**Time Series Structural Break Challenge Methodology**

Sections:

1. **Introduction to Challenge**
2. **Summary of Wavelet Based Detection, Khudhayr et al. 2022 Paper**
3. **Implementation of Wavelet Based Detection feature set for supervised learning.**
4. **Summary and Implementation of Nonparametric Structural-Break Detection (Roy et al., 2024) feature set for supervised learning.**

**Section 1: Time Series Data Science Structural Break Challenge**

🔗 Competition link: [ADIA Lab Structural Break Challenge](https://hub.crunchdao.com/competitions/structural-break)

**Objective:**  
Detect whether a structural break occurred at a given boundary point in a univariate time series.

**Definition of Structural Break (Summary):**

* A structural break is a **change in the underlying data-generating process (DGP)** at a specific time.
* Can manifest as changes in parameters (e.g., mean, variance, volatility), functional form, or regime transitions.
* Breaks may be abrupt or gradual.
* No break means the process parameters remain stable across the boundary.

**Setup:**

* Each time series is split into two candidate periods:
  + *Period 0*: before the boundary
  + *Period 1*: after the boundary
* Not every boundary represents a true break — some are false splits.
* Task: assign a score ∈ [0,1] indicating likelihood of a break.
* Evaluation: ROC AUC.

**Training Data:**

* X\_train: MultiIndex DataFrame (id, time) with columns:
  + value (float): observed time series value
  + period (0/1): before vs. after the boundary
* y\_train: Series indexed by id, Boolean values:
  + True = genuine structural break
  + False = no break
* Distribution: ~30% True, ~70% False.

**Test Data:**

* Same structure as X\_train, but labels are hidden.
* Submission: Python code with train() and infer() functions, returning one prediction per id.

**Dataset Description (Summary):**

* Tens of thousands of synthetic univariate time series, each ~1,000–5,000 values.
* Each series has a designated boundary point and label (True/False) indicating break or no break.
* Series simulate scenarios from **finance, climate, industry, biomedical signals**, with varying detection difficulty.
* Goal: algorithms must **generalize across diverse scenarios** and detect breaks in unseen data.

**Methodology Suggestions (Summary):**

* Approaches include: change-point detection, distribution comparison tests, anomaly detection, supervised ML, or deep learning.
* Useful strategies: statistical tests before/after the boundary, feature extraction from both parts, time series modeling, automated deep feature learning.
* Careful preprocessing is key for robustness.
* Aim: build reliable, generalizable detection models.

**📊 Dataset Schema Diagram**

**X\_train (DataFrame with MultiIndex)**

Index: [id, time]

Columns:

- value (float)

- period (int: {0,1})

Example:

| **id** | **time** | **value** | **period** |
| --- | --- | --- | --- |
| 0 | 0 | 0.001858 | 0 |
| 0 | 1 | -0.00166 | 0 |
| 0 | 1890 | -0.00590 | 1 |
| 0 | 1891 | 0.007295 | 1 |

**y\_train (Series)**

Index: [id]

Values: bool (True/False)

Example:

| **id** | **y** |
| --- | --- |
| 0 | True |
| 1 | False |
| 2 | False |

**Methodology for Structural Break Detection**

**Core approach:**  
Use supervised learning (e.g., gradient boosting trees, random forests, or neural nets) trained on features, tests, models extracted from periods 0 and 1.

# Section 2. Summary of Boundary Break Test via MODWT-Calibrated Residual Analysis

## Boundary Break Test Via Modwt-Calibrated Residual Analysis

(Adapted from: Rashedi, Ismail, Serroukh, Al Wadi, “Wavelet Based Detection of Outliers in Volatility Time Series Models,” CMC, 2022)

## Goal & Data Setup

* Goal: Test whether the data-generating process (DGP) in Period 1 differs materially from Period 0 at a known boundary.
* Data: Univariate time series with a known split into Period 0 (before boundary) and Period 1 (after boundary).
* Output: A decision (break vs no break), and optionally a p-value or confidence measure.

