude <input id="burstAmp" type="number" value="0.5" step="0.01"></label>
  <label>Seed <input id="seed" type="number" value="12345" min="1"></label>
 </div>
 <div style="margin:8px 0">
  <button class="btn" id="btnGen">Generate</button>
  <button class="btn ghost" id="btnClear">Clear</button>
 </div>
 <textarea id="seriesBox" placeholder="Paste comma-separated numbers here (e.g., 0.1,0.12,0.11,...)"></textarea>
 <div class="small">Tip: You can paste values separated by commas, spaces, or newlines.</div>
</div>

<div class="card ctrls">
 <h2>2) Kernel & window settings (tested canonical)</h2>
 <div class="grid">
  <label>λ (lambda) <input id="lambda" type="number" value="0.03" step="0.001" min="0.0001"></label>
  <label>Use σ (stddev) <input id="useSigma" type="checkbox"></label>
  <label>Window length <input id="win" type="number" value="60" min="4"></label>
  <label>Hop size <input id="hop" type="number" value="30" min="1"></label>
  <label>Δt per hop (time units) <input id="dt" type="number" value="1.0" step="0.1"></label>
 </div>
 <div style="margin:8px 0">
  <button class="btn" id="btnPlot">Compute & Plot</button>
  <button class="btn ghost" id="btnCSV">Export CSV</button>
 </div>
```

```
  <div id="summary" class="small"></div>
</div>

<div class="row">
  <div class="card" style="flex:1 1 480px">
    <div class="pill">Z(t)</div>
    <canvas id="cz" width="980" height="260"></canvas>
  </div>
  <div class="card" style="flex:1 1 480px">
    <div class="pill">EdgeSlope(t)</div>
    <canvas id="ces" width="980" height="260"></canvas>
  </div>
</div>

<p class="small">
Notes. No libraries, no network. ES5-safe functions mirror the reference kernel. Everything runs locally in your browser.
</p>

<script>
// ------------------------------
// ES5 utilities & kernel
// ------------------------------
function LCG(seed){var s=seed>>>0;return function(){s=(1664525*s+1013904223)>>>0;return s/4294967296;};}

function parseSeries(text){
  var parts = text.replace(/\\n/g,' ').split(/[\\s,]+/);
```

```
  var out=[], i, v;
  for(i=0;i<parts.length;i++){
    if(parts[i]==='') continue;
    v = parseFloat(parts[i]);
    if(!isNaN(v)) out.push(v);
  }
  return out;
}

function mean(arr){
  var i, n=arr.length, m=0;
  for(i=0;i<n;i++) m+=arr[i];
  return n? m/n : 0;
}

function spread(arr, useSigma){
  var n=arr.length, i, mu=0, acc=0, d;
  if(!n) return 0;
  for(i=0;i<n;i++) mu+=arr[i];
  mu/=n;
  for(i=0;i<n;i++){ d=arr[i]-mu; acc+=d*d; }
  if(useSigma) return Math.sqrt(acc/n);
  return acc/n; // variance
}

function zentrube(seriesSlice, t, lambda, useSigma){
  var S = spread(seriesSlice, useSigma);
```

```
  return Math.log(S + 1.0) * Math.exp(-lambda * t);
}

function rollingZ(series, window, hop, lambda, useSigma){
  var out=[], start=0, t=0;
  while(start + window <= series.length){
    out.push(zentrube(series.slice(start, start+window), t, lambda, useSigma));
    start += hop;
    t += 1;
  }
  return out;
}

function edgeSlope(Z, deltaT){
  var out=[], i;
  if(!Z.length) return out;
  out.push(0); // undefined at t=0 → define 0
  for(i=1;i<Z.length;i++){
    out.push(Math.abs(Z[i]-Z[i-1]) / (deltaT>0? deltaT:1));
  }
  return out;
}

// ------------------------------
// Synthetic series with silent pre-transition
// ------------------------------
function genSeries(N, baseNoise, preRamp, driftStart, burstAmp, seed){
```

