Here's a structured document addressing your Wavelet21 error, repo alignment, and cleanup strategy.

Debugging and Cleanup Guide for Wavelet21 Implementation

1. Methodological Alignment

- The **Wavelet21 path** in your repo matches the documented approach:
- Fit a single ARMA/GARCH model under H_0 (no break).
- Transform standardized residuals with MODWT/SWT.
- Localize analysis around the boundary.
- Extract scale-wise indicators (exceedance counts, local maxima, energy contrasts).
- Feed indicators into a classifier.
- The doc specifies LA(8), which ≈ `sym4` in PyWavelets. That choice is consistent.

Conclusion: The codebase is still method-aligned, but fragile due to PyWavelets API drift and variable reuse.

---

2. Root Cause of the Error

Error:

```
TypeError: len() of unsized object
```

- At some point, `Wj` becomes a **scalar** (e.g., max value or tuple element) instead of a 1-D array.
- Causes:
- PyWavelets version drift (`swt` returns arrays in newer versions, `(cA, cD)` tuples in older ones).
- Variable reuse: `Wj` reassigned from array → scalar (`Wj = np.max(...)`).

---

3. Minimal Safe Patch

In `methods/wavelet21/feature_extractor.py`:

```python
for j in range(1, cfg.J + 1):
    # Get detail coefficients robustly
    Wj_raw = coeffs[j-1]
    if isinstance(Wj_raw, tuple):
        Wj_raw = Wj_raw[1]  # detail (cD)
    Wj = np.asarray(Wj_raw).reshape(-1)
    nj = Wj.size
```

```
    if nj == 0:
        continue

    # Exceedance fraction
    exceed_count = int(np.sum(np.abs(Wj) > thresh_j))
    features[f"j{j}_exceed_count"] = float(exceed_count)
    features[f"j{j}_exceed_frac"]  = exceed_count / float(nj)

    # Localized stats (avoid reusing Wj)
    Wj_win = Wj[win_lo:win_hi]
    S_j = float(np.max(np.abs(Wj_win))) if Wj_win.size else 0.0
    features[f"j{j}_S_local_max"] = S_j

    # Energy
    energy_j = float(np.sum(Wj * Wj))
    features[f"j{j}_energy"] = energy_j
```

Key rule: **never overwrite** `Wj` **with a scalar.** Use `Wj_win`, `S_j`, etc.

---

4. Sanity Harness

Drop this in your notebook to confirm stability:

```
import numpy as np, pywt

def _swt_details(x, wavelet='sym4', J=3):
    coeffs = pywt.swt(x, wavelet, level=J, trim_approx=True, norm=True)
    out = []
    for c in coeffs:
        if isinstance(c, tuple):
            out.append(np.asarray(c[1]).reshape(-1))
        else:
            out.append(np.asarray(c).reshape(-1))
    return out

# Smoke test
x = np.random.randn(2048)
det = _swt_details(x, 'sym4', 3)
for j, Wj in enumerate(det, 1):
    assert Wj.ndim == 1 and Wj.size == x.size
```

---

5. Stability Fixes

1. **Statsmodels ARIMA**: remove deprecated `disp=0`.

```
res = ARIMA(x, order=(1,0,1)).fit(method="statespace")
```

2. **Pin dependencies** in `requirements.txt`:

```
numpy>=1.24,<2.0
scipy>=1.10
pywavelets==1.5.0
statsmodels==0.14.1
arch>=6.0
pandas>=1.5,<2.2
```

---

6. Cleanup Plan

1. **Freeze interfaces**:

```
def wavelet_details_1d(x, wavelet, J):
    """Return [W1,...,WJ], each (n,) array."""
```

   Use everywhere.

2. **Avoid variable reuse**: keep `Wj` for full array only.

3. **Unit tests**: simple white noise / mean shift / var shift.

4. **Logging**: print shapes/types when extracting features.

---

7. Example Notebook Call

```
cfg = WaveletConfig(wavelet='sym4', J=3, alpha=0.05)
run_synthetic_demo(cfg, cache_path="threshold_cache.json")
```

Final Note

With the above patch + environment pinning, the Wavelet21 method remains aligned with the documented MODWT break test methodology, and the codebase will stop “looping on errors.”