## Assumptions & Notation

* Let B be the boundary index.
* If prices are strictly positive, work with returns: R\_t = 100 \* (ln X\_t − ln X\_{t−1}); else use first differences.
* Under H0 (no break), one stationary univariate model explains the whole series across the boundary.

## Step-By-Step Procedure

0) PREPROCESSING (promote stationarity)

* If X\_t > 0, convert to returns R\_t. Otherwise difference the series.
* Optionally mean-center R\_t.

1) FIT A UNIVARIATE VOLATILITY MODEL UNDER H0 (NO BREAK)

* Fit on the entire concatenated sequence (Period 0 + Period 1).
* Recommended models: GARCH(1,1), EGARCH(1,1), or GJR-GARCH(1,1).

Example: R\_t = μ\_t + a\_t, a\_t = σ\_t ε\_t, σ\_t^2 = ω + α a\_{t−1}^2 + β σ\_{t−1}^2.

* Choose error law (Normal or Student-t). Student-t is often more robust for fat tails.
* Select model by AIC/BIC if desired.

## 2) Compute Standardized Residuals

* Obtain fitted residuals â\_t and conditional volatilities σ̂\_t.
* Standardize: ε̂\_t = â\_t / σ̂\_t. Under a well-specified H0, ε̂\_t should behave i.i.d.

## 3) Apply Modwt To Residuals

* Use MODWT with a moderate-length wavelet filter (Least Asymmetric LA(8) recommended; Haar is allowed but may cause artifacts).
* Compute wavelet coefficients W\_{j,t} for scales j = 1,…,J (typically J ≤ 3).

4) THRESHOLD CALIBRATION BY MONTE CARLO (PAPER’S CALIBRATION REGIME)

* Calibration target: max statistic per scale, W\_{j,max} = max\_t |W\_{j,t}|.
* Simulate ε\_t i.i.d. under the chosen error law and series length n.
* For each simulation, apply MODWT and record W\_{j,max}.
* Obtain quantiles q\_{j,α}(n) for desired α (e.g., 0.05 or 0.02). These are the critical values used by the paper.
* For localization near the boundary (see Step 5), also simulate window-length quantiles q\_{j,α}(m) for a local window of size m.

## 5) Localize The Test At The Known Boundary

* Define a symmetric window 𝒲 around B: 𝒲 = { t : |t − B| ≤ w }, with window size m.
* Local max statistic per scale: S\_j(B) = max\_{t∈𝒲} |W\_{j,t}|.
* Decision per scale j: Reject H0 at scale j if S\_j(B) > q\_{j,α}(m).
* Combine across scales j = 1,…,J using a multiplicity correction (e.g., Bonferroni). If any adjusted test rejects, declare a break.

Robust alternatives (optional):

* Count test: C\_j(B) = number of t ∈ 𝒲 such that |W\_{j,t}| exceeds a pointwise threshold with nominal exceedance rate α.

Under H0, C\_j(B) ≈ Binomial(m, α). Large C\_j(B) supports a break.

* Energy contrast: Compare E\_{j,0} = ∑\_{t∈Period 0} W\_{j,t}^2 and E\_{j,1} = ∑\_{t∈Period 1} W\_{j,t}^2 (normalized for segment length).

Large deviations imply a scale-specific change (variance/roughness shift). Calibrate via Monte Carlo.

6) OPTIONAL ITERATIVE CLEANING (AS IN THE PAPER)

* If large exceedances are found (global or local), zero-out the largest offending coefficient(s), inverse-MODWT to reconstruct “cleaned” residuals, and refit the volatility model.
* Iterate until no exceedances remain (stabilized residual kurtosis near 3). This step improves model adequacy but is not required for the boundary test itself.

## 7) Final Decision

* Break if any scale-specific boundary-localized test rejects after multiplicity control.
* Otherwise, fail to reject (no break detected).