```
 var rnd = LCG(seed);
 var out=[], i, mu=0, sigma, rampEnd=Math.max(0, driftStart);
 for(i=0;i<N;i++){
  sigma = baseNoise;
  // Silent pre-transition: ramp noise upward linearly during [driftStart-preRamp, driftStart)
  if(i >= (driftStart - preRamp) && i < driftStart){
   var r = (i - (driftStart - preRamp)) / Math.max(1, preRamp);
   sigma = baseNoise * (1 + 2*r); // up to ~3x noise at driftStart
  }
  // Visible burst/shift after driftStart
  var shift = (i >= driftStart) ? burstAmp : 0;
  // Box-Muller lite via uniform sum (cheap ES5): sum 12 uniforms - 6 ≈ N(0,1)
  var u=0,j; for(j=0;j<12;j++) u += rnd();
  var gauss = (u - 6);
  out.push( mu + shift + gauss * sigma );
 }
 return out;
}

// ------------------------------
// Canvas plotting (simple, ES5)
// ------------------------------
function drawLine(canvas, data, title){
 var ctx = canvas.getContext('2d');
 var W = canvas.width, H = canvas.height, pad=36;
 ctx.clearRect(0,0,W,H);
 // Frame
```

```
ctx.strokeStyle = '#ddd'; ctx.lineWidth=1;
ctx.strokeRect(0,0,W,H);
// Title
ctx.fillStyle='#111'; ctx.font='12px system-ui, -apple-system, Arial'; ctx.fillText(title, 10, 16);
if(!data || !data.length){ ctx.fillText('No data', 10, 32); return; }
// Compute min/max
var i, n=data.length, mn=Infinity, mx=-Infinity;
for(i=0;i<n;i++){ if(data[i]<mn) mn=data[i]; if(data[i]>mx) mx=data[i]; }
if(mn===mx){ mn-=1; mx+=1; }
// Axes
var x0=pad, y0=H-pad, x1=W-pad, y1=pad;
ctx.strokeStyle='#eee'; ctx.beginPath();
var gy, ticks=5;
for(i=0;i<=ticks;i++){
  gy = y0 - (y0-y1)*i/ticks;
  ctx.moveTo(x0, gy); ctx.lineTo(x1, gy);
}
ctx.stroke();
// Labels
ctx.fillStyle='#666';
ctx.fillText(mx.toFixed(4), x1-60, y1+10);
ctx.fillText(((mn+mx)/2).toFixed(4), x1-60, (y0+y1)/2+4);
ctx.fillText(mn.toFixed(4), x1-60, y0-2);
// Plot
ctx.strokeStyle='#111'; ctx.lineWidth=1.2; ctx.beginPath();
for(i=0;i<n;i++){
  var x = x0 + (x1-x0) * (i/(n-1));
```

```
    var y = y0 - ( (data[i]-mn) / (mx-mn) ) * (y0-y1);
    if(i===0) ctx.moveTo(x,y); else ctx.lineTo(x,y);
  }
  ctx.stroke();
}

// ------------------------------
// Wiring
// ------------------------------
function $(id){return document.getElementById(id);}

function summarize(Z, ES){
  function maxAbsDelta(a){
    var i,m=0; for(i=1;i<a.length;i++){ var d=Math.abs(a[i]-a[i-1]); if(d>m)m=d; } return m;
  }
  var last = Z.length? Z[Z.length-1] : 0;
  var peak = 0, i; for(i=0;i<Z.length;i++) if(Z[i]>peak) peak=Z[i];
  var esPeak = 0; for(i=0;i<ES.length;i++) if(ES[i]>esPeak) esPeak=ES[i];
  return {
    fz_last: +last.toFixed(6),
    fz_peak: +peak.toFixed(6),
    edge_slope_peak: +esPeak.toFixed(6),
    max_abs_delta: +maxAbsDelta(Z).toFixed(6)
  };
}

function exportCSV(Z, ES, deltaT){
```

```
  var i, rows=["t,Z,EdgeSlope,deltaT"];
  for(i=0;i<Z.length;i++){
    rows.push([i, Z[i], ES[i]||0, deltaT].join(","));
  }
  var blob = new Blob([rows.join("\n")], {type:"text/csv"});
  var a = document.createElement("a");
  a.href = URL.createObjectURL(blob);
  a.download = "zentrube_series.csv";
  document.body.appendChild(a);
  a.click();
  setTimeout(function(){ URL.revokeObjectURL(a.href); document.body.removeChild(a); }, 0);
}

function main(){
  $('btnGen').onclick = function(){
    var N = parseInt($('N').value,10);
    var noise = parseFloat($('noise').value);
    var preRamp = parseInt($('preRamp').value,10);
    var driftStart = parseInt($('driftStart').value,10);
    var burstAmp = parseFloat($('burstAmp').value);
    var seed = parseInt($('seed').value,10) || 12345;
    if(N<=0){ alert("Length N must be > 0"); return; }
    var s = genSeries(N, noise, preRamp, driftStart, burstAmp, seed);
    $('seriesBox').value = s.join(", ");
  };

