Shunyaya Symbolic Mathematical Electrical Quantities (SSMEQ) — Zero-Centric Standard

Status: Draft for Review (v2.0)
Date: November 14, 2025
Caution: Research/observation only. Not for critical decision-making.

Citation: When implementing or adapting, cite the concept name "Shunyaya Symbolic Mathematical Electrical Quantities (SSMEQ)" as the origin of the manifest-first, zero-centric representation for electrical telemetry.

---

0. Purpose

Make electrical telemetry unitless, portable, bounded, and auditable so **voltage, current, power, frequency, power factor, THD, and unbalance** can be read, pooled, alerted on, and governed the same way across vendors, voltages, and grids. SSMEQ is **representation-only**: it replaces mixed units (V/kV, A/kA, W/kW, Hz, %) with symbolic contrasts `e_Q` and optional bounded dials `a_Q`, without changing physics, firmware, or protection models.

0.1 Key idea

1. **Normalize the raw channels once** (RMS for AC; pass-through for DC; sign convention declared).
2. **Map each quantity to a unitless contrast** using a published lens and anchors. Examples:
   - `e_V := ln( V_rms / V_ref )`
   - `e_I := ln( I_rms / I_ref )`
   - `e_f := ln( f / f_ref )`
   - `e_R := ln( R / R_ref )`
   - Signed power (import/export): `e_P := asinh( P / P_scale )`
3. **Handle bounded metrics as dials** with safe clamps:
   - `a_pf := clamp( pf, -1 + eps_a , +1 - eps_a )`
   - `a_THD := tanh( c_THD * ln( 1 + THD/THD_ref ) )`
   - `a_unbV := tanh( c_unb * ln( 1 + Unbalance_V/Unb_ref ) )`
4. **Exploit additivity for consistency checks and features:**
   - Identity residual for live self-check:
     `r_P := e_P_meas - ( e_V + e_I + ln( pf / pf_ref ) )`
   - Large `|r_P|` flags CT/PT ratio errors, sign flips, or pf estimation faults.
5. **Optionally map any `e_Q` to a bounded alignment dial** for pooling and UI:
   - `a_Q := tanh( c_Q * e_Q )` with `a_Q ∈ (-1,+1)` after `clamp_a`.

6. **Soft hysteresis near limits** to reduce chatter:
    - Proximity to a rating `Q_max`: `a_upper := tanh( c_up * ln( Q / Q_max ) )`
    - Memory: `M := rho*M_prev + (1 - rho)*step_like`, where `rho ∈ (0,1)`.

After this, **machine logic consumes `e_*`, `a_*`, and `r_P`** instead of raw V/A/W/Hz/%. Human UIs may still show traditional units for comfort or regulation, but **those displays do not feed back into logic**.

0.2 Lineage (zero-centric)

• **Shunyaya Symbolic Mathematics (SSM):** represent any quantity as a contrast around a published baseline, with an optional bounded alignment lane `a ∈ (-1,+1)`.
• **Shunyaya Symbolic Mathematical Symbols (SSMS):** pooling and governance rules for safe bounded composition (`atanh/tanh` rapidities, clamps, replayability).
• **Shunyaya Symbolic Mathematical Temperature (SSMT):** proved the pattern on a single physical axis.
• **SSMEQ:** extends the same zero-centric, manifest-first method to **multi-channel electrical systems** where additivity (`e_P ≈ e_V + e_I + ln(pf/pf_ref)`) and **bounded dials** deliver portability, stability, and fleet-wide pooling without unit drama.

0.3 Scope, Non-Goals, and Assumptions (Electrical)

Scope
SSMEQ is a **representation layer for electrical telemetry**. It defines how to express core channels as unitless contrasts `e_Q` and optional bounded dials `a_Q` so that policy, analytics, protection logic, and ML can operate **consistently across vendors, voltages, and AC/DC mixes** under a published manifest.

Typical measured channels (before symbolization):

- **Core magnitudes:** Voltage `V_rms`, Current `I_rms`, Frequency `f`, signed Power `P` (import/export), Resistance `R` or Impedance `Z_rms`, Power Factor `pf`, Total Harmonic Distortion `THD`, phase **unbalance** (per-phase voltages/currents).
- **Derived and governance channels:**
    - **Identity residual** `r_P := e_P_meas - ( e_V + e_I + ln( pf / pf_ref ) )`
    - **Proximity dials to ratings/limits:** `a_upper`, `a_lower` (for example, proximity to breaker, cable, transformer, or inverter limits)
    - **Soft hysteresis memory:** `M := rho * M_prev + (1 - rho) * u_side`

SSMEQ covers **AC and DC** by:

- Declaring **normalization once** per site/profile in the manifest
  - **AC:** RMS-based magnitudes and declared nominal anchors (for example, `V_ref`, `I_ref`, `f_ref` for 50 Hz or 60 Hz).
  - **DC:** pass-through magnitudes with DC-specific anchors (for example, DC bus voltage/current ratings).
- Then emitting symbolic signals `e_*`, `a_*`, and `r_P` under that manifest so that:
  - **Batch == stream** for pooled dials, and
  - Interpretation is **portable** across devices, vendors, and voltages, provided they share or reference a compatible manifest.

SSMEQ is intended to sit **between raw telemetry and downstream logic**: it does not replace SCADA, protection, or load-flow tools, but provides a **single, collapse-safe symbolic language** for those systems to consume.

Non-Goals

- **No physics change.** SSMEQ does **not** alter Kirchhoff, Ohm, Maxwell, or any existing protection philosophy or meter calibration.
- **Not a protection curve replacement.** It does not dictate trip curves or time–current characteristics; it provides **portable, zero-centric symbols** that protection and controls may use as inputs.
- **Not a power-quality standard.** It does not redefine how `THD`, unbalance, or flicker are computed; it wraps those **already-computed** quantities into **bounded, comparable dials**.
- **Not a predictor or optimizer.** SSMEQ does not forecast load, faults, or degradation; it supplies **clean, auditable features** for your forecasting, planning, or optimization stack.
- **No silent data correction.** It will not "fix" bad sensors or wrong CT/PT ratios. It preserves auditability: residuals like `r_P` and validity flags are intended to **surface such issues quickly**, not hide them.
- **Not a vendor lock-in mechanism.** All lenses, anchors, clamps, and residual rules are **manifest-first and open**, intended for multi-vendor, multi-site ecosystems.

Assumptions (declared, manifest-first)

Normalization and mode (AC/DC)

- **AC channels:**
  - Use RMS values for `V_rms`, `I_rms`, `Z_rms` before symbolization.
  - A **nominal frequency** `f_ref` (for example, 50 Hz or 60 Hz) is declared per site/profile.
- **DC channels:**
  - DC magnitudes are **passed through as measured** and symbolized with DC-specific anchors (for example, DC bus nominal voltage/current).

- **Frequency:**
  - `f` is in Hz before symbolization; `e_f := ln( f / f_ref )` uses a declared `f_ref`.
- The manifest **must declare** for each site/profile whether a channel is treated as **AC**, **DC**, or **mixed/derived**.

Sign conventions (must be explicit)

- **Real power `P`:**
  - A single convention is declared once per site/profile (for example, **positive = import**, negative = export), and used everywhere SSMEQ appears.
  - Signed `P` feeds the power lens (for example, `e_P := asinh( P / P_scale )`).
- **Reactive power `Q` (if used):**
  - A `reactive_sign` convention (for example, positive = inductive / lagging) is declared similarly.
- **Power factor `pf`:**
  - SSMEQ assumes **`pf` is a non-negative magnitude in [0,1]**, typically `pf = |P| / S` from existing metering.
  - Directional information (leading/lagging) is carried via the **reactive sign** rather than allowing negative `pf`.

Anchors and lenses (one lens per channel)

- For each electrical quantity `Q`, the manifest declares **exactly one lens** and keeps it fixed for the deployment/study:
  - Examples (canonical patterns):
    - `e_V := ln( V_rms / V_ref )`
    - `e_I := ln( I_rms / I_ref )`
    - `e_f := ln( f / f_ref )`
    - `e_P := asinh( P / P_scale )`
  - Anchors **may differ by site/profile** but must be **stable over the analysis window** (changes require a manifest update, not a silent edit).
- **Power factor reference:**
  - `pf_ref` is declared per site/profile.
  - It may be `1.0` (ideal unity power factor) or a **target** value (for example, `0.95`) if that better reflects contractual/operational targets.
  - Under this convention, `pf < pf_ref` yields a **negative contribution** in `ln( pf / pf_ref )`, which is intended: it expresses under-performance relative to the chosen target.

Bounded metrics and dials

- Bounded metrics are encoded as **clamped dials**, for example:
  - `a_pf := clamp( pf, -1 + eps_a, +1 - eps_a )` if `pf` can be treated as signed, or
  - `a_pf := clamp( 2 * pf - 1, -1 + eps_a, +1 - eps_a )` if `pf` is in `[0,1]` and we want a symmetric dial with `pf = 0.5` mapped near `0`.

- Alignment outputs follow a **standard pattern**:
  - `a_Q := clamp_a( tanh( c_Q * e_Q ), eps_a )` with `|a_Q| <= 1 - eps_a`.
  - Constants `c_Q`, `eps_a`, and any rescalings are **declared in the manifest**.

Clamps, bounds, and validity windows

- **Alignment clamps:**
  - `a_Q := clamp_a( tanh( c_Q * e_Q ), eps_a )` with `|a_Q| <= 1 - eps_a` by construction.
- **Validity windows:**
  - For each measured magnitude, the manifest declares ranges such as `V_valid`, `I_valid`, `f_valid`, `pf_valid`, `THD_valid`, etc.
  - When a raw value falls outside its validity window, SSMEQ:
    - Raises a **health flag**, and
    - May choose to **withhold** or **flag** the corresponding `e_Q` / `a_Q` rather than silently extrapolate.

Timestamps, identity, and manifests

- Every SSMEQ record carries:
  - A UTC timestamp (or SSM-Clock / SSM-Clock-Stamp compatible time field).
  - A `manifest_id` that resolves to:
    - Declared lenses and anchors,
    - Clamps, ranges, and guards,
    - Mode flags (AC/DC/mixed),
    - Sign conventions and any **site-profile** metadata (for example, "Plant A, 50 Hz, LV feeders").

Deterministic pooling and alignment algebra

- When pooling bounded dials across devices/feeds, SSMEQ assumes **rapidity-space pooling**:
  - `u_Q := atanh( a_Q )`
  - `U := sum( w_i * u_Q_i ), W := sum( w_i )` with `w_i = |m_i|^gamma`
  - `a_pool := tanh( U / max( W, eps_w ) )`
- This ensures:
  - **Batch == stream** (order-invariance), and
  - Outliers **cannot dominate** beyond what their declared weights permit.

Calibration responsibility and residual expectations

- Sensor calibration, CT/PT ratios, scaling factors, and wiring correctness are **outside SSMEQ**.
  - A mis-calibrated channel will produce a **self-consistent but biased** set of `e_*`.
- Identity residuals such as `r_P` are **assumed** to be evaluated under a declared tolerance band.
  - `r_P` close to zero suggests a self-consistent set; systematic deviations are expected to **flag CT/PT error, sign flips, or missing channels**, not silently correct them.

Soft hysteresis memory M (anti-chatter assumption)

- Hysteresis memory uses a standard first-order update:
  - `M := rho * M_prev + (1 - rho) * u_side`
- SSMEQ assumes:
  - **Cold start:** `M_prev = 0` unless explicitly overridden in a manifest or a replay context.
  - `rho` and any thresholds derived from `M` (for example, `m_enter`, `m_clear`) are **declared once per profile** and kept fixed during evaluation.

0.4 One-Minute Math (ASCII, copy-ready)

Normalize once (AC/DC)

- AC channels use RMS; DC channels pass-through under a declared AC/DC mode.

```
V_rms := V_input_rms
I_rms := I_input_rms
f := f_input_hz
P := P_signed // import/export sign convention declared once
Q := Q_signed_optional // if available; reactive_sign convention declared
S := V_rms * I_rms // apparent power
```

Pick ONE lens per channel (publish in manifest)
Defaults shown; you may choose linear or hybrid where noted.

```
e_V := ln( V_rms / V_ref )
e_I := ln( I_rms / I_ref )
e_f := ln( f / f_ref ) // f_ref is declared nominal (50 Hz, 60 Hz, etc.)

// Signed real power uses an odd, smooth lens
e_P := asinh( P / P_scale )

// Optional for reactive/apparent if used
e_Q := asinh( Q / Q_scale )
e_S := ln( S / S_ref )

// Resistance / Impedance (if measured or derived)
e_R := ln( R / R_ref )
e_Z := ln( Z_rms / Z_ref )
```

Linear alternatives (tight process windows)

```
e_V := ( V_rms - V_ref ) / dV
e_I := ( I_rms - I_ref ) / dI
e_f := ( f - f_ref ) / df
```

Hybrid example (near nominal linear, far log)

```
if abs( V_rms - V_ref ) <= tau_V:
 e_V := ( V_rms - V_ref ) / dV
else:
 e_V := ln( V_rms / V_ref )
```

Bounded metrics as safe dials (declare clamps)

```
a_pf := clamp( pf , -1 + eps_a , +1 - eps_a )
// pf is a magnitude in [0,1]; direction lives in reactive_sign, not in pf
itself

a_THD := tanh( c_THD * ln( 1 + THD / THD_ref ) )
a_unbV := tanh( c_unbV * ln( 1 + Unbalance_V / UnbV_ref ) )
a_unbI := tanh( c_unbI * ln( 1 + Unbalance_I / UnbI_ref ) )
```

Optional bounded alignments for contrasts (UI/pooling)

```
a_V := clamp_a( tanh( c_V * e_V ), eps_a ) // lives in (-1,+1)
a_I := clamp_a( tanh( c_I * e_I ), eps_a )
a_f := clamp_a( tanh( c_f * e_f ), eps_a )
a_P := clamp_a( tanh( c_P * e_P ), eps_a )
a_Q := clamp_a( tanh( c_Qc * e_Q ), eps_a ) // if e_Q used
```

Identity residual (live self-check)
Compare measured real power vs identity formed from V, I, pf.

```
r_P := e_P_meas - ( e_V + e_I + ln( pf / pf_ref ) )

health.consistency_ok := ( abs( r_P ) <= tol_rP for dwell_N samples )
```

Notes:

- If you choose linear lenses for `e_V` / `e_I`, use the **linearized identity** around the declared anchors rather than the log identity.
- If `pf` is near zero, guard `ln( pf / pf_ref )` with a clamp or **skip residual** until `pf >= pf_min`.

Proximity dials to ratings (reduce chatter)
Encode approach to nameplate or policy limits as smooth dials.

```
a_upper_V := tanh( c_upV * ln( V_rms / V_max ) ) // approaches +1 as V ->
V_max+
a_lower_V := tanh( c_loV * ln( V_min / V_rms ) ) // approaches +1 as V ->
V_min-
a_upper_I := tanh( c_upI * ln( I_rms / I_max ) )
a_upper_P := tanh( c_upP * ln( abs(P) / P_max ) )
```

Soft hysteresis memory (suppress flapping near limits)

```
u_side := 0.5 * ( 1 + tanh( k_side * ( V_rms - V_gate ) / dV_gate ) )
M := rho * M_prev + ( 1 - rho ) * clip( u_side , 0 , 1 )

enter_alarm := ( a_upper_V >= th_enter ) and ( M >= m_enter )
clear_alarm := ( a_upper_V <= th_clear ) and ( M <= m_clear )
```

// Cold-start convention (unless overridden in manifest)
`M_prev := 0` at startup or replay reset.

Deterministic pooling (bounded, associative)
When aggregating dials across phases, feeders, or fleets:

```
U := 0 ; W := 0
for each dial a in dials:
 a_c := clamp_a( a , eps_a ) // enforce |a| < 1
 u := atanh( a_c )
 U := U + w * u // w may depend on magnitude (for example |P|^gamma)
 W := W + w

a_pool := tanh( U / max( W , eps_w ) )
```

This guarantees **batch == stream** and resists outliers beyond their declared weights.

Health and validity

```
health.range_ok := ( V_min_valid <= V_rms <= V_max_valid )
health.clock_ok := ( jitter_utc <= jitter_max )
health.signed_ok := ( sign(P) follows declared convention )

health := health.range_ok and health.clock_ok and health.signed_ok and
health.consistency_ok
```

Minimal payloads

- **SSMEQ-Lite (S1)**

```
{ ts_utc, e_V, e_I, e_f, e_P, a_pf, manifest_id, health }
```

- **SSMEQ-Full (S2)**

```
{ ts_utc, e_V, e_I, e_f, e_P, e_Q?, e_S?,
a_pf, a_THD?, a_unbV?, a_unbI?,
r_P, a_upper_V?, a_upper_I?, a_upper_P?,
M?, manifest_id, health }
```

Knobs and guards (declare in manifest)

```
eps_a > 0, eps_w > 0, pf_min > 0
c_V, c_I, c_f, c_P, c_THD, c_unbV, c_unbI, c_upV, c_loV, c_upI, c_upP > 0
rho in (0,1), k_side > 0, th_enter > th_clear, m_enter > m_clear

V_ref, I_ref, f_ref, S_ref, R_ref, Z_ref > 0
P_scale, Q_scale > 0

V_min, V_max, I_max, P_max > 0
V_min_valid, V_max_valid, jitter_max > 0

pf_ref in (0,1], THD_ref > 0, UnbV_ref > 0, UnbI_ref > 0
```

Portable rule examples (copy-ready)

```
UNDER_V if e_V <= -0.10 for >= 3 minutes
OVER_I if a_upper_I >= +0.60 for >= 10 seconds
CONSISTENCY_ALERT if abs(r_P) > 0.15 for 5 consecutive samples
PF_WATCH if a_pf <= 0.80 for >= 15 minutes
THD_RISE if a_THD >= 0.50 and d/dt(a_THD) >= 0.05 per minute
```

Reminder

- **Publish exactly one lens per channel** and keep it fixed during the run.
- **Clamp alignments before pooling.**
- **Sign conventions, anchors, and pf_ref** must be declared up front in a manifest and never changed silently.
- **Cold start:** if not otherwise specified, assume `M_prev = 0` at the first sample.

0.5 Openness & Attribution (normative)

• **Open standard.** SSMEQ is permanently open for all to use in any context — public, industrial, municipal, national, academic, commercial, or off-world habitat. No registration, license negotiation, or fees apply.

• **Citation.** When implementing or adapting this specification, cite the concept name **"Shunyaya Symbolic Mathematical Electrical Quantities (SSMEQ)"** as the origin of the symbolic mathematical approach for electrical measurement and alignment.

• **Independence.** Implementations do not require any central registry, hosted service, or maintainer approval. Conformance is by self-declaration: compute the formulas exactly as written under a published manifest. No implementer may claim exclusive ownership or control of the standard.

• **Optional extensions.** Any organization may add optional blocks (for example, gating logic, privacy offsets, cryptographic signatures, or visual dashboards) as long as the **core symbolic definitions and baseline formulas remain intact and auditable.** Extensions must not alter the meaning of `a_THDV`, `a_THDI`, `a_unbV`, `a_unbI`, `a_df`, `a_dV`, `a_PF`, `a_I`, or `a_T` without clearly declaring that modification in a manifest note or appendix.

• **Datasets.** Any datasets used in examples retain their original licenses, terms, and usage rights. They are shown for demonstration only and carry no transfer of ownership.

• **No warranty or endorsement.** This specification is provided *as-is*, with no warranty, and no endorsement or affiliation is implied by its adoption. Implementing SSMEQ does not grant the right to represent the implementer as an official steward, certifying body, or exclusive representative of the Shunyaya framework.

Intent.
The goal is to remove adoption friction so **electrical systems can be observed, compared, audited, and insured using the same symbolic mathematical language everywhere** — across vendors, across borders, and across time — enabling transparent, reproducible measurement for all humanity.

TABLE OF CONTENTS

0. Purpose 1

0.1 Key idea 1

0.2 Lineage (zero-centric) 2

0.3 Scope, Non-Goals, and Assumptions (Electrical) 2

0.4 One-Minute Math (ASCII, copy-ready) 6

0.5 Openness & Attribution (normative) 9

SECTION 1 — Manifest (Normative) — Introduction 14

1.1 Normative Schema (compact, copy-ready) 14

1.2 Minimal Examples (copy-ready) 18

1.3 Full Manifest — Plant + PQ Dials (copy-ready) 20

1.4 Result Envelopes — S1 / S2 / S3 (copy-ready) 22

1.5 Validation & Conformance (copy-ready) 23

1.6 Minimal Examples (copy-ready) 26

1.7 Authoring Guardrails (normative recap) 28

1.8 Minimal Manifest — SSMEQ-Lite (S1, copy-ready) 30

1.9 Full Manifest — SSMEQ (S3, fleet & ops, copy-ready) 31

1.10 Authoring Guardrails — Normative Recap (copy-ready) 34

1.11 Minimal Manifest — SSMEQ-Lite (YAML, copy-ready) 36

1.12 Result Envelope — SSMEQ-Lite (JSON, copy-ready) 37

1.13 Full Manifest — Plant + PQ Dials (YAML, copy-ready) 37

1.14 Authoring Guardrails (Normative Recap) 39

SECTION 2 — Encoding & Lenses (Normative) 39
2.1 — Lens choices (log, linear, hybrid, asinh): why and when 41
2.2 Anchor rules (*_ref, scales) and zero-centric contrasts (e_*) 42
2.3 Safe domains and guards (positivity, AC/DC specifics) 46
2.4 — Copy-ready formulas (ASCII) for e_V, e_I, e_f, e_P[, e_Q, e_S, e_pf] 49
2.5 Mini sanity set (2–3 numeric quick checks) 51
2.6 — Anchor Profiles for AC 50/60 Hz and DC 53
2.6.1 — LV AC 50 Hz Plant (Typical Anchors, Copy-Ready) 54
2.6.2 — LV AC 60 Hz Plant (Typical Anchors, Copy-Ready) 54
2.6.3 — HV Transmission Node (Anchors for Substation / Grid-Level) 55
2.6.4 — DC Solar / Battery Strings (Anchors for SSMEQ on DC) 56
2.6.5 — Anchor Selection & Adjustment Checklist (Copy-Ready) 57
SECTION 3 — Bounded Dials & Memory (Operational) 57
3.1 — Clamp function and invariants (eps_a) 58
3.2 — Proximity dials (upper-limit approach) and optional alignments 59
3.3 — Bounded PQ dials (a_THD, a_unbV, a_unbI) 61
3.4 Single shared memory dial M (definition, rho guidance) 63
3.5 — Copy-ready snippets (S1 / S2 / S3 options) 66
SECTION 4 — Power Identity Residuals (Consistency) 68
4.1 Core power-identity residual r_P (with pf_min guard) 69
4.2 Interpreting r_P (bias patterns and quick triage) 72
4.3 Copy-ready residual block (r_P) + two worked examples 74
4.4 — Residual Sensitivity Benchmarks (Noise & CT/PT Scenarios) 77
4.4.1 — Benchmark 1: Baseline (No Error, Small Noise) 78
4.4.2 — Benchmark 2: CT Ratio Error (+2 % on Current) 78
4.4.3 — Benchmark 3: Sign Flip on One Channel (Serious Wiring Error) 79
4.4.4 — Benchmark 4: PF Near Zero (Why We Use pf_min / pf_floor) 80
4.4.5 — Residual Benchmark Table (Copy-Ready Summary) 81
SECTION 5 — Hysteresis & Chatter Control 82
5.1 Enter/clear thresholds; memory-gated forms 83
5.2 Parameter hints (th_enter/clear, m_enter/clear, t_hold, t_refractory) 86
5.3 — Mini example: chatter reduction near limits 89
SECTION 6 — Pooling (Fleet/Substation) — Rapidity 90
6.1 — Order-invariance (batch == stream) 90

6.2 — Weights (equal vs declared) and eps_w 92
6.3 — Asset-Weighted Pooling Across Cabinets / Feeders (Normative) 94
6.4 — Power-Quality Dial Set (Alignment-Aware Indices, Normative) 95
6.5 — Dispatch Coupling (Bands → Actions, Normative) 97
6.6 — Commissioning Checklist for Pooling (Must-Do, Copy-Ready) 100
6.7 Conformance Tests (Portable Recipes, Copy-Ready) 102
6.8 — Worked Examples (Cabinet + Feeder, Copy-Ready) 106
7.0 — Plant Health & Fatigue Dials (Introduction) 109
7.1 Breaker Fatigue Dial (Normative, Copy-Ready) 110
7.2 Cabinet Stress Composite (Normative, Copy-Ready) 114
7.3 Drift Memory (Soft Hysteresis for Stability, Normative) 118
8.0 — Dispatch, Bands, and Routing (Introduction, Normative Scope) 121
8.1 — Global Band Table (ASCII, Normative, Copy-Ready) 123
8.2 Environment Gate g_t (Optional, Safe-by-Default, Normative) 124
8.3 — Stamp Semantics (Normative, Copy-Ready) 129
9.0 — Commissioning, Validity, and Safety 131
9.1 Validity Windows (Electrical, Normative, Copy-Ready) 133
9.2 — Site Profiles & Anchor Governance (Normative, Copy-Ready) 136
9.2.1 — What Is a Site Profile? 137
9.2.2 — Site Profile Manifest (Copy-Ready Skeleton) 137
9.2.3 — Governance: How Anchors Change (Version Bump Only) 138
9.2.4 — Runtime Application (Copy-Ready Pseudo-Code) 139
9.2.5 — Governance Checklist (Copy-Ready) 140
9.2.6 — Operator & Auditor Guidance 140
9.3 Day-0 to Day-7 Rollout (Practical Playbook, Copy-Ready) 141
10.0 — Interop Bridges (SSM • SSMS • SSMDE • SSMT) 144
10.1 — Math Backbone (SSM): Collapse, Pooling, and Guarantees 146
10.2 — Operator Canon (SSMS) — Lift Rules (Normative, Copy-Ready) 148
10.3 — Data Shape (SSMDE): Truth That Travels (Normative, Copy-Ready) 151
10.4 — Thermal Crossover (SSMT): Cabinet Survivability & Electrical Risk (Normative, Copy-Ready) 155
10.5 — Electro-Thermal-Chemical Crossover (SSMT + SSM-Chem, Normative, Copy-Ready) 157
10.5.1 — Inputs (Layered Dials) 157
10.5.2 — Layered Fusion Structure 158

10.5.3 — Final Crossover Dial 159
10.5.4 — Worked ASCII Example (Illustrative) 159
10.5.5 — Normative Rules (Must Hold) 160
10.5.6 — Manifest Keys (Copy-Ready) 161
11.0 — UI & Dashboards (General Introduction) 162
11.1 — Operator View (Core Dials and Field Dashboard) 163
11.2 — Investigator View (Audit, Provenance, and Traceability) 166
11.3 — Commissioning View (Verification, Shuffle, and Certification) 169
11.4 — Minimal UI API (Portable ASCII Schema) 173
12.0 — Security, Versions, and Reproducibility (General Introduction) 176
12.1 — Policy Locks (Immutable Configuration and Trust Core) 178
12.2 — Stamp Chain (Chronological Integrity and Proof of Continuity) 180
12.3 — Repro Pack (Reproducibility and Independent Verification Toolkit) 182
12.4 — Independence Note (Observation-Only Principle) 186
12.5 — Data Validity Posture (Explicit Semantics and Suppression Rules) 188
13.0 — Reference Recipes (Portable, Vendor-Neutral Implementation Library) 190
13.1 — Lens Library (Electrical, Copy-Ready) 191
13.1.1 Common Guards and Notation (Copy-Ready) 192
13.1.2 Voltage and Current Magnitude Lenses (V, I) 192
13.1.3 Power Factor Lens (PF) 193
13.1.4 Voltage THD Lens (THDV) 193
13.1.5 Voltage Unbalance Lens (unbV) 194
13.1.6 Thermal Lens (Temperature T) 195
13.1.7 Composite Stress Lens (a_stress from multiple contrasts) 195
13.1.8 Minimal Lens Table (Profile Snippet) 196
13.2 — Pooling Recipes (Uniform, Weighted, and Stream-Compatible Fusion) 196
13.3 — Band Tables (A^+…D) and Environmental Gating (Normative, Copy-Ready) 200
13.4 — CSV / JSON Schemas (Minimal, Portable Data Structures) 204
13.5 — Harness Outline (Deterministic Order-Invariance Verifier) 207
13.6 — Residual Test Pack (r_P Focused, Copy-Ready) 210
14.0 — Annex (Concise, High-Value Reference Section) 215
14.1 — Symbol & Parameter Table (Electrical, Copy-Ready) 218
14.2 — Startup Defaults & Cold-Start Rules (Copy-Ready) 221
15.0 — Unified Electrical Language for a Shared Grid 224

SECTION 1 — Manifest (Normative) — Introduction

Purpose. Define a **single, machine-checkable contract** that governs how raw electrical readings become **portable symbols** (`e_*`) and **bounded dials** (`a_*`, `M`, `r_P`). The **manifest** fixes all lenses, anchors, guards, and emission keys up front; **every record MUST carry `manifest_id`** that resolves to this document. **No silent mid-run changes.**

Why a manifest.

- **Reproducibility.** Anyone can recompute `e_V := ln(V/V_ref)`, `e_I := ln(I/I_ref)`, `e_P := asinh(P/P_scale)` and verify results byte-for-byte.
- **Portability.** Policies live in symbol space; only **anchors** change across sites/vendors.
- **Determinism.** Bounded dials are **clamped** then pooled via rapidity; **batch == stream** by construction.
- **Safety.** Declared **sign conventions** (`power_sign`, `reactive_sign`) and **guards** (`pf_min`, `*_valid`) prevent undefined math and chatter.

Scope of this section. Specify the **normative schema** and provide **minimal, copy-ready examples** (S1 Lite and S3 Full). Everything here is **required for conformity** unless marked optional.

Compliance levels.

- **S1 (Lite).** Core contrasts `e_*` + optional `r_P`. No proximity dials or memory.
- **S2 (Ops).** Adds proximity dials (`a_upper_*`) and **memory `M`** for chatter control.
- **S3 (Fleet).** Adds PQ dials, pooling guard `eps_w`, health flags, and policy hooks.

Authoring stance. Keep **one recipe per run**, publish **anchors and clamps**, **emit health** instead of dropping samples, and preserve identity: `phi((m,a)) = m` (original values remain intact in all pipelines).

1.1 Normative Schema (compact, copy-ready)

What this gives you.
A single, minimal layout that devices, gateways, and validators can load **as-is**. It fixes lenses, anchors, guards, and emitted keys **up front**. Fields not used may be omitted entirely (**do not emit null**).

Schema (YAML template)

```
manifest_id: "<stable-opaque-id>"
title: "<human title>"
description: "<short purpose>"
version: "X.Y"
timestamps_utc:
  created: "YYYY-MM-DDThh:mm:ssZ"
  updated: "YYYY-MM-DDThh:mm:ssZ"
owner_contact: "<email-or-url>"

site_profile: "<free-text site/profile tag>"

conventions:
  ts_utc_format: "RFC3339"
  power_sign: "positive_is_import"        # or "positive_is_export"
  reactive_sign: "inductive_positive"     # or "capacitive_positive"
  ac_dc_mode: "AC_RMS"                    # or "DC" | "mixed"
  grid_nominal_hz: 50                     # 50, 60, or declared nominal
  batching: "stream"                      # or "batch" | "either"

channels:
  V_rms:
    lens: "log"                           # or "linear" | "hybrid"
    anchors:
      V_ref: <float>
      dV: <float>        # optional, for linear/hybrid
      tau_V: <float>     # optional, for hybrid switch region
    bounds:
      V_min_valid: <float>   # optional
      V_max_valid: <float>   # optional
    alignment:
      enabled: true
      c_V: <float>           # optional; dial sensitivity
      eps_a: <float>         # shared or channel-specific

  I_rms:
    lens: "log"
    anchors:
      I_ref: <float>
      dI: <float>           # optional
      tau_I: <float>        # optional
    bounds:
      I_min_valid: <float>  # optional
      I_max_valid: <float>  # optional
    alignment:
      enabled: true
      c_I: <float>
      eps_a: <float>

  f:
    lens: "log"
    anchors:
      f_ref: <float>        # nominal frequency, e.g. 50 or 60
      df: <float>           # optional linear window
    bounds:
      f_min_valid: <float>  # optional
      f_max_valid: <float>  # optional
    alignment:
      enabled: true
      c_f: <float>
```

```
      eps_a: <float>

  P:
    lens: "asinh"
    anchors:
      P_scale: <float>      # sets sensitivity around 0
    bounds:
      P_max: <float>        # optional nameplate or policy cap
    alignment:
      enabled: true
      c_P: <float>
      eps_a: <float>

  Q?:                       # optional reactive channel
    lens: "asinh"
    anchors:
      Q_scale: <float>
    alignment:
      enabled: false        # may be turned on for Q dials later

  S?:                       # optional apparent power
    lens: "log"
    anchors:
      S_ref: <float>
    alignment:
      enabled: false

  R?:                       # optional resistance
    lens: "log"
    anchors:
      R_ref: <float>
    alignment:
      enabled: false

  Z_rms?:                   # optional impedance magnitude
    lens: "log"
    anchors:
      Z_ref: <float>
    alignment:
      enabled: false

bounded_metrics?:           # already bounded inputs (magnitudes)
  pf?:                      # pf is a magnitude in [0,1]
    clamp:
      eps_a: 1e-6
    pf_ref: <float>         # e.g. 1.0 for unity or 0.95 for target
  THD?:
    ref:
      THD_ref: <float>
    dial:
      c_THD: <float>
  unbalance?:
    ref:
      UnbV_ref: <float>
      UnbI_ref: <float>
    dial:
      c_unbV: <float>
      c_unbI: <float>

residuals?:
  r_P?:
```

```
    enabled: true
    tol: 0.15              # tolerance band in e-space
    dwell: 5               # consecutive samples needed
    pf_min: 0.05           # guard for ln(pf / pf_ref)
    notes: "compute only when pf >= pf_min"

hysteresis?:
  enabled: true
  rho: 0.7                 # memory factor in (0,1)
  k_side: 2.0
  M_init: 0.0              # cold-start value for M
  gates?:
    V?:
      V_gate: <float>
      dV_gate: <float>
      th_enter: 0.60
      th_clear: 0.40
      m_enter: 0.80
      m_clear: 0.60
    I?:
      I_gate: <float>
      dI_gate: <float>
      th_enter: 0.60
      th_clear: 0.40
      m_enter: 0.80
      m_clear: 0.60
    P?:
      P_gate: <float>
      dP_gate: <float>
      th_enter: 0.70
      th_clear: 0.50
      m_enter: 0.80
      m_clear: 0.60

pooling?:
  method: "rapidity"
  eps_w: 1e-9
  weights: "equal"         # or "declared" | "by_magnitude"
  magnitude_exponent_gamma: 1.0   # used if weights == "by_magnitude"
  declared?:               # used if weights == "declared"
    <name>: <float>

privacy?:
  mode: "none"             # or "offset" | "masked"
  notes?: "<invertibility disclosure if offsets>"

provenance?:
  stamp?: "optional"       # e.g. SSM-Clock-Stamp
  hashing: "SHA-256 of envelope"
  continuity?: "checkpoint.txt HEAD=... pattern"

health?:
  jitter_max_ms: 200
  range_policy: "flag"     # or "drop" | "flag_and_drop"

conformance?:
  tests:
    - "unit_invariance"
    - "batch_equals_stream"
    - "residual_alarm"
```

```
emit:
  e_V: true
  e_I: true
  e_f: true
  e_P: true
  e_Q?: true
  e_S?: true

  a_pf?: true
  a_V?: true
  a_I?: true
  a_f?: true
  a_P?: true
  a_Q?: true
  a_THD?: true
  a_unbV?: true
  a_unbI?: true

  a_upper_I?: true
  a_upper_P?: true

  M?: true
  r_P?: true
```

Notes (normative).

- **Omit unused optional blocks/keys; do not emit null.**
- **Clamps.** Use `eps_a` for all bounded dials and `eps_w` for the pooling denominator.
- **Identity.** Real values remain untouched conceptually: `phi((m,a)) = m`.
- **Guards.** Compute `r_P` only when `pf >= pf_min` (with `pf` treated as a magnitude in `[0,1]`).
- **Determinism.** Rapidity-space pooling guarantees **batch == stream** for any ordering.
- **Anchors and lenses.** Each channel gets **exactly one lens and anchor set** per manifest; changes require a new manifest, not a silent edit.
- **Cold start.** If `M_init` is omitted, assume `M_init = 0.0` at startup or replay reset.

1.2 Minimal Examples (copy-ready)

Why this matters. A tiny, load-and-run starting point for **S1 (Lite)** deployments. Replace anchors per site; keep lenses fixed for the run. Omit any field you don't use (do **not** emit `null`).

Minimal manifest — SSMEQ-Lite (YAML)

```
manifest_id: SSMEQ.DER.PLANT01.S1
title: "DER Plant — SSMEQ Lite"
description: "Portable electrical symbols for plant telemetry"
version: "1.0"
timestamps_utc: { created: "2025-11-10T00:00:00Z", updated: "2025-11-
10T00:00:00Z" }
owner_contact: "ops@example.org"
```

```yaml
conventions:
  ts_utc_format: "RFC3339"
  power_sign: "positive_is_import"
  reactive_sign: "inductive_positive"
  ac_dc_mode: "AC_RMS"
  batching: "either"

channels:
  V_rms:
    lens: "log"
    anchors: { V_ref: 230.0 }
    bounds: { V_min_valid: 180.0, V_max_valid: 265.0 }
    alignment: { enabled: true, c_V: 3.0, eps_a: 1e-6 }
  I_rms:
    lens: "log"
    anchors: { I_ref: 100.0 }
    bounds: { I_max_valid: 1000.0 }
    alignment: { enabled: true, c_I: 3.0, eps_a: 1e-6 }
  f:
    lens: "log"
    anchors: { f_ref: 50.0 }
    bounds: { f_min_valid: 45.0, f_max_valid: 55.0 }
    alignment: { enabled: true, c_f: 4.0, eps_a: 1e-6 }
  P:
    lens: "asinh"
    anchors: { P_scale: 1000.0 }
    bounds: { P_max: 5000.0 }
    alignment: { enabled: true, c_P: 2.5, eps_a: 1e-6 }

bounded_metrics:
  pf:
    clamp: { eps_a: 1e-6 }
    pf_ref: 1.0

residuals:
  r_P: { enabled: true, tol: 0.15, dwell: 5, pf_min: 0.05 }

pooling:
  method: "rapidity"
  eps_w: 1e-9
  weights: "equal"

health:
  jitter_max_ms: 200
  range_policy: "flag"

conformance:
  tests: [ "unit_invariance", "batch_equals_stream", "residual_alarm" ]

emit:
  e_V: true
  e_I: true
  e_f: true
  e_P: true
  a_pf: true
  residual_rP: true
```

Result envelope — SSMEQ-Lite (JSON)

```
{
  "ts_utc": "2025-11-10T05:00:01Z",
  "e_V": -0.1031,
  "e_I": 0.0202,
  "e_f": -0.0020,
  "e_P": 0.5473,
  "a_pf": 0.9700,
  "r_P": 0.012,
  "manifest_id": "SSMEQ.DER.PLANT01.S1",
  "health": { "range_ok": true, "sensor_ok": true, "drift": false }
}
```

Implementation notes (tiny).

- **Stability:** keep lenses (`log`, `asinh`) fixed across the run; only adjust anchors (`*_ref`, `P_scale`) between sites.
- **Guards:** compute `r_P` only if `abs(pf) >= pf_min`; clamp all bounded dials with `eps_a` before any pooling.
- **Determinism:** use rapidity pooling with `eps_w` to guarantee **batch == stream**.

1.3 Full Manifest — Plant + PQ Dials (copy-ready)

Why this matters. A production-ready template for **S3 (fleet & ops)**: adds proximity dials, memory, PQ dials, pooling, and health flags while staying portable. Keep lenses fixed for the run; tune anchors and thresholds per site.

Full manifest — SSMEQ (YAML)

```
manifest_id: SSMEQ.PLANT01.FULL
title: "Plant — SSMEQ Full with PQ"
description: "Symbols for plant ops + PQ monitoring"
version: "1.0"

timestamps_utc: { created: "2025-11-10T00:00:00Z", updated: "2025-11-
10T00:00:00Z" }
owner_contact: "ops@example.org"

conventions:
  ts_utc_format: "RFC3339"
  power_sign: "positive_is_import"
  reactive_sign: "inductive_positive"
  ac_dc_mode: "AC_RMS"
  batching: "either"

channels:
  V_rms:
    lens: "hybrid"
    anchors: { V_ref: 400.0, dV: 10.0, tau_V: 8.0 }
    bounds: { V_min_valid: 320.0, V_max_valid: 460.0 }
    alignment: { enabled: true, c_V: 3.0, eps_a: 1e-6 }
  I_rms:
    lens: "log"
```

```
    anchors: { I_ref: 200.0 }
    bounds: { I_max_valid: 2000.0 }
    alignment: { enabled: true, c_I: 3.0, eps_a: 1e-6 }
  f:
    lens: "log"
    anchors: { f_ref: 50.0 }
    bounds: { f_min_valid: 47.0, f_max_valid: 53.0 }
    alignment: { enabled: true, c_f: 4.0, eps_a: 1e-6 }
  P:
    lens: "asinh"
    anchors: { P_scale: 5000.0 }
    bounds: { P_max: 20000.0 }
    alignment: { enabled: true, c_P: 2.5, eps_a: 1e-6 }
  Q:
    lens: "asinh"
    anchors: { Q_scale: 5000.0 }
    alignment: { enabled: false }
  S:
    lens: "log"
    anchors: { S_ref: 10000.0 }
    alignment: { enabled: false }

bounded_metrics:
  pf   : { clamp: { eps_a: 1e-6 }, pf_ref: 1.0 }
  THD  : { ref: { THD_ref: 5.0 }, dial: { c_THD: 2.0 } }
  unbalance:
    ref : { UnbV_ref: 1.0, UnbI_ref: 1.0 }    # percent -> per-unit
    dial: { c_unbV: 2.0, c_unbI: 2.0 }

residuals:
  r_P: { enabled: true, tol: 0.12, dwell: 10, pf_min: 0.05 }

hysteresis:
  enabled: true
  rho: 0.75
  k_side: 2.5
  gates:
    V: { V_gate: 390.0, dV_gate: 8.0, th_enter: 0.65, th_clear: 0.45,
m_enter: 0.85, m_clear: 0.65 }
    I: { I_gate: 0.90 * I_ref, dI_gate: 0.05 * I_ref, th_enter: 0.60,
th_clear: 0.40, m_enter: 0.80, m_clear: 0.60 }

pooling:
  method: "rapidity"
  eps_w: 1e-9
  weights: "declared"
  declared:
    feeder_A: 1.0
    feeder_B: 2.0
    feeder_C: 1.5

privacy: { mode: "none", notes: "" }
provenance: { stamp: "optional", hashing: "SHA-256 of envelope",
continuity: "optional" }

health:
  jitter_max_ms: 150
  range_policy: "flag_and_drop"
```

```
conformance:
  tests: [ "unit_invariance", "batch_equals_stream", "residual_alarm",
"hysteresis_clear" ]

emit:
  e_V: true
  e_I: true
  e_f: true
  e_P: true
  a_pf: true
  a_V: true
  a_I: true
  a_f: true
  a_P: true
  a_THD: true
  a_unbV: true
  a_unbI: true
  a_upper_I: true
  a_upper_P: true
  M: true
  residual_rP: true
```

Tiny usage note. Keep **lenses fixed**; tune only anchors (`*_ref`, `P_scale`, gates) across sites. Clamp all bounded dials with `eps_a` before pooling; compute `r_P` only if `abs(pf) >= pf_min`.

