Audit Report: Power Data Quality Pass Report

Full verification of data, logs, and logic via deliverables alone

Affiliation: GhostDrift Mathematical Institute

Execution Time: 2026-01-01 10:08:28 (JST)

Abstract

This report presents the audit results for the "Ghost Drift Audit v9.9 (FY-UWP Strict + Scale Norm)" algorithm, which utilizes deterministic Fejér–Yukawa kernel processing and FFT convolution. Based solely on the provided deliverables (audit_bundle.zip), this report enables third-party verification of the following: (1) the utilization of specific electric power demand data (power_usage.csv; Target Column: "Actual Results (10k kW)") during execution; (2) the integrity of the tamper-evident logs (hash chain); (3) the preservation of execution logic identity (logic_proxy); and (4) the satisfaction of Quality Constraint Checks regarding windowed fluctuation and negative value inclusion.

The audit results confirm that the process achieved a **PASS** status against the defined thresholds (Fluctuation Upper Limit: 0.15 / Negative Inclusion Ratio Upper Limit: 0.40), observing a Maximum Windowed Fluctuation of **0.1491719546** and a Maximum Negative Inclusion Ratio of **0.0**.

Statement of Certification Scope:
This submission (audit_bundle.zip) guarantees the following facts:

1. **Zip Archive Identity**: The SHA-256 hash of the submission file itself matches the stated value.
2. **Tamper Detection of Audit Log**: The integrity of each record's record_sha256 and the continuity of the hash chain via prev_record_sha256 in audit_log.jsonl.
3. **Consistency of Certification**: The core_sha256 and record_sha256 values in audit_record.json strictly match the final entry of the audit log.
4. **Verification of Input CSV Usage**: The complete fingerprint (filename, sha256, bytes, rows, cols, target_col, has_datetime) of the CSV file utilized during execution is recorded in the source_fingerprint field of the certificate and matches the hash of the actual file.
5. **Logic Identity (Interactive Environment Support)**: The script_identity.type is recorded as logic_proxy, containing the SHA-256 hash calculated from the

concatenated source code of major functions, sorted by name to ensure independence from definition order.

1. Background and Purpose

In the operation of predictive and control models, accuracy verification alone is insufficient to guarantee detection of operational deviation (drift) or to ensure accountability in the event of an incident. Specifically, if subtle structural changes are dismissed as "noise," the rationale for operational decisions is lost.

This audit employs deterministic kernel processing (FY Kernel) and FFT convolution—independent of statistical estimation or machine learning—to extract time-series structures. Furthermore, it mechanically evaluates "Quality Constraints" using the UWP (Uniform Window Positivity) proxy, which monitors windowed fluctuation and sign constraints. The results are preserved in a tamper-evident log structure, submitted in a format that allows third parties to independently verify identity, continuity, and consistency.

2. Input/Output Artifacts

2.1 Submission (Zip Archive)

- **Canonical Filename**: audit_bundle.zip
- **Alias**: audit_bundle (10).zip (Note: Filename may vary due to environment handling)
- **Contents**:
  - audit_record.json: Execution Certificate (Single JSON file)
  - audit_log.jsonl: Append-only Audit Log (1 line per execution record)

2.2 Submission SHA-256 (Distribution-Level Identity)

The following hash values are valid for the file regardless of whether it is saved under the canonical name or the alias.

- audit_bundle.zip (including alias): b6aa197d4afe4e755c64d290025573a4e1649b9ac6a95d942eb7b8c26f4d137d
- audit_record.json (inside zip): 79af473967f1de2e7b4754d34de0016a4b3d5375f4f5e312ad9d4c004bde27e3
- audit_log.jsonl (inside zip): 5bfab6b08d6402f1f28bbbead7f72a392f67f86d71bd7e274b06646a84872399

2.3 Audit Target CSV (Input Utilized in This Execution)

The utilization of the following input file is confirmed by the source_fingerprint (source_meta) recorded in the certificate.

- **Utilized CSV**: power_usage.csv
- **CSV SHA-256**: 8188c6032e9cb595835b17a072eed86add050cf7c38156fb9465d074eea7829a
- **Size**: 96,743 bytes
- **Dimensions**: 2,904 rows, 7 columns
- **Target Column**: Actual Results (10k kW)
- **Datetime Presence**: True

3. Methodology

3.1 Determinism and Environment Recording

The algorithm is designed to be deterministic and does not utilize random number generation. Environment variables, such as thread counts, are fixed at runtime and recorded as env in the certificate.

The env for this execution includes the following (excerpt):

- OMP_NUM_THREADS=1, MKL_NUM_THREADS=1, OPENBLAS_NUM_THREADS=1, VECLIB_MAXIMUM_THREADS=1, NUMEXPR_NUM_THREADS=1
- PYTHONHASHSEED=42
- **Note**: As numpy was loaded prior to the apply_env call (numpy_preloaded_before_apply_env=True), the strictness of environment fixation is classified as **Best-effort determinism**.
- hash_randomization=1 results in the recording of pythonhashseed_effective=False.