  $('btnClear').onclick = function(){ $('seriesBox').value=''; };
```

```
$('btnPlot').onclick = function(){
  var arr = parseSeries($('seriesBox').value);
  if(arr.length === 0){ alert("Provide a series (generate or paste numbers)."); return; }
  var lam = parseFloat($('lambda').value);
  var useSigma = $('useSigma').checked;
  var win = parseInt($('win').value,10);
  var hop = parseInt($('hop').value,10);
  var dt = parseFloat($('dt').value);
  if(win<=1 || hop<=0){ alert("Window must be >1 and hop >0."); return; }
  if(arr.length < win){ alert("Series shorter than window length."); return; }
  var Z = rollingZ(arr, win, hop, lam, useSigma);
  var deltaT = hop * dt;
  var ES = edgeSlope(Z, deltaT);
  drawLine($('cz'), Z, "Z(t)  |  Zentrube_t = log(Var(x_o:_t) + 1) * exp(-λ*t)");
  drawLine($('ces'), ES, "EdgeSlope(t)  |  |Z_t - Z_{t-1}| / Δt  (Δt = hop * dt)");
  var s = summarize(Z, ES);
  $('summary').innerHTML =
    "fz_last: <b>"+s.fz_last+
    "</b>   fz_peak: <b>"+s.fz_peak+
    "</b>   edge_slope_peak: <b>"+s.edge_slope_peak+
    "</b>   (Δt = "+deltaT+")";
  // keep arrays for CSV export
  window.__Z = Z; window.__ES = ES; window.__deltaT = deltaT;
};

$('btnCSV').onclick = function(){
```

```
  var Z = window.__Z || [];
  var ES = window.__ES || [];
  var dT = window.__deltaT || 1;
  if(!Z.length){ alert("Nothing to export. Compute first."); return; }
  exportCSV(Z, ES, dT);
 };
}