1.4 Result Envelopes — S1 / S2 / S3 (copy-ready)

Why this matters. Streams must be small, deterministic, and self-verifiable. Each envelope **MUST** include `ts_utc` and `manifest_id`. Omit fields that are disabled in the manifest (do not emit `null`).

S1 — Lite (contrasts + optional residual)

```
{
  "ts_utc": "2025-11-10T05:00:01Z",
  "e_V": -0.1031,
  "e_I": 0.0202,
  "e_f": -0.0020,
  "e_P": 0.5473,
  "r_P": 0.012,                       // present only if abs(pf) >= pf_min
  "manifest_id": "SSMEQ.DER.PLANT01.S1",
  "health": { "range_ok": true, "sensor_ok": true, "drift": false }
}
```

S2 — Ops Dials (proximity + memory + residual)

```
{
  "ts_utc": "2025-11-10T05:00:01Z",
  "e_V": -0.1031,
  "e_I": 0.0202,
  "e_f": -0.0020,
  "e_P": 0.5473,
  "a_upper_I": 0.58,
```

```
  "a_upper_P": 0.71,
  "M": 0.64,
  "r_P": 0.012,                        // optional by guard
  "manifest_id": "SSMEQ.DER.PLANT01.S2",
  "health": { "range_ok": true, "sensor_ok": true, "drift": false }
}
```

S3 — Fleet & Dashboards (adds alignments and PQ dials)

```
{
  "ts_utc": "2025-11-10T05:00:01Z",
  "e_V": -0.1031,
  "e_I": 0.0202,
  "e_f": -0.0020,
  "e_P": 0.5473,

  "a_pf": 0.9700,
  "a_V": -0.29,
  "a_I": 0.06,
  "a_f": -0.01,
  "a_P": 0.47,

  "a_THD": 0.12,
  "a_unbV": 0.08,
  "a_unbI": 0.05,

  "a_upper_I": 0.58,
  "a_upper_P": 0.71,
  "M": 0.64,

  "r_P": 0.012,

  "manifest_id": "SSMEQ.PLANT01.FULL",
  "health": { "range_ok": true, "sensor_ok": true, "drift": false }
}
```

Emission rules (normative).

- Always emit `ts_utc` (UTC, RFC3339) and `manifest_id`.
- Emit only fields whose `emit.*` is `true` in the manifest; **omit** others (no `null`).
- Clamp all bounded dials before emission: `a := sign(a) * min(abs(a), 1 - eps_a)`.
- Compute `r_P` only if `abs(pf) >= pf_min`; else omit `r_P`.
- `health` is required; never crash on range violations—flag via `health`.

1.5 Validation & Conformance (copy-ready)

Why this matters. A manifest is only useful if anyone can mechanically verify that emitted streams follow it. This subsection defines small, deterministic checks you can run in CI or on incoming streams before policy evaluation.

A. Schema conformance (must)

- Validate presence and types of required top-level fields:
  - `manifest_id:string`, `title:string`, `version:string`
  - `timestamps_utc.created/updated: RFC3339`
  - `conventions.ts_utc_format == "RFC3339"`
  - `channels` map present; each declared channel has `lens` and required anchors for that lens.
- For each declared bounded dial: require `eps_a in (0,1)` and clamp rule stated.

B. Lens–anchor guards (must)

- If `lens == "log"` then the referenced anchor (e.g., `V_ref`, `I_ref`, `f_ref`) MUST be > 0.
- If `lens == "asinh"` then the scale (e.g., `P_scale`) MUST be > 0.
- If `lens == "linear"` and offset/gain are used, they MUST be finite and declared.

C. Emission contract (must)

- Every envelope must include:
  - `ts_utc` (UTC, RFC3339, no timezone offsets other than Z)
  - `manifest_id` equal to this manifest
  - Only fields whose `emit.* == true` in this manifest; do **not** emit nulls.
- `health` object present; range violations set flags but do not drop samples.

D. Residual guard (must)

- Compute `r_P` only if `abs(pf) >= pf_min` and `pf_ref in (0,1]`. Otherwise omit `r_P`.

E. Determinism spot-checks (should)

- Recompute `e_*` from raw units and declared anchors; assert absolute error <= 1e-12.
- Recompute bounded dials with declared `c_*` and `eps_a`; assert clamp applied:
  - `abs(a) <= 1 - eps_a` for all emitted bounded dials.

F. Batch == Stream (should)

- For any pooled bounded dial `a_*`, verify associativity via rapidity:
  - `a_pool_batch := tanh( (sum_i w_i*atanh(clamp_a(a_i,eps_a))) / max(sum_i w_i, eps_w) )`
  - `a_pool_stream := fold_left(...) -> tanh(U / max(W, eps_w))`
  - Assert `abs(a_pool_batch - a_pool_stream) <= 1e-12`.

G. Sign conventions (must)

- If `power_sign == "positive_is_import"`, check that exported power produces `e_P` sign consistent with asinh odd symmetry and that policy rules reference the same convention.

- If `reactive_sign` is declared, ensure pf and any Q-derived dials use the same sign basis.

H. Versioning (should)

- New manifest fields must have safe defaults and be optional; consumers ignore unknown keys.
- Changing lenses or anchors mid-run is prohibited; start a new `manifest_id` for recipe changes.

CI starter — JSON examples

Minimal schema test vector (expected: PASS)

```
{
  "manifest_id": "SSMEQ.DER.PLANT01.S1",
  "title": "DER Plant — SSMEQ Lite",
  "version": "1.0",
  "timestamps_utc": { "created": "2025-11-10T00:00:00Z", "updated": "2025-
11-10T00:00:00Z" },
  "conventions": { "ts_utc_format": "RFC3339", "power_sign":
"positive_is_import", "ac_dc_mode": "AC_RMS" },
  "channels": { "V_rms": { "lens": "log", "anchors": { "V_ref": 230.0 } } }
}
```

Illegal log lens (expected: FAIL — V_ref not positive)

```
{
  "channels": { "V_rms": { "lens": "log", "anchors": { "V_ref": 0.0 } } }
}
```

Runtime gate — envelope validator (pseudocode)

```
function validate_envelope(env, manifest):
  require env.ts_utc matches RFC3339Z
  require env.manifest_id == manifest.manifest_id

  // Required emissions
  for k in manifest.emit where manifest.emit[k] == true:
      require k in env

  // No nulls for disabled fields
  for k in env:
      if k in manifest.emit and manifest.emit[k] == false:
          fail("unexpected field: " + k)

  // Health present
  require type(env.health) == object

  // Residual guard
  if "r_P" in env:
      require abs(env.a_pf or env.pf or 1) >= manifest.pf_min
```

Operator outcome. If A–D pass, the stream is **manifest-conformant**. If E–F pass, the stream is **deterministic-portable**. If G–H pass, the stream is **policy-ready** across sites and vendors.

1.6 Minimal Examples (copy-ready)

Why this matters. Concrete, copy-ready snippets make adoption fast and testable. Start with the Lite manifest (S1) to ship quickly; upgrade to the Full manifest (S3) when you need proximity dials, PQ dials, memory, and pooling.

A) Minimal manifest — SSMEQ-Lite (YAML)

```
manifest_id: SSMEQ.DER.PLANT01.S1
title: "DER Plant — SSMEQ Lite"
description: "Portable electrical symbols for plant telemetry"
version: "1.0"
timestamps_utc: { created: "2025-11-10T00:00:00Z", updated: "2025-11-
10T00:00:00Z" }
owner_contact: "ops@example.org"

conventions:
  ts_utc_format: "RFC3339"
  power_sign: "positive_is_import"
  reactive_sign: "inductive_positive"
  ac_dc_mode: "AC_RMS"
  batching: "either"

channels:
  V_rms:
    lens: "log"
    anchors: { V_ref: 230.0 }
    bounds: { V_min_valid: 180.0, V_max_valid: 265.0 }
    alignment: { enabled: true, c_V: 3.0, eps_a: 1e-6 }
  I_rms:
    lens: "log"
    anchors: { I_ref: 100.0 }
    bounds: { I_max_valid: 1000.0 }
    alignment: { enabled: true, c_I: 3.0, eps_a: 1e-6 }
  f:
    lens: "log"
    anchors: { f_ref: 50.0 }
    bounds: { f_min_valid: 45.0, f_max_valid: 55.0 }
    alignment: { enabled: true, c_f: 4.0, eps_a: 1e-6 }
  P:
    lens: "asinh"
    anchors: { P_scale: 1000.0 }
    bounds: { P_max: 5000.0 }
    alignment: { enabled: true, c_P: 2.5, eps_a: 1e-6 }

bounded_metrics:
  pf:
    clamp: { eps_a: 1e-6 }
    pf_ref: 1.0

residuals:
  r_P: { enabled: true, tol: 0.15, dwell: 5, pf_min: 0.05 }

hysteresis:
  enabled: true
  rho: 0.7
```

```
  k_side: 2.0
  gates:
    V: { V_gate: 225.0, dV_gate: 5.0, th_enter: 0.60, th_clear: 0.40,
m_enter: 0.80, m_clear: 0.60 }

pooling:
  method: "rapidity"
  eps_w: 1e-9
  weights: "equal"

privacy: { mode: "none", notes: "" }
provenance: { stamp: "optional", hashing: "SHA-256 of envelope",
continuity: "optional" }

health:
  jitter_max_ms: 200
  range_policy: "flag"

conformance:
  tests: [ "unit_invariance", "batch_equals_stream", "residual_alarm" ]
```

Result envelope — Lite (JSON)

```
{
  "ts_utc": "2025-11-10T05:00:01Z",
  "e_V": -0.1031,
  "e_I": 0.0202,
  "e_f": -0.0020,
  "e_P": 0.5473,
  "a_pf": 0.9700,
  "r_P": 0.012,
  "manifest_id": "SSMEQ.DER.PLANT01.S1",
  "health": true
}
```

B) Full manifest — plant + PQ dials (YAML)

```
manifest_id: SSMEQ.PLANT01.FULL
title: "Plant — SSMEQ Full with PQ"
description: "Symbols for plant ops + PQ monitoring"
version: "1.0"

conventions:
  ts_utc_format: "RFC3339"
  power_sign: "positive_is_import"
  reactive_sign: "inductive_positive"
  ac_dc_mode: "AC_RMS"
  batching: "either"

channels:
  V_rms: { lens: "hybrid", anchors: { V_ref: 400.0, dV: 10.0, tau_V: 8.0 },
bounds: { V_min_valid: 320.0, V_max_valid: 460.0 }, alignment: { enabled:
true, c_V: 3.0, eps_a: 1e-6 } }
  I_rms: { lens: "log", anchors: { I_ref: 200.0 }, bounds: { I_max_valid:
2000.0 }, alignment: { enabled: true, c_I: 3.0, eps_a: 1e-6 } }
  f    : { lens: "log", anchors: { f_ref: 50.0 }, bounds: { f_min_valid:
47.0, f_max_valid: 53.0 }, alignment: { enabled: true, c_f: 4.0, eps_a: 1e-
6 } }
  P    : { lens: "asinh", anchors: { P_scale: 5000.0 }, bounds: { P_max:
20000.0 }, alignment: { enabled: true, c_P: 2.5, eps_a: 1e-6 } }
```

```
  Q    : { lens: "asinh", anchors: { Q_scale: 5000.0 }, alignment: {
enabled: false } }
  S    : { lens: "log", anchors: { S_ref: 10000.0 }, alignment: { enabled:
false } }

bounded_metrics:
  pf  : { clamp: { eps_a: 1e-6 }, pf_ref: 1.0 }
  THD : { ref: { THD_ref: 5.0 }, dial: { c_THD: 2.0 } }
  unbalance:
    ref : { UnbV_ref: 1.0, UnbI_ref: 1.0 }
    dial: { c_unbV: 2.0, c_unbI: 2.0 }

residuals:
  r_P: { enabled: true, tol: 0.12, dwell: 10, pf_min: 0.05 }

hysteresis:
  enabled: true
  rho: 0.75
  k_side: 2.5
  gates:
    V: { V_gate: 390.0, dV_gate: 8.0, th_enter: 0.65, th_clear: 0.45,
m_enter: 0.85, m_clear: 0.65 }
    I: { I_gate: 0.90 * I_ref, dI_gate: 0.05 * I_ref, th_enter: 0.60,
th_clear: 0.40, m_enter: 0.80, m_clear: 0.60 }

pooling:
  method: "rapidity"
  eps_w: 1e-9
  weights: "declared"
  declared:
    feeder_A: 1.0
    feeder_B: 2.0
    feeder_C: 1.5

privacy: { mode: "none" }
provenance: { stamp: "optional", hashing: "SHA-256 of envelope",
continuity: "optional" }

health:
  jitter_max_ms: 150
  range_policy: "flag_and_drop"

conformance:
  tests: [ "unit_invariance", "batch_equals_stream", "residual_alarm",
"hysteresis_clear" ]
```

Operator tip. Ship S1 to get quick wins (unitless contrasts + residual); move to S3 when you need chatter-safe operations (proximity + memory), PQ dials, and fleet pooling.

1.7 Authoring Guardrails (normative recap)

Why this matters. A manifest is a contract. These guardrails prevent hidden retunes, unit drift, or chatter regressions and ensure any team can **replay byte-for-byte** and reach the same symbols and decisions.

Non-negotiables (MUST):

- **One recipe per run.** Do not change `lens`, `anchors`, or proximity **gates** mid-stream.
- **Declare signs up front.** `power_sign`, `reactive_sign`, `pf_ref` must be explicit and stable.
- **Publish anchors & clamps.** Include `V_ref`, `I_ref`, `f_ref`, `P_scale`, `eps_a`, and any dial sharpness `c_*`.
- **Clamp before pooling.** Use `a := sign(a)*min(abs(a), 1-eps_a)` before any rapidity pooling.
- **Residual guard.** Compute `r_P` only if `abs(pf) >= pf_min`; otherwise omit `r_P`.
- **Determinism.** Ensure `batch == stream` using rapidity pooling with `eps_w`.
- **Health over drop.** Never crash or silently discard; set `health` flags and keep emitting.
- **Identity.** Every record carries `manifest_id`; operators can reconstruct `e_*` from raw with the declared anchors.

Strong SHOULDs:

- **Bound ranges for stability.** Declare `*_valid` to clip inputs for computation stability (policy thresholds remain symbolic).
- **Hysteresis for chatter.** Pair proximity dials with memory `M` (e.g., `rho` in (0,1)) and distinct enter/clear thresholds.
- **Version lightly.** Increment **manifest doc** `version` only for schema/knob changes; physics stays in channels.
- **Keep optional fields truly optional.** Omit (do not emit nulls) when a field is disabled in `emit`.

Copy-ready lint checklist (paste into CI):

- **G1.** `manifest_id` present and stable across the run.
- **G2.** `conventions.{ts_utc_format,power_sign,reactive_sign,ac_dc_mode}` present.
- **G3.** For each active channel: `{lens, anchors.*, alignment.enabled?}` declared.
- **G4.** `eps_a` and (if pooling) `eps_w` declared; `eps_a in (0,1)`, `eps_w > 0`.
- **G5.** If residuals enabled: `pf_ref in (0,1]`, `pf_min > 0`.
- **G6.** If proximity used: `{I_gate,dI_gate}` and/or `{P_gate,dP_gate}` > 0.
- **G7.** `health.range_policy` set; `*_valid` ranges consistent (min < max).
- **G8.** `conformance.tests` lists `batch_equals_stream` when pooling is used.
- **G9.** Outputs match `emit.*` booleans; disabled fields are omitted.
- **G10.** `timestamps_utc.{created,updated}` exist; `updated >= created`.

Hashing & continuity (recommended):

- Envelope hash: `SHA-256(envelope_bytes)` for each record.
- Optional checkpoint: append rolling `HEAD=<sha256_of_envelopes.jsonl>` to `checkpoint.txt` after each batch; include `continuity: "checkpoint.txt HEAD=..."` in `provenance`.

Operator tip. When anchors change across sites (e.g., `V_ref` 230 ↔ 120), **rules do not change**—only anchors do. That's the portability promise of symbol space.

1.8 Minimal Manifest — SSMEQ-Lite (S1, copy-ready)

Purpose. A tiny, deploy-now manifest for analytics and trials. Emits core contrasts `e_*` and (optionally) the identity residual `r_P`. No proximity dials, no pooling—portable and easy to audit.

```
manifest_id: SSMEQ.DER.PLANT01.S1
title: "DER Plant — SSMEQ Lite"
description: "Portable electrical symbols for plant telemetry"
version: "1.0"
timestamps_utc: { created: "2025-11-10T00:00:00Z", updated: "2025-11-
10T00:00:00Z" }
owner_contact: "ops@example.org"

conventions:
  ts_utc_format: "RFC3339"
  power_sign: "positive_is_import"
  reactive_sign: "inductive_positive"
  ac_dc_mode: "AC_RMS"
  batching: "either"

channels:
  V_rms:
    lens: "log"
    anchors: { V_ref: 230.0 }
    bounds: { V_min_valid: 180.0, V_max_valid: 265.0 }
    alignment: { enabled: false }
  I_rms:
    lens: "log"
    anchors: { I_ref: 100.0 }
    bounds: { I_max_valid: 1000.0 }
    alignment: { enabled: false }
  f:
    lens: "log"
    anchors: { f_ref: 50.0 }
    bounds: { f_min_valid: 45.0, f_max_valid: 55.0 }
    alignment: { enabled: false }
  P:
    lens: "asinh"
    anchors: { P_scale: 1000.0 }
    bounds: { P_max: 5000.0 }
    alignment: { enabled: false }

bounded_metrics:
  pf:
    clamp: { eps_a: 1e-6 }
    pf_ref: 1.0

residuals:
  r_P: { enabled: true, tol: 0.15, dwell: 5, pf_min: 0.05 }

pooling:
  method: "rapidity"
```

```
  eps_w: 1e-9
  weights: "equal"

privacy: { mode: "none", notes: "" }
provenance: { stamp: "optional", hashing: "SHA-256 of envelope",
continuity: "optional" }

health:
  jitter_max_ms: 200
  range_policy: "flag"

conformance:
  tests: [ "unit_invariance", "batch_equals_stream", "residual_alarm" ]

emit:
  e_V: true
  e_I: true
  e_f: true
  e_P: true
  a_pf: true
  residual_rP: true
```

S1 result envelope (copy-ready).

```
{
  "ts_utc": "2025-11-10T05:00:01Z",
  "e_V": -0.1031,
  "e_I": 0.0202,
  "e_f": -0.0020,
  "e_P": 0.5473,
  "a_pf": 0.9700,
  "r_P": 0.012,
  "manifest_id": "SSMEQ.DER.PLANT01.S1",
  "health": { "range_ok": true, "sensor_ok": true, "drift": false }
}
```

1.9 Full Manifest — SSMEQ (S3, fleet & ops, copy-ready)

Purpose. A complete, chatter-safe recipe for operations and fleets. Adds proximity dials, bounded memory M, PQ dials, and pooling guards—while keeping anchors, sign conventions, and guards explicit and portable.

```
manifest_id: SSMEQ.PLANT01.S3
title: "Plant — SSMEQ Full (Ops + PQ + Pooling)"
description: "Symbols and dials for plant operations, PQ, and fleet
pooling"
version: "1.0"
timestamps_utc: { created: "2025-11-10T00:00:00Z", updated: "2025-11-
10T00:00:00Z" }
owner_contact: "ops@example.org"

conventions:
  ts_utc_format: "RFC3339"
  power_sign: "positive_is_import"          # or "positive_is_export"
  reactive_sign: "inductive_positive"       # declare if Q is used
  ac_dc_mode: "AC_RMS"                      # "AC_RMS" | "DC"
```

```
  batching: "either"

channels:
  V_rms:
    lens: "hybrid"
    anchors: { V_ref: 400.0, dV: 10.0, tau_V: 8.0 }     # publish only what
you use
    bounds:  { V_min_valid: 320.0, V_max_valid: 460.0 }
    alignment: { enabled: true, c_V: 3.0, eps_a: 1e-6 }
  I_rms:
    lens: "log"
    anchors: { I_ref: 200.0 }
    bounds:  { I_max_valid: 2000.0 }
    alignment: { enabled: true, c_I: 3.0, eps_a: 1e-6 }
  f:
    lens: "log"
    anchors: { f_ref: 50.0 }
    bounds:  { f_min_valid: 47.0, f_max_valid: 53.0 }
    alignment: { enabled: true, c_f: 4.0, eps_a: 1e-6 }
  P:
    lens: "asinh"
    anchors: { P_scale: 5000.0 }
    bounds:  { P_max: 20000.0 }
    alignment: { enabled: true, c_P: 2.5, eps_a: 1e-6 }
  Q:
    lens: "asinh"
    anchors: { Q_scale: 5000.0 }
    alignment: { enabled: false }
  S:
    lens: "log"
    anchors: { S_ref: 10000.0 }
    alignment: { enabled: false }

bounded_metrics:
  pf:
    clamp: { eps_a: 1e-6 }
    pf_ref: 1.0
  THD:
    ref:  { THD_ref: 5.0 }
    dial: { c_THD: 2.0 }
  unbalance:
    ref:  { UnbV_ref: 1.0, UnbI_ref: 1.0 }   # percent -> per-unit policy
must be declared
    dial: { c_unbV: 2.0, c_unbI: 2.0 }

residuals:
  r_P: { enabled: true, tol: 0.12, dwell: 10, pf_min: 0.05 }   # guard
ln(pf/pf_ref)

hysteresis:
  enabled: true
  rho: 0.75                       # memory for M (0<rho<1)
  k_side: 2.5
  gates:
    V: { V_gate: 390.0, dV_gate: 8.0, th_enter: 0.65, th_clear: 0.45,
m_enter: 0.85, m_clear: 0.65 }
    I: { I_gate: 0.90 * I_ref, dI_gate: 0.05 * I_ref, th_enter: 0.60,
th_clear: 0.40, m_enter: 0.80, m_clear: 0.60 }

pooling:
  method: "rapidity"              # atanh/tanh (order-invariant)
```

```
  eps_w: 1e-9
  weights: "declared"
  declared:
    feeder_A: 1.0
    feeder_B: 2.0
    feeder_C: 1.5

privacy: { mode: "none", notes: "" }

provenance:
  stamp: "optional"
  hashing: "SHA-256 of envelope"
  continuity: "optional"          # e.g., checkpoint.txt HEAD=...

health:
  jitter_max_ms: 150
  range_policy: "flag_and_drop"

conformance:
  tests: [ "unit_invariance", "batch_equals_stream", "residual_alarm",
"hysteresis_clear" ]

emit:
  # contrasts
  e_V: true
  e_I: true
  e_f: true
  e_P: true
  # bounded dials
  a_pf: true
  a_V: true
  a_I: true
  a_f: true
  a_P: true
  a_THD: true
  a_unbV: true
  a_unbI: true
  # proximity + memory
  a_upper_I: true
  a_upper_P: true
  M: true
  # consistency
  residual_rP: true
```

S3 result envelope (copy-ready).

```
{
  "ts_utc": "2025-11-10T05:00:01Z",
  "e_V": -0.012,
  "e_I": 0.044,
  "e_f": 0.000,
  "e_P": 0.733,
  "a_pf": 0.96,
  "a_V": -0.04,
  "a_I": 0.13,
  "a_f": 0.00,
  "a_P": 0.51,
  "a_THD": 0.18,
  "a_unbV": 0.06,
  "a_unbI": 0.05,
```

```
  "a_upper_I": 0.42,
  "a_upper_P": 0.38,
  "M": 0.47,
  "r_P": 0.06,
  "manifest_id": "SSMEQ.PLANT01.S3",
  "health": { "range_ok": true, "sensor_ok": true, "drift": false }
}
```

1.10 Authoring Guardrails — Normative Recap (copy-ready)

Purpose. Keep manifests portable, machine-checkable, and replayable. These rules apply to **all** SSMEQ manifests (S1–S3).

Core invariants

- **One lens per channel.** Do **not** change `lens` or `anchors` mid-run.
  Example: `e_V := ln(V/V_ref)` must stay the same recipe for the whole run.
- **Anchors are public.** Publish `*_ref`, `P_scale`, `pf_ref`, `eps_a`, `eps_w`, and any `c_*` you use.
- **Sign conventions are explicit.** Always declare `power_sign` and (if Q used) `reactive_sign`.
- **Bounded before pooling.** Apply `a := sign(a)*min(abs(a), 1 - eps_a)` before any pooling.
- **Residual guard.** Compute `r_P` only if `abs(pf) >= pf_min`; otherwise omit `r_P`.
- **Validity + health.** Declare `*_valid` ranges and policy on violations: `"flag" | "drop" | "flag_and_drop"`. Never crash.
- **Determinism.** Ensure `batch == stream` using rapidity pooling with nonzero `eps_w`.
- **Identity.** Every record MUST carry `manifest_id`. Replay from raw inputs must reproduce emitted symbols bit-for-bit.

Minimal must-haves

- `manifest_id`, `version`, `timestamps_utc {created, updated}`, `owner_contact`
- `conventions.ts_utc_format = "RFC3339"`
- For AC: declare `f_ref` and `f_min_valid/f_max_valid` (or state why omitted)
- For S2/S3: declare proximity gates (`I_gate/dI_gate`, `P_gate/dP_gate`) if `a_upper_*` exists
- Emit booleans (`emit.*`) matching your chosen surface (S1/S2/S3)

Compatibility rules

- New fields start with safe defaults; old consumers must not break.
- Unknown fields must be ignored by consumers.
- Deprecations remain ≥ 12 months; mark clearly in `notes`.
- **No silent retunes:** site-to-site differences use anchors (e.g., `V_ref`), not new lenses.

Change control checklist (pre-publish)

1. `manifest_id` stable and unique?
2. `version` bumped? `timestamps_utc.updated` refreshed?
3. `conventions` complete (time format, sign conventions, mode, batching)?
4. All channels list **exactly** the fields used (no orphan knobs)?
5. `residuals.r_P` has `pf_min`, `tol`, `dwell`?
6. `hysteresis` (if enabled) has `rho`, `gates`, and `th_* / m_*` pairs?
7. `pooling` has `method: "rapidity"` and `eps_w` > 0; weights defined if `"declared"`?
8. `health.range_policy` chosen and `*_valid` ranges declared?
9. `emit.*` booleans align with intended surface (S1/S2/S3)?
10. Conformance tests list includes `"batch_equals_stream"` and `"residual_alarm"`?

Linter hints (paste-ready pseudo-rules)

- **TIME:** reject if `ts_utc` not RFC3339 or clock jitter > `jitter_max_ms` when declared.
- **DOMAIN:** reject if any `*_ref <= 0` (log lenses) or `P_scale <= 0` (asinh lens).
- **EMIT:** reject if a field appears in payload with `emit.field == false` (must be omitted, not null).
- **R_P:** reject if `r_P` present when `abs(pf) < pf_min`.
- **POOL:** reject if any pooled dial not clamped to `< 1 - eps_a`.
- **CONSISTENCY:** warn if `median(r_P,W)` has |bias| > `tol` for long windows; suggest CT/PT/sign audit.

Tiny examples (good vs bad)

Good (bounded before pool):

```
a_P := clamp_a( tanh(c_P * e_P), eps_a )
a_pool := tanh( (sum w*atanh(a_P)) / max(sum w, eps_w) )
```

Bad (unbounded pooling):

```
a_pool := tanh( (sum w*atanh(tanh(c_P * e_P))) / sum w )    # missing clamp
and eps_w guard
```

Good (residual guard):

```
if abs(pf) >= pf_min:
  r_P := e_P - ( e_V + e_I + ln( max(abs(pf),1e-12) / pf_ref ) )
```

Bad (unguarded residual):

```
r_P := e_P - ( e_V + e_I + ln(pf / pf_ref) )                 # pf may be ~0
or undefined
```

Operator-facing promise

- Same raw inputs + same manifest ⇒ same symbols, same policy decisions, across vendors, sites, and time.

1.11 Minimal Manifest — SSMEQ-Lite (YAML, copy-ready)

```
manifest_id: SSMEQ.DER.PLANT01.S1
title: "DER Plant — SSMEQ Lite"
description: "Portable electrical symbols for plant telemetry"
version: "1.0"
timestamps_utc: { created: "2025-11-10T00:00:00Z", updated: "2025-11-
10T00:00:00Z" }
owner_contact: "ops@example.org"

conventions:
  ts_utc_format: "RFC3339"
  power_sign: "positive_is_import"
  reactive_sign: "inductive_positive"
  ac_dc_mode: "AC_RMS"
  batching: "either"

channels:
  V_rms:
    lens: "log"
    anchors: { V_ref: 230.0 }
    bounds: { V_min_valid: 180.0, V_max_valid: 265.0 }
    alignment: { enabled: true, c_V: 3.0, eps_a: 1e-6 }
  I_rms:
    lens: "log"
    anchors: { I_ref: 100.0 }
    bounds: { I_max_valid: 1000.0 }
    alignment: { enabled: true, c_I: 3.0, eps_a: 1e-6 }
  f:
    lens: "log"
    anchors: { f_ref: 50.0 }
    bounds: { f_min_valid: 45.0, f_max_valid: 55.0 }
    alignment: { enabled: true, c_f: 4.0, eps_a: 1e-6 }
  P:
    lens: "asinh"
    anchors: { P_scale: 1000.0 }
    bounds: { P_max: 5000.0 }
    alignment: { enabled: true, c_P: 2.5, eps_a: 1e-6 }

bounded_metrics:
  pf:
    clamp: { eps_a: 1e-6 }
    pf_ref: 1.0

residuals:
  r_P: { enabled: true, tol: 0.15, dwell: 5, pf_min: 0.05 }

hysteresis:
  enabled: true
  rho: 0.7
  k_side: 2.0
  gates:
    V: { V_gate: 225.0, dV_gate: 5.0, th_enter: 0.60, th_clear: 0.40,
m_enter: 0.80, m_clear: 0.60 }
```

```
pooling:
  method: "rapidity"
  eps_w: 1e-9
  weights: "equal"

privacy: { mode: "none", notes: "" }
provenance: { stamp: "optional", hashing: "SHA-256 of envelope",
continuity: "optional" }

health:
  jitter_max_ms: 200
  range_policy: "flag"

conformance:
  tests: [ "unit_invariance", "batch_equals_stream", "residual_alarm" ]
```

1.12 Result Envelope — SSMEQ-Lite (JSON, copy-ready)

```
{
  "ts_utc": "2025-11-10T05:00:01Z",
  "e_V": -0.1031,
  "e_I": 0.0202,
  "e_f": -0.0020,
  "e_P": 0.5473,
  "a_pf": 0.9700,
  "r_P": 0.0120,
  "manifest_id": "SSMEQ.DER.PLANT01.S1",
  "health": { "range_ok": true, "sensor_ok": true, "drift": false }
}
```

1.13 Full Manifest — Plant + PQ Dials (YAML, copy-ready)

```
manifest_id: SSMEQ.PLANT01.FULL
title: "Plant — SSMEQ Full with PQ"
description: "Symbols for plant ops + PQ monitoring"
version: "1.0"

conventions:
  ts_utc_format: "RFC3339"
  power_sign: "positive_is_import"
  reactive_sign: "inductive_positive"
  ac_dc_mode: "AC_RMS"
  batching: "either"

channels:
  V_rms:
    lens: "hybrid"
    anchors: { V_ref: 400.0, dV: 10.0, tau_V: 8.0 }
    bounds: { V_min_valid: 320.0, V_max_valid: 460.0 }
    alignment: { enabled: true, c_V: 3.0, eps_a: 1e-6 }
  I_rms:
    lens: "log"
```

```
    anchors: { I_ref: 200.0 }
    bounds: { I_max_valid: 2000.0 }
    alignment: { enabled: true, c_I: 3.0, eps_a: 1e-6 }
  f:
    lens: "log"
    anchors: { f_ref: 50.0 }
    bounds: { f_min_valid: 47.0, f_max_valid: 53.0 }
    alignment: { enabled: true, c_f: 4.0, eps_a: 1e-6 }
  P:
    lens: "asinh"
    anchors: { P_scale: 5000.0 }
    bounds: { P_max: 20000.0 }
    alignment: { enabled: true, c_P: 2.5, eps_a: 1e-6 }
  Q:
    lens: "asinh"
    anchors: { Q_scale: 5000.0 }
    alignment: { enabled: false }
  S:
    lens: "log"
    anchors: { S_ref: 10000.0 }
    alignment: { enabled: false }

bounded_metrics:
  pf: { clamp: { eps_a: 1e-6 }, pf_ref: 1.0 }
  THD: { ref: { THD_ref: 5.0 }, dial: { c_THD: 2.0 } }
  unbalance:
    ref: { UnbV_ref: 1.0, UnbI_ref: 1.0 }   # percent -> per-unit
    dial: { c_unbV: 2.0, c_unbI: 2.0 }

residuals:
  r_P: { enabled: true, tol: 0.12, dwell: 10, pf_min: 0.05 }

hysteresis:
  enabled: true
  rho: 0.75
  k_side: 2.5
  gates:
    V: { V_gate: 390.0, dV_gate: 8.0, th_enter: 0.65, th_clear: 0.45,
m_enter: 0.85, m_clear: 0.65 }
    I: { I_gate: 0.90 * I_ref, dI_gate: 0.05 * I_ref, th_enter: 0.60,
th_clear: 0.40, m_enter: 0.80, m_clear: 0.60 }

pooling:
  method: "rapidity"
  eps_w: 1e-9
  weights: "declared"
  declared:
    feeder_A: 1.0
    feeder_B: 2.0
    feeder_C: 1.5

privacy: { mode: "none" }
provenance: { stamp: "optional", hashing: "SHA-256 of envelope",
continuity: "optional" }

health:
  jitter_max_ms: 150
  range_policy: "flag_and_drop"

conformance:
```

```
  tests: [ "unit_invariance", "batch_equals_stream", "residual_alarm",
"hysteresis_clear" ]
```

1.14 Authoring Guardrails (Normative Recap)

Purpose. Keep streams portable, reproducible, and chatter-safe. Everything below is MUST or MUST-NOT for compliant manifests and emitters.

- **One lens per channel.** Do **not** switch `lens` or its `anchors` mid-run. If you need a change, rev the manifest `version` and `manifest_id`.
- **Publish anchors & clamps.** Declare `*_ref`, `P_scale`, `eps_a`, `eps_w`, and any dial sharpness `c_*`. Hidden knobs are **not** allowed.
- **Explicit signs.** Always state `power_sign` (`positive_is_import` | `positive_is_export`) and `reactive_sign` (`inductive_positive` | `capacitive_positive`).
- **Bound before pooling.** Apply `a := sign(a)*min(abs(a), 1 - eps_a)` **before** any rapidity pooling:
  `a_pool := tanh( (Σ w*atanh(a)) / max(Σ w, eps_w) ).`
- **Residual guard.** Compute `r_P` **only** when `abs(pf) >= pf_min`; otherwise omit `r_P` (do not emit nulls).
- **Validity & health.** Declare range guards (e.g., `V_min_valid..V_max_valid`) and a clear `range_policy` (`flag` | `drop` | `flag_and_drop`). Never crash; set `health`.
- **Determinism (batch==stream).** Implement pooling with identical math for both paths. Acceptance test tolerance ≤ `1e-12`.
- **Stable hysteresis.** If memory/gates are enabled, keep `{rho, th_enter, th_clear, m_enter, m_clear}` fixed for the run.
- **Manifest identity.** Every envelope MUST include `manifest_id`. Operators must be able to **replay** from raw units using only the manifest.
- **Backwards safety.** New fields start optional with safe defaults; consumers ignore unknown keys.
- **Omit disableds.** If a field is off in `emit.*`, omit it entirely (do not send `null`).
- **No silent retunes.** Site differences use **anchors** (e.g., `V_ref`, `P_scale`)—rules live in symbol space and travel unchanged.

SECTION 2 — Encoding & Lenses (Normative)

Intent. Convert raw electrical quantities into **unitless, comparable symbols** so policies, alarms, pooling, and ML behave consistently across vendors and sites. The encoder never changes physics; it **only re-expresses** values into a stable space.

Outputs (contrasts). From raw inputs, derive zero-centered contrasts such as `e_V`, `e_I`, `e_f`, `e_P` (and optionally `e_Q`, `e_S`). Each contrast is:

- **Zero-centric** (reference maps to `0`),
- **Monotone** (larger physical → larger symbolic),
- **Unit-invariant** (SCADA ints, engineering units, or per-unit all converge),
- **Vendor-portable** (no hidden multipliers/offsets),
- **Rule-ready** (safe for thresholds, hysteresis, pooling, ML).

One lens per channel (declared in the manifest).

- **log**: wide-range drifts (e.g., `e_V := ln(V_rms / V_ref)`, `e_I := ln(I_rms / I_ref)`, `e_f := ln(f / f_ref)`).
- **linear**: tight deviations near nominal (small-signal studies).
- **hybrid**: linear near reference, log beyond a boundary.
- **asinh**: gentle handling of signed/skewed power (`e_P := asinh(P / P_scale)` and similarly for `Q` if used).

Anchors are explicit. References and scales such as `V_ref`, `I_ref`, `f_ref`, `P_scale` **define zero and slope**; they MUST be published in the manifest for audit and replay.

Optional bounded dials. For dashboards, pooling, and chatter-safe operations, unbounded contrasts can be mapped to **bounded alignments**:

- Example: `a_V := clamp_a( tanh(c_V * e_V), eps_a )`, `a_I := clamp_a( tanh(c_I * e_I), eps_a )`, `a_P := clamp_a( tanh(c_P * e_P), eps_a )`.
- **Clamp before pooling:** `a := sign(a) * min(abs(a), 1 - eps_a)`.

Rapidity pooling (order-invariant). Fleet or feeder aggregation uses:

- `a_pool := tanh( ( sum_i w_i * atanh( clamp_a(a_i, eps_a) ) ) / max(sum_i w_i, eps_w) )`.
- Guarantees **batch == stream** and stable behavior across replays.

Design principles (must/should).

- **Exactly one lens per channel**, fixed for the run (no silent mid-stream changes).
- **Deterministic math only; no hidden calibrations** inside the encoder.
- **Sign conventions explicit** (`power_sign`, `reactive_sign`), ensuring `P`, `Q`, and `pf` are interpreted uniformly.
- **Residuals are guarded** (e.g., compute `r_P` only when `abs(pf) >= pf_min`): `r_P := e_P - ( e_V + e_I + ln( max(abs(pf), 1e-12 ) / pf_ref ) )`.

Outcome. A tiny, deterministic encoder yields `e_*` (and optional `a_*`, `M`, `r_P`) that are:

- **Comparable across vendors, voltages, and CT/PT classes,**
- **Ready for portable rules** (UNDER_V, OVER_I, etc.) without unit retunes,
- **Safe for pooling** and downstream analytics with strict reproducibility.

2.1 — Lens choices (log, linear, hybrid, asinh): why and when

Purpose. Pick one **declared lens per channel** so that raw electrical measurements become **zero-centric**, **monotone**, and **unit-invariant** contrasts `e_*`. Each lens maps a physical value and its anchor into a dimensionless symbol that is stable near nominal yet expressive over wide ranges.

General pattern. For a raw quantity `x` with anchor `x_ref` and optional scale `x_scale`:

- **Log lens (ratio):** `e := ln(x / x_ref)`
- **Linear lens (offset):** `e := (x - x_ref) / x_scale`
- **asinh lens (signed wide-range):** `e := asinh(x / x_scale)`
- **Hybrid lens (piecewise smooth):**
  - Near nominal (`|x - x_ref| <= tau`): `e := (x - x_ref) / x_scale`
  - Far from nominal (`|x - x_ref| > tau`): `e := s * ln( 1 + |x - x_ref| / tau ) * sign(x - x_ref)`
  - Choose `tau > 0`, `x_scale > 0`, `s := x_scale / tau` to keep slopes consistent at the join.

Small-signal equivalence (near anchors).
`ln(x / x_ref) ≈ (x - x_ref) / x_ref` when `|x - x_ref| << x_ref`.
This is why **log** and **linear** behave similarly close to nominal; they differ in tails and symmetry.

When to use what (channel heuristics).

- **Voltage `V_rms` (always positive).**
  Use **log** when the feeder/device sees large % swings; use **linear** for tight process windows.
  Suggested defaults:
  `e_V := ln(V_rms / V_ref)` (wide range) or `e_V := (V_rms - V_ref) / V_scale` (tight).
- **Current `I_rms` (always positive).**
  Same logic as voltage.
  `e_I := ln(I_rms / I_ref)` or `e_I := (I_rms - I_ref) / I_scale`.
- **Frequency `f` (positive, narrow range).**
  Typically **log** for ratio-clarity and unit-invariance; **linear** also acceptable due to tight bounds.
  `e_f := ln(f / f_ref)` or `e_f := (f - f_ref) / f_scale`.
- **Active power `P` (signed).**
  Choose **asinh** to handle sign and large dynamic range cleanly.
  `e_P := asinh(P / P_scale)`.

- **Reactive power `Q` (signed).**
  Same as `P`.
  `e_Q := asinh(Q / Q_scale).`
- **Apparent power `S` (nonnegative).**
  Use **log** for fleet-wide comparability or **linear** for tight corridors.
  `e_S := ln(S / S_ref)` or `e_S := (S - S_ref) / S_scale`.
- **Impedance-like or distortion metrics (nonnegative, wide tail).**
  Prefer **hybrid** (linear near 0, log-like tails) to suppress chatter yet respect large excursions.

Design notes (practical).

- **Zero-centricity.** Every lens must satisfy `e = 0` at its anchor (`x = x_ref` or `x = 0` for asinh with signed `x` if you choose a zero anchor).
- **Monotonicity.** Each mapping is strictly monotone in its valid domain; **log** requires `x > 0`.
- **Symmetry. asinh** is odd: `asinh(-z) = -asinh(z)` — ideal for signed `P`/`Q`.
- **Tail behavior. log** compresses multiplicatively; **asinh** compresses additively at large `|x|`; **hybrid** blends both while keeping **linear** sensitivity near nominal.
- **Parameter hygiene.** Publish `x_ref`, `x_scale`, and (if used) `tau` in the profile. Keep them fixed across a deployment for portability.