## Implementation Notes

* Filter choice: LA(8) preferred to avoid blocky artifacts; Haar is acceptable but may be less smooth.
* Number of scales: small J (≤3) is sufficient and consistent with the paper; higher levels produce smaller thresholds and capture coarser changes.
* Error law for calibration: match the fitted residual law (Normal or t). If in doubt, use t with ν≈7–10.
* Window size m: choose so that m ≪ n but large enough to capture boundary-localized energy (e.g., 0.5–2% of n on each side).
* Multiple testing: across scales (and/or across statistics), adjust α (Bonferroni/Simes).
* Power vs specificity: the paper notes a “lower common hard threshold” trick (use a higher-level, smaller quantile across lower levels) to gain sensitivity when tails are heavier than assumed.

PSEUDOCODE (HIGH-LEVEL)

Input: series X\_t with boundary B, significance α, window half-width w, scales J, wavelet filter F.

1. Transform X\_t → R\_t (returns or differences).

2. Fit GARCH-family model on full R\_t; compute standardized residuals ε̂\_t.

3. MODWT\_F(ε̂\_t) → { W\_{j,t} for j=1..J }.

4. Calibrate q\_{j,α}(m) via Monte Carlo (error law = fitted; length = window size m).

5. For each scale j: S\_j(B) = max\_{t∈𝒲} |W\_{j,t}|; reject\_j = [ S\_j(B) > q\_{j,α}(m) ].

6. Apply multiplicity correction across j. If any reject\_j survives correction → declare BREAK; else NO BREAK.

(Optional) Clean-and-refit loop to improve residual diagnostics, following the paper’s procedure.

**Section 3: Implementation towards a rich feature set**

**Synopsis of Khudhayr et al. (2022)**

Khudhayr et al. propose a *residual-first* outlier detection scheme for volatility models. Fit a GARCH-family model to the full series; if the model is adequate, standardized residuals should be i.i.d. “noise.” Apply a **maximal-overlap (shift-invariant) wavelet transform** to those residuals; true outliers appear as large, localized coefficients. Per scale jj, compare the observed max⁡t∣Wj,t∣\max\_t |W\_{j,t}| to Monte-Carlo critical values qj,αq\_{j,\alpha} simulated under the assumed error law. Iteratively zero offending coefficients, invert the transform, and refit the null until no exceedances remain. This turns sparse, transient anomalies into a simple test on scale-specific maxima with calibrated thresholds.

**Intuitive Outline of methodology**

1. **Model the baseline (“explain what’s explainable”)**  
   Fit a single **null** (no-break) model to the full series. The leftovers (standardized residuals) are the signal we interrogate.
2. **Look at the signal through “lenses” of scale**  
   Wavelets separate short-lived spikes from slower changes. If a structural break exists, one or more scales will show a systematic shift across the known boundary.
3. **Quantify “how much it changed,” not just “did it spike”**  
   Instead of a single yes/no test, we extract multiple **per-level contrasts** (variance/MAD/energy) and **boundary-localized** features. A supervised model learns the mapping from this multi-scale fingerprint to “break vs no-break.”

**Technical Outline**

**Null (H₀) residualization (optional but recommended).**  
Fit ARMA/GARCH on Period 0 + 1; standardize residuals ε^t\hat\varepsilon\_t. Record diagnostics (e.g., Ljung–Box on ε^t\hat\varepsilon\_t and ε^t2\hat\varepsilon\_t^2, ARCH-LM, Student-t flag/df).