main();
</script>

</body>
</html>
```

Appendix J — Zentrube Domain Setup Cheatsheet

Note
Zentrube defines entropy drift with time-aware decay. Since drift exists in every system, the formula is universally applicable. This matrix is a sample guide — users should tune λ, β, and window parameters to their own data, and always follow **Section 1: Pseudoscience vs. Readiness** before considering operational use.

Domain Matrix

Domain / Function	Recommended Variants	λ (Decay Horizon)	Window / Hop	Special Params
Symbolic Intelligence / Search	Canonical Z, Z + JS	0.02–0.04 (mid horizon)	Sliding, equal length	JS weight α ≈ 0.1–0.3
AI Systems & Governance	Z_weighted, Z_ratio, Z_cluster	0.01–0.03 (long horizon for eval stability)	Aligned with eval batch	Keep weights interpretable; start uniform

Domain / Function	Recommended Variants	λ (Decay Horizon)	Window / Hop	Special Params
Cybersecurity	Z_r, Z_ratio, σ vs Var	0.02–0.05 (short horizon for anomalies)	1–5 min windows	ε ≈ 1e-9 for stability
Software Logs / DevOps	Z_weighted, Z_r	0.02–0.03	Windows aligned to release cycles	Use post-fix Δ_t to mark healing
Telecom / Network	Z_weighted, Z_ratio	0.02–0.05	1–10s rolling	Align series lengths for ratio
Audio Clarity	Z_Δ, multi (with Δ, β)	0.01–0.03	256–1024 samples, 50% overlap	$\beta < 1$ for subtle clarity; $\beta > 1$ for bursts
Imaging / Visual Entropy	Z_weighted, Z + JS	0.01–0.02 (longer horizon)	Sliding pixel/patch windows	JS weight α small (~0.1)
Physiology (ECG/EEG/HRV)	Z_r, Canonical Z, Z_pol	0.02–0.04	5–30s	Use σ for cross-subject comparability
Insurance / Risk	Canonical Z, Z_cluster	0.005–0.02 (very long horizon)	Months/years	Cluster weights ~ portfolio share
Markets / Currency (Observation)	Z_r, Z_ratio, Z_cluster	0.01–0.03	Daily/weekly	Avoid buy/sell — observation only
Geometry / Structural Modeling	Canonical Z, Z_weighted	0.01–0.03	Stress/strain cycles	Weight multi-metrics equally first
Construction / Civil	Canonical Z, Z_cluster	0.005–0.02	Long scales (months)	Cluster = locations/sensors
Combustion / Thermal	Z_Δ, multi, Z_r	0.02–0.05	ms–s range	$\beta > 1$ to emphasize ruptures
Semiconductors / Electronics	Z_ratio, Z_weighted	0.02–0.04	Aligned test cycles	Ratio = current/voltage
Materials / Chemistry	Z_weighted, Z_elemental	0.01–0.03	Lab cycle length	Element mapping optional
Flight Dynamics / Glide	Z_weighted, Z_r	0.01–0.03	Seconds–minutes	Use Δ_t polarity for stall vs. recovery
Spaceflight / Trajectories	Z_weighted, Z_r	0.005–0.02	Hours–days	Long λ horizon for orbital drift
Weather Forecasting	Z_Δ, multi, λ sweeps	0.01–0.03	1h–6h windows	Multi-param fusion
Natural Disasters (Cyclones, Floods)	Z_r, Z_weighted	0.01–0.02	Hours–days	Rupture polarity for escalation
Maritime / Navigation	Z_weighted, Z_ratio	0.01–0.03	Minutes–hours	Baseline = met-ocean reference
Transportation Safety	Z_weighted, Z_r, λ sweep	0.02–0.05	Seconds	Short horizon for accident prevention

Domain / Function	Recommended Variants	λ (Decay Horizon)	Window / Hop	Special Params
Sports Fairness (Edge Decisions)	Z_ratio, Z_r	0.02–0.04	Frame-by-frame	Ratio = event vs. reference
Education / Learning Analytics	Canonical Z, Z_pol	0.01–0.03	Session length	Polarity = engage vs. disengage
Social Polarization / Collective	Z_pol, Z_cluster, Z_ratio	0.01–0.02	Days–weeks	Polarity = sentiment split
SEO / Attention Dynamics	Z + JS, Z_r	0.01–0.03	Days	JS weight α small
Supply Chains / Ops	Z_weighted, Z_cluster	0.01–0.02	Days–weeks	Cluster = sites, regions
Energy / Smart Grids	Z_weighted, Z_r	0.01–0.02	Minutes–hours	Use λ sweeps for horizon planning
Edge / IoT Sensing	Canonical Z, Z_weighted	0.02–0.05	Seconds–minutes	Use ES5-safe browser builds

---

Appendix K — Demo Pack & Tested Kernel Applications

This appendix provides a practical set of reproducible demos and tested kernel applications. All examples rely on the canonical Zentrube formula or its direct variants. The intent is to show how Zentrube can be verified immediately through small, offline experiments, ensuring transparency and reproducibility.

K.1 Practical Demos

D1 — Annuities Tail Realism

- Compare classical projections with entropy-damped overlays.
- Sweep λ across a small range and record governance notes.