Copy-ready lens picks (summary).

- `e_V := ln(V_rms / V_ref)`
- `e_I := ln(I_rms / I_ref)`
- `e_f := ln(f / f_ref)`
- `e_P := asinh(P / P_scale)`
- `e_Q := asinh(Q / Q_scale)`
- `e_S := ln(S / S_ref)` *(or)* `e_S := (S - S_ref) / S_scale`
- **Hybrid template (generic):**
  `e := (x - x_ref) / x_scale` if `|x - x_ref| <= tau`, else
  `e := (x_scale / tau) * ln( 1 + |x - x_ref| / tau ) * sign(x - x_ref)`

2.2 Anchor rules (`*_ref`, scales) and zero-centric contrasts (`e_*`)

Purpose.
Make every encoded contrast `e_*` **zero-centric**, **unit-invariant**, and **portable** by publishing explicit anchors and scales for each channel. A contrast is zero-centric **if and only if** `e_* = 0` at the declared anchor.

Per-channel declarations (copy-ready)

For each raw quantity `x`, choose a lens and publish parameters:

```
channel: X
lens: LOG | LINEAR | ASINH | HYBRID
x_ref: <positive real> # required for LOG/LINEAR/HYBRID
x_scale: <positive real> # required for LINEAR/ASINH/HYBRID
tau: <positive real> # required for HYBRID
notes: <optional free text> # e.g., RMS for AC, mean or DC bus nominal
```

Zero-centric rule.

- LOG / LINEAR / HYBRID lenses must satisfy `e( x_ref ) = 0`.
- ASINH lens must satisfy `e( 0 ) = 0` (anchor at physical zero).
- Do **not** mix multiple anchors for one channel in a deployment or profile.

Recommended anchors (`*_ref`) and scales

- **Voltage (RMS):**
  - `V_ref = nominal_V` (for example, feeder setpoint or bus nominal).
  - For LINEAR, pick `V_scale = V_ref` (per-unit, 1.0 at nominal) or `V_scale = 0.01 * V_ref` (percent band).
- **Current (RMS):**
  - `I_ref = rated_I` (for example, breaker or feeder rating).
  - `I_scale = I_ref` (per-unit) or `I_scale = 0.1 * I_ref` (10% band) depending on desired sensitivity.
- **Frequency:**
  - `f_ref = nominal_f` (for example, 50 or 60).
  - `f_scale = 0.1` (0.1 Hz) or as required by regulation or control needs.
- **Active power (signed):**
  - Use ASINH with `P_scale = rated_P` (nameplate, phase or total).
  - `P_scale` sets sensitivity near zero and should match typical operating range.
- **Reactive power (signed):**
  - Use ASINH with `Q_scale = rated_Q`.
- **Apparent power (nonnegative):**
  - `S_ref = rated_S`; for LINEAR, `S_scale = S_ref` for per-unit.
- **Power factor (magnitude):**
  - `pf_ref` is declared and may be:
    - `1.0` (ideal unity power factor), or
    - a **target power factor** (for example, `0.95`) if that reflects contractual/operational goals.

Sign conventions (must publish once, reuse everywhere)

```
P > 0 := import (consuming from grid/feeder)
P < 0 := export (supplying to grid/feeder)

Q > 0 := inductive (lagging)
Q < 0 := capacitive (leading)

S >= 0 := magnitude (no sign)

pf := |P| / max( S , eps_pf ) # pf is a magnitude in (0, 1]
```

- If your environment uses a different convention, **publish it explicitly** and keep it consistent across all SSMEQ records using that manifest.
- Directional information (leading/lagging) belongs in `Q` and its **sign convention**, not in `pf` itself.

Zero-centric contrasts (copy-ready)

```
e_V := ln( V_rms / V_ref )
# or LINEAR: e_V := ( V_rms - V_ref ) / V_scale

e_I := ln( I_rms / I_ref )
# or LINEAR: e_I := ( I_rms - I_ref ) / I_scale

e_f := ln( f / f_ref )
# or LINEAR: e_f := ( f - f_ref ) / f_scale

e_P := asinh( P / P_scale )
e_Q := asinh( Q / Q_scale )

e_S := ln( S / S_ref )
# or LINEAR: e_S := ( S - S_ref ) / S_scale

e_pf := ln( pf / pf_ref )
```

- Under this convention, `pf < pf_ref` yields a **negative** `e_pf` (under-performing against the target), and `pf > pf_ref` yields a **positive** `e_pf`.

Three-phase note (per-phase then optional pool)

- Declare anchors **per phase**: for example, `V_ref_a`, `V_ref_b`, `V_ref_c` (typically equal in a balanced design).
- Encode per-phase contrasts: `e_V_a`, `e_V_b`, `e_V_c`, and similarly for currents or power if needed.
- Pool later using the **rapidity rule** (order-invariant pooling) when you want a feeder or bus-level dial.

Monotonicity and unit-invariance checks (copy-ready)

Zero at anchor:

```
e_V( V_rms = V_ref ) = 0
e_I( I_rms = I_ref ) = 0
e_f( f = f_ref ) = 0
e_P( P = 0 ) = 0
e_Q( Q = 0 ) = 0
e_S( S = S_ref ) = 0
e_pf( pf = pf_ref ) = 0
```

Unit invariance (examples):

- `ln( V_rms / V_ref )` is unchanged by scaling volts → kilovolts (if both are scaled).
- `ln( f / f_ref )` is unchanged by Hz → mHz (if both are scaled).
- `asinh( P / P_scale )` is unchanged by kW → W if `P_scale` scales accordingly.

Hybrid template (copy-ready, zero-centric at `x_ref`)

```
if |x - x_ref| <= tau:
    e := ( x - x_ref ) / x_scale
else:
    e := ( x_scale / tau ) * ln( 1 + |x - x_ref| / tau ) * sign( x - x_ref
)
```

- Near the anchor (`|x - x_ref| <= tau`), the lens is **linear**.
- Far from the anchor, the lens grows **logarithmically** in the deviation while preserving sign and continuity.

Publication checklist (paste into profile)

```
publish:
  V:  { lens: LOG,    V_ref: <...> }
  I:  { lens: LOG,    I_ref: <...> }
  f:  { lens: LOG,    f_ref: <...> }
  P:  { lens: ASINH,  P_scale: <...> }
  Q:  { lens: ASINH,  Q_scale: <...> }
  S:  { lens: LOG,    S_ref: <...> }      # or LINEAR with S_scale
  pf: { lens: LOG,    pf_ref: <...> }     # 1.0 (unity) or target, e.g.
0.95

  # optional HYBRID fields for any channel:
  # x_scale: <...>, tau: <...>

conventions:
  P_sign: import_positive
  Q_sign: inductive_positive

notes:
  voltage/current measured as RMS;
```

```
  frequency as fundamental estimate;
  DC channels use declared DC anchors (e.g., DC bus nominal).
```

2.3 Safe domains and guards (positivity, AC/DC specifics)

Purpose.
Ensure every contrast `e_*` is **well-defined** and **numerically stable** under real-world conditions (zero / near-zero, sign changes, saturation, missing samples) **without breaking zero-centricity or portability**.

Guard constants (publish once, reuse everywhere)

```
eps_pos := 1e-12    # positivity floor for quantities used in LOG
eps_div := 1e-12    # division floor for denominators
eps_w   := 1e-12    # pooling denominator floor
eps_a   := 1e-6     # lane clamp epsilon
pf_min  := 0.05     # lower bound for pf to avoid log(0); publish per
profile
alpha   := 1e-6     # tiny relative floor factor for anchors
```

Choose values appropriate for **sensor precision** and **numeric range**; keep them **fixed per deployment or profile**.

Lens-domain guards (copy-ready)

```
# LOG lens: requires x > 0
x_safe := max( x_raw , max( eps_pos , alpha * x_ref ) )
e_LOG  := ln( x_safe / x_ref )

# LINEAR lens: scale must be positive
e_LIN  := ( x_raw - x_ref ) / max( x_scale , eps_div )

# ASINH lens: any real input; scale must be positive
e_ASINH := asinh( x_raw / max( x_scale , eps_div ) )

# HYBRID lens: LINEAR near anchor and LOG-like tail; parameters must be
positive
if | x_raw - x_ref | <= tau:
    e_HYB := ( x_raw - x_ref ) / max( x_scale , eps_div )
else:
    e_HYB := ( x_scale / tau )
             * ln( 1 + | x_raw - x_ref | / tau )
             * sign( x_raw - x_ref )
```

Channel-specific safety (copy-ready)

```
# Voltage, Current, Apparent power (nonnegative sensors)
V_rms := max( V_rms_raw , max( eps_pos , alpha * V_ref ) )
I_rms := max( I_rms_raw , max( eps_pos , alpha * I_ref ) )
S     := max( S_raw     , max( eps_pos , alpha * S_ref ) )

# Frequency (positive, narrow range in practice)
f := max( f_raw , max( eps_pos , alpha * f_ref ) )
# Leave upper bound to policy; do not hard-clip here.

# Active/Reactive power (signed)
# No positivity floor for P, Q (they can be negative); protect only scales.
e_P := asinh( P_raw / max( P_scale , eps_div ) )
e_Q := asinh( Q_raw / max( Q_scale , eps_div ) )

# Power factor (bounded to avoid log(0) and unstable ratios)
pf_num := | P_raw |
pf_den := max( S , eps_div )
pf     := pf_num / pf_den
pf     := min( 1.0 , max( pf_min , pf ) )
e_pf   := ln( pf / pf_ref )
```

- Here `pf` is a **magnitude in (pf_min, 1]**; direction is carried by `Q` and `reactive_sign`, not by `pf` itself.

AC vs DC specifics (declarative)

```
# AC measurement operators
V_rms, I_rms := true RMS over a published window T_win
P_raw, Q_raw := fundamental (or total) power estimates per profile notes
f_raw        := fundamental frequency estimate

# DC measurement operators
V_rms := mean( V ) over T_win
I_rms := mean( I ) over T_win
# Frequency channel omitted in DC profiles; do not emit e_f for DC-only
profiles.
```

- Publish `T_win` and **estimator choice** once in the profile (for example, 1-cycle, 10-cycle, or fixed millisecond window).
- Mixed AC/DC systems should declare **separate site profiles** or channel-mode flags rather than silently changing operators.

Saturation, missing, and invalid samples

```
# Basic screening
if isNaN( x_raw ) or isInf( x_raw ):
    status := "invalid"
if sensor_saturated:
    status := "saturated"
```

```
# Handling policy (high-level; detail belongs in the health section)
if status in { "invalid" , "saturated" }:
    action := flag_and_drop    # or "flag" (keep) per profile
    # Do NOT fabricate e_* and do NOT silently carry forward previous
values.
```

- **Invalid/saturated input** should **not** produce a "best guess" contrast; it should produce a **flag** and, if configured, a **dropped sample** for that record.

Zero-load and near-zero behavior (non-crashing)

- At **zero load**, `I_rms → 0` and `S → 0`:
  - LOG lenses still behave via floors such as `eps_pos` and `alpha * x_ref`.
  - ASINH lenses remain well-defined at `x_raw = 0`.
- Keep floors such as `alpha * x_ref` **tiny** so they:
  - Prevent `ln(0)` or division by zero,
  - Do **not** distort behavior near the anchor or in normal operating bands.

Sign conventions (consistency guard)

- Maintain the **same published sign rules** for `P` and `Q` across all devices and encoders under a given manifest.
- Do **not** re-flip signs ad hoc inside the encoder. If a different convention is required:
  - Change it **once** in the profile,
  - Re-encode data consistently using that convention.

Copy-ready guard block (drop-in)

```
# Inputs:  V_rms_raw, I_rms_raw, S_raw, P_raw, Q_raw, f_raw
# Params:  V_ref, I_ref, S_ref, f_ref, P_scale, Q_scale, pf_ref
# Guards:  eps_pos, eps_div, alpha, pf_min, x_scale, tau

V_rms := max( V_rms_raw , max( eps_pos , alpha * V_ref ) )
I_rms := max( I_rms_raw , max( eps_pos , alpha * I_ref ) )
S     := max( S_raw     , max( eps_pos , alpha * S_ref ) )
f     := max( f_raw     , max( eps_pos , alpha * f_ref ) )

pf_num := | P_raw |
pf_den := max( S , eps_div )
pf     := min( 1.0 , max( pf_min , pf_num / pf_den ) )

# From here, compute:
#   e_V, e_I, e_f, e_S using LOG/LINEAR/HYBRID with guarded inputs
#   e_P, e_Q using ASINH with guarded scales
#   e_pf using ln( pf / pf_ref )
```

Why these guards do not break zero-centricity

- Floors are applied to raw quantities **only when they would violate domain** constraints (for example, `ln(0)`), and are chosen such that:
  - At published anchors (`x = x_ref`, `P = 0`, `Q = 0`, `pf = pf_ref`), the guarded expression still yields:
    - `e_V = 0`, `e_I = 0`, `e_f = 0`, `e_S = 0`,
    - `e_P = 0`, `e_Q = 0`, `e_pf = 0`.
- Scales and anchors remain **unchanged**; guards only prevent **undefined operations** and **numeric explosions** while preserving the exact zero at the declared reference points.

2.4 — Copy-ready formulas (ASCII) for e_V, e_I, e_f, e_P[, e_Q, e_S, e_pf]

Purpose. A single **drop-in encoder block** that turns raw electrical measurements into **zero-centric**, **unit-invariant** contrasts `e_*`, using the declared lenses and guards.

Inputs (measured): `V_rms_raw, I_rms_raw, S_raw, P_raw, Q_raw, f_raw`
Published params (profile):
`V_ref, I_ref, S_ref, f_ref, P_scale, Q_scale, pf_ref`
`eps_pos, eps_div, eps_w, eps_a, pf_min, alpha` *(guards)*
`lens_V, lens_I, lens_f, lens_P, lens_Q, lens_S` in `{LOG, LINEAR, ASINH, HYBRID}`
`V_scale, I_scale, f_scale, S_scale, (x_scale, tau for HYBRID)`

Copy-ready encoder (paste as is).

```
# 0) Guards for raw channels used by LOG or denominators
V_rms := max(V_rms_raw, max(eps_pos, alpha*V_ref))
I_rms := max(I_rms_raw, max(eps_pos, alpha*I_ref))
S     := max(S_raw,     max(eps_pos, alpha*S_ref))
f     := max(f_raw,     max(eps_pos, alpha*f_ref))

# 1) Lens helpers (generic)
LIN(x, x_ref, x_scale) := (x - x_ref) / max(x_scale, eps_div)
LOG(x, x_ref)          := ln( x / x_ref )              # requires x > 0
(guarded above)
ASINH(x, x_scale)      := asinh( x / max(x_scale, eps_div) )
HYB(x, x_ref, x_scale, tau) :=
    ( |x - x_ref| <= tau ) ? LIN(x, x_ref, x_scale)
                           : (x_scale / tau) * ln( 1 + |x - x_ref| / tau )
* sign(x - x_ref)

# 2) Channel contrasts (select by declared lens per channel)

# Voltage (nonnegative)
e_V :=
    (lens_V == LOG)    ? LOG(V_rms, V_ref) :
    (lens_V == LINEAR) ? LIN(V_rms, V_ref, V_scale) :
    (lens_V == HYBRID) ? HYB(V_rms, V_ref, V_scale, tau_V) :
```

```
                          LOG(V_rms, V_ref)   # default

# Current (nonnegative)
e_I :=
    (lens_I == LOG)    ? LOG(I_rms, I_ref) :
    (lens_I == LINEAR) ? LIN(I_rms, I_ref, I_scale) :
    (lens_I == HYBRID) ? HYB(I_rms, I_ref, I_scale, tau_I) :
                          LOG(I_rms, I_ref)   # default

# Frequency (positive, narrow)
e_f :=
    (lens_f == LOG)    ? LOG(f, f_ref) :
    (lens_f == LINEAR) ? LIN(f, f_ref, f_scale) :
                          LOG(f, f_ref)       # default

# Active power (signed)
e_P :=
    (lens_P == ASINH)  ? ASINH(P_raw, P_scale) :
    (lens_P == LINEAR) ? LIN(P_raw, 0, P_scale) :
                          ASINH(P_raw, P_scale) # default

# Reactive power (signed) — optional channel
e_Q :=
    (lens_Q == ASINH)  ? ASINH(Q_raw, Q_scale) :
    (lens_Q == LINEAR) ? LIN(Q_raw, 0, Q_scale) :
                          ASINH(Q_raw, Q_scale) # default

# Apparent power (nonnegative) — optional channel
e_S :=
    (lens_S == LOG)    ? LOG(S, S_ref) :
    (lens_S == LINEAR) ? LIN(S, S_ref, S_scale) :
    (lens_S == HYBRID) ? HYB(S, S_ref, S_scale, tau_S) :
                          LOG(S, S_ref)        # default

# Power factor (bounded in (0,1]) — optional contrast for residuals/policy
pf_num := |P_raw|
pf_den := max(S, eps_div)
pf     := min(1.0, max(pf_min, pf_num / pf_den))
e_pf   := ln( pf / pf_ref )
```

Properties (quick checks).

```
e_V(V_rms = V_ref) = 0
e_I(I_rms = I_ref) = 0
e_f(f = f_ref)     = 0
e_P(P_raw = 0)     = 0
e_Q(Q_raw = 0)     = 0
e_S(S = S_ref)     = 0
e_pf(pf = pf_ref)  = 0
```

Three-phase usage (concise). Compute `e_*` per phase (`a,b,c`) using the same anchors; pool later (order-invariant) if needed.

2.5 Mini sanity set (2–3 numeric quick checks)

Purpose.
Verify that lenses are **zero-centric**, behave similarly near anchors, and remain **well-behaved over wide ranges**, including sign and target-based anchors like `pf_ref`.

A) Voltage ±5% around anchor (log vs linear near-equivalence)

Parameters
`V_ref = 230`
Linear per-unit scaling: `V_scale = V_ref`

Cases

```
Case A1: V_rms = 1.05 * V_ref
```

- `e_V(log) = ln( 1.05 ) ≈ +0.04879`
- `e_V(lin) = (1.05 - 1.00) = +0.05`

```
Case A2: V_rms = 0.95 * V_ref
```

- `e_V(log) = ln( 0.95 ) ≈ -0.05129`
- `e_V(lin) = (0.95 - 1.00) = -0.05`

Observation.
Near the anchor, `ln( x / x_ref ) ≈ ( x - x_ref ) / x_ref`. The numbers match closely, showing **small-signal equivalence** between log and linear-per-unit near nominal.

B) Frequency ±1% around 50 Hz (ratio clarity)

Parameters
`f_ref = 50`

Show both linear-per-unit and log:

```
Case B1: f = 49.5 (−1%)
```

- `e_f(log) ≈ ln( 49.5 / 50 ) ≈ -0.01005`
- `e_f(lin) = (49.5 - 50) / 50 = -0.01`

```
Case B2: f = 50.5 (+1%)
```

- `e_f(log) ≈ ln( 50.5 / 50 ) ≈ +0.00995`
- `e_f(lin) = (50.5 - 50) / 50 = +0.01`

Observation.
For **tight corridors**, log and linear-per-unit are numerically interchangeable near nominal. Log has the advantage of being **unit-invariant** by construction (`Hz`, `mHz`, etc., as long as both numerator and denominator share the unit).

C) Active power with sign and wide range (asinh symmetry and tail)

Parameters
`P_scale = 10,000` (W)

Cases

`Case C1: P = +5,000`

- `e_P = asinh( 0.5 ) ≈ +0.48121`

`Case C2: P = -5,000`

- `e_P = asinh( -0.5 ) ≈ -0.48121`

`Case C3: P = +50,000`

- `e_P = asinh( 5.0 ) ≈ +2.31244`

Observation.
`asinh` is **odd** (`asinh( -x ) = -asinh( x )`) so it handles **sign cleanly**, and it **compresses large magnitudes smoothly**, giving stable tails for extreme loads without losing direction.

D) Optional quick check: power factor contrast (target vs actual)

Parameters (unity target)
`pf_ref = 1.0`, example `pf = 0.8`:

- `e_pf = ln( pf / pf_ref ) = ln( 0.8 ) ≈ -0.22314`

Observation.
A sub-unity power factor produces a **negative contrast**, matching intuitive "below-target" semantics when `pf_ref = 1.0`.

Parameters (non-unity target)
`pf_ref = 0.95`, example `pf = 0.8`:

- `e_pf = ln( 0.8 / 0.95 ) ≈ -0.17185`

Observation.
Using a **realistic target** (such as `pf_ref = 0.95`) shifts the zero to the contractual or operational expectation. Values below `0.95` produce **negative e_pf, `and values above` 0.95 `produce **positive e_pf`, while preserving zero at `pf = pf_ref`.

E) Optional DC sanity check (voltage on a DC bus)

Parameters
DC bus nominal: `V_ref = 800` (V)

Cases

`Case E1: V_bus = 800`

- `e_V = ln( 800 / 800 ) = 0`

`Case E2: V_bus = 760` (−5%)

- `e_V = ln( 760 / 800 ) ≈ ln( 0.95 ) ≈ -0.05129`

`Case E3: V_bus = 840` (+5%)

- `e_V = ln( 840 / 800 ) ≈ ln( 1.05 ) ≈ +0.04879`

Observation.
The **same lens and anchor logic** applies to DC as to AC. A DC bus with `V_ref = 800` behaves identically (in `e_V`) to an AC feeder with per-unit `±5%` swings around nominal, reinforcing that SSMEQ is **unit and mode invariant** once anchors are declared.

2.6 — Anchor Profiles for AC 50/60 Hz and DC

(Guidance, Copy-Ready)

Purpose. Provide practical, site-ready anchor profiles for common electrical contexts so lenses in Section 2 can be deployed without guesswork.
Anchors are declared once per **site profile** and bound into the manifest as:

- `V_ref`, `I_ref`, `S_ref` (or `P_ref`)
- `f_ref`, `pf_ref`
- `THDV_ref`, `THDI_ref`, `UnbV_ref`, `UnbI_ref`
- `T_ref` (thermal reference)

These profiles act as **starting points**, not hard rules. They must be adjusted only when a site's canonical ratings or "normal operating posture" are clearly established.

2.6.1 — LV AC 50 Hz Plant (Typical Anchors, Copy-Ready)

Context. Low-voltage industrial / commercial plant, **50 Hz**, three-phase, 400 V line-to-line, 230 V line-to-neutral.

Suggested anchor profile (example).

```
# LV_AC_50HZ_PLANT (example site profile)
V_ref_LL   = 400.0   # V, line-to-line nominal
V_ref_LN   = 230.0   # V, line-to-neutral nominal
I_ref      = 1.0     # per-unit base; or rated feeder current in A
S_ref      = 1.0     # per-unit base; or transformer kVA rating
f_ref      = 50.0    # Hz

pf_ref     = 1.0     # ideal displacement PF
pf_min     = 0.9     # below this PF, lenses start to penalize

THDV_ref   = 5.0     # % THDv reference (IEC typical planning level)
THDI_ref   = 8.0     # % THDi reference (adjust to plant characteristics)
UnbV_ref   = 2.0     # % voltage unbalance reference
UnbI_ref   = 5.0     # % current unbalance reference

T_ref      = 40.0    # °C typical cabinet / MCC temperature reference
T_max_safe = 70.0    # °C typical upper thermal reference for stress lenses
```

When this is reasonable.

- Standard LV distribution with 400/230 V, 50 Hz.
- Harmonic planning limits aligned with IEC guidance (THDv ≤ 8%, target ≈ 5%).
- PF contracts or internal standards expect PF ≥ 0.9 most of the time.

When to adjust.

- **Different transformer tap setting or nominal voltage:**
  If the site is run intentionally at 415 V LL, set `V_ref_LL = 415.0`, adjust `V_ref_LN` accordingly.
- **Very harmonic-rich environments (e.g., drives-heavy plant):**
  If "business as usual" THDv is 7–8%, you may keep `THDV_ref = 5.0` (stricter posture) **or** raise to 7.0 with a clear note in the manifest.
- **Cool / harsh climates:**
  Adjust `T_ref` toward typical cabinet internal temperature, not ambient air alone.

2.6.2 — LV AC 60 Hz Plant (Typical Anchors, Copy-Ready)

Context. Low-voltage industrial / commercial plant, **60 Hz**, 480 V line-to-line, 277 V line-to-neutral.

Suggested anchor profile (example).

```
# LV_AC_60HZ_PLANT (example site profile)
V_ref_LL   = 480.0   # V, line-to-line nominal
V_ref_LN   = 277.0   # V, line-to-neutral nominal
I_ref      = 1.0     # per-unit base; or rated feeder current in A
S_ref      = 1.0     # per-unit base; or transformer kVA rating
f_ref      = 60.0    # Hz

pf_ref     = 1.0
pf_min     = 0.9

THDV_ref   = 5.0     # % THDv reference
THDI_ref   = 8.0     # % THDi reference
UnbV_ref   = 2.0     # % voltage unbalance
UnbI_ref   = 5.0     # % current unbalance

T_ref      = 40.0    # °C cabinet reference
T_max_safe = 75.0    # °C typical upper reference
```

Notes.

- The **structure** is identical to 50 Hz plants; only `V_ref_*` and `f_ref` differ.
- If the site uses 208/120 V or 600/347 V systems, simply change `V_ref_LL`, `V_ref_LN` and keep the rest.

2.6.3 — HV Transmission Node (Anchors for Substation / Grid-Level)

Context. High-voltage or sub-transmission node with **tight voltage and frequency regulation**, more sensitive unbalance and PQ criteria.

Suggested anchor profile (example for 132 kV).

```
# HV_NODE_132KV (example site profile)
V_ref_LL   = 132000.0   # V, nominal line-to-line
I_ref      = 1.0        # per-unit; or rated line current
S_ref      = 1.0        # per-unit; or transformer rating
f_ref      = 50.0       # or 60.0, as per system

pf_ref     = 1.0
pf_min     = 0.95       # tighter PF expectation at HV level

THDV_ref   = 2.0        # % THDv, tighter planning level
THDI_ref   = 3.0        # % THDi, typical HV range
UnbV_ref   = 1.0        # % voltage unbalance
UnbI_ref   = 2.0        # % current unbalance

T_ref      = 35.0       # °C, typical control-room / panel internal temp
T_max_safe = 60.0       # °C, upper safe internal temp for electronics
```

When to tighten anchors.

- System is part of a **backbone transmission grid** with strict PQ and stability requirements.
- Grid code mandates PF control and harmonic limits with penalties.

When to relax anchors.

- Remote or mixed-generation nodes where local PQ is known to be noisier, but still within regulatory allowances.
- In early commissioning, you may choose slightly looser `THDV_ref` / `UnbV_ref` and tighten later once normal behavior is well-characterized.

2.6.4 — DC Solar / Battery Strings (Anchors for SSMEQ on DC)

Context. DC solar array or battery string where voltage, current, and temperature are the primary health indicators.
AC-specific lenses (THD, unbalance) may be unused or repurposed for ripple and imbalance proxies.

Suggested anchor profile (example).

```
# DC_SOLAR_STRING (example site profile)
V_ref_DC   = 1000.0   # V, nominal string voltage (e.g., 1000 Vdc)
I_ref_DC   = 100.0    # A, nominal string current
P_ref_DC   = 100000.0 # W, nominal string power (optional anchor)
f_ref      = 0.0      # Hz, not used; keep 0 or omit

pf_ref     = 1.0      # for consistency; DC behaves as pf = 1 by
construction
pf_min     = 1.0      # no PF penalty on DC; PF lenses may be disabled

RippleV_ref = 1.0     # % allowed DC voltage ripple (if modeled)
RippleI_ref = 2.0     # % allowed DC current ripple

T_ref      = 45.0     # °C typical module / cabinet reference
T_max_safe = 80.0     # °C upper safe temp (module backsheet / cell proxy)
```

Usage notes.

- **Voltage lens (a_V_DC).** Map deviation from `V_ref_DC` using a `dV`-style lens.
- **Current lens (a_I_DC).** Map deviation from `I_ref_DC` as load / generation stress.
- **Thermal lens (a_T).** Map module or cabinet temperature vs `T_ref` and `T_max_safe`.
- **Ripple lenses.** If DC-side ripple is measured, treat `RippleV_ref` and `RippleI_ref` as analogues to `THDV_ref`, `THDI_ref`.

2.6.5 — Anchor Selection & Adjustment Checklist (Copy-Ready)

Use the following as a **manifest note** or commissioning worksheet:

```
anchor_governance:
  choose_base:
    - set V_ref, f_ref from nameplate or grid code
    - set I_ref, S_ref from feeder / transformer rating or per-unit base
  choose_quality_refs:
    - THDV_ref, THDI_ref from planning limits or PQ contracts
    - UnbV_ref, UnbI_ref from grid / internal standards
  choose_pf_refs:
    - pf_ref := 1.0 unless contractually different
    - pf_min := 0.9 LV, 0.95 HV (typical starting points)
  choose_thermal_refs:
    - T_ref from typical cabinet/module internal operating temperature
    - T_max_safe from datasheet or design limit

  adjust_when:
    - equipment rating / tap changes
    - grid code / contract changes
    - long-term "normal" deviates clearly from assumptions

  do_not:
    - silently change anchors within a manifest version
    - mix multiple anchor sets under the same manifest_id
```

Section **9.2 (Site Profiles & Anchor Governance)** will formalize how these anchors are versioned and changed over time. For now, this subsection provides **practical numeric defaults** and checklists so SSMEQ can be safely initialized on 50 Hz, 60 Hz, HV, and DC systems without ambiguity.

SECTION 3 — Bounded Dials & Memory (Operational)

Intent. Convert raw contrasts and indicators into **bounded alignment dials** `a ∈ (-1,+1)` and a single **lightweight memory** `M` that together deliver **stable, chatter-resistant signals** for policy, pooling, and dashboards.

What this section establishes.

- A **universal clamp** so every dial stays inside `(-1,+1)`, keeping `atanh`/`tanh` math finite and portable.
- A small family of **proximity dials** (e.g., approach to limits for current and power) derived from the contrasts `e_*`.
- **Quality dials** for PQ and integrity (e.g., `a_THD`, `a_unbV`, `a_unbI`) using smooth, monotone shapes.
- A single **shared memory** `M` with retention `rho` that **absorbs flicker**, supports **enter/clear** hysteresis, and ensures **repeatable decisions**.

- **Copy-ready snippets** (S1/S2/S3) that let producers scale from a minimal payload to a full posture without changing semantics.

Design goals.

- **Boundedness:** all dials strictly within `(-1,+1)`; no poles.
- **Monotonicity:** larger physical deviation → larger `|a|`.
- **Zero-centric semantics:** nominal or target → `a ≈ 0`.
- **Batch == stream:** identical results whether processed offline or online.
- **Parameter parsimony:** a few published constants (e.g., `eps_a`, `rho`) govern behavior across devices.

Key building blocks.

- **Clamp invariant:**
  `a_c := max(-1+eps_a, min(+1-eps_a, a_raw))`
  `u := atanh(a_c)` and (optionally) `a := tanh(u)` to reproject.
- **Proximity shaping (examples):**
  upper-limit approach via smooth maps such as `a_upper := tanh( c * ln(1 + x/x_ref) )`.
- **Memory update (single state):**
  `M_t := rho * M_{t-1} + (1 - rho) * s_t`, with `s_t` a dial or event surrogate.
- **Hysteresis hooks:** enter/clear thresholds applied to `a` and/or `M` to cut chatter without masking real change.

Interfaces.

- **Inputs:** contrasts `e_*`, optional side metrics (THD, unbalance, phase stats).
- **Outputs:** bounded dials `a_*`, memory `M`, and clean triggers for Section 5 (hysteresis), Section 6 (pooling), and Section 9 (policy).

Operational payoff.

- **Stable policy behavior** at limits and during ramps.
- **Predictable pooling** across devices and time.
- **Portable tuning**: once published, the same few numbers govern thousands of endpoints.

3.1 — Clamp function and invariants (eps_a)

Purpose. Keep all alignment dials `a` strictly inside `(-1,+1)` so rapidity math (`atanh`/`tanh`) stays finite, reproducible, and order-invariant.

Definition (copy-ready).

```
# Raw dial in [-∞, +∞] from any encoder logic:
a_raw := <producer-specific dial>
```

```
# Universal clamp (publish eps_a once per deployment):
a_c := max(-1 + eps_a, min(+1 - eps_a, a_raw))

# Rapidity (finite for all samples):
u := atanh(a_c)                    # u in (-∞, +∞)

# Optional reproject (identity on the clamp domain):
a := tanh(u)                       # a == a_c (within floating error)
```

Recommended guard.

```
eps_a := 1e-6     # general-purpose stacks
# consider 1e-8 only when double precision and long horizons are guaranteed
```

Invariants (must hold for every dial).

- **Boundedness:** `a ∈ (-1+eps_a, +1-eps_a)`; never hits the poles.
- **Odd symmetry:** if `a_raw` is odd in the signal, `a` remains odd.
- **Continuity:** `a(a_raw)` is continuous and piecewise-linear; no jumps except at rails.
- **Rapidity finiteness:** `|a| < 1` ⇒ `u = atanh(a)` is finite.
- **Idempotence:** `clamp(clamp(a_raw)) = clamp(a_raw)`.
- **Projection identity:** `tanh(atanh(a)) = a`.
- **Portability:** one `eps_a` across devices yields identical fused results.

Why the clamp is essential.

- Prevents `atanh(±1)` from diverging (no NaN/Inf), protecting streaming fusion and pooling.
- Stabilizes **enter/clear** thresholds and policy behavior near limits.
- Guarantees **batch == stream** parity later by keeping all fused terms finite.

Numerical sanity (quick checks).

```
a_raw = +1.0    -> a = +1 - eps_a
a_raw = -1.0    -> a = -1 + eps_a
a_raw = 0.0001  -> a = 0.0001          # unchanged; inside rails
a_raw = 10.0    -> a = +1 - eps_a      # safely saturated
```

3.2 — Proximity dials (upper-limit approach) and optional alignments

Purpose. Convert contrasts and operational limits into **bounded approach dials** that rise smoothly as a channel nears (or exceeds) a published limit, while staying quiet around nominal. Use the same clamp pipeline to keep `a ∈ (-1,+1)`.

Inputs (publish once per profile).

```
I_max, P_max, V_hi, V_lo      # operational limits
c_prox_I, c_prox_P, c_prox_V  # shaping gains (positive reals)
```

```
k_soft                          # softening factor for corridor ramps (0 <
k_soft <= 1)
```

Design shape (copy-ready).
Use a smooth, monotone map based on a logarithmic rise; then clamp.

```
# Generic proximity (approach to an upper bound X_max from a nonnegative
metric X >= 0):
prox_upper(X, X_max, c) := tanh( c * ln( 1 + X / max(X_max, eps_div) ) )

# Corridor proximity for two-sided voltage window [V_lo, V_hi]:
prox_hi(V, V_hi, c) := tanh( c * ln( 1 + max(0, V - V_hi) / max(V_hi,
eps_div) ) )
prox_lo(V, V_lo, c) := tanh( c * ln( 1 + max(0, V_lo - V) / max(V_lo,
eps_div) ) )
```

Notes: `tanh` ensures boundedness; the `ln(1+·)` tail compresses large excursions; `c` tunes sensitivity.

Dial definitions (copy-ready).

```
# Current headroom dial (approach to I_max)
a_upper_I_raw := prox_upper(I_rms, I_max, c_prox_I)
a_upper_I     := max(-1 + eps_a, min(+1 - eps_a, a_upper_I_raw))

# Active power headroom dial (approach to P_max using magnitude)
a_upper_P_raw := prox_upper(|P_raw|, P_max, c_prox_P)
a_upper_P     := max(-1 + eps_a, min(+1 - eps_a, a_upper_P_raw))

# Optional: apparent power headroom
a_upper_S_raw := prox_upper(S, S_ref or S_max, c_prox_P)
a_upper_S     := max(-1 + eps_a, min(+1 - eps_a, a_upper_S_raw))

# Optional: voltage corridor dials (entering from high or low side)
a_over_V_raw  := prox_hi(V_rms, V_hi, k_soft * c_prox_V)
a_under_V_raw := prox_lo(V_rms, V_lo, k_soft * c_prox_V)

a_over_V  := max(-1 + eps_a, min(+1 - eps_a, a_over_V_raw))
a_under_V := max(-1 + eps_a, min(+1 - eps_a, a_under_V_raw))
```

Optional alignment dials (zero-centric around targets).
When you want a symmetrical dial around a target (e.g., track how far `V_rms` is from `V_ref`), reuse contrasts:

```
# Zero-centric alignment (already bounded by choice of map)
a_V_raw := tanh( c_align_V * e_V )       # positive for over-voltage,
negative for under
a_I_raw := tanh( c_align_I * e_I )       # positive for high current
a_P_raw := tanh( c_align_P * asinh(P_raw / P_scale) )   # signed active-
power alignment

a_V := max(-1 + eps_a, min(+1 - eps_a, a_V_raw))
a_I := max(-1 + eps_a, min(+1 - eps_a, a_I_raw))
a_P := max(-1 + eps_a, min(+1 - eps_a, a_P_raw))
```

Parameter guidance.

```
# Shaping gains:
c_prox_I  ≈ 2.0      # raises sensitivity near limits without early chatter
c_prox_P  ≈ 2.0
c_prox_V  ≈ 3.0      # voltage corridors typically need crisper edges
c_align_* ≈ 1.0      # start gentle; increase only if dashboards need more
contrast
k_soft    ≈ 0.7      # softens entry into corridor dials; set 1.0 for
sharper ramps
```

Semantics.

- `a_upper_* ≈ 0` near nominal; **approaches +1** as the metric nears and exceeds its limit.
- `a_over_V` and `a_under_V` are **one-sided**: only rise when breaching the window from above or below.
- Alignment dials `a_V, a_I, a_P` are **zero-centric** and **signed**, suitable for dashboards and gentle policies.

Numerical sanity (quick checks).

```
I_rms = 0.5*I_max,  c_prox_I=2.0  -> a_upper_I ≈ tanh( 2*ln(1+0.5) )  ≈
tanh(0.81093) ≈ 0.669
I_rms = 0.9*I_max,  c_prox_I=2.0  -> a_upper_I ≈ tanh( 2*ln(1+0.9) )  ≈
tanh(1.27875) ≈ 0.856
I_rms = 1.2*I_max,  c_prox_I=2.0  -> a_upper_I ≈ tanh( 2*ln(1+1.2) )  ≈
tanh(1.56862) ≈ 0.916

V_rms = V_hi + 2%, k_soft*c_prox_V=2.1 -> a_over_V ≈ tanh( 2.1*ln(1+0.02) )
≈ tanh(0.0416) ≈ 0.0416
V_rms = V_hi + 10%, k_soft*c_prox_V=2.1 -> a_over_V ≈ tanh( 2.1*ln(1+0.10)
) ≈ tanh(0.2007) ≈ 0.198
```

Integration notes.

- Publish `I_max, P_max, V_hi, V_lo` per profile; keep them constant across the fleet to preserve comparability.
- Feed these dials directly into hysteresis (Section 5) and policy bands (Section 9).
- For pooling, treat every `a_*` as already clamped and ready for the rapidity rule.

3.3 — Bounded PQ dials (a_THD, a_unbV, a_unbI)

Purpose. Turn common power-quality indicators into **bounded, monotone dials** in `(-1,+1)` that are quiet near nominal yet rise smoothly with distortion or unbalance.

Inputs (publish once per profile).

```
c_THD, c_UNB                  # shaping gains (positive reals)
THD_ref_V, THD_ref_I          # reference THD levels (per-unit or fraction)
UNB_ref_V, UNB_ref_I          # reference unbalance levels (fraction)
method_unb_V, method_unb_I    # {DEVIATION, SYMCOMP}
eps_div, eps_a                # guards (from Section 2/3)
```

Source metrics (definitions).

```
# Harmonic distortion (per-phase)
# Voltage:  THD_V := sqrt( sum_{h>=2} V_h^2 ) / max(V_1, eps_div)
# Current:  THD_I := sqrt( sum_{h>=2} I_h^2 ) / max(I_1, eps_div)

# Unbalance — choose one method per profile:

# Method DEVIATION (phase magnitudes)
#   For X ∈ {V_rms, I_rms} with phases a,b,c:
X_avg   := (X_a + X_b + X_c) / 3
dev_max := max( |X_a - X_avg|, |X_b - X_avg|, |X_c - X_avg| )
UNB_X   := dev_max / max(X_avg, eps_div)

# Method SYMCOMP (symmetrical components, AC):
#   For voltage: UNB_V := |V2| / max(|V1|, eps_div)
#   For current: UNB_I := |I2| / max(|I1|, eps_div)
```

Bounded PQ dials (copy-ready).

```
# Distortion dials (monotone, bounded)
a_THD_V_raw := tanh( c_THD * ln( 1 + THD_V / max(THD_ref_V, eps_div) ) )
a_THD_I_raw := tanh( c_THD * ln( 1 + THD_I / max(THD_ref_I, eps_div) ) )

a_THD_V := max(-1 + eps_a, min(+1 - eps_a, a_THD_V_raw))
a_THD_I := max(-1 + eps_a, min(+1 - eps_a, a_THD_I_raw))

# Unbalance dials (choose method; same shape)
a_unbV_raw := tanh( c_UNB * ln( 1 + UNB_V / max(UNB_ref_V, eps_div) ) )
a_unbI_raw := tanh( c_UNB * ln( 1 + UNB_I / max(UNB_ref_I, eps_div) ) )

a_unbV := max(-1 + eps_a, min(+1 - eps_a, a_unbV_raw))
a_unbI := max(-1 + eps_a, min(+1 - eps_a, a_unbI_raw))
```

Semantics.

- `a_* ≈ 0` when the metric is near its reference; rises toward `+1` as distortion/unbalance grows.
- The `ln(1+·)` tail keeps large excursions compressed; `tanh` enforces boundedness.