**Wavelet analysis.**  
Decompose either the standardized residuals (preferred) or mean-centered values. In code, the default *example* CLI shows wavelet\_type='db4' and decomposition\_levels=4; in analysis we often use **sym4 (LA(8))** to align with the paper’s least-asymmetric filter. ([GitHub](https://github.com/YehudaDayan99/StructualBreakV2))

**Period-aware, per-level contrasts (implemented).**  
For each level jj:

* log⁡\log **variance ratio**: log⁡(Varpost/Varpre)\log\big(\mathrm{Var}\_{\text{post}} / \mathrm{Var}\_{\text{pre}}\big)
* log⁡\log **MAD ratio**: log⁡(MADpost/MADpre)\log\big(\mathrm{MAD}\_{\text{post}} / \mathrm{MAD}\_{\text{pre}}\big)

**Boundary-localized features (optional/extendable).**  
Around the known boundary index BB: local maxima of ∣Wj,t∣|W\_{j,t}| within half-windows w∈{16,32,64}w \in \{16,32,64\}; **exceedance counts** vs calibrated qj,α(w)q\_{j,\alpha}(w) (MC under Normal or Student-t with ν\nu). Thresholds are cached for reproducibility/speed.

**Outputs & usage.**  
Write per-ID features to Wavelet.csv, merge with Roy24 (if used), filter/prune, train a calibrated classifier, and evaluate on ROC/Brier. The repo’s README shows both Roy24 and Wavelet21 entry points via API/CLI. ([GitHub](https://github.com/YehudaDayan99/StructualBreakV2))

**Implementation for the Structural Break Challenge**

**Where this lives (repo)**

* methods/wavelet21/feature\_extractor.py — core feature logic (residuals → wavelets → contrasts; boundary windowing and threshold cache hooks).
* methods/wavelet21/config.py — configuration (e.g., wavelet, J, alpha, flags like use\_residuals, null-model settings).
* Batch/CLI wiring exposed via run\_batch(..., method='wavelet21', ...) and corresponding CLI flags (--wavelet-type, --decomposition-levels, --threshold-factor). ([GitHub](https://github.com/YehudaDayan99/StructualBreakV2))

**What is implemented now (per your latest code path)**

* **Residual-first option**: use\_residuals=True with a NullModelCfg controlling ARMA/GARCH and error law.
* **Diagnostics surfaced**: h0\_ljungbox\_p, h0\_archlm\_p, h0\_err\_is\_t, h0\_t\_nu.
* **Per-level period-aware contrasts** (on detail reconstructions with contrast\_engine='recon'):  
  wav\_{family}\_L{j}\_var\_logratio, wav\_{family}\_L{j}\_mad\_logratio.
* **Defaults in examples**: CLI shows wavelet\_type='db4', decomposition\_levels=4; notebooks/configs can switch to sym4 (LA(8)). ([GitHub](https://github.com/YehudaDayan99/StructualBreakV2))

**What is ready to toggle/extend next**

* Add energy\_logratio alongside var/MAD;
* Enable SWT/MODWT coefficient-domain engine (shift-invariant contrasts) alongside recon;
* Boundary-localized features with per-window MC thresholds qj,α(w)q\_{j,\alpha}(w) cached to threshold\_cache.json.

**Step-by-Step Algorithm**

Below is the *feature-generation* algorithm used to produce the Wavelet21 block for each series id. (Training/inference simply reuse this block to build the design matrix.)

1. **Inputs**  
   Series values (xt)t=1n(x\_t)\_{t=1}^n, period labels pt∈{0,1}p\_t \in \{0,1\} (single boundary), config C=\mathcal{C} = {wavelet, J, alpha, flags, null-model params}.
2. **Residualization (if use\_residuals=True)**  
   Fit null on full (xt)(x\_t) → standardized residuals ε^t\hat\varepsilon\_t.  
   Emit: h0\_ljungbox\_p, h0\_archlm\_p, h0\_err\_is\_t (0/1), h0\_t\_nu (df).
3. **Wavelet transform**  
   Transform xtx\_t (centered) or ε^t\hat\varepsilon\_t according to engine:

* contrast\_engine='recon' (current): compute per-level **detail reconstructions**.
* *(optional next)* contrast\_engine='coef\_swt': compute **SWT** detail coefficients.