- Observe how Zentrube moderates extremes while collapsing to classical values when variance is low.

D2 — Symbolic Search Overlay

- Log Zentrube_t against successive suggestion hops in a symbolic query.
- Visualize drift-aware search behavior and suppress results when drift exceeds an ethical threshold.

D3 — Idea Expansion Stability

- Attach $Z\_weighted_u$ to multi-signal or metadata-enriched primitives.
- Show how drift fields stabilize expansions without uncontrolled divergence.

D4 — Telemetry Micro-Shifts

- Compute $Zentrube_t$ on jitter, SNR, or pixel noise streams.
- Derive $EdgeSlope_t = |Zentrube_t - Zentrube_{t-1}| / \Delta t$ as an early-warning indicator.
- Demonstrate lead-time alerts before visible failures.

K.2 Canonical Formulas (Plain Text)

- **Canonical:**
  $Zentrube_t = \log(Var(x_{0:t}) + 1) \times \exp(-\lambda t)$

- **Weighted (multi-signal):**
  $Z\_weighted_u = \log(\Sigma_i w_i \times Var(x_{i0:u}) + 1) \times \exp(-\lambda u)$

- **Edge slope:**
  $EdgeSlope_t = |Zentrube_t - Zentrube_{t-1}| / \Delta t$

- **Directional split:**
  - $Z\_rupture_t = Zentrube_t \times \max(\Delta_t, 0)$
  - $Z\_recovery_t = Zentrube_t \times \max(-\Delta_t, 0)$
  - with $\Delta_t = (x_t - x_0) / (x_0 + \varepsilon)$

- **Zero divergence:**
  $D\_zero(u) = \log(Var\_obs / Var\_ref) \times \exp(-\lambda u)$

- **Adaptive decay:**
  $\lambda_t = clamp(\lambda_0 \times (1 + \gamma \times (EdgeSlope_t/\tau - 1)), \lambda\_min, \lambda\_max)$

K.3 Demo Protocols

P1 — Early-Warning Telemetry (Noise/Jitter)

- Load a univariate time series ($\Delta t = 1$).
- Compute $Zentrube_t$ and $EdgeSlope_t$.
- Trigger a WARN if $EdgeSlope_t > \tau$ for N-of-M windows.
- Plot a marker before the visible failure to demonstrate lead-time.

P2 — Insurance Tail Governance

- Compute classical present value and entropy-damped present value using $W_t = \exp(-\lambda t)$.
- Overlay both plots and compare $\Delta(t)$.
- Run λ sweep to find the stability "knee" and capture governance snapshots.

P3 — Symbolic Search Drift

- For each step in a symbolic expansion, append results to a session list.
- Compute Zentrube_t on the anchor variance.
- Display drift-distance badges per step and enforce limits when drift exceeds a threshold.

P4 — Multi-Sensor Fusion

- Inputs: e.g., speed, vibration, and temperature with weights [0.4, 0.4, 0.2].
- Compute Z_weighted_u.
- Apply adaptive λ when EdgeSlope_t spikes to handle transient anomalies.

K.4 Geometry and Dimensional Mapping (Tested Kernel)

Canonical symbolic drift (tested):

- Canonical:
  $\text{Zentrube}_t = \log(\text{Var}(x_{0:t}) + 1) \times \exp(-\lambda t)$
- Weighted:
  $\text{Z\_weighted}_u = \log(\ \Sigma_i\ w_i \times \text{Var}(x_{i0:u}) + 1\ ) \times \exp(-\lambda u)$

Dimension-safe overlays (examples):

- Length: $L_s = L + k \times s\_L \times fz$
- Angle: $\theta_s = \theta + k \times s\_\theta \times fz$
- Radius → Curvature: $R_s = R + k \times s\_R \times fz$; $\kappa_s = 1 / R_s$
- Perimeter: $P_s = P + k \times s\_P \times fz$
- Area: $A_s = A + k \times s\_A \times fz$
- Volume: $V_s = V + k \times s\_V \times fz$
- Tolerance stack: $S_s = S + k \times s\_S \times fz$ with $\text{Var}(S) = \Sigma\ \text{Var}(\text{part}_i)$

Guards: $\lambda \geq 0$, $t \geq 1$, $\text{Var} \geq 0$. Collapse to classical when $\text{Var} \to 0$ or $\lambda t \to \infty$.

K.5 Domains × Functions (Sample Applications)

All rows below rely on the same kernel (fz), tested via the canonical and weighted forms:

- **Geometry, CAD, CAE:** Gap breathing in thermal joints, curvature-aware checks, tolerance stacks, material yield margins, metadata exports.
- **Materials & Manufacturing:** Fatigue/yield drift overlays, process capability adjustments.
- **Biomed & Physiology (non-clinical):** HRV/ECG variability, recovery vs. stress detection with directional splits.
- **Imaging & Audio:** Multi-band variance for clarity drift, subtle residual detection.

- **Cybersecurity & Telemetry:** Auth/packet variance, drift in reference baselines, early warning via EdgeSlope.
- **Markets & Insurance (non-trading):** Non-trading analytics, annuities tail realism overlays, governance-safe observation.