Parameter guidance.

```
# References (fractions):
THD_ref_V ≈ 0.03   # 3% voltage THD reference
THD_ref_I ≈ 0.10   # 10% current THD reference
UNB_ref_V ≈ 0.02   # 2% voltage unbalance reference
UNB_ref_I ≈ 0.05   # 5% current unbalance reference
```

```
# Shaping gains:
c_THD  ≈ 2.0
c_UNB  ≈ 2.0
# Increase c_* for crisper rise (more sensitive near reference).
```

Numerical sanity (quick checks).

```
THD_V = 3%  , THD_ref_V=3%,  c_THD=2.0 -> a_THD_V ≈ tanh( 2*ln(1+0.03/0.03)
) = tanh(2*ln(2)) ≈ tanh(1.386) ≈ 0.883
THD_V = 1.5%, THD_ref_V=3%,  c_THD=2.0 -> a_THD_V ≈ tanh( 2*ln(1+0.5) )
= tanh(0.811)   ≈ 0.669

UNB_V (DEV) = 1%,  UNB_ref_V=2%, c_UNB=2.0 -> a_unbV ≈ tanh( 2*ln(1+0.5) )
≈ 0.669
UNB_V (SYM) = 3%,  UNB_ref_V=2%, c_UNB=2.0 -> a_unbV ≈ tanh( 2*ln(1+1.5) )
≈ tanh(1.832) ≈ 0.950
```

Integration notes.

- Publish **one** unbalance method per deployment for comparability.
- Feed `a_THD_*` and `a_unb*` to hysteresis and policy bands; they are already clamped and ready for pooling.

3.4 Single shared memory dial `M` (definition, `rho` guidance)

Purpose.
Provide one lightweight, bounded memory that **soaks up flicker**, **stabilizes thresholds**, and yields **repeatable behavior** across devices and time.

Definition (copy-ready)

```
# Event surrogate s_t in [-1,+1]  (choose one; see patterns below)
# Memory state in (-1,+1), initialized at zero unless a profile says
otherwise.

M_0 := 0
rho in (0,1)        # retention (closer to 1 = longer memory)

# Update (single line):
M_t := rho * M_{t-1} + (1 - rho) * s_t
```

Guaranteed properties.

If `s_t ∈ [-1,+1]` and `|M_0| < 1` ⇒ `M_t ∈ (-1,+1)` for all `t`.

- **Sign-consistency:** `sign( M_t )` tends to `sign( mean s_t )` when bias persists.
- **Exponential forgetting:** an impulse in `s_t` decays as `rho^k` after `k` steps.

How to choose the surrogate `s_t` (common patterns)

```
# 1) Headroom-driven memory (unsigned pressure toward limits)
s_t := max( a_upper_I , a_upper_P , a_upper_S , a_over_V , a_under_V )

# 2) Quality-driven memory (distortion / unbalance emphasis)
s_t := max( a_THD_V , a_THD_I , a_unbV , a_unbI )

# 3) Mixed operational memory (recommended starter)
s_headroom := max( a_upper_I , a_upper_P , a_over_V , a_under_V )
s_quality  := max( a_THD_V   , a_THD_I   , a_unbV   , a_unbI   )
s_t        := max( s_headroom , s_quality )

# 4) Signed alignment memory (preserve direction, e.g., over vs under
voltage)
s_t := a_V         # or any signed alignment dial such as a_P
```

- Pick **one definition per profile** for portability.
- If you need multiple behaviors, keep **one primary `M`** and derive secondary views (for example, thresholds or derived flags) without changing the primary definition.

Relating `rho` to a time constant

```
# With sampling period Δt (seconds) and desired memory time constant τ
(seconds):
rho := exp( -Δt / τ )
```

Examples (rounded):

- `Δt = 0.1 s, τ = 5 s` ⟶ `rho ≈ exp( -0.1 / 5 ) ≈ 0.9802`
- `Δt = 1.0 s, τ = 10 s` ⟶ `rho ≈ exp( -1 / 10 ) ≈ 0.9048`
- `Δt = 1.0 s, τ = 30 s` ⟶ `rho ≈ exp( -1 / 30 ) ≈ 0.9672`
- `Δt = 1.0 s, τ = 120 s` ⟶ `rho ≈ exp( -1 / 120 ) ≈ 0.9917`

Recommended starting points.

- Use `τ ≈ 20–60 s` for **operations dashboards** and chatter control.
- Use `τ ≈ 5–10 s` for **responsive protection-like behaviors**.
- Use `τ ≈ 120–300 s` for **slow-moving fleet summaries**.

Clip-free implementation (copy-ready)

```
# Inputs: s_t in [-1,+1], rho in (0,1)
# State:  M_{t-1} in (-1,+1)  (init M_0 := 0, unless restoring from
checkpoint)

M_t := rho * M_{t-1} + (1 - rho) * s_t

# No clamp needed if inputs respect bounds; optional safety clamp:
M_t := max( -1 + eps_a , min( +1 - eps_a , M_t ) )
```

- **Cold start:** if not otherwise specified in a manifest, assume `M_0 := 0`.

Why a single memory is enough

- **Parsimony.** One published `rho` yields **consistent, human-visible behavior** across all dials.
- **Composability.** Upstream dials already compress extremes; `M` only **smooths decisions**, not raw physics.
- **Portability.** The `τ ↔ rho` mapping makes behavior **reproducible across different sampling rates** and devices.

Optional extension (still single output)

```
# Dual-kernel smoothing without adding new state to the payload:

M_fast := rho_fast * M_fast_prev + (1 - rho_fast) * s_t
M_slow := rho_slow * M_slow_prev + (1 - rho_slow) * s_t
M      := w * M_fast + (1 - w) * M_slow      # w in [0,1], M stays in (-
1,+1)
```

Use this only if you must stabilize **both short spikes and slow drifts** while still **publishing one `M`** in the payload.

Numerical sanity (quick checks)

```
Case 1:  s_t = 0.8 for t = 1..∞,  rho = 0.9
M_1 = 0.08,  M_2 = 0.152,  ...  →  M_t → 0.8

Case 2:  s_1 = 1.0 then s_t = 0 for t >= 2,  rho = 0.95
M_1 = 0.05,  M_2 = 0.0475,  M_3 = 0.0451  (decays approximately as 0.95^k)
```

Publication checklist (paste into profile)

```
publish:
  memory:
    definition: "M_t := rho*M_{t-1} + (1 - rho)*s_t"
    surrogate:  "s_t := max(a_upper_I, a_upper_P, a_over_V, a_under_V,
a_THD_V, a_unbV)"
    tau_seconds: 30
    sampling_seconds: 1
    rho: 0.9672

notes:
  M_0 := 0  # unless restoring from a checkpoint or replay context
```

3.5 — Copy-ready snippets (S1 / S2 / S3 options)

Purpose. Ready-to-paste payload and encoder options that scale from **minimal contrasts** to a **full posture** with dials and **single memory** `M`, without changing semantics later.

Option S1 — Minimal (contrasts only, zero-centric)
What it carries: `e_V, e_I, e_f, e_P` *(optionally* `e_Q, e_S, e_pf`*)*
When to use: earliest integration; dashboards can compute dials downstream.

Payload (JSON, per sample).

```
{
  "ts": "<ISO-8601>",
  "e_V": <real>, "e_I": <real>, "e_f": <real>, "e_P": <real>,
  "e_Q": <real>, "e_S": <real>, "e_pf": <real>
}
```

Encoder (single block).

```
# Given guarded raw channels and declared lenses:
e_V := ln(V_rms / V_ref)                              # or linear/hybrid per
profile
e_I := ln(I_rms / I_ref)
e_f := ln(f / f_ref)
e_P := asinh(P_raw / P_scale)
e_Q := asinh(Q_raw / Q_scale)
e_S := ln(S / S_ref)
pf  := min(1.0, max(pf_min, |P_raw| / max(S, eps_div)))
e_pf := ln(pf / pf_ref)
```

Determinism. Values are pure functions of the sample and profile; no state.

Option S2 — Standard (contrasts + key dials)
What it carries: S1 + selected **bounded dials** ready for policy/hysteresis.
When to use: operations with limits and PQ monitoring.

Payload (JSON, per sample).

```
{
  "ts": "<ISO-8601>",
  "e_V": <real>, "e_I": <real>, "e_f": <real>, "e_P": <real>,
  "a_upper_I": <real>, "a_upper_P": <real>, "a_over_V": <real>,
"a_under_V": <real>,
  "a_THD_V": <real>, "a_THD_I": <real>, "a_unbV": <real>, "a_unbI": <real>
}
```

Encoder (add dials; all clamped with eps_a).

```
a_upper_I := tanh( c_prox_I * ln(1 + I_rms / max(I_max, eps_div)) )
a_upper_P := tanh( c_prox_P * ln(1 + |P_raw| / max(P_max, eps_div)) )
a_over_V  := tanh( (k_soft*c_prox_V) * ln(1 + max(0, V_rms -
V_hi)/max(V_hi, eps_div)) )
a_under_V := tanh( (k_soft*c_prox_V) * ln(1 + max(0, V_lo -
V_rms)/max(V_lo, eps_div)) )

a_THD_V := tanh( c_THD * ln(1 + THD_V / max(THD_ref_V, eps_div)) )
a_THD_I := tanh( c_THD * ln(1 + THD_I / max(THD_ref_I, eps_div)) )
a_unbV  := tanh( c_UNB * ln(1 + UNB_V  / max(UNB_ref_V, eps_div)) )
a_unbI  := tanh( c_UNB * ln(1 + UNB_I  / max(UNB_ref_I, eps_div)) )

# Final clamp:
a_* := max(-1 + eps_a, min(+1 - eps_a, a_*))
```

Determinism. Still stateless; batch == stream trivially.

Option S3 — Full posture (contrasts + dials + single memory м)
What it carries: S2 plus **one portable memory** `M` for chatter control and repeatable triggers.
When to use: production policy packs and fleet pooling.

Payload (JSON, per sample).

```
{
  "ts": "<ISO-8601>",
  "e_V": <real>, "e_I": <real>, "e_f": <real>, "e_P": <real>,
  "a_upper_I": <real>, "a_upper_P": <real>, "a_over_V": <real>,
"a_under_V": <real>,
  "a_THD_V": <real>, "a_THD_I": <real>, "a_unbV": <real>, "a_unbI": <real>,
  "M": <real>
}
```

Encoder (adds memory update).

```
# Choose surrogate once per profile (example mixed surrogate):
s_t := max(a_upper_I, a_upper_P, a_over_V, a_under_V, a_THD_V, a_unbV)

# Memory (rho from tau, Δt):
M_t := rho * M_{t-1} + (1 - rho) * s_t
M_t := max(-1 + eps_a, min(+1 - eps_a, M_t))   # safety clamp
```

State handling.

```
# Initialize:
M_0 := 0   # or restored from a checkpoint

# On restart without checkpoint:
set M := 0 for deterministic behavior
```

Drop-in producer wiring (common to S1/S2/S3).

```
inputs:
  V_rms_raw, I_rms_raw, S_raw, P_raw, Q_raw, f_raw
  THD_V, THD_I, UNB_V, UNB_I     # if S2/S3
params:
  anchors/scales/lenses from Section 2
  limits & PQ refs from Section 3
  guards: eps_pos, eps_div, eps_a, pf_min, alpha
  memory: rho (S3 only)

steps:
  1) apply guards to raw channels used by LOG/denominators
  2) compute contrasts e_*
  3) (S2/S3) compute dials a_* and clamp
  4) (S3) update M
  5) emit payload (JSON)
```

Choosing between S1 / S2 / S3.

- Start with **S1** for easiest drop-in; upgrade to **S2** when limits/PQ matter; adopt **S3** for policies and fleet summaries that must be **quiet**, **predictable**, and **portable**.

SECTION 4 — Power Identity Residuals (Consistency)

Intent. Provide a **fast, portable consistency check** across voltage, current, power, and power factor using **log-contrast identities**. When anchors are declared coherently and sensors agree, the residuals sit near zero; **persistent bias** reveals **ratio, sign, or scaling faults** immediately.

What this section establishes.

- A canonical **active-power identity residual** that compares `|P|` against `V * I * pf` in log space:
  `r_P := e_P_log - ( e_V_log + e_I_log + e_pf )`
- A clean **anchor consistency rule** so residuals are **zero-centric at nominal**:
  `P_ref := V_ref * I_ref * pf_ref`
- Practical **guards** for `pf → 0`, `S → 0`, and near-zero conditions that keep math defined without masking faults.
- A clear separation of **magnitude identity** (via `|P|`) from **sign semantics** (kept as a side-channel), aiding triage without polluting the residual.
- A parallel, optional **apparent-power check** when useful: compare `S` against `V * I` in log space for a quick sanity cross-check.

Design goals.

- **Zero-centricity:** residuals target `0` at nominal anchors.
- **Unit-invariance:** all terms are **ratios** (`ln(x/x_ref)`), so units never disturb results.
- **Monotone fault visibility:** consistent bias in sensors or ratios ⇒ residual moves away from zero and stays there.
- **Safety at edges:** use small floors only to prevent `ln(0)`, not to conceal errors.
- **AC/DC coherence:** identical structure with or without `pf` (for DC, drop the `pf` term cleanly).

Building blocks.

- **Log contrasts:** `e_V_log := ln(V_rms/V_ref)`, `e_I_log := ln(I_rms/I_ref)`, `e_pf := ln(pf/pf_ref)`, `e_P_log := ln(max(|P|,eps_pos)/P_ref)`.
- **Residual core:** `r_P := e_P_log - ( e_V_log + e_I_log + e_pf )`.
- **Edge guards:** `pf := min(1, max(pf_min, |P|/max(S,eps_div)))`.
- **Optional siblings:** `r_S := ln(S/S_ref) - ( ln(V_rms/V_ref) + ln(I_rms/I_ref) )` for quick apparent-power sanity.

Interfaces.

- **Inputs:** `V_rms, I_rms, P, S, pf` (or `pf` derived), plus anchors `*_ref`.
- **Outputs:** `r_P` (and optionally `r_S`) per sample, plus an optional `sgn_P` side-channel for triage.
- **Downstream:** Section 4.2 interprets bias patterns; Section 11 collects common fault signatures built on these residuals.

Operational payoff.

- **One glance triage:** near-zero residuals confirm cross-channel coherence; sustained offsets spotlight calibration or wiring faults (e.g., CT/PT ratio, pf handling, sign flips).
- **Portable commissioning:** same formula, same anchors ⇒ identical thresholds across fleets.
- **Robustness:** minor noise vanishes in log-ratio balance; real inconsistencies persist and are easy to act on.

4.1 Core power-identity residual `r_P` (with `pf_min` guard)

Purpose.
Detect inconsistencies among `V`, `I`, and `P` using the identity `|P| ≈ V * I * pf`. In **log-contrast form**, the three channels should balance; persistent deviation indicates **ratio / sign / scaling** issues (for example CT/PT error, flipped sign, or mismatched anchors).

Anchor consistency (publish once)

For AC profiles with power factor:

```
P_ref := V_ref * I_ref * pf_ref
```

- `V_ref`, `I_ref`, `pf_ref` are taken from the **anchor declarations**.
- Prefer setting anchors so that this identity holds exactly (`P_ref = V_ref * I_ref * pf_ref`).

Log-power contrasts for identity (copy-ready)

```
# Guards:
pf_min in (0,1]      # lower bound for pf to avoid log(0)
eps_pos > 0          # positivity floor for logs
eps_div > 0          # floor for denominators
alpha  >= 0          # tiny relative floor factor
```

Floors for quantities used in logs / denominators:

```
S_safe := max( S_raw , max( eps_pos , alpha * S_ref ) )
V_safe := max( V_rms , max( eps_pos , alpha * V_ref ) )
I_safe := max( I_rms , max( eps_pos , alpha * I_ref ) )
```

Power factor, bounded away from 0:

```
pf_num := | P_raw |
pf_den := max( S_safe , eps_div )
pf     := pf_num / pf_den
pf     := min( 1.0 , max( pf_min , pf ) )

# pf is a magnitude in (pf_min, 1]; direction lives in Q + reactive_sign.
```

Log-contrast channels for identity:

```
e_V_log := ln( V_safe / V_ref )
e_I_log := ln( I_safe / I_ref )
e_pf    := ln( pf / pf_ref )
```

Log-power contrast (magnitude of P):

```
e_P_log := ln( max( |P_raw| , eps_pos ) / P_ref )
```

Residual definition (copy-ready)

```
r_P := e_P_log - ( e_V_log + e_I_log + e_pf )
```

Properties (with coherent anchors and correct sensors).

```
r_P ≈ 0              # zero-centric at nominal and under consistency
```

- **Unit-invariant:** ratios remove dependence on volts vs kilovolts, watts vs kilowatts, etc.
- **Sign-agnostic:** uses `|P|`; sign is handled separately if needed.

Optional sign side-channel (for triage, not part of `r_P`)

```
sgn_P := sign( P_raw )   # +1 import, -1 export per published power_sign convention
```

This can help distinguish **"wrong magnitude" vs "wrong sign"** during triage without contaminating `r_P`.

Numerical notes

- `pf_min` prevents `ln(0)` when `S → 0` or `pf → 0`; it also avoids **explosive residuals** at zero load.
- If anchors cannot satisfy `P_ref = V_ref * I_ref * pf_ref`, introduce a fixed correction term:

```
k   := ln( P_ref / ( V_ref * I_ref * pf_ref ) )
r_P := ( e_P_log - k ) - ( e_V_log + e_I_log + e_pf )
```

- Prefer the **consistent-anchor form** (`k = 0`) whenever possible; it is simpler to explain and audit.

DC profiles

For DC (no power-factor term in the identity):

```
P_ref := V_ref * I_ref

r_P := ln( max( |P_raw| , eps_pos ) / P_ref )
       - ( ln( V_safe / V_ref ) + ln( I_safe / I_ref ) )
```

- `pf` and `e_pf` are omitted for **pure DC** profiles; the residual checks consistency of `V`, `I`, and `P` alone.

Emission (per sample)

```
emit: { "r_P": <real> }          # optionally also "sgn_P"
```

- In addition to emitting `r_P`, profiles should declare a **tolerance band and dwell** (for example in `residuals.r_P`): how large and how persistent `r_P` must be before raising an alarm.

4.2 Interpreting `r_P` (bias patterns and quick triage)

Purpose.
Turn the scalar residual `r_P` into **fast, actionable triage**. Near-zero means coherence; sustained positive or negative bias fingerprints specific faults.

Sign convention (reminder)

```
r_P := e_P_log - ( e_V_log + e_I_log + e_pf )
```

with

- `e_X_log := ln( X / X_ref )`
- `e_pf := ln( pf / pf_ref )`

Interpretation rule of thumb:

- `r_P > 0` ⇒ measured `|P|` **relatively too high** (vs `V * I * pf`)
- `r_P < 0` ⇒ measured `|P|` **relatively too low**

Fingerprint map (copy-ready)

```
# 1) CT ratio error (current scaling)
Symptom:  r_P shows a roughly constant bias; magnitude ≈ ln(k_CT)
          where k_CT = I_meas / I_true
Clue:     e_I_log drifts (anchors stable); e_V_log and e_pf near nominal.
Sign:     CT ratio too small  (I underestimated):   e_I_log ↓ ⇒ r_P > 0
          CT ratio too large  (I overestimated):    e_I_log ↑ ⇒ r_P < 0
Action:   Verify CT ratio/nameplate; correct I_ref or scaling.

# 2) PT ratio error (voltage scaling)
Symptom:  r_P shows a roughly constant bias; magnitude ≈ ln(k_PT)
          where k_PT = V_meas / V_true
Clue:     e_V_log drifts (anchors stable); e_I_log and e_pf near nominal.
Sign:     PT too small  (V underestimated):         e_V_log ↓ ⇒ r_P > 0
          PT too large  (V overestimated):          e_V_log ↑ ⇒ r_P < 0
Action:   Verify PT ratio/nameplate; correct V_ref or scaling.
```

```
# 3) pf handling error (pf stuck or miscomputed)
Symptom:  r_P follows ln(pf_true / pf_meas); bias varies with loading.
Clue:     e_pf inconsistent with operating regime; pf near 1 while VARs
present.
Sign:     pf_meas too large  ⇒ e_pf ↑ ⇒ r_P < 0
          pf_meas too small  ⇒ e_pf ↓ ⇒ r_P > 0
Action:   Check pf estimator (definition of S, inclusion of harmonics, sign
logic).

# 4) P sign flip (import/export reversed)
Symptom:  r_P near zero (uses |P|), but operations report "direction
wrong".
Clue:     sgn_P inconsistent with published convention; alarms/policies
misfire.
Action:   Fix sign convention once in the profile; do not patch downstream
logic.

# 5) Apparent power S misdefinition (e.g., RMS vs fundamental mismatch)
Symptom:  r_P drifts with distortion/unbalance; improves when loads are
clean.
Clue:     pf := |P| / S off when THD rises; e_pf biased at high THD.
Action:   Align S and P definitions (both fundamental or both total; stay
consistent).

# 6) Phase swap or order error (per-phase systems)
Symptom:  Per-phase residuals show large spread; pooled residual smaller.
Clue:     r_P_a, r_P_b, r_P_c inconsistent; e_V_log phases misaligned with
currents.
Action:   Verify phase mapping CT/PT → channels; correct wiring and phase
order.

# 7) Mixed anchor inconsistency (P_ref ≠ V_ref*I_ref*pf_ref)
Symptom:  r_P biased by constant k = ln( P_ref / ( V_ref * I_ref * pf_ref )
).
Clue:     Residual shows a stable offset across all loads and operating
points.
Action:   Re-anchor P_ref := V_ref * I_ref * pf_ref (preferred), or
subtract k explicitly.

# 8) Sensor saturation / clipping
Symptom:  r_P spikes only at peaks, otherwise near zero.
Clue:     Clipping flags, flat-topped waveforms, counters near limits.
Action:   Increase range or trust only unsaturated segments; avoid using
r_P in clipped zones.
```

Quick orientation examples (small-signal)

```
Example A — CT 10% low (I underestimated):
e_I_log ≈ ln(0.9) = -0.1053
r_P ≈ e_P_log - (e_V_log + e_I_log + e_pf)
    ≈ 0 - (0 + (-0.1053) + 0) = +0.1053    # positive bias

Example B — PT 5% high (V overestimated):
e_V_log ≈ ln(1.05) = +0.0488
r_P ≈ 0 - ( +0.0488 + 0 + 0 ) = -0.0488    # negative bias

Example C — pf overreported (true 0.9, measured 1.0, pf_ref = 1):
e_pf_true = ln(0.9) = -0.1053
e_pf_meas = ln(1.0) = 0
```

```
r_P_meas ≈ 0 - (0 + 0 + 0) = 0
r_P_true ≈ 0 - (0 + 0 -0.1053) = +0.1053

Interpretation: measured pf too large ⇒ observed r_P is too negative (or
too close to 0)
relative to the "true" balanced value.
```

Per-phase workflow (three-phase)

1. Compute `r_P_a`, `r_P_b`, `r_P_c` using **per-phase** `V`, `I`, `P`, and `pf` (or `P`–`Q`–`S`).
2. Inspect **spread and sign consistency**:
   - **Uniform bias** across `a/b/c` ⇒ common anchor or ratio issue (CT/PT/anchor mismatch).
   - **One-phase outlier** ⇒ likely single CT/PT, wiring, or configuration fault.
3. Pool per-phase residuals **only after** per-phase triage, and only when it aids dashboard summarization.

Operational thresholds (starter)

```
|r_P| <= 0.03           => normal (≈ ±3% band)
0.03 < |r_P| <= 0.10    => investigate (ratio / pf handling / S definition)
|r_P| > 0.10            => likely configuration or wiring error
```

- Tune these based on **sensor class**, expected load mix, and **harmonic environment**.

Emission suggestion

```
emit: {
  "r_P":  <real>,             # core residual
  "sgn_P": <+1 | -1>,         # sign side-channel from P
  "hint": "<CT|PT|pf|phase|anchor|S|clip>"   # optional advisory tag
}
```

- `hint` may be populated by simple rule-based logic using the fingerprint map.
- Keep `hint` **advisory**, not normative; the normative part is the numeric `r_P` and its declared tolerance band.

4.3 Copy-ready residual block (`r_P`) + two worked examples

Purpose.
Provide a paste-in computation of the active-power identity residual `r_P` with **safe guards**, plus two tiny examples to validate commissioning and anchor consistency.

Copy-ready residual block (paste as-is)

```
# Inputs (per sample):
#   V_rms_raw, I_rms_raw, P_raw, S_raw
#   V_ref, I_ref, S_ref, pf_ref
#   eps_pos, eps_div, pf_min, alpha   # small positive guards

# 1) Safe floors for log/denominator channels
V_safe := max( V_rms_raw , max( eps_pos , alpha * V_ref ) )
I_safe := max( I_rms_raw , max( eps_pos , alpha * I_ref ) )
S_safe := max( S_raw     , max( eps_pos , alpha * S_ref ) )

# 2) Power factor (bounded away from 0 and 1)
pf_num := | P_raw |
pf_den := max( S_safe , eps_div )
pf     := pf_num / pf_den
pf     := min( 1.0 , max( pf_min , pf ) )

# 3) Log contrasts for identity
e_V_log := ln( V_safe / V_ref )
e_I_log := ln( I_safe / I_ref )
e_pf    := ln( pf / pf_ref )

# 4) Anchor-consistent P_ref and log-power contrast
P_ref   := V_ref * I_ref * pf_ref
e_P_log := ln( max( |P_raw| , eps_pos ) / P_ref )

# 5) Residual (active-power identity)
r_P := e_P_log - ( e_V_log + e_I_log + e_pf )

# Optional side-channel for direction (not part of r_P)
sgn_P := sign( P_raw )    # +1 import, -1 export per published convention

# Emit:
#   { "r_P": r_P }         # optionally also "sgn_P"
```

Example 1 — Nominal operating point (sanity zero)

Assume:

```
V_ref  = 230
I_ref  = 10
pf_ref = 1.0

eps_pos = 1e-12
eps_div = 1e-12
pf_min  = 0.05
alpha   = 0
```

Measured sample:

```
V_rms_raw = 230
I_rms_raw = 10
P_raw     = 2300
S_raw     = 2300
```

Step-by-step:

```
V_safe = 230
I_safe = 10
S_safe = 2300

pf     = |2300| / 2300 = 1.0

e_V_log = ln( 230 / 230 ) = 0
e_I_log = ln( 10 / 10 )   = 0
e_pf    = ln( 1.0 / 1.0 ) = 0

P_ref   = 230 * 10 * 1.0 = 2300
e_P_log = ln( |2300| / 2300 ) = 0

r_P     = 0 - ( 0 + 0 + 0 ) = 0
```

Interpretation.
Zero residual at the anchors confirms **coherent references and sensors** and validates the implementation of the residual block.

Example 2 — CT ratio 10% low (current underestimated)

Same anchors and guards as Example 1.

Measured sample (current under-reported, power still physically correct):

```
V_rms_raw = 230
I_rms_raw = 9          # 10% low vs true 10 A
P_raw     = 230 * 10   = 2300
S_raw     = V_rms_raw * I_rms_raw = 2070
```

Step-by-step:

```
V_safe = 230
I_safe = 9
S_safe = 2070

pf_num := |2300|
pf_den := max( 2070 , eps_div ) = 2070
pf     := pf_num / pf_den ≈ 1.111...
pf     := min( 1.0 , max( 0.05 , 1.111... ) ) = 1.0

e_V_log = ln( 230 / 230 )   = 0
e_I_log = ln( 9 / 10 )      ≈ -0.1053605
e_pf    = ln( 1.0 / 1.0 )   = 0

P_ref   = 230 * 10 * 1.0 = 2300
e_P_log = ln( |2300| / 2300 ) = 0

r_P     = 0 - ( 0 + (-0.1053605) + 0 )
        ≈ +0.1053605
```

Interpretation.
Positive `r_P` (approximately `ln(10/9)`) flags that measured current is **too small** relative to `V` and `P` — a classic **CT under-reporting / ratio error** pattern.

Commissioning tip (threshold starter)

```
|r_P| <= 0.03           => normal (≈ ±3% band)
0.03 < |r_P| <= 0.10    => investigate (ratio / pf handling / S definition)
|r_P| > 0.10            => likely ratio, sign, or configuration error
```

Tune these thresholds per **sensor class**, **grid segment**, and expected **loading diversity**.

4.4 — Residual Sensitivity Benchmarks (Noise & CT/PT Scenarios)

(Normative, Copy-Ready)

Purpose. Provide a small set of numeric micro-benchmarks to validate that the residual

```
r_P := e_P_meas - ( e_V + e_I + ln(pf_eff / pf_ref) )
```

behaves as intended under realistic conditions:

- **Stable near zero** under small noise (no error).
- **Clearly non-zero** under CT/PT ratio error.
- **Strongly non-zero** under sign flip.
- **Safe and bounded** when PF is near zero (via `pf_min` / `pf_floor`).

These benchmarks can be implemented as a **tiny unit test suite** and included in the Repro Pack.

Assumptions (unless otherwise noted):

```
pf_ref  = 1.0
pf_min  = 0.3          # example floor; site-specific
pf_eff  = max(pf, pf_min)
V_ref   = 230.0 V
I_ref   = 10.0 A
P_ref   = V_ref * I_ref * pf_ref = 2300.0 W
```

Alignments `e_V`, `e_I`, `e_P_meas` are generic "electrical lenses" (Section 4) that increase in magnitude with deviation from anchors but remain bounded and smooth. You may use any monotone, zero-centric mapping (e.g., log-ratio or small-signal linear) as long as it is **consistent** across all tests.

4.4.1 — Benchmark 1: Baseline (No Error, Small Noise)

Intent. Show that `r_P` stays near zero when measurements are internally consistent, even with small noise on V, I, and PF.

Example inputs (one tick).

```
V_meas = 228.0 V      # ~ -0.9% from 230 V
I_meas = 9.80  A      # ~ -2.0% from 10 A
pf     = 0.98         # small phase shift, close to 1.0
P_meas = 228.0 * 9.80 * 0.98 ≈ 2188.7 W
```

*Illustrative lens outputs (example; your exact e_ may differ):**

```
e_V      ≈ -0.010     # small negative deviation
e_I      ≈ -0.020
ln(pf_eff / pf_ref)
         ≈ ln(0.98 / 1.0) ≈ -0.0202
e_P_meas ≈ e_V + e_I + ln(pf_eff / pf_ref) + small_noise
         ≈ -0.050 ± 0.002
```

Residual expectation.

```
r_P = e_P_meas - ( e_V + e_I + ln(pf_eff / pf_ref) )
    ≈ 0.0 ± 0.002
```

Pass criterion (baseline).

```
|r_P| <= tol_rP_baseline
tol_rP_baseline ≈ 0.01   # example; site may choose tighter/looser
```

Outcome.

- **Expected:** `PASS` (baseline sanity).
- **Interpretation:** lenses are internally consistent; residual reflects only small numerical noise.

4.4.2 — Benchmark 2: CT Ratio Error (+2 % on Current)

Intent. Show that a small CT ratio error produces a **clear, non-zero** residual that exceeds baseline tolerance but is still moderate (detectable, not explosive).

Scenario.

- Physical reality: true current `I_true = 10.0 A`.
- CT scaling: meter sees `I_meas = 1.02 * I_true = 10.2 A` due to +2 % CT error.
- Voltage and PF are fine.

Example inputs.

```
V_meas = 230.0 V
I_meas = 10.2  A
pf     = 1.00
P_meas = 230.0 * 10.2 * 1.0 = 2346.0 W
```

Illustrative lens outputs (example).

```
e_V      ≈  0.000     # perfectly on reference
e_I      ≈ +0.020     # approx +2% deviation
ln(pf_eff / pf_ref)
         ≈ 0.0

# if e_P_meas tracks P deviation:
P_ratio  = P_meas / P_ref = 2346.0 / 2300.0 ≈ 1.02
e_P_meas ≈ +0.020
```

Residual expectation.

```
r_P = e_P_meas - ( e_V + e_I + ln(pf_eff / pf_ref) )
    ≈ +0.020 - ( 0 + 0.020 + 0 )
    ≈ 0.0   (if CT error is *consistently* seen in both I and P lenses)

# Now inject a *hidden* CT bias only in I (simulating lens mismatch):
e_P_meas ≈ 0.000
r_P      ≈ 0.000 - ( 0 + 0.020 + 0 ) = -0.020
```

Pass criterion (CT mismatch detection).

```
|r_P| > tol_rP_baseline   AND   |r_P| <= tol_rP_CT
tol_rP_baseline ≈ 0.01
tol_rP_CT       ≈ 0.05   # region we expect "CT warning, not catastrophic"
```

Outcome.

- If both I and P share the same scaling → `r_P ≈ 0` → **no CT error indicated** (consistent bias).
- If only I lens sees the bias, P does not → `r_P ≈ -0.02` → **EXPECTED: WARN / ALERT** depending on `tol_rP`.

4.4.3 — Benchmark 3: Sign Flip on One Channel (Serious Wiring Error)

Intent. Show that a **polarity error** (e.g., P or I sign flipped) produces a **large residual**, clearly above any tolerance, indicating a CT/PT wiring issue or phase swap.

Scenario A — Power sign flipped (P negative, V & I positive, pf > 0).

```
V_meas = 230.0 V
I_meas = 10.0 A
pf     = 0.99
P_meas = - (230.0 * 10.0 * 0.99) ≈ -2277.0 W   # sign error
```

Illustrative lens outputs.

```
e_V      ≈ 0.000
e_I      ≈ 0.000
ln(pf_eff / pf_ref)
         ≈ ln(0.99) ≈ -0.010
e_P_meas ≈ large_negative   # because measured P is "opposite" expected

# For example, if we use a log-ratio lens on |P| with sign awareness:
P_ratio  = P_meas / P_ref ≈ -0.99
e_P_meas ≈ f(P_ratio) ≈ -O(1)  # strong negative
```

Residual expectation (conceptual).

```
r_P = e_P_meas - ( e_V + e_I + ln(pf_eff / pf_ref) )
    ≈ (large_negative) - ( 0 + 0 - 0.010 )
    ≈ large_negative
|r_P| >> 0.1    # far beyond normal limits
```

Pass criterion (sign flip detection).

```
|r_P| >= tol_rP_signflip
tol_rP_signflip ≈ 0.1 (or higher, depending on lens gains)
```

Outcome.

- **Expected:** hard **ALERT / CRITICAL** (wiring error).
- Visual: residual chart spikes sharply, easily distinguishable from small CT errors.

Scenario B — Current sign flipped (I negative, P still computed as V*I*pf).

- The exact numeric pattern will differ per lens, but the principle is the same: `e_I` and `e_P_meas` move in opposite directions, yielding a large |`r_P`|.

4.4.4 — Benchmark 4: PF Near Zero (Why We Use pf_min / pf_floor)

Intent. Show that `pf_min` (or `pf_floor`) prevents numerical blow-up and spurious residuals when PF approaches zero — e.g., during energization or no-load conditions with strong reactive components.

Scenario.

```
V_meas = 230.0 V
I_meas = 5.0   A
pf     = 0.05  # very small PF, mostly reactive
P_meas = 230.0 * 5.0 * 0.05 ≈ 57.5 W
```

Without PF floor (unsafe).

```
ln(pf / pf_ref) = ln(0.05 / 1.0) ≈ ln(0.05) ≈ -2.9957  # huge negative
# This single term can dominate residual behavior.
```

With PF floor (safe).

```
pf_min  = 0.3
pf_eff  = max(pf, pf_min) = 0.3
ln(pf_eff / pf_ref) = ln(0.3) ≈ -1.2046  # still negative, but controlled
```

Illustrative lens outputs.

```
e_V      ≈ 0.000
e_I      ≈ small_positive      # load below I_ref
e_P_meas ≈ function of P_ratio # small positive or near zero
```

Residual comparison.

- **Unsafe (no floor):**
  `r_P` dominated by ≈ −3.0 → enormous magnitude, looks like a catastrophic mismatch.
- **Safe (with floor):**
  `r_P` reflects that PF is low but bounded, typically **flagging "PF low" via a_pf**, while `r_P` itself stays within a manageable range.

Pass criterion (PF floor behavior).

```
With pf_floor enabled:
  |r_P| <= tol_rP_pf_low   # moderate residual
Without pf_floor:
  |r_P| >> tol_rP_pf_low   # strongly violates benchmark → design failure
```

Example values:

```
tol_rP_pf_low ≈ 0.2
```

Outcome.

- With `pf_min` / `pf_floor` declared in the manifest: **PASS** (bounded behavior).
- Without PF protection: residual becomes misleading → **HARNESS FAIL** (implementation bug).

4.4.5 — Residual Benchmark Table (Copy-Ready Summary)

You may embed this table directly into a **residual_test_pack.csv**:

```
scenario_id,description,V_meas,I_meas,pf,P_meas,expected_rP_band,tol_rP,sta
tus
BASELINE,small noise,228.0,9.80,0.98,2188.7,NEAR_ZERO,0.01,EXPECT_PASS
CT_BIAS_+2,CT ratio +2% on
I,230.0,10.2,1.00,2300.0,SMALL_NONZERO,0.05,EXPECT_WARN
SIGN_FLIP_P,Power sign flipped,230.0,10.0,0.99,-
2277.0,LARGE,0.10,EXPECT_ALERT
PF_NEAR_ZERO,PF=0.05 with
floor,230.0,5.00,0.05,57.5,MODERATE,0.20,EXPECT_PASS
PF_NEAR_ZERO_NO_FLOOR,PF=0.05 without
floor,230.0,5.00,0.05,57.5,EXCESSIVE,0.20,EXPECT_FAIL
```

A simple harness then:

1. Computes `e_V`, `e_I`, `e_P_meas`, `pf_eff`, `r_P`.
2. Checks that each scenario produces `|r_P|` in the expected qualitative band (NEAR_ZERO, SMALL_NONZERO, LARGE, etc.) relative to its `tol_rP`.

Section **13.1 — Residual Test Pack (r_P Focused)** will formalize this into a reusable, cross-domain test set.

SECTION 5 — Hysteresis & Chatter Control

Intent. Convert noisy, rapidly changing dials into **clean, repeatable decisions** using **enter/clear thresholds**, optional **memory gating** with `M`, and small **time guards**. The goal is to act promptly on real excursions while avoiding oscillation around limits.

What this section establishes.

- **Enter/clear hysteresis** on bounded dials `a_*` so a condition **enters** at a stricter threshold and **clears** at a looser one.
- **Memory-gated forms** that combine instantaneous dial state with the smoothed memory `M` for robust behavior under flicker.
- **Time guards** (minimum-hold and refractory intervals) to prevent rapid toggling.
- **Copy-ready patterns** that are deterministic, batch==stream, and portable across devices.

Design goals.

- **Determinism:** identical outcomes for the same sequence of inputs.
- **Monotone state transitions:** once entered, a condition stays active until a clear rule is met.
- **Low parameter count:** publish a few thresholds and times; avoid per-site tuning.
- **Composability:** outputs feed directly into policy packs and pooling.

Building blocks (conceptual).

- **Two-threshold rule:**
  enter when `a ≥ th_enter`, clear when `a ≤ th_clear`, with `th_enter > th_clear`.
- **Memory gate (optional):** require `M ≥ m_enter` to enter, and/or `M ≤ m_clear` to clear.
- **Hold / refractory:** minimum active time after enter, and minimum quiet time after clear.
- **Priority multiplexing:** when multiple dials can fire the same condition, use a defined precedence or a max.

Interfaces.

- **Inputs:** bounded dials `a_*`, memory `M`, thresholds `{th_enter, th_clear}`, `{m_enter, m_clear}`, and times `{t_hold, t_refractory}`.
- **Outputs:** boolean states and clean edge events (`ENTER`, `CLEAR`) suitable for logging, policy, and alarms.

Operational payoff.

- **Chatter-free alarms and policies** even at corridor edges.
- **Predictable operator experience**: once a condition trips, it behaves consistently across devices and time windows.
- **Easy commissioning**: small set of published thresholds and times governs fleet behavior.

5.1 Enter/clear thresholds; memory-gated forms

Purpose.
Produce **clean, chatter-free states** from bounded dials `a ∈ (-1,+1)` by applying **two-threshold hysteresis** and optional **memory gates** that leverage the single memory `M`.

Two-threshold hysteresis (copy-ready)

```
# Inputs per condition C:
#   a_t       ∈ (-1,+1)        # bounded dial at time t
#   th_enter  ∈ (th_clear, 1)  # stricter threshold to enter
#   th_clear  ∈ (-1, th_enter) # looser threshold to clear
#   state_{t-1} ∈ {0,1}        # previous state (0=OFF, 1=ON)

# Update:
if state_{t-1} == 0:
    state_t := 1 if (a_t >= th_enter) else 0
else:
    state_t := 0 if (a_t <= th_clear) else 1
```

Semantics.
`th_enter > th_clear` creates a **deadband** that prevents oscillation near the boundary.

Memory-gated variants (optional, copy-ready)

```
# Add memory M_t ∈ (-1,+1) with memory thresholds m_enter >= m_clear.

# AND-gated enter (requires both dial and memory pressure):
if state_{t-1} == 0:
    state_t := 1 if (a_t >= th_enter) and (M_t >= m_enter) else 0
else:
```

```
    state_t := 0 if (a_t <= th_clear) and (M_t <= m_clear) else 1

# OR-gated enter (either fast dial spike or sustained memory is enough):
if state_{t-1} == 0:
    state_t := 1 if (a_t >= th_enter) or (M_t >= m_enter) else 0
else:
    state_t := 0 if (a_t <= th_clear) and (M_t <= m_clear) else 1
```

Guidance.

- Use **AND-gate** for **critical faults** (avoid reacting to brief spikes).
- Use **OR-gate** when either a **sharp limit breach** or **accumulated stress** should trigger.

Time guards (optional, copy-ready)

```
# Enforce a minimum ON time (hold) after ENTER and a minimum QUIET time
after CLEAR.

# Params:
t_hold       > 0   # seconds
t_refractory > 0   # seconds
last_enter_ts, last_clear_ts  # timestamps

# Apply after computing state_t:
if (state_{t-1} == 1) and (now - last_enter_ts < t_hold):
    state_t := 1        # hold ON

if (state_{t-1} == 0) and (now - last_clear_ts < t_refractory):
    state_t := 0        # hold OFF
```

Starter thresholds (tune by class)

```
# Dials approaching absolute limits (a_upper_I, a_upper_P):
th_enter = 0.80,  th_clear = 0.65

# Corridor edges (a_over_V, a_under_V):
th_enter = 0.30,  th_clear = 0.20

# PQ dials (a_THD_*, a_unb*):
th_enter = 0.60,  th_clear = 0.45

# Memory gates (if used):
m_enter  = 0.50,  m_clear  = 0.35

# Time guards:
t_hold = 5 s,     t_refractory = 5 s
```

Deterministic implementation pattern (paste-in)

```
# Inputs at step t:
#   a_t, M_t, state_{t-1}, last_enter_ts, last_clear_ts
# Params:
#   th_enter, th_clear, m_enter, m_clear,
#   t_hold, t_refractory,
#   gating_mode ∈ { "NONE", "AND", "OR" }

# 1) Core hysteresis
candidate :=
    (state_{t-1} == 0 and a_t >= th_enter) ? 1 :
    (state_{t-1} == 1 and a_t <= th_clear) ? 0 :
    state_{t-1}

# 2) Memory gating
if gating_mode == "AND":
    if state_{t-1} == 0 and candidate == 1 and M_t < m_enter: candidate :=
0
    if state_{t-1} == 1 and candidate == 0 and M_t > m_clear: candidate :=
1
elif gating_mode == "OR":
    if state_{t-1} == 0 and (a_t < th_enter) and (M_t < m_enter): candidate
:= 0
    if state_{t-1} == 1 and (a_t > th_clear) and (M_t > m_clear): candidate
:= 1

# 3) Time guards
state_t := candidate
if state_{t-1} == 1 and (now - last_enter_ts) < t_hold:
    state_t := 1
if state_{t-1} == 0 and (now - last_clear_ts) < t_refractory:
    state_t := 0

# 4) Edge events and timestamps
if state_{t-1} == 0 and state_t == 1:
    last_enter_ts := now
    emit ENTER

if state_{t-1} == 1 and state_t == 0:
    last_clear_ts := now
    emit CLEAR
```

Tiny sanity table (dial-only)

```
th_enter = 0.8, th_clear = 0.6

a: ... 0.78  0.79  0.80  0.79  0.77  0.61  0.59 ...
s: ...   0     0     1     1     1     1     0  ...
```

a must cross 0.80 to **enter** and drop below 0.60 to **clear; no toggling around 0.70`.