3.2 Log-binning

The time series x_t ($t=0,\dots,T-1$) is aggregated into a sequence $b[n]$ with dimension M_{bins}.

$$\begin{aligned} n_t &:= t+1, \quad y_t := \log(n_t), \\ \text{edges}[k] &:= y_{\min} + \frac{k}{M_{\text{bins}}}(y_{\max} - y_{\min}), \quad k=0,\dots,M_{\text{bins}}, \\ \text{idx}(t) &:= \text{clip}(\text{searchsorted}(\text{edges}, y_t, \text{right}) - 1, 0, M_{\text{bins}} - 1), \\ b[\text{idx}(t)] &\leftarrow b[\text{idx}(t)] + x_t, \\ \text{centers}[n] &:= \frac{1}{2}(\text{edges}[n] + \text{edges}[n+1]). \end{aligned}$$
In this execution, $M_{\text{bins}}=1452$ was used.

3.3 Scale-Invariant Normalization

The binned output b is normalized using the RMS scale to ensure scale invariance.

$$s := \sqrt{\frac{1}{M_{\text{bins}}} \sum_{n=0}^{M_{\text{bins}}-1} b[n]^2}, \quad b_{\text{norm}}[n] := \frac{b[n]}{s + \varepsilon}.$$
Parameters for this execution:

- Method: rms
- Scale s: 12391.1335...
- ε: 1×10^{-9}

3.4 Fejér–Yukawa Composite Kernel (FY-Kernel)

The Fejér kernel (discrete triangular kernel) is defined by the self-convolution of a normalized rectangular kernel.

$$r_\Gamma[n] = \frac{1}{2\Gamma+1} \mathbf{1}\{|n| \le \Gamma\}, \quad F_\Gamma = r_\Gamma * r_\Gamma.$$
The Yukawa kernel (finite support exponential decay) is defined as:

$$Y_\lambda[m] = \frac{e^{-\lambda|m|}}{\sum_{\ell=-M}^{M} e^{-\lambda|\ell|}} \mathbf{1}\{|m| \le M\}.$$
The composite kernel is defined as:

$$K_{FY} := F_\Gamma * Y_\lambda.$$
In terms of implementation, each component kernel is L1-normalized, and the resulting composite kernel maintains L1 normalization.
Parameters for this execution:

- $\Gamma = 10$
- $\lambda = 0.4$
- $M = 256$

3.5 FFT Convolution (Overlap-save)

The convolution of the normalized bin sequence b_{norm} and the kernel K_{FY} is performed using the Fast Fourier Transform (FFT) via the overlap-save method. The output signal is denoted as b_{smooth}.

3.6 UWP Proxy Metrics (Epsilon Guarded)

For a sliding window of width W, the window mean μ_i, window RMS σ_i, and negative inclusion ratio ν_i are defined.

$$\begin{aligned} \mu_i &:= \frac{1}{W} \sum_{j=0}^{W-1} b_{\text{smooth}}[i+j], \\ \sigma_i &:= \sqrt{\frac{1}{W} \sum_{j=0}^{W-1} (b_{\text{smooth}}[i+j] - \mu_i)^2}, \\ \nu_i &:= \frac{1}{W} \sum_{j=0}^{W-1} \mathbf{1}\{b_{\text{smooth}}[i+j] < -\text{eps}\}. \end{aligned}$$
The worst-case metrics are defined as:

$$\text{rms\_worst} := \max_i \sigma_i, \quad \text{rneg\_worst} := \max_i \nu_i$$
The criteria for passing are:

$$\text{PASS} \iff (\text{rms\_worst} \le \tau_{\text{rms}}) \land (\text{rneg\_worst} \le \tau_{\text{neg}})$$
Settings for this execution:

- $W=128$
- $(\tau_{rms}, \tau_{neg})=(0.15, 0.40)$
- $\text{eps} = 1 \times 10^{-9}$ (Epsilon guard to filter numerical noise)

4. Execution Results

4.1 Execution Identifiers

The following identifiers are confirmed via the certificate and logs:

- **run_id (core_sha256)**: 45c99fda3f89222ca9f8339b70cafeb189fb4fefa52a0b97bbd3950b0d200675
- **record_sha256**: 2f7d3b2f0ae8325996b963e038e1d023db4ce456c6300ff166a282a04e259aad
- **prev_record_sha256**: bdc5ac72b64b204313389b1b2d5fedb373b2029c5fb942d502cb27fe6d92dacd

4.2 Input/Output Digests (Array SHA-256)

Derived from the outputs field in the final record of audit_log.jsonl:

- bins_raw_sha256: 464f2f5e4f14e0aae061d57cd6f4c7857d5392419479e7dfacd0a5714a84d229
- b_smooth_array_sha256: de6847d352e874f15188d9613164e53ff7cea7f62c754654ad588e4f972968df
- S_array_sha256: 5187b708e1d94d4249e5508258b3e93c8a75186dcf76cc25edaf8ca413f207c8
- T_array_sha256: bc69cdd7e2d5004cdf63c18ce4232d9753e684a5396d031d1ab7262200fdc135
- centers_array_sha256: ac1cc5249ada48f1641410365ccf9fc75d28dbe529bb807496b2d9ace1361166