1. **Per-level period contrasts (implemented)**  
   For each level j∈{1,…,J}j\in\{1,\dots,J\}:

* wav\_{w}\_L{j}\_var\_logratio =log⁡(Varpost/Varpre)=\log(\mathrm{Var}\_{\text{post}}/\mathrm{Var}\_{\text{pre}})
* wav\_{w}\_L{j}\_mad\_logratio =log⁡(MADpost/MADpre)=\log(\mathrm{MAD}\_{\text{post}}/\mathrm{MAD}\_{\text{pre}})
* *(optional next)* wav\_{w}\_L{j}\_energy\_logratio =log⁡(Epost/Epre)=\log(E\_{\text{post}}/E\_{\text{pre}}), E=E[Wj2]E=\mathbb{E}[W\_j^2].

1. **Boundary-localized features (optional/next)**  
   Let B=min⁡{t:pt=1}B=\min\{t: p\_t=1\}. For w∈{16,32,64}w\in\{16,32,64\} and each jj:

* ...\_localmax\_w{w} =max⁡∣t−B∣≤w∣Wj,t∣=\max\_{|t-B|\le w}|W\_{j,t}|.
* ...\_exceed\_w{w} =∑∣t−B∣≤w1{∣Wj,t∣>qj,α(w)}=\sum\_{|t-B|\le w} \mathbf{1}\{|W\_{j,t}|>q\_{j,\alpha}(w)\}, with qj,α(w)q\_{j,\alpha}(w) from MC (Normal vs Student-t(ν\nu)) and **cached**.

1. **Output assembly**  
   Return a dict keyed by id with all h0\_\* and wav\_\* features. The batch runner concatenates rows into Wavelet.csv. Downstream: join with Roy24 predictors, filter/prune, fit model, calibrate, evaluate.

**Notes & defaults**

* **Wavelet family & levels.** Examples show db4 with 4 levels via CLI; analysis often prefers **sym4 (LA(8))** for least-asymmetric filters per the paper. Choose per dataset; keep J∈[3,4]J\in[3,4] absent very long series. ([GitHub](https://github.com/YehudaDayan99/StructualBreakV2))
* **Caching.** MC thresholds are stored to threshold\_cache.json with keys incorporating (wavelet,J,α,window length,error law/ν)(\text{wavelet}, J, \alpha, \text{window length}, \text{error law/}\nu).
* **Stability.** All extensions are **flag-gated**; legacy columns remain, so downstream notebooks won’t break as you enrich the feature set.

**Repo pointers & usage**  
See README for API/CLI examples for both methods and project layout; Wavelet21 arguments include --wavelet-type, --decomposition-levels, and a threshold parameter in the CLI pathway. ([GitHub](https://github.com/YehudaDayan99/StructualBreakV2))

## Reference

Rashedi, K. A., Ismail, M. T., Serroukh, A., & Al Wadi, S. (2022). Wavelet Based Detection of Outliers in Volatility Time Series Models. Computers, Materials & Continua, 72(2), 3835–3847.

**Section 4. Summary of Nonparametric Structural-Break Detection (Roy et al., 2024)**

**Synopsis.** Roy, Podder & Deb (2024) propose a *nonparametric* change-detection framework for time series in the location–scale model

Y\_t = μ(X\_t) + σ(X\_t)·ε\_t,

where ε\_t are i.i.d. with unit variance. The idea is to compare the *entire shapes* of the conditional mean μ(·) and conditional variance σ²(·) **before vs. after** a split, using kernel estimators. The test statistic is the **standardized sup-norm** of the difference between segment-wise curves, evaluated on a fine grid. Under H₀ (no break), this supremum obeys a **Gumbel** extreme-value limit, giving analytic critical values. They also give **CPFind**, a two-stage binary-segmentation algorithm to locate unknown breaks.

**4.1 Intuitive outline**

* **Smooth each side:** Nonparametrically smooth the data in Period 0 and Period 1 to estimate μ₀(x), μ₁(x) (and similarly σ²₀(x), σ²₁(x)).
* **Compare shapes:** Look for the **largest absolute gap** between the two smoothed curves across x. If the biggest gap is unusually large relative to sampling noise → a break.
* **Mean and variance:** Do this once for the conditional mean and once for the conditional variance. Either can signal a structural break.