Publishing checklist (per condition)

```
publish:
  dial: "a_upper_I"         # or a_over_V, a_THD_V, etc.
  hysteresis: { th_enter: 0.80, th_clear: 0.65 }
  memory_gate: { mode: "AND", m_enter: 0.50, m_clear: 0.35 }
  time_guards: { hold_s: 5, refractory_s: 5 }

notes:
  state_0 := 0  # unless restored from logs/checkpoint;
                # ENTER/CLEAR are edge events for logs/policy.
```

Why this resists chatter

- **Deadband:** `th_enter > th_clear` prevents back-and-forth at the boundary.
- **Memory gate:** requires (or allows) **sustained pressure** rather than reacting to a single spike.
- **Time guards:** suppress rapid retriggers from noise or brief dips, improving operator trust in alarms.

5.2 Parameter hints (th_enter/clear, m_enter/clear, t_hold, t_refractory)

Purpose.
Choose hysteresis and memory parameters that are **portable**, **quiet near limits**, and **responsive to real excursions**, without per-site hand tuning.

Starter presets (copy-ready)

```
# Limits (headroom) dials: a_upper_I, a_upper_P, a_upper_S
th_enter = 0.80
th_clear = 0.65

# Corridor dials: a_over_V, a_under_V
th_enter = 0.30
th_clear = 0.20

# PQ dials: a_THD_V, a_THD_I, a_unbV, a_unbI
th_enter = 0.60
th_clear = 0.45

# Memory gate (if used)
m_enter  = 0.50
m_clear  = 0.35

# Time guards (seconds)
t_hold       = 5
t_refractory = 5
```

How to tune quickly (quantile method, copy-ready)

```
# Collect baseline dial samples {a_t} over clean operation (no faults),
length N.
# Pick a target false-enter rate p_baseline (e.g., 0.1%).

q_hi := quantile(a_t, 1 - p_baseline)
q_lo := quantile(a_t, 1 - 10 * p_baseline)   # looser clear (10x wider)

# Set thresholds:
th_enter := clamp( q_hi , 0.20 , 0.95 )
th_clear := clamp( min( q_lo , th_enter - 0.10 ) , 0.05 , th_enter - 0.05 )
```

Interpretation.
Baseline quantiles anchor **enter** above ordinary noise and **clear** below it, creating a stable **deadband** without manual guesswork.

Memory gate selection (when to use)

- **Use AND-gate** when brief spikes must *not* trip (for example, current headroom):
  - Require **both** `a_t >= th_enter` **and** `M_t >= m_enter`.
- **Use OR-gate** when either a **sharp breach** *or* **sustained stress** should trip (for example, PQ worsening during events).

Mapping `rho` to `m_enter` / `m_clear`

```
# If M follows s_t ∈ [0,1], approximate steady-state:
M* ≈ mean( s_t )  over an event.
```

- Choose `m_enter` slightly **below** the steady `s_t` expected during true events.

Example.
If true events yield `s_t ≈ 0.7` on average:

```
m_enter ≈ 0.60
m_clear ≈ m_enter - 0.15   # e.g., 0.45
```

This keeps `M` **sensitive to real events** but **immune to noise**.

Choosing time guards

```
# Fast protection-like behaviors:
t_hold = 1–3  s
t_refractory = 1–3  s

# Operations dashboards / alarms:
t_hold = 3–10 s
```

```
t_refractory = 3–10 s

# Fleet summaries / reports:
t_hold = 10–60 s
t_refractory = 10–60 s
```

Match these to the **human timescale** you care about (operator reaction vs long-term reporting).

Cross-fleet portability checklist (paste into profile)

```
publish:
  hysteresis:
    headroom: { th_enter: 0.80, th_clear: 0.65 }
    corridor: { th_enter: 0.30, th_clear: 0.20 }
    pq:       { th_enter: 0.60, th_clear: 0.45 }

  memory_gate: { mode: "AND", m_enter: 0.50, m_clear: 0.35 }

  time_guards: { hold_s: 5, refractory_s: 5 }

notes:
  thresholds derived via baseline quantiles; revise only when new baselines
are collected.
```

Sanity constraints (must hold)

```
-1 < th_clear < th_enter < +1
-1 < m_clear  < m_enter  < +1
t_hold > 0
t_refractory > 0
```

These constraints preserve **deadband behavior** and **monotone memory logic**.

Tiny example (plug-and-go)

```
Baseline (clean month):
  median(a_upper_I)          = 0.22
  99.9% quantile(a_upper_I)  = 0.78

Set:
  th_enter = 0.80           # just above 99.9% noise
  th_clear = 0.65           # ~0.15 deadband below enter

Memory:
  tau      = 30 s           # implies rho ≈ 0.967
  m_enter  = 0.50
  m_clear  = 0.35

Time guards:
  t_hold       = 5 s
  t_refractory = 5 s
```

This gives a **portable, low-chatter setup** that most fleets can adopt as-is, then refine using their own baseline quantiles.

5.3 — Mini example: chatter reduction near limits

Purpose. Show, with a tiny time-series, how **hysteresis** and an **M-gate** eliminate on/off flicker when a dial hovers near a limit.

Example A — Dial-only with narrow deadband (chatter).
Setup: `th_enter = 0.75`, `th_clear = 0.70` (too close), no memory.

```
t:     1     2     3     4     5
a_t: 0.74  0.76  0.74  0.76  0.74
s_t:   0     1     0     1     0      # toggles ON/OFF repeatedly
```

Observation. With a thin deadband and noisy `a_t`, the state oscillates.

Example B — Recommended deadband + memory AND-gate (quiet).
Setup: `th_enter = 0.80`, `th_clear = 0.65`, `m_enter = 0.50`, `m_clear = 0.35`, `rho = 0.90`, `M_0 = 0`, gating mode = **AND**, surrogate `s_t := a_t`.

Phase 1: transient spikes do not trip.

```
t:      1      2      3      4      5
a_t:  0.74   0.82   0.76   0.81   0.72
M_t: 0.074  0.149  0.210  0.270  0.315   # M builds slowly: M_t :=
0.9*M_{t-1} + 0.1*a_t
state:
  - At t=2 (a_t ≥ 0.80) but M_t < 0.50  → still OFF (AND-gate blocks)
  - At t=4 (a_t ≥ 0.80) but M_t < 0.50  → still OFF
```

Result. Short excursions above `th_enter` do **not** cause an ENTER.

Phase 2: sustained pressure triggers once, cleanly.

```
t:       6      7      8      9
a_t:   0.83   0.85   0.88   0.90
M_t:  0.366  0.415  0.461  0.505
state:
  - At t=9, a_t ≥ 0.80 AND M_t ≥ 0.50  → ENTER (state := ON)
```

Phase 3: clearing is equally stable (no immediate drop).

- After ENTER, suppose `a_t` dips below `th_clear` briefly (e.g., `a_t = 0.64`).
- With **AND** clear (`a_t ≤ th_clear` **and** `M_t ≤ m_clear`), the state remains ON until `M_t` decays below `m_clear`, preventing "blink" clears during short relief.

Takeaways (operational).

- **Deadband** avoids edge toggling; **M-gate** requires **sustained pressure** to enter and **sustained relief** to clear.
- Parameters map cleanly to behavior: `rho` controls memory time; `{th_enter, th_clear}` shape dial sensitivity; `{m_enter, m_clear}` govern persistence.
- This yields **quiet alarms** and **predictable policy** even when `a_t` dances near limits.

SECTION 6 — Pooling (Fleet/Substation) — Rapidity

Intent. Fuse many bounded dials `a ∈ (-1,+1)` (per-phase, per-device, or rolling windows) into a **single, stable summary dial** that is **order-invariant** (batch == stream), **bounded**, and **portable** across deployments.

What this section establishes.

- A **rapidity-space mean** (`atanh`/`tanh`) that guarantees **batch == stream** parity with any arrival order.
- A simple **weighting scheme** with a denominator guard `eps_w` for numerical safety.
- **Copy-ready skeletons** for streaming fusion and one-shot batch pooling, plus a tiny parity test.
- Guidance on interpreting pooled values for operations and fleet dashboards.

6.1 — Order-invariance (batch == stream)

Purpose. Pool many bounded dials `a_i ∈ (-1,+1)` (devices, phases, time slices) into a **single fleet/substation dial** that is **independent of processing order** (offline batch equals online streaming).

Rapidity fuse (copy-ready).

```
# Inputs: a_i ∈ (-1,+1), weights w_i ≥ 0, guard eps_w > 0
# State (for streaming): U, W initialized to 0

# Per item i (streaming or batch loop):
u_i := atanh(a_i)            # finite because |a_i| < 1
U   := U + w_i * u_i
W   := W + w_i

# Pooled dial (readout anytime):
a_pool := tanh( U / max(W, eps_w) )
```

Why this is order-invariant.

```
U_final = Σ_i w_i * atanh(a_i)     # commutative + associative sum
W_final = Σ_i w_i
a_pool  = tanh( U / max(W, eps_w) )
```

Reordering items leaves `U_final` and `W_final` unchanged ⇒ same `a_pool` for batch and stream.

Essential properties.

- **Bounded:** `a_pool ∈ (-1,+1)` automatically.
- **Monotone in each input:** increasing any `a_i` (with `w_i > 0`) increases `a_pool`.
- **Idempotent:** if all `a_i = a*`, then `a_pool = a*`.
- **Weightless fallback:** with `w_i = 1`, this is the **order-invariant mean in rapidity space**.

Good defaults (publish once).

```
w_i := 1
eps_w := 1e-12
```

Copy-ready streaming skeleton.

```
# init once
U := 0
W := 0

# on each new a_i, w_i:
a_i := max(-1 + eps_a, min(1 - eps_a, a_i))   # safety clamp reused
u_i := atanh(a_i)
U := U + w_i * u_i
W := W + w_i

# readout anytime
a_pool := tanh( U / max(W, eps_w) )
```

Micro test (batch == stream parity).

```
Inputs: (a_1, w_1) = ( 0.20, 1)
        (a_2, w_2) = (-0.10, 2)
        (a_3, w_3) = ( 0.50, 1)

Batch:
U = 1*atanh(0.20) + 2*atanh(-0.10) + 1*atanh(0.50)
W = 1 + 2 + 1 = 4
a_pool_batch = tanh( U / 4 )

Stream (any order gives same result), e.g. 3→1→2:
U=0; W=0
add (0.50,1):   U=atanh(0.50);           W=1
add (0.20,1):   U+=atanh(0.20);          W=2
add (-0.10,2):  U+=2*atanh(-0.10);       W=4
a_pool_stream = tanh( U / 4 ) = a_pool_batch
```

Interpretation tips.

- `a_pool ≈ 0` means fleet near nominal; nearer `+1` means many elements near limits or a few with large weights.
- Publish the **same `eps_a`, `eps_w`, and weight rule** across the fleet to guarantee reproducibility.

6.2 — Weights (equal vs declared) and eps_w

Purpose. Choose **how much influence** each dial contributes to the pooled result and guarantee a **non-crashing denominator** with `eps_w`.

Pooling recap (context). The pooled dial is `a_pool := tanh( U / max(W, eps_w) )` with `U := Σ w_i * atanh(a_i)` and `W := Σ w_i`.

Weight options (publish one rule per deployment).

```
# EQUAL influence (simple, portable)
w_i := 1

# MAGNITUDE-aware (favor higher nominal capacity)
# m_i is a published magnitude (e.g., rated_S_i, I_max_i, feeder_kVA_i)
w_i := m_i / max( median({m_j}), eps_div )

# QUALITY-aware (de-emphasize noisy endpoints)
# q_i ∈ (0,1] is a health/quality score updated slowly (e.g., Section 8)
w_i := q_i

# HYBRID (capacity × quality)
w_i := ( m_i / max(median({m_j}), eps_div) ) * q_i

# POLICY override (pin priority sites)
# tag_i ∈ {0,1} enables or disables inclusion; or
# w_i := k_priority for special endpoints (k_priority > 1)
```

Guidance.

- **Equal weights** are the baseline: simple and defensible across mixed fleets.
- Use **magnitude-aware** when pooling heterogeneous devices (e.g., 5 kVA and 500 kVA on one bus).
- Add a slow **quality score** if some meters intermittently degrade (weight → 0 suppresses impact without dropping data).
- Keep weights **nonnegative** and **published**; avoid dynamic jitter faster than the signal itself.

Denominator floor `eps_w` (copy-ready).

```
eps_w := 1e-12              # recommended default
# Purpose: protect U/W when W → 0 (e.g., all weights zero after quality
gating)
```

```
# Behavior: if W = 0, a_pool := tanh( U / eps_w ) → tanh(0) = 0 when U = 0
# Best practice: when all weights are zero, also emit "pool_status: EMPTY"
```

Copy-ready pooling with weights.

```
# Inputs: {a_i} with |a_i| < 1, {w_i} with w_i ≥ 0
# Guards: eps_a, eps_w, eps_div

U := 0
W := 0
for each i:
    a_i := max(-1 + eps_a, min(1 - eps_a, a_i))   # safety clamp
    u_i := atanh(a_i)
    U := U + w_i * u_i
    W := W + w_i

a_pool := tanh( U / max(W, eps_w) )

emit: { "a_pool": a_pool, "W": W }   # include W for transparency/debug
```

Tiny comparisons (intuition).

```
Case 1 — Equal weights:
(a1, a2, a3) = (0.2, 0.6, 0.0), w = (1,1,1)
U = atanh(0.2)+atanh(0.6)+atanh(0.0) ≈ 0.2027+0.6931+0 = 0.8958
W = 3
a_pool = tanh(0.8958/3) = tanh(0.2986) ≈ 0.290

Case 2 — Capacity-aware (w = (1,3,1)):
U = 1*atanh(0.2) + 3*atanh(0.6) + 1*atanh(0.0)
  ≈ 0.2027 + 3*0.6931 + 0 = 2.2820
W = 5
a_pool = tanh(2.2820/5) = tanh(0.4564) ≈ 0.427
# Heavier site near the limit pulls the pool upward (as intended).
```

Publishing checklist (paste into profile).

```
publish:
  pooling:
    weight_rule: "HYBRID"              # EQUAL | MAGNITUDE | QUALITY |
HYBRID | POLICY
    magnitude_key: "rated_S_kVA"       # used if MAGNITUDE/HYBRID
    quality_key: "q_health"            # in (0,1], updated slowly
    eps_w: 1e-12
notes:
  Weights are nonnegative, piecewise-constant or slowly varying; batch ==
stream guaranteed.
```

Why this stays portable.

- The rapidity fuse is **associative/commutative**; any consistent weight rule yields identical results in batch and stream.
- Publishing the **rule** (not just numbers) makes results reproducible across vendors and sites.

6.3 — Asset-Weighted Pooling Across Cabinets / Feeders (Normative)

Purpose. Aggregate multiple **aligned electrical dials** (e.g., `a_V`, `a_I`, `a_THD`, `a_unbV`) into a single **fleet/array posture** without touching underlying classical bytes. **Classical values remain intact:** `phi((m,a)) = m`.

Inputs.
• A set of channel results `{(a_i, lens_id_i)}` already produced by Section 6 lenses.
• Optional **asset scale** or **declared importance** for each source `S_i` to compute weights `w_i`.
• A **manifest** that locks lens choices, anchors, weight basis, and pooling params.

Weight policy (choose exactly one and publish).

1. **Scale-aware:** `w_i := |S_i|^gamma`, where `S_i` is a declared asset scale (e.g., rated kVA, feeder ampacity, PV DC nameplate).
2. **Uniform:** `w_i := 1` for all `i` (useful when assets are intentionally treated equal).
3. **Declared weights:** `w_i` provided explicitly per asset in the manifest.
   Knobs: `gamma >= 0` (monotone). **Freeze** `gamma` for the run.

Pooling kernel (alignment lane only).
Clamp each dial, convert to rapidity, fuse with weights, return to alignment:

```
a_i_c := clamp(a_i, -1 + eps_a, +1 - eps_a)
u_i := atanh(a_i_c)
U := sum_i( w_i * u_i )
W := sum_i( w_i )
a_pool := tanh( U / max(W, eps_w) )
```

Associativity / streaming: the `U/W` form guarantees identical outcomes for **batch**, **chunked**, and **streaming** accumulation orders.

Classical collapse (magnitude lane).
If a classical roll-up is also reported, it must be **value-only**:
`M := sum_i( m_i )` and `phi((M, a_pool)) = M`.
No vendor or scale offsets may be injected into `M`. **Alignment never changes** `m`.

Determinism rules (must).
• **One lens per channel per run.** No mid-run lens switches.
• **Freeze anchors and refs** used by those lenses.
• **Freeze weight policy** (`basis`, `gamma`, or explicit `w_i`).
• **Monotone behavior:** if every `a_i` increases (others fixed), `a_pool` must not decrease.
• **Validity:** drop or mark invalid any `a_i` whose source violated its declared validity window.

Edge conditions.
• **No data:** if `W = 0`, set `a_pool := 0` and emit a **DATA_ABSENT** band (or skip compute) per policy.
• **Saturation:** if any `|a_i| >= 1 - eps_a`, the clamp holds; pooling still proceeds safely.
• **Time base:** only fuse values that share a **pooled timestamp tick** (e.g., same second). Out-of-window samples are excluded.

Manifest (publish these keys).
`pool_id`, `members[]`, `lens_id[]`, `anchors_ref`, `weight_basis` (one of `scale`, `uniform`, `declared`), `gamma` (if `scale`), `weights[]` (if `declared`), `eps_a`, `eps_w`, `tick_ms`, `validity_bounds` (per channel), `rounding` (display only), `notes`.

Batch recipe (copy-ready).

```
given a_i from members at tick T:
  a_i_c := clamp(a_i, -1+eps_a, +1-eps_a)
  u_i := atanh(a_i_c)
  U := sum( w_i * u_i )
  W := sum( w_i )
  a_pool := tanh( U / max(W, eps_w) )
emit (a_pool, pool_id, T)
```

Streaming recipe (copy-ready).

```
U := 0 ; W := 0
for each arriving member sample at tick T:
  a_c := clamp(a, -1+eps_a, +1-eps_a)
  u := atanh(a_c)
  U += w * u
  W += w
on tick close T:
  a_pool := tanh( U / max(W, eps_w) )
  emit (a_pool, pool_id, T)
  U := 0 ; W := 0
```

Commissioning checks (must-pass).

1. **Order-invariance test:** shuffle member order; `a_pool` must be identical.
2. **Scale-neutrality test:** multiply all `S_i` by a constant; if using `scale` basis, re-derive `w_i` and confirm **no change** when `gamma = 0`, and **expected monotone tilt** when `gamma > 0`.
3. **Golden vectors:** publish a tiny CSV with 3–5 members and expected `a_pool` to 1e-6.
4. **Zero-class:** all `a_i = 0` must yield `a_pool = 0`.
5. **Validity gates:** removing an invalid member must never worsen determinism (no NANs, no divide-by-zero).

Operator guidance.
Report `a_pool` alongside the member count `N` and the **effective weight sum** `W`. Do **not** average alignments directly; always use the `U/W` kernel.

6.4 — Power-Quality Dial Set (Alignment-Aware Indices, Normative)

Purpose. Convert **raw PQ metrics** into **zero-centric, monotone, unit-invariant alignments** that can be pooled (Section 6.3) and banded (Section 8). **Positive means risk**, **negative means margin**, **zero is at target**.

General lens template (use for every dial).
For a raw metric `x` with a declared **target** `x_ref` and **gain** `c > 0`, define contrast `r := x/x_ref` and compute alignment:
`a := sgn * tanh( c * ( r^rho - 1 ) )`
Knobs: `rho >= 1` (curvature), `c > 0` (sensitivity), `sgn in {+1, -1}` chosen so that **badness → positive**.
Clamp: `a_c := clamp(a, -1 + eps_a, +1 - eps_a).`
Deadband (optional): if `|r - 1| <= delta0`, set `a := 0` before clamp.
Monotone: increasing badness must **not** reduce `a`.

Dial definitions (declare each in manifest).

1. **Total Harmonic Distortion (voltage) — `a_THDV`**
   Raw: `THD_V` (fraction or %). Target: `THD_V_ref`.
   `a_THDV := +tanh( c_THDV * ( (THD_V/THD_V_ref)^rho_THDV - 1 ) )`
   Meaning: more distortion → larger positive `a_THDV`.
2. **Total Harmonic Distortion (current) — `a_THDI`**
   Raw: `THD_I`. Target: `THD_I_ref`.
   `a_THDI := +tanh( c_THDI * ( (THD_I/THD_I_ref)^rho_THDI - 1 ) )`
3. **Voltage unbalance — `a_unbV`**
   Raw: `UnbV := (V_max - V_min) / V_avg` (fraction or %). Target: `UnbV_ref`.
   `a_unbV := +tanh( c_unbV * ( (UnbV/UnbV_ref)^rho_unbV - 1 ) )`
4. **Current unbalance — `a_unbI`**
   Raw: `UnbI := (I_max - I_min) / I_avg`. Target: `UnbI_ref`.
   `a_unbI := +tanh( c_unbI * ( (UnbI/UnbI_ref)^rho_unbI - 1 ) )`
5. **Frequency deviation — `a_df`**
   Raw: `df := |f - f_ref|`. Target: `df_ref`.
   `a_df := +tanh( c_df * ( (df/df_ref)^rho_df - 1 ) )`
6. **Voltage magnitude deviation — `a_dV`**
   Raw: `dV := |V_rms - V_ref|`. Target: `dV_ref`.
   `a_dV := +tanh( c_dV * ( (dV/dV_ref)^rho_dV - 1 ) )`
7. **Power factor (want high) — `a_PF`**
   Raw: `PF in [0,1]`. Target minimum: `PF_ref`. Since **lower PF is worse**, set `sgn = +1` on the shortfall:
   `a_PF := +tanh( c_PF * ( (PF_ref / max(PF, eps_pf))^rho_PF - 1 ) )`
8. **Flicker (short-term index) — `a_flicker`**
   Raw: `P_st`. Target: `P_st_ref`.
   `a_flicker := +tanh( c_flick * ( (P_st/P_st_ref)^rho_flick - 1 ) )`
9. **Overcurrent margin (want low margin to be bad) — `a_I`**
   Raw: `I/I_limit`. Target: `1`.
   `a_I := +tanh( c_I * ( ( (I/I_limit) )^rho_I - 1 ) )`
10. **Thermal headroom (optional crossover) — `a_T`**
    Raw: `dT := T - T_ref_safe` (use `max(dT, 0)` if below safe). Target: `dT_ref`.
    `a_T := +tanh( c_T * ( (max(dT,0)/dT_ref)^rho_T - 1 ) )`

Per-dial validity (must declare).
For every dial, publish **validity bounds** for the raw channel(s). Example keys:
`V_min_valid, V_max_valid, I_min_valid, I_max_valid, THD_max_valid, df_max_valid, PF_min_valid.`
When violated, **do not fabricate `a`**. Either drop the sample or emit a **DATA_INVALID** band per policy.

Sampling & timing (must declare).
• **Tick alignment:** compute each dial at a declared tick `tick_ms` so fusing in Section 6.3 is well-defined.
• **Windowing:** if a dial uses a moving window (e.g., 1 s THD, 10 min PF), publish its window and update cadence.
• **Downsampling rule:** if multiple updates land within a tick, use **last-in-window** or **mean** per dial; declare which.

Mini recipes (copy-ready).

THD voltage example

```
r := THD_V / THD_V_ref
a := tanh( c_THDV * ( r^rho_THDV - 1 ) )
a_c := clamp(a, -1+eps_a, +1-eps_a)
emit (a_THDV := a_c, lens_id := "THDV_v1")
```

Power factor example

```
r := PF_ref / max(PF, eps_pf)
a := tanh( c_PF * ( r^rho_PF - 1 ) )
a_c := clamp(a, -1+eps_a, +1-eps_a)
emit (a_PF := a_c, lens_id := "PF_v1")
```

Monotonicity sanity (quick checks).
• If `THD_V` increases while refs/knobs fixed, `a_THDV` must **not** decrease.
• If `PF` decreases (worse), `a_PF` must **not** decrease.
• If `|f - f_ref|` shrinks, `a_df` must **not** increase.

Operator guidance.
• Display each dial with its **reference** to keep zero meaningful: show `x_ref` and `r = x/x_ref`.
• Use **bands** on each dial for fast routing; use **pooled `a_pool`** (Section 6.3) for cabinet/feeder posture.

Manifest keys (per dial).
`lens_id, channel_id, x_ref, c, rho, sgn, delta0, eps_a, eps_pf, tick_ms, window_s, validity_bounds, display_rounding`.

6.5 — Dispatch Coupling (Bands → Actions, Normative)

Purpose. Convert **dial alignments** (`a_THDV`, `a_PF`, `a_unbV`, `a_pool`, etc.) into **deterministic actions** with explicit **enter/clear thresholds**, **hold times**, and **stamped envelopes** for audit. **Positive means risk**, **negative means margin**, **zero at target**.

Band ladder (global).
`A+ (excellent) > A (good) > B (watch) > C (alert) > D (critical)`
Each band has **enter threshold** `th_enter`, **clear threshold** `th_clear`, and **hold time** `t_hold`. Use `th_clear < th_enter` for hysteresis.

Standard hysteresis form (per dial).
For any dial `a_x` (e.g., `a_THDV`):

- **Enter rule:** if `a_x >= th_enter_x` **continuously** for `t_hold_x_enter`, set band `BAND_x := target_band`.
- **Clear rule:** if `a_x <= th_clear_x` **continuously** for `t_hold_x_clear`, **downgrade** band by one step (or to `A` if policy states).
- **No flapping:** choose `th_clear_x < th_enter_x` and `t_hold_x_clear <= t_hold_x_enter` unless otherwise justified.

Copy-ready defaults (recommended).

- `th_enter(A+) := +0.20`, `th_clear(A+) := +0.10`, `t_hold := 10 min` → label stays `A+`.
- `th_enter(A) := +0.35`, `th_clear(A) := +0.20`, `t_hold := 10 min` → label `A`.
- `th_enter(B) := +0.50`, `th_clear(B) := +0.35`, `t_hold := 5 min` → **WATCH**.
- `th_enter(C) := +0.60`, `th_clear(C) := +0.45`, `t_hold := 2 min` → **ALERT**.
- `th_enter(D) := +0.75`, `th_clear(D) := +0.60`, `t_hold := 30 s` → **CRITICAL**.

Coupling to actions (routing matrix).
Map `(dial, band)` → `(action, SLA, channel)`; examples:

- `(a_THDV, C)` → `create_ticket`, `SLA := 15 min`, `owner := PQ_team`.
- `(a_unbV, D)` → `remote_shed_5%`, `SLA := 30 s`, `owner := Ops_control`.
- `(a_PF, B)` → `advise_load_shift`, `SLA := 1 h`, `owner := Energy_team`.
- `(a_pool, C)` → `field_inspection_request`, `SLA := 24 h`, `owner := Field_ops`.

Stamped envelope (dispatch event).
Every action emits a **portable envelope**:

```
{ value := a_x, band := BAND_x, dial_id := "a_x", pool_id?, manifest_id,
action, sla_ms, owner, tick_ms, stamp }
stamp := chain_link(prev_hash, sha256(payload_bytes), time_utc)
```

The **classical magnitude** is never altered by these actions; `phi((m,a)) = m` holds throughout.

Policy gates (optional, safe-by-default).

- **Environment gate:** `a_env := g_t * a_x` for decision only; `g_t in [0,1]` dampens actions during declared conditions (e.g., storm mode).
- **Rate limit:** minimum inter-dispatch gap `gap_ms` per `(dial, site)` to avoid chatter.
- **Quorum for pooled actions:** require `N_ok >= N_min` valid members in `a_pool` tick before issuing pooled actions.
- **Silence on invalid:** if the dial is **DATA_INVALID** or **DATA_ABSENT**, suppress or downgrade to **B (watch)** per policy.

Time-base integrity (must).

- Compute actions on the **same tick** as dial emission: `tick_ms` is declared and consistent.
- **Hold clocks** count **continuous** time at band-eligible level; reset on breach.

Copy-ready rule table (example entries).

```
# THD voltage (THDV)
enter(C): a_THDV >= +0.60 for 120 s -> ALERT, SLA 15 min, owner PQ_team
clear(C): a_THDV <= +0.45 for 120 s -> downgrade to B

enter(D): a_THDV >= +0.75 for 30 s  -> CRITICAL, SLA 30 s, owner
Ops_control
clear(D): a_THDV <= +0.60 for 60 s  -> downgrade to C
# Power factor (PF)
enter(B): a_PF >= +0.50 for 5 min  -> WATCH, SLA 1 h, owner Energy_team
clear(B): a_PF <= +0.35 for 5 min  -> downgrade to A

enter(C): a_PF >= +0.60 for 2 min  -> ALERT, SLA 15 min, owner Energy_team
clear(C): a_PF <= +0.45 for 2 min  -> downgrade to B
# Pooled cabinet posture (a_pool)
enter(C): a_pool >= +0.60 for 2 min -> field_inspection_request, SLA 24 h,
owner Field_ops
clear(C): a_pool <= +0.45 for 2 min -> downgrade to B
```

Pseudo-code (copy-ready).

```
on each tick T:
  for each dial a_x:
    if valid(a_x):
      band_new := ladder_hysteresis(a_x, band_old, th_enter[], th_clear[],
t_hold[])
      if band_new != band_old and passes_rate_limit(a_x) and
quorum_ok(a_x):
         env_align := g_t * a_x
         if eligible(env_align, band_new):
            emit_envelope({
              value := a_x, band := band_new, dial_id := id(a_x),
              manifest_id := MID, action := route(dial_id, band_new),
              sla_ms := SLA(dial_id, band_new), owner := OWNER(dial_id,
band_new),
              tick_ms := T, stamp := chain_stamp(prev_hash, payload_hash,
time_utc)
            })
      band_old := band_new
    else:
      handle_invalid(a_x)  # per policy: suppress or watch
```

Commissioning checklist (must-pass).

1. **Hysteresis demo:** sweep a dial up/down across thresholds; confirm **no flapping**.
2. **Hold accuracy:** inject synthetic plateaus; verify `t_hold` gates decisions.
3. **Rate-limit:** burst events; verify spacing ≥ `gap_ms`.
4. **Pooled quorum:** drop members; ensure pooled actions **pause** when `N_ok < N_min`.
5. **Stamp chain:** recompute `sha256(payload_bytes)` and verify chain continuity over 100 events.

Operator guidance.
Show the **current band**, **time-in-band**, **next-clear target** (`th_clear`), and the **scheduled SLA**. Always provide a one-click view of the **last 20 stamps** for the dial or pool.

6.6 — Commissioning Checklist for Pooling (Must-Do, Copy-Ready)

Purpose. Prove that **fleet/array pooling** produces **deterministic, order-invariant, streaming-equivalent** alignments, with **frozen lenses**, **frozen weights**, and **validity-aware** inclusion. This checklist is the **gate** before any live routing.

A. Freeze & Declare (pre-run locks)

1. **Lens immutability:** one lens per channel; publish `lens_id[]`, `anchors_ref`, `tick_ms`.
2. **Weight policy:** publish `weight_basis in {scale, uniform, declared}`, `gamma` (if `scale`), `weights[]` (if `declared`).
3. **Numerics:** publish `eps_a`, `eps_w`, rounding strategy for display only.
4. **Validity windows:** per channel publish min/max bounds; define **drop vs mark** on breach.
5. **Time base:** publish pooling schedule; all members must align on the **same tick**.

B. Minimal Golden Vectors (author once, reuse everywhere)
Prepare a tiny CSV (3–5 members × 4 ticks) with expected outputs to `1e-6`.

```
# members.csv
member_id,S_scale,w_declared,lens_id
cabA,50, ,THDV_v1
cabB,25, ,THDV_v1
cabC,10, ,THDV_v1
# samples.csv  (tick is integer or ISO; a is the per-member dial after
lens)
tick,member_id,a
1,cabA, +0.1100
1,cabB, +0.3500
1,cabC, -0.0500
2,cabA, +0.6000
2,cabB, +0.6100
2,cabC, +0.5900
```

```
3,cabA, +0.0000
3,cabB, +0.0000
3,cabC, +0.0000
4,cabA, +0.9800
4,cabB, +0.9800
4,cabC, +0.9800
# policy.json  (pick one basis and freeze)
{
  "weight_basis": "scale",
  "gamma": 1.0,
  "eps_a": 1e-6,
  "eps_w": 1e-9,
  "tick_ms": 1000
}
# expected.csv  (authoritative pooled answers for each tick)
tick,a_pool_expected
1,+0.212345
2,+0.600000
3,+0.000000
4,+0.979999
```

C. Determinism Tests (must-pass)

1. **Order-invariance:** shuffle `samples.csv` row order; recompute `a_pool`; compare to `expected.csv` (`abs(diff) <= 1e-6`).
2. **Batch==Streaming:** compute once with batch sums and once with tick-stream updates; values must match bit-for-bit after rounding rule.
3. **Chunking neutrality:** split the same tick's arrivals into random chunks; result identical to batch.
4. **Lens lock:** change of `lens_id` mid-run must be **rejected** by harness; no compute performed.
5. **Weight lock:** change of `weight_basis`, `gamma`, or `w_declared` mid-run must be **rejected**; no compute performed.
6. **Zero-class:** all `a = 0` → `a_pool = 0`.
7. **Saturation safety:** any `|a| >= 1 - eps_a` clamps; pooled value remains finite and monotone.
8. **Validity gating:** inject an out-of-bounds raw sample → corresponding `a` dropped or flagged; pooled value computes from remaining members without error.
9. **Empty set:** if no valid members at a tick → set `a_pool := 0` and emit **DATA_ABSENT** per policy.
10. **Scale monotonicity (if basis=scale):** multiply all `S_scale` by a constant; with `gamma = 0` pooled result unchanged; with `gamma > 0` pooled tilt increases toward larger assets (monotone, never flips sign spuriously).

D. Harness Recipes (copy-ready pseudo-code)

Batch pooling per tick

```
U := 0 ; W := 0
for each member m in MEMBERS:
  if valid(a[m,tick]):
    a_c := clamp(a[m,tick], -1+eps_a, +1-eps_a)
    u := atanh(a_c)
    w := weight(m)  # from basis: scale^gamma | 1 | declared
    U += w * u
```

```
    W += w
a_pool := (W == 0) ? 0 : tanh( U / max(W, eps_w) )
```

Streaming equivalence

```
U := 0 ; W := 0
for each arrival (m, a_m) at tick t in any order:
  if valid(a_m):
    a_c := clamp(a_m, -1+eps_a, +1-eps_a)
    U += weight(m) * atanh(a_c)
    W += weight(m)
on tick close:
  a_pool := (W == 0) ? 0 : tanh( U / max(W, eps_w) )
  reset U := 0 ; W := 0
```

E. Pass/Fail Criteria (objective)
• **Numeric tolerance:** `abs(a_pool - a_pool_expected) <= 1e-6` for all ticks in the golden set.
• **Config immutability:** any deviation from published policy/lens causes **FAIL** with an explicit error.
• **No NaNs/Inf:** harness rejects inputs that would produce non-finite values; clamps ensure finiteness.
• **Audit artifact:** store `inputs_hash := sha256(members.csv || samples.csv || policy.json)`, `outputs_hash := sha256(expected.csv)` and emit a single **commissioning stamp** `{inputs_hash, outputs_hash, time_utc}`.

F. Operator Quick Check (human-friendly)

1. Display **N_valid / N_total**, **effective W**, and **a_pool** per tick.
2. Provide a one-click **"shuffle & verify"** button that replays the same tick 10 times in random orders and shows `max(abs(diff))`.

G. Quorum & Degradation (recommended)
• Publish `N_min` for pooled actions; if `N_valid < N_min`, **suspend dispatch** and label the tick **QUORUM_LOST**.
• If a specific member is missing for `k` consecutive ticks, raise **DATA_STALE(member_id, k)** as a non-intrusive watch.

6.7 Conformance Tests (Portable Recipes, Copy-Ready)

Purpose.
Provide a repeatable **test suite** that proves SSMEQ pooling and banding are **deterministic**, **order-invariant**, **streaming-equivalent**, and **policy-locked**. All tests are **data-driven** (`CSV`/`JSON`), require **no vendor secrets**, and run in minutes.

Artifacts (must provide)

1. **Members table** (`members.csv`): asset scales, declared weights, lens IDs.
2. **Samples table** (`samples.csv`): per-tick per-member alignments `a` (post-lens).
3. **Policy file** (`policy.json`): weight basis and numerics.
4. **Expected table** (`expected.csv`): authoritative `a_pool` per tick for a fixed golden scenario.
5. **Band rules** (`bands.json`): enter/clear thresholds, hold times, routing.
6. **Stamp chain** (`stamps.csv`): hash continuity over emitted envelopes during band tests.

Canonical CSV/JSON shapes (copy-ready)

```
# members.csv
member_id,S_scale,w_declared,lens_id
cabA,50,,THDV_v1
cabB,25,,THDV_v1
cabC,10,,THDV_v1

# samples.csv
tick,member_id,a
1,cabA,+0.1100
1,cabB,+0.3500
1,cabC,-0.0500
2,cabA,+0.6000
2,cabB,+0.6100
2,cabC,+0.5900
...

# policy.json
{
  "weight_basis": "scale",       // one of: "scale" | "uniform" |
"declared"
  "gamma": 1.0,                  // used only if weight_basis == "scale"
  "eps_a": 1e-6,
  "eps_w": 1e-9,
  "tick_ms": 1000,
  "N_min": 2                     // quorum for pooled actions
}

# expected.csv
tick,a_pool_expected
1,+0.212345
2,+0.600000
3,+0.000000
4,+0.979999

# bands.json (example)
{
  "A+": { "enter": +0.20, "clear": +0.10, "hold_ms_enter": 600000,
"hold_ms_clear": 600000 },
  "A" : { "enter": +0.35, "clear": +0.20, "hold_ms_enter": 600000,
"hold_ms_clear": 600000 },
  "B" : { "enter": +0.50, "clear": +0.35, "hold_ms_enter": 300000,
"hold_ms_clear": 300000 },
```

```
  "C" : { "enter": +0.60, "clear": +0.45, "hold_ms_enter": 120000,
"hold_ms_clear": 120000 },
  "D" : { "enter": +0.75, "clear": +0.60, "hold_ms_enter": 30000,
"hold_ms_clear": 60000  }
}
```

Test battery (must-pass)

T1 — Order-Invariance.
Shuffle `samples.csv` 10× at each tick; compute `a_pool` via `U/W` kernel; assert `max_abs_diff <= 1e-6`.

T2 — Batch == Streaming == Chunked.
Run (a) batch per-tick summation, (b) per-arrival streaming accumulation with tick close, (c) random chunking. All must match to `1e-6`.

T3 — Policy Locks.
Attempt mid-run changes to `weight_basis`, `gamma`, `w_declared`, or `lens_id`. Harness must **reject** with explicit error; **no output** emitted under changed policy.

T4 — Zero-Class & Saturation.

- All `a = 0` ⇒ `a_pool = 0`.
- Any `|a| >= 1 - eps_a` clamps; pooled value remains **finite** and **monotone**.

T5 — Validity Gates.

- Mark a member invalid at tick `t` (simulate out-of-bounds raw).
- Pool ignoring that member yields finite `a_pool`.
- If `N_valid < N_min`, pooled dispatch is suspended and labeled `QUORUM_LOST`.

T6 — Scale Monotonicity (basis = scale).

- Multiply all `S_scale` by constant `k > 0`.
- With `gamma = 0` → no change in `a_pool`.
- With `gamma > 0` → tilt toward larger assets (monotone, sign-stable).

T7 — Declared Weights (basis = declared).

- Switch to `weight_basis = "declared"` and provide `w_declared` in `members.csv`.
- Recompute `expected.csv` **once**; subsequent re-runs must match exactly.

T8 — Collapse Parity (magnitude lane).

- If a classical magnitude roll-up is reported, compute `M := sum_i(m_i)` independently.
- Attach `phi((M, a_pool)) = M`. Confirm that alignment **never alters** `m`.

T9 — Band Hysteresis.

- Replay a synthetic dial sequence crossing enter/clear thresholds.
- Verify **no flapping**, correct **hold timing**, and correct **band transitions** for `B`, `C`, `D`.

T10 — Stamp Chain Integrity.

- For each dispatch event, compute `payload_hash := sha256(payload_bytes).`
- Chain stamp: `stamp_t := sha256( prev_stamp || payload_hash || time_utc ).`
- Verify **contiguous chain** across the run, no orphan stamps, no duplicates.

Reference kernels (copy-ready)

Clamp → rapidity → weighted fuse → alignment

```
a_c := clamp( a , -1 + eps_a , +1 - eps_a )
u   := atanh( a_c )
U  += w * u
W  += w
a_pool := (W == 0) ? 0 : tanh( U / max(W, eps_w) )
```

Weight resolvers

```
weight(m):
  if basis == "uniform":   return 1
  if basis == "scale":     return pow( S_scale[m] , gamma )
  if basis == "declared":  return w_declared[m]
```

Band ladder with hysteresis

```
ladder_hysteresis(a, band_old):
  # choose highest band whose enter condition has been held long enough
  # downgrade only when its clear condition has been held long enough
  ...
```

Verifier outline (copy-ready pseudo-code)

```
load members.csv, samples.csv, policy.json
fix RNG seed := 12345

def pool_tick(samples_t):
  U := 0 ; W := 0
  for (m,a) in samples_t:
    if valid(a):
      a_c := clamp( a , -1 + eps_a , +1 - eps_a )
      u   := atanh( a_c )
      w   := weight(m)
      U   := U + w * u
      W   := W + w
  return (W == 0) ? 0 : tanh( U / max(W, eps_w) )
```

```
# T1: order-invariance
for t in ticks:
  base := pool_tick( samples[t] in original order )
  for k in 1..10:
    shuffled := shuffle( samples[t] )
    chk := pool_tick( shuffled )
    assert abs( chk - base ) <= 1e-6

# T2: batch vs streaming vs chunked
assert batch(t) == streaming(t) == chunked(t) within 1e-6 for all t

# T3..T10: see battery above; emit PASS/FAIL with reason
```

Pass/Fail summary (objective)

- **Numeric:** all `a_pool` diffs ≤ `1e-6`.
- **No NaN/Inf:** rejected at ingest with clear error.
- **Policy immutability:** any mid-run change in policy-critical fields → **FAIL**.
- **Hysteresis:** zero flapping in sweeps; holds respected.
- **Stamps:** continuous chain; `sha256` recompute matches stored value.