Input/Output hashes recorded in the log:

- input_sha256: 5308014ffd2874bbbbfb56b92ed0b9d2e4f79b97bae37a8140f1f11c686cb617
- output_sha256: de6847d352e874f15188d9613164e53ff7cea7f62c754654ad588e4f972968df

4.3 UWP Proxy Metrics and Judgment

Based on the metrics in the certificate:

- **rms_worst**: 0.14917195463863175
- **rneg_worst**: 0.0
- **Judgment**: uwp_pass = True (**PASS**)

Comparing these values against the thresholds $(\tau_{rms}, \tau_{neg})=(0.15, 0.40)$, both conditions are satisfied, confirming a **PASS** result.

4.4 Logic Identity (script_identity)

As recorded in script_identity:

- type: logic_proxy
- sha256: d44698a53891f4b845604b5401f2c83863762eff0f8069fd3716cc52b0644d1d

5. Discussion

5.1 Implications of the PASS Result

In this execution, the smoothed signal b_{smooth}, derived after RMS scale normalization, exhibited a worst-case window RMS below $\tau_{rms}=0.15$ and a worst-case negative inclusion ratio below $\tau_{neg}=0.40$. Consequently, based on the definition of the UWP proxy, it is mechanically concluded that **"Quality conditions regarding fluctuation and sign constraints within the window are satisfied."**

5.2 Factual Comparison with v9.3 Report

The previous v9.3 report confirmed "Non-adoption of attached CSV" due to config.source=synthetic_demo.
In contrast, this v9.9 execution confirms Real Data Adoption by strictly binding the fingerprint of power_usage.csv within the certificate.
Furthermore, logic identity is now securely recorded via logic_proxy even in Notebook/interactive environments, rendering constraint declarations regarding script_sha256 being "missing" unnecessary (this submission explicitly includes script_identity).

6. Conclusion

The "Ghost Drift Audit v9.9 (FY-UWP Strict + Scale Norm)" verified that the execution utilizing power_usage.csv resulted in a PASS for the UWP proxy.
The observed metrics were rms_worst=0.14917195463863175 and rneg_worst=0.0, well within

the thresholds $(\tau_{rms}, \tau_{neg})=(0.15, 0.40)$.
The submission bundle provides comprehensive verification capabilities, including Zip Identity, JSONL Hash Chain, Consistency between Final Log and Certificate, Input CSV Fingerprint, and Logic Identity.

Appendix A. Third-Party Verification Code (Verification Snippet)

The following minimal Python code verifies "Zip Identity," "Chain Consistency of audit_log.jsonl," and "Consistency between audit_record.json and Final Record," adhering to the v9.9 strict JSON specification.

```
import hashlib, json, zipfile

ZIP_PATH = "audit_bundle (10).zip"  # or "audit_bundle.zip"
EXPECTED_ZIP_SHA256 =
"b6aa197d4afe4e755c64d290025573a4e1649b9ac6a95d942eb7b8c26f4d137d"

def sha256_file(path):
    h = hashlib.sha256()
    with open(path, "rb") as f:
        for chunk in iter(lambda: f.read(1024 * 1024), b""):
            h.update(chunk)
    return h.hexdigest()

def strict_sha256_json(obj: dict) -> str:
    # Ensure strict separators for consistent hashing
    b = json.dumps(obj, ensure_ascii=False, sort_keys=True, separators=(',', ':')).encode("utf-
8")
    return hashlib.sha256(b).hexdigest()

def calc_record_sha256(rec: dict) -> str:
    tmp = dict(rec)
    tmp.pop("record_sha256", None)
    return strict_sha256_json(tmp)

print("1) Verifying zip hash...")
assert sha256_file(ZIP_PATH) == EXPECTED_ZIP_SHA256
print(" -> Zip hash MATCH.")
```

```
print("2) Reading artifacts from zip...")
with zipfile.ZipFile(ZIP_PATH, "r") as z:
    log_lines = z.read("audit_log.jsonl").decode("utf-8").splitlines()
    cert = json.loads(z.read("audit_record.json").decode("utf-8"))

print("3) Verifying log chain...")
prev = None
last = None
for i, line in enumerate(log_lines):
    rec = json.loads(line)

    # Record integrity check
    assert calc_record_sha256(rec) == rec["record_sha256"], f"Record hash mismatch at line
{i+1}"

    # Chain continuity check
    if prev is not None:
        assert rec.get("prev_record_sha256") == prev, f"Chain broken at line {i+1}"

    prev = rec["record_sha256"]
    last = rec

print(" -> Log chain VALID.")

print("4) Verifying certificate matches last record...")
assert cert["core_sha256"] == last["core_sha256"]
assert cert["record_sha256"] == last["record_sha256"]
print(" -> Certificate matches last log record.")

print("OK: zip / hash-chain / record consistency verified.")
```