**4.2 Technical outline (known-boundary version)**

**Model.** Y\_t = μ(X\_t) + σ(X\_t)ε\_t; X\_t may be the lagged response (X\_t = Y\_{t-1}) or a short lag-vector. In our challenge we use a univariate regressor by default.

**Segments.** Split at the *given* boundary B into T₋ (Period 0) and T₊ (Period 1). Let g(·) denote μ(·) or σ²(·). Define g\_diff(x) = g₊(x) − g₋(x).

**Estimators.** Use Nadaraya–Watson (NW) kernel estimators with bandwidth b\_n and kernel K(·):

* \hat μ\_±(x) = Σ\_{t∈T\_±} K((x − X\_t)/b\_n)·Y\_t / Σ\_{t∈T\_±} K((x − X\_t)/b\_n).
* For variance, estimate \hat σ²\_±(x) from squared residuals using a bias-reduced mean (μ̂\*), per Roy24.

**Grid.** Evaluate on Π\_n = {x\_j : x\_j = Λ₁ + 2j·b\_n, j=0,…,m\_n−1}, where [Λ₁,Λ₂] is the observed support of X and m\_n ≈ (Λ₂−Λ₁)/(2b\_n).

**Standardized sup statistics.**

* Mean-break statistic:

T\_μ = c\_μ · max\_{x∈Π\_n} [ √{f̂\_X(x)} · | μ̂*\_+(x) − μ̂*\_−(x) | / √{σ̂²(x)} ],

where f̂\_X is the kernel density of X and c\_μ is the standard Roy24 normalizer ∝ √{m\_n/(n b\_n)} with a kernel-specific constant.

* Variance-break statistic:

T\_σ = c\_σ · max\_{x∈Π\_n} [ √{f̂\_X(x)} · | σ̂*²\_+(x) − σ̂*²\_−(x) | / S\_b(x) ],

where S\_b(x) and ν̂\_ε follow Roy24 to stabilize the variance of the variance-estimator.

**Null calibration.** Under H₀ and regularity conditions, T\_μ and T\_σ converge to **Gumbel**; so the critical value is B\_{m\_n}(z\_α) with z\_α = −log{−log(1−α)}. A rejection in either channel (mean or variance) implies a break.

**Unknown-break (reference).** Roy24’s **CPFind** recursively tests midpoints, splits on rejections, and runs confirmatory tests on candidate cutpoints to avoid over-segmentation. We only borrow elements of CPFind as *features* (see §5) since our boundary is known.

**4.3 Practical choices**

* **Regressor X\_t.** Default X\_t = Y\_{t−1}. (Optionally augment with short-lag vector and reduce via first PC.)
* **Kernel & bandwidth.** Symmetric bounded kernel (Epanechnikov or Gaussian). Bandwidth via CV on the NW MSE or weighted-MSE criterion described in Roy24.
* **Trimming.** For stability, restrict Π\_n to the central 98% of the empirical X-range.
* **Minimal segment length.** Enforce n\_± ≥ n\_min (e.g., 300) before computing statistics.

**Section 5. Implementation toward a Roy24 feature set (for supervised learning)**

We transform Roy24’s test apparatus into a **feature generator** evaluated at the known boundary B. Features are designed to be stable across series lengths and informative under class imbalance.

**5.1 Pointwise and sup-norm signals (per channel)**

Compute on Π\_n:

1. **Sup gap (raw):** sup\_x | μ̂*\_+(x) − μ̂−(x) |, and sup\_x | σ̂\*²*+(x) − σ̂\*²\_−(x) |.
2. **Sup gap (standardized):** T\_μ and T\_σ (the Roy24-scaled statistics).
3. **L₁/L₂ gap integrals:** mean\_x |Δμ̂\*(x)|, mean\_x Δμ̂\*(x)²; same for variance.