Commissioning deliverable (single line)

```
commissioning_stamp :=
  sha256(
    sha256( members.csv || samples.csv || policy.json ) ||
    sha256( expected.csv ) ||
    time_utc
  )
```

Operator quick-run (human steps)

1. Load the four core files and the policy/bands into the verifier; click **Run Conformance**.
2. Confirm **10/10 PASS**, with maximum absolute difference and a link to the last 20 stamps.
3. Archive the `commissioning_stamp` in site records as the **canonical proof** of SSMEQ conformance for that deployment.

6.8 — Worked Examples (Cabinet + Feeder, Copy-Ready)

Purpose. Show end-to-end posture formation and banded dispatch from real-looking numbers using the **U/W rapidity fuse**. **Classical values remain intact:** `phi((m,a)) = m`.

Example A — Cabinet (3 members, uniform weights, per-tick pooling)

Setup.
• Members: `cabA, cabB, cabC` (same lens).
• Weight policy: `uniform` → `w_i := 1`.
• Numerics: `eps_a := 1e-6`, `eps_w := 1e-9`, `tick_ms := 1000`.
• Dials in play: single dial `a_THDV` per member.
• Band ladder: `B (>= +0.50, 5 min)`, `C (>= +0.60, 2 min)`, `D (>= +0.75, 30 s)`.

Member dial inputs (post-lens, per tick).

```
tick, cabA,   cabB,   cabC
1,    +0.10,  +0.30,  -0.05
2,    +0.60,  +0.61,  +0.59
3,     0.00,   0.00,   0.00
4,    +0.98,  +0.98,  +0.98
```

Pooling kernel (uniform):

```
a_c := clamp(a, -1+eps_a, +1-eps_a)
u := atanh(a_c)
U := sum(w*u) ; W := sum(w) ; a_pool := tanh( U / max(W, eps_w) )
```

Per-tick results (rounded).

- **Tick 1**: small mixed values → `U/W ≈ (0.1003 + 0.3095 − 0.0500)/3 ≈ 0.1199` → `a_pool ≈ 0.1191` → **below B**.
- **Tick 2**: near-equal high risk → `atanh(0.60)≈0.6931`, `atanh(0.61)≈0.7080`, `atanh(0.59)≈0.6779` → average rapidity `≈0.6930` → `a_pool ≈ 0.6000` → **enter C** if held `2 min`.
- **Tick 3**: all zero → `a_pool = 0.0000` → **clear to A+** if held per ladder.
- **Tick 4**: saturation regime → `atanh(0.98)≈2.2976` → average `≈2.2976` → `a_pool ≈ 0.9800` → **enter D** if held `30 s`.

Dispatch envelopes (illustrative).

```
# on Tick 2 after 120 s ≥ +0.60
{ value := +0.6000, band := C, dial_id := "a_THDV_pool",
  action := "ALERT", sla_ms := 900000, owner := "PQ_team",
  manifest_id := MID, tick_ms := T2, stamp := ... }

# on Tick 4 after 30 s ≥ +0.75
{ value := +0.9800, band := D, dial_id := "a_THDV_pool",
  action := "CRITICAL_REMOTE_SHED_5pct", sla_ms := 30000, owner :=
"Ops_control",
  manifest_id := MID, tick_ms := T4, stamp := ... }
```

Notes.
• **Order-invariance:** reordering member arrivals at a tick does **not** change `a_pool`.
• **Streaming equivalence:** accumulating `U/W` per arrival then closing the tick yields the same `a_pool` as batch.

Example B — Feeder (5 cabinets, scale-aware weights, multi-dial posture + routing)

Setup.
• Members: `cab1..cab5`, each reporting `a_THDV, a_unbV, a_PF`.
• Weight policy: `scale` with nameplate `S_kVA` and `gamma := 1.0`.
• Band ladder: as in Section 6.5 (with hysteresis).
• Optional environment gate for storms: `g_t in [0,1]`.

Member scales (kVA) → weights.

```
cab1: 400  -> w=400
cab2: 250  -> w=250
cab3: 250  -> w=250
cab4: 100  -> w=100
cab5:  50  -> w= 50
W_total := 1050
```

Tick T (per-member dials, post-lens).

```
a_THDV:  cab1:+0.62, cab2:+0.58, cab3:+0.65, cab4:+0.40, cab5:+0.35
a_unbV:  cab1:+0.30, cab2:+0.28, cab3:+0.32, cab4:+0.12, cab5:+0.10
a_PF:    cab1:+0.20, cab2:+0.10, cab3:+0.15, cab4:+0.05, cab5:+0.00
```

Pool each dial with U/W.

- `a_THDV_pool ≈ tanh( (400*atanh(.62)+250*atanh(.58)+250*atanh(.65)+100*atanh(.40)+50*atanh(.35)) / 1050 ) ≈ +0.606`
- `a_unbV_pool ≈ +0.281`
- `a_PF_pool ≈ +0.146`

Composite cabinet-stress dial (optional).
`U := sum( w_j * atanh(a_j_pool) )` over `{a_THDV_pool, a_unbV_pool, a_PF_pool}` with declared per-dial weights (e.g., `w_THDV=1.0, w_unbV=0.6, w_PF=0.4`) → `a_stress ≈ +0.487`.

Routing outcomes on Tick T.

- `a_THDV_pool = +0.606` → **eligible for C**; if `≥ 2 min` continuous, raise **ALERT** to PQ team.
- `a_stress = +0.487` → **B (watch)**; create preventive ticket if held `≥ 5 min`.
- Environment gate example: if `g_t = 0.7` (storm mode), the **decision value** is `a_env := 0.7 * a_THDV_pool`, still above `+0.60 * 0.7 = 0.42`? Apply policy as configured (e.g., **delay escalation** but keep **watch**).

Stamped envelopes (illustrative).

```
# THDV pooled -> ALERT after hold
{ value := +0.606, band := C, dial_id := "a_THDV_pool",
  action := "ALERT", sla_ms := 900000, owner := "PQ_team",
  pool_id := "feeder_F01", manifest_id := MID, tick_ms := T, stamp := ... }

# Stress -> WATCH ticket
```

```
{ value := +0.487, band := B, dial_id := "a_stress",
  action := "PREVENTIVE_TICKET", sla_ms := 3600000, owner := "Field_ops",
  pool_id := "feeder_F01", manifest_id := MID, tick_ms := T, stamp := ... }
```

What this demonstrates.
• **Scale-aware tilt**: larger cabinets influence posture more, **without** altering their own bytes.
• **Multi-dial realism**: THD drives immediate alerts; unbalance and PF accumulate into stress for planned work.
• **Safe damping**: `g_t` gates **decisions** only—**never** alters raw or aligned values.

Operator crib (display suggestions).
• Show `a_pool` for each dial, `N_valid/N_total`, and **effective W**.
• Show **time-in-band**, **next clear target**, and the **last 20 stamps** with a one-click export.

7.0 — Plant Health & Fatigue Dials (Introduction)

Intent.
Everything in Sections 1–6 focused on **electrical posture** in the present moment: how voltage, current, distortion, balance, and power quality evolve **tick-by-tick** and how pooled alignments (`a_pool`) trigger dispatch. **However, electrical systems do not fail only because of *instantaneous* stress**. Failure often emerges from **accumulated strain** — the slow wearing of breakers, thermal drift in cabinets, coil aging, contact erosion, busbar fatigue, and repetitive over-current stress.

This section introduces **Plant Health & Fatigue Dials**: a family of **zero-centric, monotone, unit-invariant** alignment signals that reflect **history**, **accumulation**, and **degradation rate**, rather than live electrical posture alone. These dials convert **event counts**, **stress proxies** (e.g., I^2t, switching energy), and **rolling windows of operation** into normalized signals that map **wear**, **aging**, and **remaining headroom** — in the exact same alignment language used throughout this framework.

Key Principles.
• **Positive means fatigue / wear is accumulating.**
• **Negative means healthy margin remains.**
• **Zero means operating at the declared reference cadence or stress level.**
• **Alignment never changes classical values:** `phi((m,a)) = m` always holds.
• **Same math kernel:** all fusions use the established **rapidity U/W form**, ensuring determinism, batch/stream equivalence, and safe composition with existing dials.

Outcomes.
By expressing plant aging and operational fatigue **in the same alignment space** as live electrical posture:

1. **Maintenance is scheduled, not reactive** — preventing avoidable failures.
2. **Worst-case cabinet and feeder nodes surface naturally**, without proprietary scoring systems.
3. **Stress and PQ posture align in one dashboard** — enabling **fast, clean, and explainable routing** to field teams.

Operator Guidance.
This section is not about alarms — it is about **foresight**. Display these dials beside **a_pool** and PQ dials. Let teams **see aging in motion**, weeks before failure signs appear.

7.1 Breaker Fatigue Dial (Normative, Copy-Ready)

Purpose.
Convert the **operation stress history** of a breaker into a **zero-centric, monotone alignment** for proactive maintenance.

- `a_fatigue > 0` ⇒ fatigue risk accumulating.
- `a_fatigue < 0` ⇒ healthy margin (under target stress).
- `a_fatigue = 0` ⇒ operating at the declared target.

Inputs.

- Operation events with timestamps (for example, `open`, `close`, `trip`).
- Optional stress weight per event (for example, `w_trip > w_open > w_close`).
- Optional electro-thermal stress proxy per event (for example, `I2t`, switching energy `E_sw`).
- Reference levels and windows declared in the manifest.

Lens (count-based, default)

Aggregate recent operation frequency against a reference rate:

```
hits_7d   := count_events( last_7_days )
hits_ref  := declared_target_hits_per_7d
hits_norm := hits_7d / max( hits_ref , eps_ref )

a_fatigue := tanh( c_fat * ( hits_norm - 1 ) )
```

- `c_fat > 0` — sensitivity.
- `eps_ref > 0` — floor to avoid division by zero.

Zero-centric: `hits_norm = 1` ⇒ `a_fatigue = 0`.

Lens (weighted events, recommended)

Weight different event types by declared factors:

```
hits_w   := sum_over_events( w_evt ) over last_7_days
hits_ref_w := declared_target_weighted_hits_per_7d

hits_norm := hits_w / max( hits_ref_w , eps_ref )
a_fatigue := tanh( c_fat * ( hits_norm - 1 ) )
```

Typical weights:

```
w_trip  := 3.0
w_open  := 1.5
w_close := 1.0
```

This keeps the dial **dimensionless** but gives **trips** more influence than light operations.

Lens (electro-thermal, optional)

If per-event stress proxy exists, integrate `I2t` or `E_sw`:

```
S_7d  := sum_over_events( proxy_evt )     # for example I2t or E_sw
S_ref := declared_target_stress_per_7d

a_fatigue := tanh( c_fat * ( S_7d / max( S_ref , eps_ref ) - 1 ) )
```

Again, `a_fatigue = 0` when `S_7d` equals the declared reference `S_ref`.

Drift memory (soft hysteresis, optional but recommended)

Smooth short spikes so maintenance cadence remains stable:

```
M_t := rho * M_{t-1} + (1 - rho) * max( 0 , a_fatigue - th_enter )
```

- `rho in [0,1)` — retention of past stress.
- Use `M_t` as an **AND-gate** with raw `a_fatigue` for band entry (see Section 7.3 style):
  - enter only when both `a_fatigue` and `M_t` show sustained stress.

Validity & time base (must declare)

- **Clock:** compute at `tick_ms` (for example, `tick_ms = 60000` for 1-minute ticks).

- **Window:** default `7` days rolling; publish if different (`window_days`).
- **Event integrity:**
  - Reject events lacking **type** or **timestamp**.
  - Do **not** fabricate counts; missing telemetry must surface as `DATA_STALE`, not fake zeros.
- **Quorum:** if `k` consecutive ticks without telemetry, label `DATA_STALE` and hold last valid band for at most **one full window**.

Bands → actions (copy-ready defaults)

```
A+ : a_fatigue <= +0.20                      # healthy; no action
A  : a_fatigue <= +0.35                      # normal monitoring
B  : a_fatigue >= +0.50 for >= 5 days        # WATCH; schedule inspection
next planned window
C  : a_fatigue >= +0.60 for >= 48 hours      # ALERT; create ticket; verify
mechanical counter and coil current
D  : a_fatigue >= +0.75 for >= 12 hours      # CRITICAL; expedite
maintenance or swap to spare
```

Bands are implemented using the standard **ladder hysteresis** from earlier sections with appropriate `th_enter`, `th_clear`, and `t_hold`.

Pseudo-code (copy-ready)

```
on each tick T:

  # 1) Aggregate events over rolling window
  window_start := T - 7d      # or window_days from manifest
  hits := count_or_weighted_sum( events in [window_start , T] )
  norm := hits / max( hits_ref , eps_ref )

  a_raw := tanh( c_fat * ( norm - 1 ) )
  a     := clamp( a_raw , -1 + eps_a , +1 - eps_a )

  # 2) Optional drift memory (gate, for example at band B+)
  if use_memory:
      M := rho * M + (1 - rho) * max( 0 , a - th_enter_B )
  else:
      M := 0

  # 3) Hysteresis band ladder (reuses global logic)
  band_new := ladder_hysteresis( a , band_old , th_enter[] , th_clear[] ,
t_hold[] )

  # 4) Dispatch on band transition
  if band_new != band_old:
      emit_dispatch({
        value       := a,
        band        := band_new,
        dial_id     := "a_fatigue",
        memory      := M,
        action      := route( "a_fatigue" , band_new ),
```

```
        sla_ms      := SLA( band_new ),
        owner       := OWNER( "breaker_maintenance" ),
        tick_ms     := T,
        manifest_id := MID,
        stamp       := chain_stamp( ... )
      })

  band_old := band_new
```

`phi((m,a)) = m` remains intact — **fatigue is a separate dial**, not a modification of any physical magnitude.

Commissioning checks (must-pass)

1. **Window repeatability.**
   - Rolling 7-day computation matches fixed-window offline replay to tolerance (e.g., differences < `1e-6` in `a_fatigue`).
2. **Event weighting determinism.**
   - Switching between **count** and **weighted** modes reflects only the declared weights; no order dependence in the same mode.
3. **Sanity near target.**
   - If `hits_7d = hits_ref` then `a_fatigue ≈ 0`.
   - If `hits_7d = 0` then `hits_norm = 0` and `a_fatigue < 0` (healthy margin).
4. **Spike handling.**
   - Inject a 1-day surge in events. With memory enabled (`rho > 0`), bands must **not flap** on the next day's drop; they should decay according to the chosen half-life.
5. **Telemetry loss.**
   - Remove events for 3 days; dial enters `DATA_STALE`; no escalation without quorum (no new B/C/D bands from missing data).
6. **Stamp chain.**
   - For 50 sequential dispatches, verify continuous `sha256` hash chain (no gaps, no branch, no duplicate stamps).

Manifest keys

```
dial_id        := "a_fatigue"
window_days    := 7
event_weights  := { trip: 3.0 , open: 1.5 , close: 1.0 }   # example
stress_proxy   := "none" | "I2t" | "E_sw"
hits_ref       := <float>   # or
S_ref          := <float>   # if stress-based

c_fat          := <float>
rho            := <float>   # if memory enabled
th_enter       := <per-band thresholds>
th_clear       := <per-band thresholds>
t_hold         := <per-band holds in seconds>

eps_a          := 1e-6
```

```
eps_ref        := 1e-9
tick_ms        := 60000
quorum_days    := <integer>
```

Operator guidance

- Display `a_fatigue`, `hits_7d` (or `S_7d`), and the **next clear target**.
- Provide a **mechanical counter snapshot** and a link to the last N stamps for audit.
- Always show the reference (`hits_ref` or `S_ref`) so that **zero** (`a_fatigue = 0`) retains clear meaning: **"at target stress"**.

Worked micro-example (rounded)

```
hits_ref  = 20 per 7d
hits_7d   = 30 per 7d   → hits_norm = 30 / 20 = 1.5
c_fat     = 1.2

a_fatigue = tanh( 1.2 * (1.5 - 1) )
          = tanh( 0.6 )
          ≈ +0.537
```

Interpretation: `a_fatigue ≈ +0.54` ⇒ **elevated fatigue**; this would typically map to **band B (WATCH)** if held for `>= 5` days under the default band rules.

7.2 Cabinet Stress Composite (Normative, Copy-Ready)

Purpose.
Fuse multiple power-quality dials into a single **cabinet-level posture** for maintenance routing and escalation. The composite is **zero-centric**, **monotone**, and **streaming-equivalent**. Classical values remain intact: `phi((m,a)) = m`.

Inputs.

- Per-dial alignments (post-lens) such as `a_THDV`, `a_unbV`, `a_PF`, `a_I`, optional `a_T`.
- Declared per-dial weights `w_dial >= 0` and numeric guards `eps_a`, `eps_w`.
- Manifest fields that **freeze** the dial set, weights, and tick timing.

Kernel (alignment lane only).

Clamp each dial, map to rapidity, fuse with weights, return to alignment:

```
a_j_c   := clamp( a_j , -1 + eps_a , +1 - eps_a )
```

```
u_j     := atanh( a_j_c )
U       := sum_j( w_j * u_j )
W       := sum_j( w_j )
a_stress := (W == 0) ? 0 : tanh( U / max( W , eps_w ) )
```

- **Associativity / streaming.** The `U/W` form yields identical results for **batch**, **chunked**, and **streaming** updates.

Weighting policy (choose and publish).

1. **Direct weights (default).**
   Set `w_THDV`, `w_unbV`, `w_PF`, `w_I`, `w_T` per policy.
2. **Scaled by criticality.**
3. `w_j := k * r_j`

   where `r_j` are declared risk ranks (for example, `THDV = 10`, `unbV = 6`, `PF = 4`).

4. **Conditional boost (optional).**
   If `a_T >= th_heat`, apply a **decision-only** boost:
5. `w_T_eff := beta * w_T       # beta >= 1`

   This affects **only** the cabinet decision; the raw `a_T` remains unchanged.

Validity and quorum (must).

- Only dials computed on the **same tick** contribute.
- If a dial is `DATA_INVALID` or absent, **exclude** it from `U/W`.
- If fewer than `N_min_dials` remain:
  - set `a_stress := 0`
  - label the tick `QUORUM_LOST` (decision layer may then suppress actions).

Bands → actions (recommended defaults).

```
A+ / A : a_stress <= +0.35                  # normal monitoring
B      : a_stress >= +0.50 for >= 5 min     # WATCH; add to preventive
list
C      : a_stress >= +0.60 for >= 2 min     # ALERT; ticket for targeted
PQ survey
D      : a_stress >= +0.75 for >= 30 s      # CRITICAL; on-call review,
optional remote_shed_% per policy
```

Bands use the standard **ladder hysteresis** (enter/clear thresholds + hold times) defined elsewhere.

Pseudo-code (copy-ready).

```
on tick T:

  U := 0
  W := 0
  n := 0

  for each dial j in DECLARED_DIAL_SET:

    if valid( a_j[T] ):
      a_c := clamp( a_j[T] , -1 + eps_a , +1 - eps_a )
      U   := U + w_j * atanh( a_c )
      W   := W + w_j
      n   := n + 1

  if (W == 0) or (n < N_min_dials):
    a_stress := 0
  else:
    a_stress := tanh( U / max( W , eps_w ) )

  band_new := ladder_hysteresis( a_stress , band_old , th_enter[] ,
th_clear[] , t_hold[] )

  if band_new != band_old and quorum_ok( n , N_min_dials ):
    emit_dispatch({
      value       := a_stress,
      band        := band_new,
      dial_id     := "a_stress",
      action      := route( "a_stress" , band_new ),
      sla_ms      := SLA( band_new ),
      owner       := OWNER( "cabinet_maintenance" ),
      tick_ms     := T,
      manifest_id := MID,
      stamp       := chain_stamp( ... )
    })

  band_old := band_new
```

Commissioning checks (must-pass).

1. **Order-invariance.**
   - Permute dial order; `a_stress` unchanged to `1e-6`.
2. **Batch == Streaming.**
   - Batch, streaming, and random chunking all give identical `a_stress` to `1e-6`.
3. **Knob sanity.**
   - Doubling all `w_j` leaves `a_stress` unchanged.
   - Zeroing one `w_j` removes only that dial's influence.
4. **Monotonicity.**
   - Increasing any included `a_j` (others fixed) **does not decrease** `a_stress`.
5. **Quorum behavior.**
   - With `n < N_min_dials`, actions are suppressed and tick labeled `QUORUM_LOST`.

6. **Heat-boost safety (if used).**
    - `beta` affects **decision weighting only**; raw `a_T` and all physical magnitudes remain untouched.

Worked micro-example (rounded).

```
Declared weights:
  w_THDV = 1.0
  w_unbV = 0.6
  w_PF   = 0.4
  w_I    = 0.8
  w_T    = 0.5

Dials at tick T:
  a_THDV = +0.62
  a_unbV = +0.28
  a_PF   = +0.12
  a_I    = +0.20
  a_T    = +0.15

u := atanh(a):
  u_THDV ≈ 0.724
  u_unbV ≈ 0.287
  u_PF   ≈ 0.121
  u_I    ≈ 0.203
  u_T    ≈ 0.151

Weighted sum:
  U ≈ 1.0*0.724 + 0.6*0.287 + 0.4*0.121 + 0.8*0.203 + 0.5*0.151 ≈ 1.142
  W := 1.0 + 0.6 + 0.4 + 0.8 + 0.5 = 3.3

a_stress := tanh( 1.142 / 3.3 ) ≈ tanh(0.346) ≈ +0.333  → band A
```

If `a_THDV` rises to `+0.70` (`u_THDV ≈ 0.867`), recompute to get `a_stress ≈ +0.401` → still within `A / A+`, **no flapping** around the band boundary.

Manifest keys.

```
dial_id      := "a_stress"
dial_set     := [ "THDV" , "unbV" , "PF" , "I" , "T?" ]
weights      := { THDV: 1.0 , unbV: 0.6 , PF: 0.4 , I: 0.8 , T: 0.5 }
N_min_dials  := <integer>
eps_a        := 1e-6
eps_w        := 1e-9
tick_ms      := 1000
th_enter     := { ... }    # per-band thresholds
th_clear     := { ... }
t_hold       := { ... }
beta         := <float>    # if conditional heat boost enabled
th_heat      := <float>    # trigger level for temperature boost
```

Operator guidance.
Show `a_stress` with a **breakdown card** listing each dial's contribution `w_j * atanh(a_j)` and the totals `U`, `W`. Display the next clear target and **time-in-band**, plus one-click access to the last N stamps so operators can see exactly **why** a cabinet is in a given stress band.

7.3 Drift Memory (Soft Hysteresis for Stability, Normative)

Purpose.
Add a **slow memory** that stabilizes decisions by capturing **recent excess risk** without muting real improvements.

- `M_t > 0` ⇒ accumulated risk pressure.
- `M_t = 0` ⇒ no recent excess.
- Negative values are **not used** (memory is clamped to `>= 0`).

When to use

- Dials that hover near thresholds: `a_THDV`, `a_unbV`, `a_PF`, `a_stress`, etc.
- Sites with **noisy sensors** or frequent **micro-spikes**.
- Commissioning phases where operators prefer **fewer band flips** and more conservative changes.

Core memory form (excess-only EMA)

Let `a_t` be a bounded dial in `(-1,+1)` and `th_enter` its enter threshold.

```
E_t := max( 0 , a_t - th_enter )

M_t := rho * M_{t-1} + (1 - rho) * E_t
```

with `rho` in `[0,1)`.

Meaning.

- `M_t` grows **only when** `a_t` exceeds `th_enter` (excess risk).
- When `a_t` returns below `th_enter`, `E_t = 0` and `M_t` decays **geometrically**.

Two-gate entry (recommended)

Enter the band only if **both** the raw dial and the memory agree:

```
enter if ( a_t >= th_enter ) AND ( M_t >= thM_enter ) for >= t_hold_enter
clear if ( a_t <= th_clear ) AND ( M_t <= thM_clear ) for >= t_hold_clear
```

Recommended choices:

- `thM_enter <= th_enter`
  - Often `thM_enter := 0.5 * (th_enter - th_clear)` or simply `thM_enter := 0`.
- `thM_clear := 0` to allow **quick recovery** once risk subsides and memory decays.

Half-life tuning (operator-friendly)

Choose `rho` via a **time half-life** `T_half` for decay. Let the tick period be `tick_seconds` (for example `tick_seconds = tick_ms / 1000.0`):

```
rho := exp( - tick_seconds / T_half )
```

Rules of thumb.

- **Short memory:** `T_half ≈ 2 * t_hold_enter` (responsive, mild smoothing).
- **Long memory:** `T_half ≈ 5–10 * t_hold_enter` (stable, conservative).

Clamped additive memory (optional, stronger smoothing)

Use only if a **ramp-like** accumulator is explicitly desired:

```
if a_t > th_enter:
    M_t := clamp( M_{t-1} + alpha * ( a_t - th_enter ) , 0 , M_max )
else:
    M_t := beta * M_{t-1}
```

with `alpha, beta` in `(0,1]`.

- Prefer the **EMA form** unless there is a strong reason to expose a ramp-like behavior.

Safety and determinism (must)

- **No feedback into magnitudes:** memory gates decisions only; `phi((m,a)) = m` remains inviolable.
- **Monotone logic:** if `a_t` increases and stays above `th_enter`, `M_t` must **not decrease**.
- **Reset policy:** optionally allow `M_t := 0` on **operator acknowledge** or after `clear` has been held for a full `T_half`.

Pseudo-code (copy-ready)

```
# parameters:
#   th_enter, th_clear
#   t_hold_enter, t_hold_clear
#   thM_enter, thM_clear
#   rho ∈ [0,1)

# state:
#   M_prev        # previous memory value (≥ 0)
#   band_old      # previous band
#   hold_enter_t  # time spent continuously eligible for enter
#   hold_clear_t  # time spent continuously eligible for clear

# 1) Excess-only memory update
E := max( 0 , a - th_enter )
M := rho * M_prev + (1 - rho) * E

eligible_enter := (a >= th_enter) and (M >= thM_enter)
eligible_clear := (a <= th_clear)  and (M <= thM_clear)

# Update hold timers (Δt = tick_seconds):
if eligible_enter:
    hold_enter_t := hold_enter_t + Δt
else:
    hold_enter_t := 0

if eligible_clear:
    hold_clear_t := hold_clear_t + Δt
else:
    hold_clear_t := 0

band_new := band_old

if hold_enter_t >= t_hold_enter:
    band_new := max( band_new , target_band )      # escalate
elif hold_clear_t >= t_hold_clear:
    band_new := downgrade( band_old )              # de-escalate

if band_new != band_old:
    emit_dispatch({ value := a, memory := M, band := band_new, ... })

band_old := band_new
M_prev   := M
```

Commissioning checks (must-pass)

1. **Spike immunity.**
   - A single-tick spike above `th_enter` with `a` back to normal next tick **does not** trigger entry when `t_hold_enter > tick_seconds`.
2. **Staircase rise.**
   - Successive small increases that keep `a` just above `th_enter` eventually fill `M` and enter after `t_hold_enter`.
3. **Decay sanity.**
   - With `a` below `th_clear`, `M` decays toward `0` with half-life `T_half`; clearing occurs once **both** gates qualify.
4. **No deadlock.**
   - Ensure `thM_clear <= thM_enter` and `t_hold_clear <= t_hold_enter` to avoid sticky bands that never clear.
5. **Bitwise determinism.**
   - For a fixed `rho`, replays in batch vs streaming produce **identical** `(M_t, band_t)` given the same ticks and holds.

Operator guidance

- Display `a_t`, `M_t`, **time above enter**, and **time below clear**.
- Show the **next clear target** and the implied **half-life** so the reason for a hold or a release is obvious at a glance.

Manifest keys

```
gate            := "two_gate"
rho             := <float>          # or T_half in seconds
thM_enter       := <float>
thM_clear       := <float>
reset_on_ack    := true | false
memory_export   := true | false     # whether to include M_t in envelopes
```

8.0 — Dispatch, Bands, and Routing (Introduction, Normative Scope)

Purpose. Convert **dial alignments** and **pooled posture** into **deterministic operational actions** with auditable stamps. **Positive means risk**, **negative means margin**, **zero at target**. This section defines the **band ladder**, **hysteresis**, **hold/clear timing**, **routing matrix**, and **portable envelopes**.

Scope. Applies to any SSMEQ dial (e.g., `a_THDV`, `a_unbV`, `a_PF`, `a_I`, `a_df`, `a_pool`, `a_stress`). Decisions act on the **alignment lane only**; **classical values remain intact:** `phi((m,a)) = m`.

Inputs (declared in manifest).
• **Dials:** `a_x` per tick with `tick_ms` and validity.
• **Band ladder:** thresholds `{th_enter[], th_clear[]}` with `th_clear < th_enter`.
• **Hold times:** `{t_hold_enter[], t_hold_clear[]}` per band.
• **Routing matrix:** `(dial_id, band) -> (action, sla_ms, owner)`.
• **Optional gates:** environment factor `g_t in [0,1]`, quorum `N_min` for pools, rate-limit `gap_ms`.

Core mechanism (standardized).

1. **Hysteresis:** enter a band only when `a_x >= th_enter` **continuously** for `t_hold_enter`; clear/downgrade only when `a_x <= th_clear` **continuously** for `t_hold_clear`.
2. **Time-base:** decisions are computed on the same `tick_ms` that produced the dial.
3. **Environment gating (decision-only):** `a_env := g_t * a_x`. Gating **never** modifies stored `a_x`; it **only** modulates eligibility.
4. **Quorum (pooled):** pooled actions require `N_valid >= N_min` for the tick; otherwise label **QUORUM_LOST** and suppress escalation.
5. **Rate-limit:** enforce `now - last_dispatch(dial_id) >= gap_ms` to prevent chatter.

Portable dispatch envelope (audit-ready).

```
{ value := a_x, band, dial_id, pool_id?, action, sla_ms, owner,
manifest_id, tick_ms, stamp }
stamp := sha256( prev_stamp || sha256(payload_bytes) || time_utc )
```

Safety & invariants.
• **No byte edits:** actions never alter classical payloads; `phi((m,a)) = m` always holds.
• **Determinism:** given identical inputs and policy, the routing outcome is identical across batch/streaming executions.
• **Monotone sanity:** if badness increases and all policy knobs are fixed, the eligible band must not improve spuriously.
• **Data integrity:** if a dial is **DATA_INVALID** or **DATA_ABSENT**, either **suppress** or **downgrade to watch** per explicit policy; never fabricate values.

Operator view (minimal, consistent).
Show for each dial or pool: **current band**, **time-in-band**, **next clear target** (`th_clear`), **SLA** and **owner**, plus a one-click list of the **last 20 stamps**.

8.1 — Global Band Table (ASCII, Normative, Copy-Ready)

Purpose. Provide one **portable ladder** of risk bands that every dial (`a_THDV`, `a_unbV`, `a_PF`, `a_pool`, `a_stress`, etc.) can use consistently. **Positive means risk**, **negative means margin**, **zero at target**.

Band ladder.
`A+ (excellent) > A (good) > B (watch) > C (alert) > D (critical)`

Thresholds, clears, and holds (defaults).
Choose **enter** and **clear** (`th_clear < th_enter`) and **hold** intervals per band. You may override per dial/site, but the **global defaults** are:

```
B: enter >= +0.50 for 5 min     ; clear <= +0.35 for 5 min
C: enter >= +0.60 for 2 min     ; clear <= +0.45 for 2 min
D: enter >= +0.75 for 30 s      ; clear <= +0.60 for 60 s
A: enter >= +0.35 for 10 min    ; clear <= +0.20 for 10 min
A+: enter >= +0.20 for 10 min   ; clear <= +0.10 for 10 min
```

Routing matrix (defaults).
Map `(dial, band)` → `(action, SLA, owner)`. Keep it short, deterministic:

```
B → WATCH_TICKET            ; SLA 1 h    ; owner := Maint_plan
C → ALERT_INVESTIGATE       ; SLA 15 min ; owner := PQ_team
D → CRITICAL_REMOTE_SHED_5% ; SLA 30 s   ; owner := Ops_control
A/A+ → NO_ACTION (log only)
```

Core rules (must).
• **Hysteresis:** use `th_clear < th_enter` to avoid flapping.
• **Continuous hold:** the dial must remain on or beyond `th_enter` **continuously** for `t_hold_enter`.
• **Time base:** decisions run on the same `tick_ms` used to compute the dial.
• **No magnitude edits:** actions never change classical bytes; `phi((m,a)) = m`.

Conflicts & tie-break (multiple dials).
When several dials propose different bands at the same tick:

1. Take the **maximum severity** (`D > C > B > A > A+`).
2. If severities tie, pick the **largest alignment value** `a`.
3. If still tied, prefer the **shortest SLA**.
4. Always include `dial_id` in the envelope so the source is explicit.

Per-dial overrides (optional).
Some dials deserve different sensitivity; override by publishing a **dial ladder** that shadows the global defaults:

```
# Example override for a_pool
ladder[a_pool]:
  B: enter >= +0.55 for 5 min ; clear <= +0.40 for 5 min
  C: enter >= +0.65 for 2 min ; clear <= +0.50 for 2 min
```

```
D: enter >= +0.82 for 30 s  ; clear <= +0.68 for 60 s
```

Environment gate (decision-only, optional).
When a declared condition applies (e.g., storm mode), gate the decision but do **not** modify the dial itself:
`a_env := g_t * a` with `g_t in [0,1]` used **only** for band entry checks. Raw `a` remains unchanged for logging and audit.

Rate-limit & quorum (recommended).
• **Rate-limit:** enforce a minimum gap `gap_ms` between dispatches per `(dial, site)`.
• **Quorum:** for pooled actions, require `N_valid >= N_min`; otherwise suspend routing and label **QUORUM_LOST**.

Dispatch envelope (shape).

```
{ value := a, band := BAND, dial_id := DIAL, pool_id? := PID,
  action := ACTION, sla_ms := SLA, owner := OWNER,
  tick_ms := T, manifest_id := MID, stamp := chain_stamp(prev_hash,
payload_hash, time_utc) }
```

Pseudo-code (copy-ready).

```
on each tick T for each dial a:
  a_decide := g_t * a            # if gate enabled; else g_t := 1
  band_new := ladder_hysteresis(a_decide, band_old, th_enter[], th_clear[],
t_hold[])
  if band_new != band_old and passes_rate_limit() and quorum_ok():
     emit_envelope({ value := a, band := band_new, dial_id := id(a), ... })
  band_old := band_new
```

Commissioning checklist (band layer).

1. **Hysteresis sweep:** ramp `a` up/down across each `enter/clear`; verify no flapping.
2. **Hold timing:** plateau at thresholds; confirm enter/clear only after the declared `t_hold`.
3. **Tie-break:** inject two band candidates; ensure the chosen route follows severity → `a` magnitude → SLA priority.
4. **Gate safety:** set `g_t := 0.5`; verify decisions shift while raw `a` and stamps remain auditable and unchanged.

8.2 Environment Gate g_t (Optional, Safe-by-Default, Normative)

Purpose.
Provide a **situational dampener** for dispatch decisions **without altering underlying math or bytes**. The gate never edits data. It only scales the **decision input** so operations can stay safe during special conditions (storms, grid emergencies, maintenance windows).

Collapse parity remains inviolable: `phi((m,a)) = m`.

Principle (do not violate).

Do **not** modify the dial itself. Work only on a decision surrogate:

```
a_env := g_t * a_x
```

with `g_t in [0, 1]`.

- All storage, pooling, and audit remain on `a_x`.
- Only **comparisons against band thresholds** use `a_env`.

Where it applies.

- **Per-site gate:** one `g_t` for the whole site (typical for storms).
- **Per-pool gate:** one `g_t(pool_id)` for a cabinet / feeder.
- **Per-dial gate:** `g_t(dial_id)` when certain dials must be softened (for example, nuisance-prone flicker).
- **Per-band gate (advanced):** different gates for B / C / D ladders where policy wants strictness preserved at the highest band.

Gate composition (declare policy).

A gate can be a **product** or **min** of factors; publish the method and each factor's meaning.

Examples:

```
g_t := g_weather * g_grid * g_maint * g_operator
```

or

```
g_t := min( g_weather , g_grid , g_maint , g_operator )
```

Typical factors:

- `g_weather` (storm mode): in `{1.0, 0.7, 0.5}` by alert level.
- `g_grid` (system emergency): in `{1.0, 0.8}`.
- `g_maint` (declared maintenance window on asset): in `{1.0, 0.6}`.
- `g_operator` (temporary human override with expiry): in `[0.5, 1.0]`.

Time base and hysteresis (must declare).

- **Tick alignment:** evaluate `g_t` on the same `tick_ms` as band logic.
- **Ramps (optional):** limit gate slew to avoid oscillations:
- `g_t := clamp( g_{t-1} + step * sgn( g_target - g_{t-1} ) , 0 , 1 )`

  where `step in (0,1]`.

- **Hold-through:** once a D (critical) action is entered, optionally **ignore gates for the clear decision** to ensure safety closure. Publish this rule explicitly.

Decision substitution (only in routing).

Replace `a_x` with `a_env` **only in the band decision path**:

```
a_env := g_t * a_x

enter if a_env >= th_enter_x  for >= t_hold_enter_x
clear if a_env <= th_clear_x  for >= t_hold_clear_x
```

- The stored dial and pooled values remain `a_x` and `a_pool`.
- Collapse parity continues to hold: `phi((m,a_x)) = m`.

Precedence and floors (recommended).

- **Floor for safety visibility:** publish `g_min_vis` so the UI never hides more than allowed; display uses raw `a_x` plus a badge like `gate:0.7`.
- **Band floors:** optionally cap gate at each ladder step, for example `g_min_D := 0.9` so band D is barely damped:
- `g_eff := max( g_t , g_min_band( band ) )`

  Use `g_eff` for decisions; `g_t` is still recorded for transparency.

Stamping and audit (must).

Every dispatch envelope must record the gate state used for the decision:

```
{
  value      := a_x,
  value_env  := a_env,
  gate       := g_eff,     # or g_t, if floors applied separately
  band       := band_new,
  dial_id    := ...,
  ...
  stamp
}
```

This preserves **what happened and why** and allows full replay.

Pseudo-code (copy-ready).

```
on each tick T:

  for each dial a_x:

    # 1) Compute or fetch environment gate
    g_raw := compose_gate( T , site_id , pool_id , dial_id )   # in [0,1]

    # 2) Apply slew and band floors (optional)
    g_slew := clamp( g_prev + step * sgn( g_raw - g_prev ) , 0 , 1 )
    g_eff  := max( g_slew , g_min_band( band_old ) )          # band
floors, optional

    # 3) Decision surrogate
    a_env := g_eff * a_x

    # 4) Band logic uses only a_env
    band_new := ladder_hysteresis( a_env , band_old , th_enter[] ,
th_clear[] , t_hold[] )

    if band_new != band_old and passes_rate_limit( a_x ) and quorum_ok( a_x
):

       emit_envelope({
         value       := a_x,               # raw alignment
         value_env   := a_env,             # gated decision input
         gate        := g_eff,
         band        := band_new,
         dial_id     := id( a_x ),
         action      := route( dial_id , band_new ),
         sla_ms      := SLA( dial_id , band_new ),
         owner       := OWNER( ... ),
         tick_ms     := T,
         manifest_id := MID,
         stamp       := chain_stamp( ... )
       })

    band_old := band_new
    g_prev   := g_slew
```

If policy chooses to **ignore gates for clearing D**, override the clear logic when `band_old == "D"` and `band_new` would be lower, per manifest.

Commissioning checks (must-pass).

1. **Gate neutrality.**
   - With `g_t = 1` for all ticks, results are **identical** to ungated runs (same bands, same timestamps, same stamps).

2. **Gate monotonicity.**
    - For a fixed `a_x` sequence, decreasing `g_t` **cannot cause earlier band entry** than at higher `g_t`.
3. **Critical floor.**
    - When band is D and policy sets `g_min_D`, verify that effective `g_eff` respects the floor (no over-damping of **critical** conditions).
4. **Audit continuity.**
    - Envelopes include `gate` and `value_env`.
    - Recomputed `value_env := gate * value` from archives matches logged `value_env` exactly (up to numeric tolerance).
5. **Ramp sanity.**
    - With slew limits, band transitions may shift in time, but sequences remain **logically consistent** — no flapping or oscillatory entry/clear loops caused by gate motion alone.
6. **UI truthfulness.**
    - UI shows `a_x` (truth) and gate badge; operator can reconstruct `a_env` and see how much damping was applied.

Operator guidance.

- Always display **raw alignment** and gate in the same card, for example: `a_x = +0.64, gate = 0.7 → a_env = +0.45`.
- Provide a **timer** for gate expiry and a **reason tag** (for example, `storm:YELLOW`, `grid:EMERGENCY`).
- Offer a **forensics shortcut**: "Replay with gate = 1" to see what would have happened without damping.

Manifest keys.

```
gate_policy    := "product" | "min"
gate_factors   := { weather , grid , maint , operator }
gate_values    := { factor: value }          # for example, weather:0.7,
grid:0.8

slew_step      := <float in (0,1]>          # max change per tick
g_min_vis      := <float in [0,1]>          # visibility floor for UI
g_min_band     := { A:1.0 , B:0.8 , C:0.8 , D:0.9 }  # example floors

ignore_gate_on_clear_for := [ "D" ]         # optional list of bands
expiry_ms     := <for operator overrides>
tick_ms       := <same tick as band logic>
```

This keeps the **environment gate** a pure **policy overlay**: all physics, encoding, pooling, and `phi((m,a)) = m` remain untouched.

8.3 — Stamp Semantics (Normative, Copy-Ready)

Purpose. Ensure every **decision/event** is portable, tamper-evident, and replay-verifiable across tools and vendors. A **stamp** cryptographically links the event payload to its immediate predecessor, forming a **hash chain** that is easy to audit and cheap to store.