- **Navigation & Orbits:** Orbital element drift annotations, planning margin overlays.
- **Symbolic Systems:** Meaning drift in symbolic fields, bounding expansions or search steps with entropy-aware limits.

K.6 Summary

This demo pack shows that:

- The kernel is compact: a few lines of code in ES5/NumPy.
- The collapse-to-classical property ensures safety: if variance vanishes or λ is very large, outputs reduce to classical forms.
- Interpretability is preserved: all overlays use unit-consistent sensitivities and remain auditable.

Zentrube's demos are deliberately small and offline, ensuring reproducibility, auditability, and ethical framing.

QUICK REFERENCE TABLE:

Domain	Canonical Zentrube	Weighted Variant	Directional (Z_r)	Edge Slope	Cluster / Ratio	Polarity / Elemental
Geometry / CAD / CAE	✓	✓	✓	✓		
Materials & Manufacturing	✓	✓		✓		
Physiology (non-clinical)	✓		✓	✓		✓
Imaging & Audio	✓	✓		✓		
Cybersecurity & Telemetry	✓	✓	✓	✓	✓	
Markets & Insurance	✓	✓			✓	
Navigation & Orbits	✓	✓		✓	✓	
Symbolic Systems	✓	✓	✓	✓		✓

Appendix L — Philosophical Inspiration of Zentrube

"Zero is not emptiness, but the frontier of transformation.
From its silence arises drift, and from drift, the first measure of readiness."

Purpose

This appendix records the **conceptual inspiration** behind Zentrube. These reflections are not scientific claims, but conceptual metaphors that inspired the search for a reproducible kernel.

L.1 Shunyaya as Framework, Zentrube as Formula

- **Shunyaya (framework):** Reimagines Zero not as a void, but as a *living baseline of alignment*. Instead of treating zero as inert absence, it is reframed as an active reference point — one that systems drift around, depart from, and sometimes recover toward. In this view, entropy is reframed not as an irreversible march to disorder, but as a time-aware signal of readiness and drift.

- **Zentrube (formula):** Distills this vision into mathematics:
  $\text{Zentrube}_t = \log(\text{Var}(x_{0:t}) + 1) \times \exp(-\lambda t)$

  Variance expresses drift from alignment; exponential decay encodes readiness. What began as a philosophical spark became a measurable signal of drift.

L.2 Ground Zero vs. Edge Zero

- **Ground Zero** — the silent baseline of potential, the fertile source where subtle fluctuations begin long before they can be seen. It is equilibrium holding the possibility of motion, the quiet readiness before thresholds are crossed.

- **Edge Zero** — the first visible threshold where science begins to measure and classify. It is the frontier where drift becomes manifest — the storm eye forming, the first ice crystals appearing, the arrhythmia declared, the call buffering.

Zentrube seeks to capture signals nearer the **ground**, where readiness begins, rather than only at the **edge**, where the event is already visible.

L.3 Entropy Reimagined

- **Traditional entropy:** a one-way measure of disorder or randomness.
  Shunyaya-inspired entropy: a ***readiness drift*** — rising as systems diverge, settling as they recover.
  Zentrube: captures this readiness mathematically, transforming zero from an abstract void into a living signal of alignment, instability, and renewal.

L.4 The Power of Dynamic Zero: Everyday Insights

Nature offers simple metaphors that echo this vision: Zero is not emptiness, but a living frontier where stillness gives rise to transformation.

- **Freezing point of water:** At 0 °C, water rests in dynamic balance — poised between states.
- **Melting of ice:** At 0 °C, structure dissolves into flow — a frontier of renewal.
- **Biological dormancy:** Seeds wait in suspended readiness until a small shift awakens life.

These are not scientific claims, but metaphors that inspired the re-imagination of Zero as a *dynamic threshold* rather than a void.

Closing Note

Zentrube is not philosophy disguised as science; it is science sparked by philosophy. Shunyaya's re-imagination of Zero — **as a source of potential** — opened a path where mathematics could give form to the silent drift before rupture. The formulas are reproducible; the philosophy simply preserves the inspiration.