**5.2 Exceedance geometry on the grid**

On each channel, form standardized pointwise scores S\_μ(x) and S\_σ(x) (the bracketed terms in T\_μ/T\_σ):  
4) **Exceedance count:** #{x∈Π\_n : S\_μ(x) > c\_α}, #{x∈Π\_n : S\_σ(x) > c\_α}, where c\_α is the Roy24 Gumbel-based pointwise threshold.  
5) **Exceedance mass:** mean of S\_μ(x) over exceedance set; likewise for S\_σ(x).  
6) **Cluster width near boundary:** length (in x) of the largest contiguous exceedance cluster intersecting the boundary neighborhood.

**5.3 Boundary-localization features**

1. **Argmax position:** x*\_{μ} = argmax\_x S\_μ(x), x*\_{σ} similarly.
2. **Offset from boundary:** |x\* − x\_B| after mapping the time-boundary B to the X-scale (for X\_t = Y\_{t−1}, use contemporaneous X at B).
3. **Two-stage confirmation (CPFind-lite):** run the Roy24 confirmatory check at the nearest gridpoint to B; include its binary outcome as a feature.

**5.4 Bandwidth/complexity descriptors (regularizers)**

1. **Selected bandwidths:** b̂\_μ, b̂\_σ from CV; **grid size** m\_n; **effective sample sizes** n\_−, n\_+.

These 10–20 features (per series) integrate smoothly with the existing Wavelet21 feature block.

**5.5 Defaults and hyperparameters**

* Kernel = Epanechnikov; α = 0.05; trim = 1% per tail; grid step = 2·b̂; n\_min = 300.
* For σ²(·) estimation use the mean-bias correction (μ̂\*), S\_b(·) and ν̂\_ε per Roy24’s variance theory.
* Scale/clip S\_μ, S\_σ to [0, p99] for robustness before computing L₂ features.

**Section 6. Repository mapping and integration**

**Code location.** methods/roy24/ (mirrors the structure of methods/wavelet21/).

**Proposed modules (sketch):**

* nw.py — fast NW regression, CV bandwidth, density f̂\_X.
* roy24\_stats.py — build grid Π\_n; compute S\_μ(x), S\_σ(x); return T\_μ, T\_σ and pointwise arrays.
* roy24\_features.py — assemble the features in §5.
* \_\_init\_\_.py — user-facing extract\_roy24\_features(series\_df, cfg).

**Pipeline hooks.**

* Training: feature union [wavelet21 | roy24] → model → calibration.
* Inference: identical extraction path; ensure same kernel/bandwidth policy.

**Config keys.**

roy24:

x\_type: "lag1" # or "pc-lags:k"

kernel: "epa" # or "gauss"

alpha: 0.05

trim: 0.01

min\_seg\_len: 300

cv\_objective: "MSE" # or "WMSE"

grid\_step\_factor: 2.0

store\_pointwise: false

**Section 7. Risks, checks, and unit tests**

* **Edge support:** if Π\_n has < 10 points after trimming, fall back to larger b\_n or no-trim.
* **Small segments:** if min(n\_±) < n\_min, skip Roy24 features for that series (impute NaNs and flag).
* **Bandwidth CV:** cap b\_n ∈ [c₁·n^{-1/5}, c₂·n^{-1/5}] and monitor over-smoothing.
* **Numerical:** clip f̂\_X(x) away from 0; add ε to denominators; handle ties in argmax.

**Unit tests (minimal):**

1. No-break synthetic AR(1): T\_μ, T\_σ below B\_{m\_n}(z\_0.05) ≥ 95% of runs.
2. Mean-shift only: T\_μ power ≫ T\_σ.
3. Volatility-shift only: T\_σ power ≫ T\_μ.
4. Heavy tails (t\_ν): robustness vs. Gaussian; confirm Gumbel calibration within tolerance.