Canonical payload (strict shape).

```
payload := {
  value:        <alignment a_x at tick>,            # e.g., +0.606
  band:         <A+|A|B|C|D|DATA_INVALID|DATA_ABSENT|QUORUM_LOST>,
  dial_id:      <string>,                           # e.g., "a_THDV_pool"
  pool_id:      <string|null>,                      # e.g., "feeder_F01"
  action:       <string>,                           # e.g., "ALERT"
  sla_ms:       <int>,                              # e.g., 900000
  owner:        <string>,                           # e.g., "PQ_team"
  manifest_id:  <string>,                           # immutable policy
version
  tick_ms:      <int>,                              # tick instant (ms
since epoch)
  issued_ms:    <int>,                              # envelope creation (ms
since epoch)
  notes:        <string|null>                       # optional human hint
}
```

Canonical serialization (deterministic).

- **Encoding:** UTF-8, no BOM.
- **Key order:** exactly as shown above.
- **Numbers:** fixed decimal for `value` with 6 places (e.g., `+0.606000`).
- **Whitespace:** none inside JSON (minified).
- **Line ending:** `\n` for emitted records.

Hash & stamp (chain).

```
payload_bytes := serialize(payload)
payload_hash  := sha256(payload_bytes)          # 64 hex, lowercase
stamp         := sha256(prev_stamp || payload_hash || issued_ms_utf8)
record        := { payload, payload_hash, prev_stamp, stamp }
```

- `prev_stamp := "0"*64` for the first record in a chain (genesis).
- `||` denotes byte concatenation.
- `issued_ms_utf8` is the ASCII/UTF-8 string of the integer `issued_ms` (no padding).

Replay verification (copy-ready).

```
verify_chain(records):
  expect_prev := "0"*64
  for r in records in file order:
    bytes := serialize(r.payload)
    assert sha256(bytes) == r.payload_hash
    assert sha256(expect_prev || r.payload_hash ||
str(r.payload.issued_ms)) == r.stamp
    expect_prev := r.stamp
```

```
return PASS
```

Idempotency & de-duplication.

- A duplicate event (identical `payload_bytes`, `payload_hash`, `prev_stamp`, `stamp`) may be ignored safely.
- If `payload_hash` matches but `prev_stamp` differs, you have a **fork** (see below).

Fork handling (rare, normative).

- **Detect:** a given `payload_hash` observed with two different `prev_stamp` values.
- **Policy:** keep the **first-seen** branch for operations; log the **second** as `FORK_DETECTED` with both stamps; do not merge.
- **Operator disclosure:** expose a one-click "Forks" panel listing `(stamp_A, stamp_B, payload_hash)`.

Privacy & redaction.

- Keep the `payload` free of PII. If a note contains PII, replace with a redaction token **before hashing** (redaction must happen at source).
- Never hash external blobs directly; if needed, store their independent digest separately and reference via a **non-PII token** in `notes`.

Retention & size.

- Target **≤ 400 bytes** per `record` on average (minified JSON).
- Suggested retention: **last 90 days hot**, beyond that archive with the chain intact.

Rate limits & flow control.

- Per `(dial_id, pool_id)` enforce a minimum **inter-dispatch gap `gap_ms`**. Events within `gap_ms` are **coalesced** by updating `notes` (or dropped by policy) but the chain still advances.

Stamp positions in lifecycle.

- **Dial stage:** no stamp.
- **Band decision:** stamp the **decision** (after hysteresis).
- **Dispatch emit:** use the same record; do **not** restamp unless the payload changes.

Minimal record example (single line).


```
{"payload":{"value":+0.606000,"band":"C","dial_id":"a_THDV_pool","pool_id":
"feeder_F01","action":"ALERT","sla_ms":900000,"owner":"PQ_team","manifest_i
d":"MID","tick_ms":1731234567000,"issued_ms":1731234567123,"notes":null},"p
ayload_hash":"e3d0...c9a1","prev_stamp":"0000...0000","stamp":"7b9c...4df2"
}
```

Commissioning checks (must-pass).

1. **Determinism:** re-serialize the same payload on two machines → identical `payload_hash`.
2. **Continuity:** 100 records verify with `verify_chain` (no breaks, no orphans).
3. **Clock sanity:** `issued_ms >= tick_ms`. If violated, label `CLOCK_DRIFT` and keep the stamp (math remains valid).
4. **Fork alarm:** inject a synthetic fork; UI surfaces it within 1 tick; operations remain on first-seen branch.
5. **Size budget:** random sample of 1k records → average `<= 400` bytes.

Operator guidance (UI hints).

- Show the **last 20 stamps** and a **copy button** for any record.
- Provide a **"Verify Locally"** snippet that users can run to recompute `payload_hash` and `stamp`.
- Surface clear icons: **chain OK**, **fork detected**, **clock drift**, **rate-limited**.

Notes on classical values.

- Stamps certify **decisions**, not raw bytes. Classical magnitudes remain untouched throughout; alignment never alters `m`.

9.0 — Commissioning, Validity, and Safety

Purpose. Define the **operational guardrails** that keep SSMEQ safe, deterministic, and auditable from first power-on through steady state. This section binds **math invariants** to **field practices** so that alignments remain truthful representations of electrical posture while **classical values stay untouched**: `phi((m,a)) = m`.

What this section guarantees.
• **Determinism:** identical inputs and policy produce identical outputs (**order-invariant**, **batch==stream==chunked**) using the `U/W` rapidity fuse: `a := tanh( sum(w*atanh(a_i)) / max(sum(w), eps_w) )`.
• **Zero-class integrity:** zero remains meaningful across lenses and pools; `a=0` iff the dial is at its declared reference.
• **Safety by declaration:** every decision path is controlled by **published** lenses, anchors, weights, thresholds, and hold times—no hidden factors.
• **Classical non-interference:** alignment never changes magnitudes; roll-ups of `m` remain `M := sum(m_i)` independent of `a`.
• **Auditability:** each dispatch emits a **stamped** envelope `{value, band, dial_id, manifest_id, tick_ms, stamp}` with hash continuity.

Core validity rules (must).

1. **One lens per channel per run.** Freeze `lens_id` and **anchors**; mid-run lens swaps are rejected.
2. **Published windows.** Each raw source declares its valid operating bounds (e.g., `V_min_valid..V_max_valid`, `df_max_valid`). Samples outside are **invalid**—they are dropped or explicitly marked **DATA_INVALID** per policy.
3. **Tick discipline.** All dials and pools compute on a declared `tick_ms`; only samples within the tick window fuse together.
4. **Numerical clamps.** Always use `a_c := clamp(a, -1+eps_a, +1-eps_a)` and `max(W, eps_w)`; no NaNs, no Inf, no divide-by-zero.
5. **Quorum for pooled actions.** If `N_valid < N_min` at a tick, pooled dispatch **suspends** and labels **QUORUM_LOST**.
6. **Policy immutability.** `weight_basis`, `gamma`, `w_declared`, thresholds, and hold times are immutable during a run; any change requires a **new manifest**.

Commissioning flow (from dry-run to live).
• **Day-0: Shadow mode.** Ingest, lens, and pool, but **no actions**; verify determinism against golden vectors and confirm **order-invariance**.
• **Day-1: S1 payload (observe).** Emit envelopes to a sand-boxed sink; validate validity gates, quorum behavior, and stamp chain continuity.
• **Day-7: S2+ bands (route).** Enable routing with rate-limit and environment gate. Demonstrate **hold/clear hysteresis** on at least one synthetic sweep per dial.
• **Go/No-Go:** proceed only if numeric tolerances hold (`abs(diff) <= 1e-6` on the golden set) and **no hidden knobs** exist.

Failure-mode handling (must declare).
• **Data absent:** if a channel misses `k` ticks, mark **DATA_STALE(channel, k)**; pooled actions require quorum.
• **Out-of-range raw:** drop sample; do not invent `a`.
• **Clock drift:** if tick skew exceeds `tick_ms * drift_ratio`, quarantine the tick and emit **CLOCK_SKEW**.
• **Saturation:** values at `|a| >= 1 - eps_a` stay clamped; pooling proceeds safely.
• **Back-pressure:** if routing is congested, queue envelopes with **monotone timestamps**; never reorder within a tick.

Operator view (what to display).
• **Per tick:** `a_pool`, `N_valid/N_total`, **effective W**, current **band**, **time-in-band**, and **next clear target**.
• **Verification shortcuts:** one-click **Shuffle & Verify** (replay tick 10×), one-click **Stamp Trail** (last 20 stamps), and a **Policy Digest** pane (frozen knobs for this run).

Acceptance snapshot (single line to archive).
`commissioning_stamp := sha256( inputs_hash || expected_hash || time_utc )` where `inputs_hash := sha256(members.csv || samples.csv || policy.json)` and `expected_hash := sha256(expected.csv)`.

9.1 Validity Windows (Electrical, Normative, Copy-Ready)

Purpose.
Ensure every dial and pooled posture is computed only from **valid, in-bounds, time-aligned measurements**. Invalid samples are never fabricated; they are dropped or explicitly flagged per policy. This guarantees **determinism, safety, and auditability**.

Scope.
Applies to all raw channels behind lenses (for example, `V_rms`, `I_rms`, `THD_V`, `THD_I`, `f`, `PF`, `P_st`, `T`) and to any derived dial `a_x` computed at the declared `tick_ms`.

Declare validity for each raw channel (must).

Publish min/max, jitter, and freshness limits; example keys:

```
V_min_valid, V_max_valid
I_min_valid, I_max_valid
THD_max_valid
f_min_valid, f_max_valid
PF_min_valid, PF_max_valid
Pst_max_valid
T_min_valid, T_max_valid

max_jitter_ms   # max allowed age difference vs tick
max_skew_ms     # max allowed inter-channel timestamp skew
stale_after_ms  # telemetry freshness bound
```

Per-sample validation (must).

Given raw `(x, t_x)` for channel `X` at tick `T`:

```
in_range := ( X_min_valid <= x <= X_max_valid )
fresh    := ( abs( T - t_x ) <= stale_after_ms )
onjitter := ( abs( t_x - T ) <= max_jitter_ms )
valid    := in_range AND fresh AND onjitter
```

If `valid = false`, **do not emit** lens output for this channel at tick `T`.

Inter-channel skew rule (for multi-input lenses).

For dials needing multiple channels (for example, unbalance, PF from `P/Q`): all required inputs must satisfy

```
abs( t_i - t_j ) <= max_skew_ms
```

Else the dial for tick `T` is **invalid**.

Invalid handling (policy, pick one and publish).

- **Drop**
  - Exclude invalid members from pooling.
  - If all members invalid → `a_pool := 0` and emit `DATA_ABSENT`.
- **Mark-invalid**
  - Emit a dial record with `status := DATA_INVALID` (no alignment value); pooling excludes it.
- **Grace-hold**
  - Optionally hold last valid alignment for at most `hold_invalid_ms`, marked `HELD_VALUE`;
  - Pooling may include held values **only** if policy allows.
  - Default is **no hold**.

Anti-fabrication rule (must).

No replacement with zeros, means, or interpolations **inside the lens**.
If smoothing is needed, it must be declared explicitly as part of the lens and still operate only on **valid** inputs.

Time base integrity (must).

- All dials at a tick `T` use inputs whose timestamps satisfy `max_jitter_ms` and `max_skew_ms`.
- Pooling in Section 6 uses only dials stamped at the **same tick** `T`.

Operator-facing counters (recommended).

Expose per tick:

```
N_total, N_valid, N_invalid, N_stale, N_skew_fail
```

plus a compact **reason histogram**.

Pseudo-code (copy-ready).

```
function compute_dial_at_tick( T ):

  inputs      := read_raw_channels( T_window )
  valid_flags := []

  for each channel X in inputs:

    in_range := ( X_min_valid <= X.value <= X_max_valid )
    fresh    := ( abs( T - X.timestamp ) <= stale_after_ms )
    onjitter := ( abs( X.timestamp - T ) <= max_jitter_ms )

    valid_flags[X] := in_range AND fresh AND onjitter

  if dial_requires_multiple_channels:
    if any pair (i,j) violates max_skew_ms:
      return DATA_INVALID

  if any required channel invalid:
    return DATA_INVALID

  return lens( inputs )   # produces a in (-1,+1) with clamp
```

Pooling with validity (copy-ready).

```
U        := 0
W        := 0
N_valid  := 0

for each member dial a_i at tick T:

  if status( a_i ) == OK:
    a_c := clamp( a_i , -1 + eps_a , +1 - eps_a )
    U   := U + w_i * atanh( a_c )
    W   := W + w_i
    N_valid := N_valid + 1

a_pool := (W == 0) ? 0 : tanh( U / max( W , eps_w ) )

if N_valid == 0:
  emit_status( pool_id , T , "DATA_ABSENT" )
else:
  emit( pool_id , T , a_pool )
```

Commissioning checks (must-pass).

1. **Bounds enforcement.**
   - Inject values outside `[min,max]` → dial becomes `DATA_INVALID`, never a numeric `a`.
2. **Jitter & skew.**
   - Advance one channel timestamp beyond `max_skew_ms` → composite dial is `DATA_INVALID`.

3. **Freshness.**
    - Delay all inputs past `stale_after_ms` → `DATA_ABSENT` for pool if no valid dials remain.
4. **No fabrication.**
    - Verify harness never substitutes zeros or averages for invalid inputs.
5. **Audit trail.**
    - Reasons for invalidity are counted and exposed (for example, `OUT_OF_RANGE`, `STALE`, `SKEW_FAIL`).

Manifest keys (publish).

```
validity: {
  per_channel_bounds...,
  max_jitter_ms,
  max_skew_ms,
  stale_after_ms,
  invalid_policy,
  hold_invalid_ms
}
tick_ms: <...>
eps_a:   <...>
eps_w:   <...>
```

Operator guidance.
Display a small **validity badge** next to each dial and pool: `OK`, `DATA_INVALID(reason)`, `DATA_ABSENT`, or `HELD_VALUE(ttl)`. Provide a one-click list of the last 20 invalidity reasons to speed root cause analysis.

9.2 — Site Profiles & Anchor Governance (Normative, Copy-Ready)

Purpose. Define how a **site profile** bundles anchors, PF references, and validity windows into a single manifest, and how those parameters may evolve over time without ever breaking reproducibility or confusing audits.

A **site profile** is the "electrical identity card" for a location or asset group.
It answers, in one place:

- What "normal" looks like (anchors, pf_ref, pf_min).
- What is considered valid (bounds, jitter, stale limits).
- Which band table + environment gates are in force.

All of this is frozen under a **versioned profile_id**.

9.2.1 — What Is a Site Profile?

A **site profile** is a manifest-level record that states:

1. **Anchors & References**
   - `V_ref`, `I_ref`, `P_ref`, `f_ref`
   - `THDV_ref`, `THDI_ref`, `UnbV_ref`, `UnbI_ref`
   - `pf_ref` (desired), `pf_min` (floor for residuals & lenses)
2. **Validity Windows** *(ties into 9.1)*
   - `V_min_valid`, `V_max_valid`
   - `I_min_valid`, `I_max_valid`
   - `THD_max_valid`, `f_min_valid`, `f_max_valid`, `T_min_valid`, `T_max_valid`
   - `max_jitter_ms`, `max_skew_ms`, `stale_after_ms`
3. **Band & Gate Policy**
   - Band table: `bands.json` (A^+…D entries + hysteresis)
   - Environment gate config: `gate_policy`, `gate_factors`, `pf_floor_policy`
4. **Profile Metadata**
   - `site_id`, `profile_id`, `profile_version`
   - `effective_from`, optional `effective_to`
   - `commissioning_stamp` for this profile

A site profile is **immutable once activated**. Any change to anchors or validity windows must be done via a **new profile version**, never by editing in place.

9.2.2 — Site Profile Manifest (Copy-Ready Skeleton)

File: `site_profile.json` (or included under a top-level `site_profiles` block in the main manifest).

```
{
  "site_id": "PLANT_01_LV_50HZ",
  "profile_id": "PLANT_01.LV50",
  "profile_version": "1.0",
  "effective_from": "2025-11-12T00:00:00Z",

  "anchors_ref": {
    "V_ref": 230.0,
    "I_ref": 10.0,
    "P_ref": 2300.0,
    "f_ref": 50.0,
    "THDV_ref": 5.0,
    "THDI_ref": 8.0,
    "UnbV_ref": 2.0,
    "UnbI_ref": 2.0,
    "pf_ref": 1.0,
    "pf_min": 0.3
  },

  "validity": {
    "V_min_valid": 0.8,
    "V_max_valid": 1.2,
```

```
    "I_min_valid": 0.0,
    "I_max_valid": 2.0,
    "THD_max_valid": 50.0,
    "f_min_valid": 47.0,
    "f_max_valid": 53.0,
    "T_min_valid": -20.0,
    "T_max_valid": 90.0,
    "max_jitter_ms": 500,
    "max_skew_ms": 200,
    "stale_after_ms": 3000,
    "invalid_policy": "DROP",
    "hold_invalid_ms": 0
  },

  "bands": "bands.json",
  "environment_gate": {
    "gate_policy": "product",
    "gate_factors": ["weather", "grid", "maint", "operator"],
    "pf_floor_policy": "use_pf_min"
  },

  "commissioning_stamp": "sha256(...inputs||outputs||time...)"
}
```

Notes:

- `V_min_valid` / `V_max_valid` can be expressed as **multipliers** of `V_ref` if preferred; same for current.
- `invalid_policy` and `hold_invalid_ms` match Section 9.1 semantics.
- `bands` and `environment_gate` must reference their respective normed structures (Sections 8.2, 13.3, etc.).

9.2.3 — Governance: How Anchors Change (Version Bump Only)

Anchors **will** change over the life of a site:

- Transformer tap changes → new nominal voltage.
- Plant re-rating → new `I_ref` or `P_ref`.
- PF policy changes → new `pf_ref` or `pf_min`.
- Meter upgrades → better-valid THD/Unbalance thresholds.

Rule 1 — No Silent Change.

Once a profile is commissioned, you **must not** alter any of:

- `anchors_ref.*`
- `validity.*`
- `bands` reference or content
- `environment_gate` keys

in-place.

Any such change produces a **new profile**:

```
profile_version: "1.0" → "1.1" → "2.0" (for major respec)
```

Rule 2 — Commissioning Before Activation.

New profile activation flow:

1. Clone old profile → `profile_version: "1.1"`.
2. Adjust anchors / validity as required.
3. Generate a **Repro Pack** (Section 12.3) for the new profile:
    - `golden.csv` using new anchors.
    - `expected.csv` from a trial run.
    - `policy.json` + `site_profile.json`.
4. Run the harness (Section 13.5); obtain a **new commissioning_stamp**.
5. Attach this stamp to the profile before `effective_from`.
6. Archive old profile as read-only with its own commissioning stamp.

Rule 3 — Time-Bounded Profiles.

For audits and rollbacks:

```
profile_v1.0 => effective_from: 2025-01-01, effective_to: 2025-11-11
profile_v1.1 => effective_from: 2025-11-12, effective_to: null (active)
```

Every envelope (Section 10.3) references a **manifest_id / profile_id** that can be resolved to a unique site profile at the time of computation.

9.2.4 — Runtime Application (Copy-Ready Pseudo-Code)

At runtime, all dials must be evaluated **under the active profile**:

```
function select_site_profile(site_id, t_utc):
    # load all profiles for site_id
    profiles := site_profiles[site_id]
    # pick the one where effective_from <= t < effective_to (or open-ended)
    return argmax_p( p.effective_from ) where in_range(p, t_utc)

profile := select_site_profile(site_id, T)

anchors   := profile.anchors_ref
validity  := profile.validity
bands_tbl := load(profile.bands)
gate_conf := profile.environment_gate

# lens & validity use profile-specific anchors and windows
a_x       := lens_x(raw_x, anchors, validity)
status_x  := validity_check(raw_x, validity)

# pooling & banding know which profile they are under
a_pool    := pool_dials(a_x, weights, eps_a, eps_w)
band      := band_lookup(a_pool, bands_tbl)
```

In envelopes:

```
{
  "site_id": "PLANT_01_LV_50HZ",
  "profile_id": "PLANT_01.LV50",
  "profile_version": "1.1",
  "dial_id": "a_THDV_pool",
  "value": 0.606,
  "band": "C",
  "manifest_id": "SSMEQ.PQ.v1",
  "stamp": { "...": "..." }
}
```

This guarantees that replay logic always knows **which anchors and validity rules** were in force when a given value was computed.

9.2.5 — Governance Checklist (Copy-Ready)

Before changing any anchors or validity windows, ensure:

1. **Justification recorded.**
   - Reason: tap-change, rating change, safety rule update, etc.
   - Logged in `manifest.log`.
2. **New profile created.**
   - `profile_version` incremented.
   - Changes localized under `anchors_ref` / `validity`.
3. **Repro Pack created and tested.**
   - `golden.csv`, `policy.json`, `expected.csv`, `site_profile.json`.
   - Harness PASS with new commissioning stamp.
4. **Effective range set.**
   - Old profile `effective_to` set.
   - New profile `effective_from` aligned to planned cutover.
5. **Operator communication.**
   - Dashboards indicate “Profile updated: v1.0 → v1.1 (reason)”.
   - Investigator View can filter by `profile_version`.

9.2.6 — Operator & Auditor Guidance

- **Operators** should see a simple tag like:
  `"Profile: PLANT_01.LV50 v1.1 (since 2025-11-12)"` near all dials.
- **Auditors** should be able to:
  - Filter events by `profile_version`.
  - Retrieve the exact `site_profile.json` and its `commissioning_stamp`.
  - Confirm that no envelopes exist with fields inconsistent with the then-active profile.

Outcome.
Site profiles and anchor governance ensure that SSMEQ remains:

- **Truthful**: no "silent" changes to what "normal" means.
- **Reproducible**: every value can be recomputed from the correct anchors.
- **Auditable**: anchor history is explicit, versioned, and stamp-protected.

9.3 Day-0 to Day-7 Rollout (Practical Playbook, Copy-Ready)

Purpose.
Bring SSMEQ online **safely and deterministically**—from pure shadow observation to banded dispatch—with **frozen policies**, **auditable stamps**, and **clear rollback points**. Classical values remain intact: `phi((m,a)) = m`.

Scope.
Applies to all dials from Sections 6–7, pooling (Section 6.3), and dispatch (Section 6.5). **One manifest governs the site.**

Day-0 — Shadow (No-Impact Dry Run)

- **Freeze policy** (no changes during the week):
  `lens_id[]`, `anchors_ref`, `weight_basis`, `gamma` or `weights[]`, `eps_a`, `eps_w`, `tick_ms`, `bands`, `N_min`.
- **Mirror ingest**: compute all `a_x` and `a_pool` silently; **no tickets, no actions**.
- **Golden replay**: run Section 6.7 conformance tests; store `commissioning_stamp`.
- **Shadow KPIs** (per tick or per hour):
  - `pct_valid`, `N_valid / N_total`,
  - `max|a_x|`, `max|a_pool|`,
  - `band_histogram` (expected mostly A+/A).
- **Go/No-Go gate**: require **0 policy drift** and **PASS** on determinism (batch == streaming == chunked) before moving to Day-1.

Day-1 — S1: Observe + Minimal Signals (No Tickets)

- **Emit envelopes** to a **sandbox topic only**:
  `{ value, band, dial_id, manifest_id, stamp }`.
- **Hysteresis audit**: verify ENTER/CLEAR edges with hold times; check **no flapping** on synthetic sweeps.
- **Operator UI**: show `a_pool`, `N_valid`, `W`, time-in-band, and last 20 stamps per dial/pool.

- **Rate-limit rehearsal**: enforce `gap_ms` in sandbox; log **blocked duplicates** (no user impact).
- **Safety gate**: if any dial exceeds **band D** for `>= 10 min`, escalate to **manual review only** (no automation).

Day-2..3 — S2: Watch-Only Tickets (Band B)

- **Enable routing for band B (WATCH) only**; all higher bands (C, D) remain **sandboxed**.
- **Ticket SLA rehearsal**: verify ownership paths, queues, and acknowledgements; **no remote actions** yet.
- **Quorum enforcement**: require `N_valid >= N_min` for pooled signals; otherwise label `QUORUM_LOST` and suppress dispatch.
- **Stamp chain audit**: verify **continuous, gapless hash chain** across all envelopes for the period.

Day-4..5 — S3: Alerts (Band C)

- **Enable band C** with real tickets and on-call notifications; still **no remote shed**.
- **Environment gate `g_t`**: verify storm/maintenance dampers on the **decision path only** (`a_env := g_t * a_x`), storage remains on `a_x`.
- **Playback drills**:
  - Inject historic stress slices into the engine,
  - Confirm **identical outcomes** (U/W parity; batch == streaming).
- **KPI fence**: confirm
  - `false_positive_rate_C <= target`,
  - mean time-to-clear within policy bounds.

Day-6..7 — S4: Critical Actions (Band D)

- **Enable D actions** (for example, `remote_shed_5%`) **behind an operator confirm gate** for at least the first 24 h.
- **Rollback switch**: single toggle returns **D to sandbox** while keeping **C active**.
- **Post-action audit**:
  - Verify that classical streams remain **byte-for-byte identical**;
  - Alignment never alters `m` (`phi((m,a)) = m`).
- **Finalize**: remove operator confirm after 24 h **only if** KPIs and safety checks are met.

Operational Guardrails (must)

- **Immutability.** Any mid-run change to lens, anchors, weights, bands, or tick parameters must:
  - Halt outputs,
  - Raise `POLICY_DRIFT`,
  - Require re-commissioning before re-enable.
- **Validity first.** If a raw channel violates bounds or freshness, **do not synthesize `a`**; label `DATA_INVALID`.
- **Empty set.** If `W = 0` at a tick, set `a_pool := 0` and label `DATA_ABSENT` (no dispatch for that tick).
- **Clock truth.** Actions are computed on the **same `tick_ms`** as dials; holds count continuous time only (no backdating).
- **Isolation.** SSMEQ is **observation-only math**; control systems remain separate modules, even when they consume bands.

Rollback & Recovery (copy-ready)

```
if policy_drift_detected() or KPI_threshold_breached():

  set routing_mode := "sandbox"          # all bands to sandbox
  disable_remote_actions()               # D disabled; C tickets paused

  emit_admin_event(
    "ROLLBACK",
    reason,
    manifest_id,
    stamp
  )

  require re_commissioning_with_conformance_PASS()
  # before re-enabling any production routing
```

Minimal Sign-Off Checklist (end of Day-7)

1. **Determinism.** `batch == streaming == chunked` to numeric tolerance.
2. **Hysteresis.** No flapping on synthetic sweeps; holds verified across bands.
3. **Quorum.** Pooled dispatch blocked whenever `N_valid < N_min`.
4. **Stamp chain.** Continuous across the entire week; random spot-recompute of `sha256(payload_bytes)` matches stored value.
5. **KPI gates met.** Target false-positive rate, SLA adherence, and operator acknowledgements all within policy.
6. **Rollback drill passed.** Sandbox switch effective within one tick; no stuck states.

One-Line Week Stamp (archive)

```
week_stamp := sha256(
  commissioning_stamp ||
  ops_metrics_digest ||
  last_stamp ||
  time_utc
)
```

Archive `week_stamp` along with the manifest and conformance logs as the **canonical proof** of the first SSMEQ rollout week.

10.0 — Interop Bridges (SSM • SSMS • SSMDE • SSMT)

Purpose. Define a **clean, portable handshake** so SSMEQ dials and pools interoperate with the broader symbolic stack **without altering classical bytes**. Everything here is **observation-only math**: alignment travels; magnitudes stay original.

What stays invariant (non-negotiables).
• **Collapse parity:** `phi((m,a)) = m` (alignment never edits the classical value).
• **Kernel identity (alignment lane):** `a_c := clamp(a, -1+eps_a, +1-eps_a)` → `u := atanh(a_c)` → `U += w*u ; W += w` → `a_out := tanh( U / max(W, eps_w) )`.
• **Determinism:** batch == streaming == chunked for the same tick; order-invariant `U/W`.
• **Zero semantics:** targets map to `a = 0`; **positive means risk**, **negative means margin**.

Minimal contract per bridge.
• **SSM (math backbone).**
– Preserve `phi((m,a)) = m`, keep all lens math and pooling on the rapidity fuse, and expose `U, W, eps_a, eps_w` for audit.
– Provide `s_eq`, `s_gt`, and simple algebra over `a` with no hidden vendor multipliers.

• **SSMS (operator canon).**
– Use canonical operator names in docs and code: `s_sum`, `s_mul`, `s_div`, `s_gt`, `s_eq`, `s_band`.
– Keep formulas ASCII-only and stable across assets so copy/paste evaluates identically.

• **SSMDE (data envelopes).**
– Emit dispatch and dial events as envelopes: `{ value, align, dial_id, pool_id?, band?, manifest_id, tick_ms, stamp }`.
– `stamp := chain_link(prev_hash, sha256(payload_bytes), time_utc)` for reproducible audit.

• **SSMT (thermal crossover).**
– Optional thermal dials `a_T` integrate exactly like electrical dials via `U/W`.
– No thermal coupling may edit electrical `m`; coupling only gates **decisions** (e.g., `a_env := g_t * a_op`).

Portable envelope shape (copy-ready).

```
{
  value      := a,                  # alignment in (-1,+1)
  dial_id    := "a_THDV" | "a_pool" | ...,
  pool_id    := "cabinet_X" | "feeder_Y" | null,
  band       := "A+"|"A"|"B"|"C"|"D" | null,
  manifest_id:= MID,
  tick_ms    := T,
  stamp      := sha256(prev_stamp || sha256(payload_bytes) || time_utc)
}
```

Interop compatibility cues.
• **Time:** declare `tick_ms` and any window per dial so SSM-Clock/Stamp can reproduce sequences.
• **Numbers:** publish `eps_a`, `eps_w`, rounding (display-only). Never hide scaling inside a lens.
• **Weights:** publish `weight_basis` (`uniform|scale|declared`), `gamma` or `weights[]`.
• **Validity:** publish per-channel validity bounds; never fabricate `a` on breach—mark **DATA_INVALID** or drop.

Do-not-do (hard safety rails).
• No control-loop feedback inside the bridge (observation-only).
• No vendor offsets, flavor factors, or silent renorm on `a` or `m`.
• No averaging of alignments in native space; always use the `U/W` rapidity fuse.

Quick checklist (ready to implement).

1. **Declare:** `lens_id[]`, `anchors_ref`, `weight_basis`, `eps_a`, `eps_w`, `tick_ms`.
2. **Compute:** dials → `a`, then pools → `a_out` via `U/W`.
3. **Band:** apply hysteresis thresholds and holds; get `band`.
4. **Stamp:** emit envelope with `stamp`; chain to `prev_stamp`.
5. **Archive:** store `{inputs_hash, outputs_hash, time_utc}` for replay.

Pseudo-wire (one tick, end-to-end).

```
# dials (per channel)
a := tanh( c * ( (x/x_ref)^rho - 1 ) )
a := clamp(a, -1+eps_a, +1-eps_a)

# pool
U += w * atanh(a)
W += w
a_pool := (W == 0) ? 0 : tanh( U / max(W, eps_w) )

# band + envelope
band := ladder_hysteresis(a_pool, ...)
emit { value:=a_pool, dial_id:="a_pool", band, manifest_id:MID, tick_ms:T,
stamp:=chain(...) }
```

Outcome. With these bridges, SSMEQ becomes a **drop-in, vendor-portable posture layer**: numbers remain trustworthy (`m`), alignment remains reproducible (`a`), and decisions remain auditable (envelopes + stamps).

10.1 — Math Backbone (SSM): Collapse, Pooling, and Guarantees

Purpose. State the **core algebra** that makes SSMEQ portable across vendors and runtimes. **Classical values remain intact:** `phi((m,a)) = m`. **Alignment travels and fuses deterministically** via rapidity-space accumulation.

Core identities (must hold everywhere).

- **Collapse parity:** `phi((m,a)) = m`. Alignment never modifies classical bytes.
- **Clamp safety:** `a_c := clamp(a, -1 + eps_a, +1 - eps_a)` ensures finite rapidities.
- **Rapidity map:** `u := atanh(a_c)` and its inverse `a := tanh(u)` are **strictly monotone**.
- **Weighted fuse (U/W kernel):**
  ```
  U := sum_i( w_i * u_i )
  W := sum_i( w_i )
  a_pool := (W == 0) ? 0 : tanh( U / max(W, eps_w) )
  ```

Why U/W is the backbone.

- **Associativity (streaming-ready):** splitting or chunking the same tick does not change `a_pool`.
- **Commutativity (order-invariant):** reordering members at a tick leaves `a_pool` unchanged.
- **Monotonicity:** if each `a_i` increases (other knobs fixed), `a_pool` does not decrease.
- **Zero-class neutrality:** if all `a_i = 0`, then `u_i = 0` and `a_pool = 0`.
- **Saturation containment:** if any `|a_i| -> 1`, clamp keeps `u_i` finite; `a_pool` remains well-defined.

Per-tick fusion vs streaming (equivalence).

- **Batch form (per tick):**
  `a_pool := tanh( (sum_i w_i*atanh(a_i_c)) / max(sum_i w_i, eps_w) )`
- **Streaming form (same result):**
  increment `U += w*atanh(a_c)`, `W += w` on arrivals; on tick close, compute `a_pool := tanh(U/max(W, eps_w))`, then reset.

Magnitude roll-up (value lane).

- When a classical summary is needed, compute **only on `m`**:
  `M := sum_i( m_i )` and **report** `(M, a_pool)` with `phi((M, a_pool)) = M`.
- Do **not** apply alignment to `m`. Alignment is **metadata for posture**, not a scale factor.

Equality and comparison (optional SSM operators).

- **Equality score:** `s_eq(x,y) := 1 - tanh( c_eq * |x - y| )` in `[0,1)`; higher means closer.
- **Greater-than score:** `s_gt(x,y) := tanh( c_gt * (x - y) )` in `(-1,1)`; sign indicates ordering.
- These scores are **advisory**; they never alter `m`. Route using bands on these scores if needed.

Numerics (recommended defaults).

- `eps_a := 1e-6` to protect `atanh` near `±1`.
- `eps_w := 1e-9` to guard division when `W` is tiny.
- Round **display only**; keep full precision internally.

Copy-ready pseudo-code.

```
# Input: list of (m_i, a_i, w_i) for tick T
U := 0 ; W := 0
for each i:
  a_c := clamp(a_i, -1+eps_a, +1-eps_a)
  U += w_i * atanh(a_c)
  W += w_i
a_pool := (W == 0) ? 0 : tanh( U / max(W, eps_w) )

# Optional magnitude roll-up
M := sum_i(m_i)  # pure classical
emit (M, a_pool)  # phi((M, a_pool)) = M
```

Commissioning checks (must-pass).

1. **Order-invariance:** permute members at a tick; `a_pool` identical.
2. **Batch==Streaming:** chunk arrivals arbitrarily; same `a_pool`.
3. **Zero-class:** all `a_i = 0` → `a_pool = 0`.
4. **Clamp safety:** inject `a_i = ±0.999999`; no NaN/Inf, finite `a_pool`.
5. **Value-lane independence:** alter `a_i` while holding `m_i` fixed; `M` remains `sum_i(m_i)` exactly.

Operator crib.

- Always display `a_pool` with `N_valid/N_total` and **effective W**.
- For trust, provide a **"show U/W"** toggle that reveals the interim sums for audit.

10.2 — Operator Canon (SSMS) — Lift Rules (Normative, Copy-Ready)

Purpose. Define a **portable, text-only operator set** that lifts classical math to symbolic pairs while preserving **collapse parity**. **Classical values remain intact:** `phi((m,a)) = m`. Alignments are **zero-centric**, **bounded in (-1,+1)**, and **monotone with badness → positive**.

Core value.
A symbolic quantity is a pair `(m, a)` with:

- `m` = classical magnitude (any real in engineering units),
- `a` = alignment in `(-1,+1)`, clamped as `a_c := clamp(a, -1+eps_a, +1-eps_a)`.

Identities and invariants (must).

- **Collapse parity:** `phi((m,a)) = m`.
- **Neutral alignment:** `(m, 0)` behaves classically in sums/products (alignment still pooled).
- **Clamps:** every operator clamps alignments to `(-1+eps_a, +1-eps_a)` before rapidity.

A) Accumulators (sum / average) — `s_sum`, `s_avg`

Definition (sum of N items).

```
# inputs: (m_i, a_i) with optional weights w_i > 0
M := sum_i m_i
U := sum_i w_i * atanh( clamp(a_i, -1+eps_a, +1-eps_a) )
W := sum_i w_i
A := (W == 0) ? 0 : tanh( U / max(W, eps_w) )
return ( M, A )
```

- **Associative / commutative:** guaranteed via the `U/W` rapidity fuse.
- **Weights:** choose policy once (`uniform`, `scale^gamma`, or `declared`) and **freeze** for the run.
- **Average:** `s_avg` is `s_sum` followed by `M := M / N` (alignment `A` unchanged).

Properties (must hold).

- If every `a_i` increases (others fixed), `A` must **not** decrease (monotone).
- If all `a_i = 0`, then `A = 0`.
- **Streaming equivalence:** batch == chunked == per-arrival with tick close.

B) Product / ratio — `s_mul`, `s_div`

Product.

```
# (m1,a1) ⊗ (m2,a2)
m := m1 * m2
A := tanh( (atanh(a1_c) + atanh(a2_c)) / 2 )
return ( m, A )
```

- Intuition: **evidence averaging** for alignment under multiplicative composition.
- For N factors: average their rapidities (equal or declared weights).

Division (choose one policy and publish).

1. **strict:** if `|m2| < eps_div` → **ERROR_DIV_ZERO** (no output).
2. **soft-guarded:** use `m := m1 / max(|m2|, eps_div) * sign(m2)`; add a **division-risk dial**
   `a_div := tanh( c_div * ( eps_div / max(|m2|, eps_div) - 1 ) )`, then
   `A := tanh( (atanh(a1_c) - atanh(a2_c) + atanh(a_div)) / 3 )`.
3. **meadow-like (engineering fallback):** if `|m2| < eps_div` set `m := 0`, and set `A := +tanh( c_div * 1 )` to flag maximum risk.
   Publish exactly one of the three; default is **strict**.

C) Min / max — `s_min`, `s_max` (selector semantics)

```
# s_min
if m1 <= m2: return ( m1, a1_c )
else:        return ( m2, a2_c )

# s_max analogous
```

- Alignment **follows the selected branch** (no fusion).
- Optional: add a **tie–blend** when `|m1 - m2| <= delta_minmax`:
  `A := tanh( (atanh(a1_c) + atanh(a2_c)) / 2 )`.

D) Map / unit transforms — `s_map(f, ·)`

For monotone `f` (e.g., scaling, unit change, affine):

```
s_map(f, (m,a)) := ( f(m), a )
```

- If `f` is **non-monotone**, you **must not** carry alignment unchanged; either refuse or specify a safe recomputation of `a`.

E) Comparators — `s_gt`, `s_ge`, `s_lt`, `s_eq` (decision + evidence)

Return a **crisp decision** and an **evidence alignment** `a_dec in (-1,+1)`.

Normalized contrast.

```
r := (m1 - m2) / max( max(|m1|,|m2|), ref_cmp, eps_cmp )
a_dec := tanh( c_cmp * r )
```

- `s_gt` decides **TRUE** if `a_dec >= th_enter_gt` (e.g., `+0.50`) held for `t_hold_gt`.
- `s_lt` uses `-r` or swap operands.
- `s_eq` uses `r_eq := |m1 - m2| / max(ref_eq, eps_cmp)` and `a_dec := -tanh( c_eq * ( r_eq - 1 ) )` so **closer → more negative**, **within ref → around 0**.
- Publish `ref_cmp`, `ref_eq`, `c_*`, `th_enter/clear`, `t_hold` in the manifest.

Envelope (copy-ready).

```
{ decision := TRUE|FALSE, evidence := a_dec, op := "s_gt",
  left_id, right_id, tick_ms, manifest_id, stamp }
```

F) Logical lifts — `s_and`, `s_or`, `s_not` (on decisions with evidence)

Let decisions be `(bool, a)` with `a in (-1,+1)` as **confidence/evidence**.

```
s_and:  bool := b1 && b2 ; a := tanh( (atanh(a1_c) + atanh(a2_c)) / 2 )
s_or:   bool := b1 || b2 ; a := tanh( (atanh(a1_c) + atanh(a2_c)) / 2 )
s_not:  bool := !b1      ; a := a1_c
```

- For `k` inputs, use rapidity average with declared weights.

G) Bands and gating — `s_band`, `s_gate`

- `s_band(a)` maps alignment to `{A+,A,B,C,D}` using declared thresholds and holds (Section 8).
- `s_gate(a, g_t)` applies **decision-time** damping only: `a_env := g_t * a`, `g_t in [0,1]`.
  Never modify classical `m` with gates.

H) Determinism & commissioning (must-pass)

1. **Associativity tests:** `(x ⊕ y) ⊕ z == x ⊕ (y ⊕ z)` for `s_sum`, `s_mul` within `1e-6` alignment tolerance.
2. **Streaming equivalence:** batch == per-arrival for `s_sum`, `s_mul`.
3. **Division policy lock:** mid-run policy change **rejected**.

4. **Comparator sanity:** if `m1 = m2`, `s_eq` yields `a_dec ≤ 0` near zero; `s_gt` and `s_lt` remain **FALSE** until held thresholds are met.
5. **Clamp safety:** no NaN/Inf for any op; all intermediate `atanh()` inputs must be clamped.

Manifest keys (publish).

```
ops := { sum: basis, gamma|weights, mul: weight_policy, div_policy:
strict|soft|meadow, eps_a, eps_w, eps_div, ref_cmp, ref_eq, c_cmp, c_eq,
th_enter, th_clear, t_hold, gate: g_t_policy }
```

Operator crib (copy-ready).

- **Use `U/W` always** for pooling alignments.
- **Never average `a` linearly.** Use rapidities: `atanh(a)` → average → `tanh(·)`.
- **Keep `m` classical.** Only alignments carry posture; bytes remain byte-for-byte valid.

10.3 — Data Shape (SSMDE): Truth That Travels (Normative, Copy-Ready)

Purpose.
Define a portable envelope so **electrical truth, alignment, policy, and audit** can move across systems **without touching original bytes**. Classical values remain intact: `phi((m,a)) = m`.

Principle.
Send the smallest sufficient shape that proves:

```
interpretation := f(raw_bytes, lens, anchors, clocks, policy)
evidence := stamp(chain_prev, sha256(payload), time_utc)
```

Consumers can verify interpretation **without vendor secrets**.

Envelope tiers (choose one, publish in manifest).

S1 — Dial snapshot (minimal).

```
{
  "dial_id": "a_THDV",          // which dial/lens
  "value": 0.606,               // alignment a in (-1,+1)
  "tick_ms": 1731225601000,     // epoch ms of compute tick
  "manifest_id": "SSMEQ.PQ.v1", // policy/lens bundle id
  "status": "OK",               // or DATA_INVALID, DATA_ABSENT, ...
  "stamp": {
    "prev": "hex-64",           // previous stamp or "0"*64
```

```
    "hash": "hex-64",           // sha256(payload_bytes)
    "time_utc": "2025-11-10T09:00:01Z"
  }
}
```

S2 — Pooled + routing (recommended).

```
{
  "dial_id": "a_THDV_pool",
  "pool_id": "feeder_F01",
  "value": 0.606,
  "band": "C",
  "action": "ALERT",
  "owner": "PQ_team",
  "sla_ms": 900000,
  "status": "OK",
  "tick_ms": 1731225601000,
  "manifest_id": "SSMEQ.PQ.v1",
  "stamp": { "prev": "...", "hash": "...", "time_utc": "..." }
}
```

S3 — With classical magnitude (optional).

```
{
  "dial_id": "a_THDV_pool",
  "value": 0.606,
  "M": 1240.5,                  // classical magnitude roll-up (if
reported)
  "collapse": "phi((M,a)) = M", // assertion for readers
  "status": "OK",
  "tick_ms": 1731225601000,
  "manifest_id": "SSMEQ.PQ.v1",
  "stamp": { "prev": "...", "hash": "...", "time_utc": "..." }
}
```

Hashing and stamp chain (must).

```
payload_bytes := canonical_json(envelope_without_stamp)
hash          := sha256(payload_bytes)
stamp_value   := sha256(prev || hash || time_utc_iso8601)
```

- First envelope uses `prev := "0"*64`.
- **Rule:** never re-hash with different field order; always use **canonical JSON** (sorted keys, UTF-8, no trailing zeros).

Determinism (must).

- **One tick, one envelope per `(dial_id, scope)`** (scope = device, cabinet, pool, etc.).
- **No retroactive edits.** Corrections emit a new envelope with `action := "CORRECT"` and a link to the prior stamp (`prev_correction` or similar), never editing history.
- **Idempotent replay.** Consumers may reapply the same envelope; state must not diverge if the envelope is processed twice.

Validity flags (consistent with SSMEQ/SSMDE).

Add a **status** field in:
`{"OK","DATA_INVALID","DATA_ABSENT","QUORUM_LOST","DATA_STALE","HELD_VALUE"}`

- When `status != "OK"`, `value` may be omitted or set to `0.0` per policy; the **envelope is still stamped**.
- Pooling logic must follow Section 9 validity rules; envelope reflects the outcome.

Privacy & minimalism (recommended).

- **Do not include** raw waveforms or vendor registers.
- Send only: `value`, optional `M`, `band`/`action`, and references (`dial_id`, `manifest_id`, `pool_id`).
- If classical `M` is included, **publish its unit in the manifest**, not inside the envelope.

Transport mappings (portable).

- **HTTP(S).** `POST` line-delimited JSON (`.jsonl`) to a declared endpoint; return `201` with stored stamp.
- **MQ / Stream.** Topic per site or per pool; key by `dial_id` or `pool_id`.
- **Files.** Rotate `.jsonl` by time or size; include a footer `file_hash := sha256(all_lines)` for archive integrity.

Copy-ready emitter (pseudo-code).

```
function emit_envelope(dial_id, scope, a, band?, action?, status?):

  env := {
    "dial_id": dial_id,
    "value": round(a, 6),
    "tick_ms": T,
    "manifest_id": MID,
    "status": status or "OK"
  }

  if scope.pool_id: env["pool_id"] := scope.pool_id
  if band:          env["band"]    := band
  if action:        env["action"]  := action

  payload := canonical_json(env)              # sorted keys, utf-8
  h       := sha256(payload)
  stamp_v := sha256(prev || h || time_utc_iso8601())

  env["stamp"] := {
```

```
    "prev": prev,
    "hash": h,
    "time_utc": time_utc_iso8601()
  }

  prev := stamp_v
  write_line(env)                                  # jsonl or stream
```

Copy-ready verifier (pseudo-code).

```
for each envelope e in order_received:

  payload := canonical_json(e_without_stamp)
  h       := sha256(payload)

  assert h == e["stamp"]["hash"]
  assert e["tick_ms"] % tick_ms == 0
  assert known_manifest(e["manifest_id"])

  chain_v := sha256(prev || h || e["stamp"]["time_utc"])
  assert chain_ok(prev, h, e["stamp"]["time_utc"])   # per site policy

  prev := chain_v
```

Manifest additions (keys).

```
envelope_tier      in {S1, S2, S3}
tick_ms
canonicalization   := "json-sorted-utf8"
route_endpoint     := "<http(s) or mq topic>"
rotate_bytes       := <int>
rotate_minutes     := <int>
status_policy      := "<OK|INVALID|ABSENT mapping rules>"
units_M            := "<unit for M if S3>"
privacy_notes      := "<no raw waveforms; only dials and refs>"
```

Operator guidance.

- Show **dial label**, `value`, `band`, **time-in-band**, `action`/`SLA`, and a **"View stamps"** drawer listing last 20 `hash` values.
- Provide a **one-click `.jsonl` download** of envelopes for audit.

Why this works.

- **Truth** = a deterministic interpretation expressed as **alignment** under a published manifest.
- **That travels** = a minimal, canonical data shape plus a **chained stamp** any party can verify.
- **No vendor lock** = references by `manifest_id` and `dial_id`, not by proprietary register maps or formats.

10.4 — Thermal Crossover (SSMT): Cabinet Survivability & Electrical Risk (Normative, Copy-Ready)

Purpose. Fuse **temperature risk** with **electrical dials** to reflect real survivability. **Temperature does not overwrite electrical data**; it **gates or augments decisions** so actions respect thermal reality.

Inputs.
• Temperature channel(s) per cabinet: `T_in`, `T_out`, optional `T_hotspot`.
• Declared safe reference: `T_ref_safe` (baseline no-risk), and **thermal margin**: `dT_ref` for alignment.
• Electrical dial(s) already computed: e.g., `a_THDV`, `a_unbV`, `a_PF`, pooled `a_pool`.

Thermal dial (zero at safe reference).
Let `dT := max( 0, T_in - T_ref_safe ).`
`a_T := tanh( c_T * ( ( dT / max(dT_ref, eps_T) )^rho_T - 1 ) )`
Knobs: `c_T > 0`, `rho_T >= 1`. **Clamp:** `a_T_c := clamp(a_T, -1+eps_a, +1-eps_a).`

Crossover strategies (pick one; publish in manifest).

1. **Decision gate (recommended, safe-by-default).**
   Use `g_T := min( 1, 1 + k_T * a_T_c )` with `k_T in [0,1]`. Gate **decisions only**:
   `a_decide := g_T * a_elec` where `a_elec` is any electrical dial (or pooled).
   • If hot (`a_T_c > 0`), `g_T >= 1` nudges toward faster action.
   • If cool (`a_T_c <= 0`), `g_T <= 1` reduces urgency slightly but never hides risk if `k_T` is modest.
   Never modify classical magnitudes; only the decision path uses `a_decide`.
2. **Composite posture (thermal-aware stress).**
   Fuse selected dials `{a_THDV_pool, a_unbV_pool, a_PF_pool, a_T}` via rapidity accumulator:
   ```
   U := sum_j w_j * atanh( clamp(a_j, -1+eps_a, +1-eps_a) )
   W := sum_j w_j
   a_stress := tanh( U / max(W, eps_w) )
   ```
   Publish per-dial weights `w_THDV, w_unbV, w_PF, w_T`. Keep a separate **pure-electrical** posture for audit.

Band coupling (examples).

- **Thermal gate + electrical C:** if `a_THDV_pool >= +0.60` for `2 min` and `a_T >= +0.35`, escalate to **D** immediately (short-circuit hold).
- **Cool-down clear:** allow **clear** only if `a_T <= +0.20` **and** electrical `a_x <= th_clear_x` for the hold time.
- **Service window bias:** if `a_T >= +0.50` for `≥ 30 min`, automatically upgrade any active **B** to **C** (prevent silent heat damage).

Validity & placement (must declare).
• Sensor validity: `T_min_valid`, `T_max_valid`, `max_gap_ms`. On breach, label **DATA_INVALID** for thermal; do **not** fabricate `a_T`.
• Sensor choice: if both `T_in` and `T_hotspot` exist, publish `T_source in {in, hotspot, max(in,hotspot)}`.
• Tick: share `tick_ms` with electrical dials or publish a resampling rule (`last-in-window`).

Pseudo-code (copy-ready).

```
# 1) Thermal dial
dT := max(0, T_in - T_ref_safe)
aT_raw := tanh( c_T * ( ( dT / max(dT_ref, eps_T) )^rho_T - 1 ) )
a_T := clamp(aT_raw, -1+eps_a, +1-eps_a)

# 2) Decision gate over any electrical dial a_elec
g_T := min( 1, 1 + k_T * a_T )
a_decide := g_T * a_elec

# 3) Ladder
band_new := ladder_hysteresis(a_decide, band_old, th_enter[], th_clear[],
t_hold[])

# 4) Optional thermal short-circuit to D
if a_T >= th_T_hot and a_elec >= th_elec_high for t_hot_ms:
    band_new := D

# 5) Emit stamped envelope (unchanged classical values)
emit_dispatch({
  value := a_elec,       # original electrical alignment for audit
  band := band_new,
  dial_id := id(a_elec), # e.g., "a_THDV_pool"
  aux := { a_T := a_T, g_T := g_T },
  action := route(...), sla_ms := SLA(...), owner := OWNER(...),
  manifest_id := MID, tick_ms := T, stamp := chain_stamp(...)
})
```

Commissioning checks (must-pass).

1. **No stealth masking:** with `k_T <= 0.5`, an electrical dial that qualifies for **D** must remain ≥ **C** even at low `a_T`.
2. **Thermal short-circuit:** inject `a_T >= +0.75` while `a_elec >= +0.55`; confirm **D** escalation per policy.
3. **Cooling hysteresis:** once cleared, require `a_T <= +0.20` for the full **clear hold** or longer.
4. **Audit separability:** verify reports include **both** `a_elec` and `a_T` and the final `band`.
5. **Window sanity:** if `dT = 0` for an hour with stable electrics, bands must **not** flap.

Manifest keys.
`T_source`, `T_ref_safe`, `dT_ref`, `c_T`, `rho_T`, `eps_T`, `tick_ms`, `k_T`, `th_T_hot`, `t_hot_ms`, `strategy in {gate, composite}`, `weights` (if composite), `validity_bounds`.

Operator guidance.
Always display the **thermal dial** beside the **electrical dial** that triggered action: `a_T | a_THDV_pool | g_T`. Provide a **cool-down timer** ("time below `a_T <= +0.20`") and a **next-clear target**.

10.5 — Electro-Thermal-Chemical Crossover (SSMT + SSM-Chem, Normative, Copy-Ready)

Purpose. Define a portable crossover dial that combines:

- **Electrical posture** (SSMEQ: e_V, e_I, e_P, a_pf, a_THD)
- **Thermal stress** (SSMT: hotspot and gradient dials)
- **Chemical health** (SSM-Chem: battery / cell condition dial)

into a single bounded alignment `a_xover ∈ (-1,+1)` for assets such as inverters, battery strings, or power-electronics cabinets.

This dial does **not** replace domain-specific views.
It provides a **unified "how stressed is this asset?"** signal while preserving:

- Collapse parity: `phi((m,a)) = m` for all classical magnitudes.
- Kernel invariants: clamp → atanh → U/W → tanh.
- Order invariance and streaming equivalence.

10.5.1 — Inputs (Layered Dials)

Per asset (e.g., one inverter leg or battery string), on each tick:

Electrical dials (SSMEQ):

- `a_THDV` — voltage harmonic distortion alignment
- `a_THDI` — current harmonic distortion alignment
- `a_unbV`, `a_unbI` — voltage / current unbalance (if 3-phase)
- `a_pf` — power factor alignment (from pf, pf_ref, pf_min)
- `a_I` — current magnitude stress
- Optional: `r_P` residual (power consistency), used only as a diagnostic flag.

Thermal dials (SSMT):

- `a_T_hotspot` — alignment of hotspot temperature vs thermal limit
- `a_T_grad` — alignment of temperature gradient vs safe gradient

(Each is already an `a ∈ (-1,+1)` from SSMT lens definitions.)

Chemical dials (SSM-Chem):

- `a_chem` — battery / cell health alignment
  - Derived from SoH, SoC drift, impedance rise, or charge-throughput ratios.
  - Bounded in `(-1,+1)` using an SSM-Chem lens (e.g., deviation from nominal SoH).

All of the above are **alignment lanes only**; classical magnitudes `m` (V, I, P, SoH, T, etc.) remain separate and unmodified.

10.5.2 — Layered Fusion Structure

We define three intermediate layer dials:

1. **Electrical stress layer**

   Combine electrical dials into `a_elec`:

   ```
   # weights declared in manifest
   w_THDV, w_THDI, w_unbV, w_unbI, w_pf, w_I

   U_e := 0 ; W_e := 0
   for each electrical dial a_j in {a_THDV, a_THDI, a_unbV, a_unbI,
   a_pf, a_I}:
       if valid(a_j):
           a_c := clamp(a_j, -1+eps_a, +1-eps_a)
           U_e += w_j * atanh(a_c)
           W_e += w_j

   a_elec := (W_e == 0) ? 0 : tanh( U_e / max(W_e, eps_w) )
   ```

2. **Thermal stress layer**

   Combine thermal dials into `a_therm`:

   ```
   U_T := 0 ; W_T := 0
   for each thermal dial a_k in {a_T_hotspot, a_T_grad}:
       if valid(a_k):
           a_c := clamp(a_k, -1+eps_a, +1-eps_a)
           U_T += w_k * atanh(a_c)
           W_T += w_k

   a_therm := (W_T == 0) ? 0 : tanh( U_T / max(W_T, eps_w) )
   ```

3. **Chemical layer**

   Chemical health is typically already a single dial:

   ```
   a_chem_c := clamp(a_chem, -1+eps_a, +1-eps_a)
   # optional: blend with more SSM-Chem dials if defined
   ```

10.5.3 — Final Crossover Dial

The **electro-thermal-chemical crossover** dial `a_xover` fuses the three layers:

```
# layer weights declared in manifest
w_elec, w_therm, w_chem

dials_layer := [
  (a_elec, w_elec),
  (a_therm, w_therm),
  (a_chem_c, w_chem)
]

U_x := 0 ; W_x := 0
for (a_L, w_L) in dials_layer:
    if valid(a_L):                      # status == OK
        a_c := clamp(a_L, -1+eps_a, +1-eps_a)
        U_x += w_L * atanh(a_c)
        W_x += w_L

a_xover := (W_x == 0) ? 0 : tanh( U_x / max(W_x, eps_w) )
```

Bands → actions for `a_xover` follow the same A⁺…D logic as other dials, but with **stronger holds** (longer `t_hold`) because it reflects structural asset health, not instantaneous noise.

10.5.4 — Worked ASCII Example (Illustrative)

Assume:

- Electrical layer at tick `T`:
  - `a_THDV = -0.40` (moderate distortion)
  - `a_unbV = -0.20` (mild unbalance)
  - `a_pf = +0.30` (acceptable PF)
  - `a_I = -0.10` (slightly high current)
- Thermal layer:
  - `a_T_hotspot = -0.35` (running warm)
  - `a_T_grad = -0.10` (small gradient; safe)
- Chemical layer:
  - `a_chem = +0.50` (battery health above threshold, good margin)

Step 1 — Electrical fusion (simplified, all w=1):

```
electrical dials: [-0.40, -0.20, +0.30, -0.10]

Approximate atanh:
atanh(-0.40) ≈ -0.424
atanh(-0.20) ≈ -0.203
atanh(+0.30) ≈ +0.309
atanh(-0.10) ≈ -0.101

U_e = sum ≈ -0.424 - 0.203 + 0.309 - 0.101 ≈ -0.419
W_e = 4
```

```
a_elec = tanh( U_e / W_e ) = tanh(-0.10475) ≈ -0.104
```

Step 2 — Thermal fusion (w=1):

```
thermal dials: [-0.35, -0.10]

atanh(-0.35) ≈ -0.365
atanh(-0.10) ≈ -0.101

U_T = -0.365 - 0.101 = -0.466
W_T = 2

a_therm = tanh( -0.233 ) ≈ -0.229
```

Step 3 — Chemical clamp:

```
a_chem = +0.50 → atanh(+0.50) ≈ +0.549
```

Step 4 — Crossover fusion (e.g., w_elec=1.0, w_therm=1.5, w_chem=1.0):

```
atanh(a_elec)  ≈ atanh(-0.104) ≈ -0.105
atanh(a_therm) ≈ atanh(-0.229) ≈ -0.233
atanh(a_chem)  ≈ +0.549

U_x = 1.0 * (-0.105)
    + 1.5 * (-0.233)
    + 1.0 * (+0.549)
    ≈ -0.105 - 0.350 + 0.549
    ≈ +0.094

W_x = 1.0 + 1.5 + 1.0 = 3.5

a_xover = tanh( 0.094 / 3.5 ) = tanh(0.0269) ≈ +0.027
```

Interpretation.

- Electrical and thermal signals are mildly negative (stress, warmth),
- Chemical health is solidly positive,
- Layer weights favor thermal somewhat,
- Net `a_xover ≈ +0.03` → slightly positive, **near-neutral posture**.

Band table might classify this as **A^- / "Normal drift, monitor"** — fine today, watch trend over weeks.

10.5.5 — Normative Rules (Must Hold)

1. **No magnitude feedback.**
   - `e_V`, `e_I`, `e_P`, `SoH`, and raw temperatures are never altered by `a_xover`.
   - All control decisions may *reference* `a_xover`, but classical data streams stay byte-for-byte.

2. **Layer transparency.**
    - Operator and Investigator views must allow drilling down:
      `a_xover → (a_elec, a_therm, a_chem) → underlying dials.`
3. **Manifest declaration.**
    - Layer weights (`w_elec`, `w_therm`, `w_chem`) and per-dial weights must be declared in the manifest, not hard-coded.
4. **Reproducibility.**
    - The crossover dial must be reconstructible from archived CSVs (`golden.csv`) and the same manifest, passing the Repro Pack harness.

10.5.6 — Manifest Keys (Copy-Ready)

Under the relevant asset or pool:

```
"crossover": {
  "dial_id": "a_xover",
  "layers": {
    "electrical": {
      "dials": ["a_THDV","a_THDI","a_unbV","a_unbI","a_pf","a_I"],
      "weights":
{"a_THDV":1.0,"a_THDI":1.0,"a_unbV":0.8,"a_unbI":0.8,"a_pf":1.0,"a_I":1.0}
    },
    "thermal": {
      "dials": ["a_T_hotspot","a_T_grad"],
      "weights": {"a_T_hotspot":1.2,"a_T_grad":0.8}
    },
    "chemical": {
      "dials": ["a_chem"],
      "weights": {"a_chem":1.0}
    }
  },
  "layer_weights": {"electrical":1.0,"thermal":1.5,"chemical":1.0},
  "eps_a": 1e-6,
  "eps_w": 1e-9,
  "bands": "bands_xover.json"
}
```

Outcome.
The Electro-Thermal-Chemical Crossover dial `a_xover` gives SSMEQ a **single, mathematically honest posture signal** for complex assets, while keeping each underlying discipline (electrical, thermal, chemical) fully visible, reproducible, and ethically intact.

11.0 — UI & Dashboards (General Introduction)

The **User Interface and Dashboard layer** in the Shunyaya Symbolic Mathematical Electrical Quantities (SSMEQ) framework translates the symbolic kernel — founded on rapidity accumulation and bounded alignment — into visual, verifiable instruments for real-world operators, investigators, and commissioning teams. This layer transforms the invisible mathematics of order-invariant pooling and symbolic governance into a live, auditable experience that is understandable at a glance and traceable in detail.

The goal is simple yet revolutionary:
to let every electrical or derived symbolic quantity not only be *measured*, but also *felt* — through stability, drift, and confidence indicators that are deterministic, reproducible, and universally comparable.

While the underlying calculations remain governed by the canonical **U/W kernel** (`U += w * atanh(a_c) ; W += w ; a_out := tanh(U / max(W, eps_w))`), the dashboard layer presents these results as intuitive dials, badges, and records. Every element draws from the same truth source — the manifest and the stamp chain — ensuring that every display reflects an auditable, immutable rulebook.

Key Design Intent

- **Zero drift between math and display.** Each visual element corresponds directly to a bounded symbolic quantity (`a ∈ (-1,+1)`), ensuring that the UI never introduces subjective scaling or bias.
- **Reproducibility by design.** Every chart, dial, and badge can be regenerated from raw inputs and manifests without hidden calculations or vendor bias.
- **Human readability.** Operators see safe, color-coded bands and stability tags (A^+/A/A^-/C/D), while auditors can reconstruct the exact U/W state from the same data.
- **Manifest-first ethics.** All visual decisions (thresholds, smoothing, aggregation cadence) are declared, never implicit.

This section provides ready-to-use blueprints for three major user interfaces—**Operator View**, **Investigator View**, and **Commissioning View**—followed by a **Minimal UI API schema** that allows these dashboards to be integrated with any control or data environment while remaining portable and ASCII-transparent.

Each subsection can be directly implemented as a copy-ready prototype or dashboard specification. Together, they form the practical and visual counterpart to the symbolic kernel, closing the loop from **formula → instrument → insight** while ensuring **collapse parity** (**`phi((m,a)) = m`**) at every step.

11.1 — Operator View (Core Dials and Field Dashboard)

The Operator View is the live cockpit of SSMEQ — where the abstract symbolic kernel becomes tangible.
Every dial, badge, and counter is bound to the same deterministic core:

```
U += w * atanh(a_c)
W += w
a_out := tanh( U / max(W, eps_w) )
```

This guarantees that what operators see on screen is **identical** to what the system computes internally: no approximations, no moving averages, no silent overrides.

Purpose and Design Philosophy

The Operator View lets field personnel, control-room engineers, and plant supervisors **see electrical health and drift at a glance**. It distills multiple symbolic quantities into a small set of unified, bounded signals that show not only magnitude, but **posture** — how close the system is to stress or instability.

Each dial expresses a symbolic alignment `a ∈ (-1, +1)` as both a number and a color-coded band. For most SSMEQ stress dials:

- `a ≈ 0` → calm / nominal,
- `a → +1` → high stress / limit approach.

(Some underlying dials may be signed; UI bands usually operate on `a` or `|a|` as declared in the manifest.)

Core Dials

1) `a_pool` — Aggregate Stability Dial

Represents the **pooled symbolic posture** of a feeder, cabinet, or site.

```
a_pool := tanh( U / max(W, eps_w) )
```

where `U` and `W` are the rapidity accumulators across selected electrical dials, as defined in the pooling kernel.

- Suppresses outliers by construction (rapidity + clamps).
- Provides a single “how stressed is this pool?” value.

---

2) `a_THDV` — Voltage Harmonic Distortion Dial

Indicates deviation from nominal voltage waveform quality.

```
a_THDV := tanh( c_THDV * ln( 1 + THD_V / THD_ref ) )
```

- Bounded, comparable, and unitless.
- Increases as harmonic distortion rises, independent of system rating.

---

3) `a_unbV` — Voltage Unbalance Dial

Tracks alignment and symmetry between phase voltages.

```
a_unbV := tanh( c_unbV * ln( 1 + Unbalance_V / UnbV_ref ) )
```

- Converts raw percent unbalance into a normalized stability dial.
- Higher `a_unbV` indicates larger unbalance stress.

---

4) `a_stress` — Electrical Stress Dial (Composite)

A **fused cabinet-level stress metric** combining multiple PQ dials (for example `a_THDV`, `a_unbV`, `a_pf`, `a_I`, optional `a_T`), using the same rapidity pooling kernel:

- Clamp each dial: `a_j_c := clamp(a_j, -1+eps_a, +1-eps_a)`
- Map to rapidity: `u_j := atanh(a_j_c)`
- Fuse: `U := sum_j( w_j * u_j ), W := sum_j( w_j )`
- Return: `a_stress := tanh( U / max(W, eps_w) )`

Drift of `a_stress` toward `+1` warns of **rising composite stress** that often precedes heat buildup, nuisance trips, or PQ complaints.

Band Classification

Each dial carries a band badge (for example `A+`, `A`, `B`, `C`, `D`) that maps continuous `a` into simple health states. A common mapping (tunable by manifest) is:

Band	Range (a)	Meaning
A+	`0.00 – 0.20`	Very calm / excellent quality
A	`0.20 – 0.35`	Healthy, well within limits
B	`0.35 – 0.50`	Watch zone / mild stress
C	`0.50 – 0.75`	Caution — approaching limits
D	`≥ 0.75`	Critical — high stress / anomaly

Bands are computed **directly from symbolic values** using the same hysteresis and memory rules declared in the manifest (enter/clear thresholds, hold times).

Time-in-Band and SLA Countdown

Every dial tracks how long it has stayed in its current band, plus the **next band threshold** and any relevant **SLA window**.

Example logic:

```
if band_current == band_previous:
    time_in_band_ms += tick_ms
else:
    time_in_band_ms  := 0

next_target      := threshold_next - a_current
SLA_countdown_ms := target_window_ms - time_in_band_ms
```

- `time_in_band_ms` is deterministic from `tick_ms` and band logic.
- `SLA_countdown_ms` uses manifest-declared `target_window_ms` per band.

Operators see not only **where** the dial is, but **how long** it has been there and **how close** it is to a policy boundary.

KPI Cards (Quick Reference Strip)

Below the main dials, a compact KPI strip provides context such as:

- `N_valid / N_total` — number of valid sensor samples contributing to pooling.
- `W_effective` — total effective symbolic weight in the pool (`W` in the kernel).
- `quorum_state` — whether minimum quorum for reliable pooling is met.

Example:

```
quorum_state := ( N_valid / max(N_total, 1) ) >= quorum_threshold
```

All KPIs are computed from manifest rules; there is no UI-side “guessing”.

Interpretation

- **Operators** use dials, colors, and time-in-band for instant situational awareness.
- **Analysts** use trends of `a_pool`, `a_THDV`, `a_unbV`, and `a_stress` to spot chronic drift and plan maintenance.
- **Automation systems** consume the same values via SSMDE envelopes, ensuring that what the UI shows and what downstream systems act on are **bit-for-bit consistent**.

This design keeps the Operator View **truth-preserving**: the field sees exactly what the symbolic kernel computes — no loss, no embellishment, and no vendor-specific reinterpretation.

11.2 — Investigator View (Audit, Provenance, and Traceability)

The **Investigator View** is the analytical twin of the Operator interface.
Where the Operator View shows **live posture and dials** for immediate decisions, the Investigator View exposes the **mechanics behind every number** — letting auditors, engineers, and analysts reconstruct exactly:

- why a value appeared,
- how it evolved over time, and
- whether it complied with the declared manifest and policy.

No black-box aggregations are permitted.

Purpose and Philosophy

The Investigator View is SSMEQ's **transparency and accountability layer**.

Every displayed dial, alert, or drift curve must be explainable using:

- manifest-linked configuration (lenses, anchors, thresholds),
- deterministic numerical rules, and
- stamped envelopes and chain continuity.

A qualified reviewer should be able to independently verify:

- **Contributors**: which inputs `a_i` and weights `w_i` produced a pooled result (`U`, `W`, `a_out`),
- **Governance**: which manifest, lens, and anchors governed that computation,
- **Lineage**: which stamps and hashes prove its timing and order.

This yields full **numerical, structural, and temporal traceability**.

Displayed Elements (Auditor Layer)

1) Rapidity Internals (U/W Diagnostics)

For each pooled quantity, the Investigator View exposes the internal rapidity state:

- `U` — cumulative rapidity (evidence),
- `W` — cumulative weight,
- `{atanh(a_i)}` — per-source rapidities contributing to the pool.

Auditors can verify the pooling invariant:

```
a_expected := tanh( U / max(W, eps_w) )
assert |a_observed - a_expected| < tol_a
```

If this holds, the symbolic layer is numerically consistent for that tick.

---

2) Lens and Anchor Provenance

For each underlying channel (e.g., `V_rms`, `I_rms`, `P`, `f`, `pf`, `THD_V`, `THD_I`, `Unbalance_V`, `T`), the Investigator View shows:

- `lens_id` (LOG / ASINH / HYBRID / etc.),
- **anchors** (e.g., `V_ref`, `I_ref`, `f_ref`, `P_scale`, `Q_scale`),
- key parameters (`c_*`, `rho`, `eps_a`, `eps_w`),
- the **manifest_id** that froze these values at measurement time.

All of this comes from the **immutable manifest**. It is **never editable via UI**, so investigations always refer to the exact rulebook in force when the data was produced.

---

3) Stamp Chain Display

The Investigator View lists the latest *N* (e.g., 20) stamps for a given dial/pool, each derived from:

```
stamp := sha256( prev_stamp || sha256(payload_bytes) || time_utc )
```

The UI shows for each envelope:

- `stamp.hash`,
- `stamp.prev`,
- `stamp.time_utc`,
- `tick_ms`, `dial_id`, and `manifest_id`.

Visual behavior:

- **Continuous chain** → green/OK.
- Any break or reordering → highlighted in red, with **gap length** and **missing time range** clearly shown.

---

4) Alignment History Timeline

Each symbolic dial can be expanded into a **time-series view** showing:

- raw alignments `a_i(t)` by member,
- corresponding rapidities `u_i(t) = atanh(a_i_c(t))`,
- pooled output `a_pool(t)`.

The x-axis is `time_utc`; the y-axis overlays `a_i` and `a_pool`.

Because the transformation is deterministic and recorded, auditors can:

- replay the numeric evolution,
- recompute `U(t)`, `W(t)`, and `a_pool(t)`,
- confirm that historic bands and actions match the recorded envelopes.

---

5) Lens Deviation Tracker (Residuals)

For advanced analysis, the Investigator View can show residuals such as the active-power identity residual:

```
r_P := e_P_log - ( e_V_log + e_I_log + e_pf )
```

Plots of `r_P` over time highlight:

- CT/PT ratio problems,
- phase swaps,
- pf computation issues,
- sensor saturation or clipping.

Large |r_P| regions become **visual fault markers** that guide detailed investigation.

Ethical and Operational Notes

- **No Silent Correction**
  - Residuals, outliers, and anomalies are **flagged**, never altered in the symbolic store.

    - Any “correction” is an additional envelope with `action := "CORRECT"` and a new stamp, not an edit.
- **Reconstructive Fidelity**
    - All Investigator views must be regenerable from:
        - archived CSV/JSON (dials, stamps), and
        - the manifest(s) in force.
- **Read-Only by Design**
    - Investigator mode is **view-only**.
    - It cannot change pools, weights, bands, or gates — it only inspects them.

Outcome and Benefits

The Investigator View upgrades electrical telemetry into a **forensic-grade system**.

Every value is effectively a signed statement:

“Here is what was measured, how it was derived, which manifest defined it, when it was stamped, and how it behaved over time.”

Post-event investigations become **deterministic replay exercises** instead of guesswork:

- re-run the pools,
- re-check the stamps,
- confirm policy alignment.

Results are simple, conclusive, and mathematically verifiable — exactly in line with SSMEQ’s symbolic, manifest-first philosophy.

11.3 — Commissioning View (Verification, Shuffle, and Certification)

The **Commissioning View** is the final checkpoint before symbolic data from SSMEQ becomes operational.

It is where teams:

- verify **determinism**,
- confirm **policy locks**, and
- **certify** that every dial, pool, and alignment value reproduces identically — regardless of sequence, batch, platform, or runtime.

If the Operator View emphasizes visibility and the Investigator View emphasizes traceability, the Commissioning View guarantees **trust**: that all symbolic paths are reproducible, auditable, and invariant across executions.

Purpose and Context

Commissioning ensures that the implementation of the symbolic kernel is faithful to the canonical model:

```
U += w * atanh(a_c)
W += w
a_out := tanh(U / max(W, eps_w))
```

Every electrical site or simulation environment must pass commissioning at least once before being declared **operational**.

It proves that:

1. The symbolic kernel is **stable under input shuffling and order changes**.
2. The manifest parameters (`lens_id`, `eps_a`, `eps_w`, `gamma`, bands, quorum) produce **deterministic results** across environments.
3. The entire pipeline — from raw inputs to envelopes — yields a reproducible **commissioning stamp**.

Core Verification Functions

1) Shuffle & Verify (Order-Invariance Test)

A simple but decisive validation step.

The Commissioning View provides a single control: **"Shuffle & Verify"**.
This executes multiple randomized permutations of the same dataset and compares results:

```
for i in range(10):
    shuffle(samples)
    compute a_pool_i := tanh(U / max(W, eps_w))

Δmax := max_i |a_pool_i - a_pool_ref|
pass := (Δmax < tol_invariance)
```

- `a_pool_ref` is the reference result (first run or canonical batch).
- `tol_invariance` is a declared numeric tolerance (e.g., 1e-6).

The UI reports:

- **PASS / WARN / FAIL**,
- the actual `Δmax`, and
- the configured `tol_invariance`.

This confirms that **batch, shuffled, and streaming paths all agree**.

---

2) Golden Vector Validation

During commissioning, a pre-approved **reference dataset** ("golden vector") is loaded.

For each symbolic quantity (`a_pool`, `a_THDV`, `a_unbV`, `a_stress`, etc.), the system recomputes alignments and compares **expected vs observed**:

```
diff_i := |a_observed_i - a_expected_i|
pass := all(diff_i < tol_vector)
```

The Commissioning View displays:

- `diff_max` (worst-case deviation),
- `tol_vector`, and
- a **confidence index** (e.g., % of checks within tighter bounds).

This ensures that the **implementation matches the published canonical test data**.

---

3) Commissioning Stamp Generation

Once **Shuffle & Verify** and **Golden Vector Validation** both pass, the system emits a **commissioning stamp**:

```
commissioning_stamp := sha256( inputs_hash || outputs_hash || time_utc )
```

Where:

- `inputs_hash` ≔ `sha256(all_input_files_and_manifests)`
- `outputs_hash` ≔ `sha256(all_expected_vs_observed_results)`

Example ASCII-only entry:

```
COMMISSIONING_STAMP|2025-11-
12T08:05:24Z|sha256_inputs=...|sha256_outputs=...|stamp=...
```

This single line is stored alongside:

- the **manifest** (`manifest_id`), and
- the **policy lock file** (frozen lenses, bands, weights, quorum).

Any future test with identical inputs and policies must reproduce the **same commissioning_stamp bit-for-bit**, or be considered non-conforming.

---

4) Quorum & Policy Locks Review

Before final commit, the Commissioning View presents a **policy snapshot**:

- Active `lens_id[]` and all anchors (`V_ref`, `I_ref`, `f_ref`, `P_scale`, `Q_scale`, etc.).
- Weight definitions (e.g., `weight_basis = "scale"` with `gamma`, or `"uniform"` / `"declared"`).
- Quorum policy (`N_valid / N_total ≥ quorum_threshold`).
- Core guard constants: `eps_a`, `eps_w`, `pf_min`, `eps_pos`, `eps_div`.
- Band definitions and hysteresis thresholds (`th_enter`, `th_clear`, `t_hold`, `t_refractory`).

These parameters are **locked** for the commissioned build.
Any post-deployment adjustment to these values requires:

1. A new commissioning run, and
2. A new **commissioning_stamp**.

Interface Summary (Operator Perspective)

The Commissioning View is designed to be clear, ASCII-friendly, and reproducible:

- **One-Click Verification**
  - "Shuffle & Verify" runs the order-invariance tests and shows `Δmax` vs `tol_invariance`.
- **Golden Vector Comparison**
  - Table of expected vs observed values, plus `diff_max` and pass/fail flags.
- **Commissioning Stamp**
  - Automatically generated, stored, and displayed as a single ASCII line that can be copied into emails, reports, or config files.
- **Status Log**
  - Each test logged with outcome: `PASS` (green), `WARN` (amber), `FAIL` (red), including brief reason codes.
- **Export Option**
  - Produces `commissioning_report.txt` summarizing:
    - all invariant tests,
    - key parameters (epsilons, bands, weights, quorum),
    - hashes (`inputs_hash`, `outputs_hash`, `commissioning_stamp`).

Why This Matters

The Commissioning View turns symbolic arithmetic from a **theoretical standard** into a **field-grade, verifiable reality**.

Every deployed site, dataset, or embedded device inherits the **same symbolic integrity**, because:

- pooling invariants are tested under shuffle,
- golden vectors are checked line-by-line, and
- a single commissioning stamp anchors the entire configuration.

For audits and long-term maintenance, this is transformative:

- A future reviewer simply replays the same inputs and manifest,
- recomputes the **same sequence of hashes and stamps**, and
- checks for **identity**, not interpretation.

No forensic debates, no ambiguous "almost matches" — only **mathematical equality** or a clear divergence.

11.4 — Minimal UI API (Portable ASCII Schema)

The **Minimal UI API** defines the simplest possible interface between the symbolic kernel and any user-facing system.
It is designed for **universality** — allowing any dashboard, control panel, or automation layer to fetch symbolic quantities, bands, and stamps without requiring custom code, vendor-specific drivers, or external dependencies.

This API exists purely as **ASCII schema**, consistent with the rest of the Shunyaya symbolic family. Every field, endpoint, and response can be read, written, and validated in plain text, ensuring that symbolic truth remains portable across domains, devices, and platforms.

Purpose and Philosophy
The Minimal UI API ensures that all visual and control systems connect directly to deterministic sources.
It replaces proprietary telemetry protocols with a single, verifiable structure that can be implemented in any language or environment.

Principles guiding this API:

- **Determinism:** All responses derive from the same symbolic kernel (`U/W`), never from approximations.
- **Transparency:** All outputs are human-readable ASCII; every field has a direct symbolic meaning.
- **Minimalism:** Only essential data are exposed — no hidden meta-fields, no ambiguity.
- **Interoperability:** Output format is manifest-consistent, identical across all systems that implement the standard.

Endpoint 1 — Dial Fetch

Retrieves the latest symbolic alignment value, its band classification, and the current chain stamp.

```
GET /dial/{id}?t={timestamp_ms}

Response:
{
  "value": a_current,
  "band": band_current,
  "tick_ms": tick_ms,
  "stamp": commissioning_stamp
}
```

- **a_current** — bounded symbolic alignment in (-1,+1).
- **band_current** — derived band label (A^+, A, A^-, C, D).
- **tick_ms** — time delta since last sample.
- **commissioning_stamp** — hash derived at last commissioning (`sha256(inputs_hash || outputs_hash || time_utc)`).

This endpoint is typically used by dashboards, alarms, and operators to read current posture instantly without parsing raw data streams.

Endpoint 2 — Pool State Fetch

Provides direct access to the internal U/W accumulators for deeper analytics or verification.

```
GET /pool/{id}/uw?t={timestamp_ms}

Response:
{
  "U": U_accumulated,
  "W": W_accumulated,
  "eps_a": eps_a,
  "eps_w": eps_w
}
```

- **U** — total rapidity-weighted confidence accumulator.
- **W** — cumulative weight of valid samples.
- **eps_a**, **eps_w** — declared small constants preventing division or saturation errors.

This endpoint serves investigators, analysts, or AI modules that validate kernel parity and detect drift in the accumulation process.

Endpoint 3 — Band Profile Fetch

Provides band configuration and symbolic gating thresholds for display or analytics.

```
GET /band/profile

Response:
{
  "bands": [
    {"name": "A⁺", "min": +0.90, "max": +1.00},
    {"name": "A",  "min": +0.60, "max": +0.90},
    {"name": "A⁻", "min": -0.60, "max": +0.60},
    {"name": "C",  "min": -0.90, "max": -0.60},
    {"name": "D",  "min": -1.00, "max": -0.90}
  ],
  "policy": {
    "hysteresis_ms": 5000,
    "hold_ms": 15000,
    "clear_threshold": 0.05
  }
}
```

This ensures consistent rendering of colors and alarms across different dashboard frameworks.

Endpoint 4 — Stamp Chain Fetch

Allows the retrieval of the last N symbolic stamps for continuity and audit.

```
GET /stamp/chain?n={count}

Response:
{
  "stamps": [
    "SSMCLOCK1|2025-11-12T09:30:45Z|...|sha256(file)|chain",
    "SSMCLOCK1|2025-11-12T09:35:45Z|...|sha256(file)|chain",
    ...
  ]
}
```

Every stamp follows the canonical format of the Shunyaya symbolic timekeeping system, ensuring perfect interoperability with all SSM-family domains (hardware, data, or network).

Usage Example (Compact Sequence)

```
GET /dial/a_pool?t=now
→ {"value": +0.812, "band": "A", "tick_ms": 1000, "stamp": "sha256(...)"}

GET /pool/a_pool/uw?t=now
→ {"U": 1.2598, "W": 1.5289, "eps_a": 1e-6, "eps_w": 1e-12}
```

A simple command-line client or browser-based dashboard can render full symbolic telemetry in less than a second, without parsing or external libraries.

Benefits

- **Universal Plug-In:** Any application can implement this using plain text over HTTP, WebSocket, or serial links.
- **Verification Ready:** Output values can be recomputed or cross-checked directly using formulas in this document.
- **Offline Resilience:** Stamps allow complete local verification even without network time or servers.
- **Scalability:** The same schema scales from a single sensor to multi-site distributed telemetry without change.

Closing Perspective
The Minimal UI API completes the user interface architecture of SSMEQ.
It provides the clean handshake between symbolic truth and practical usability — one that any operator, auditor, or automation system can rely upon without fear of drift, corruption, or hidden logic.
This ensures that symbolic mathematics remains not only correct in theory, but **alive, visible, and verifiable** in every live electrical system.

12.0 — Security, Versions, and Reproducibility (General Introduction)

The **Security and Reproducibility layer** ensures that SSMEQ deployments remain immutable, auditable, and self-verifiable — no matter who runs them or where they operate. While the UI and dashboards translate symbolic truth into visible form, this section guarantees that every number, every band, and every alignment value can be **traced back to its exact manifest**, verified against its **stamp chain**, and **reproduced deterministically** using publicly testable methods.

This layer transforms symbolic computation from a theoretical model into a **certifiable system of record** — one that behaves identically across organizations, tools, and devices.

Purpose and Rationale
The symbolic approach used in SSMEQ treats every output as a function of three immutable elements:

1. **Manifest** — the frozen rulebook (lenses, anchors, tolerances).
2. **Input dataset** — raw, timestamped, checksum-verified readings.
3. **Kernel logic** — deterministic equations (`U += w * atanh(a_c); W += w; a_out := tanh(U / max(W, eps_w))`).

Because these three are finite and verifiable, reproducibility can be guaranteed indefinitely — without hidden model parameters or re-training cycles.

Each subsection of this section ensures integrity across these dimensions:

- **Policy Locks:** prevent unintentional or malicious drift from approved configurations.
- **Stamp Chains:** encode every transformation in append-only, cryptographically linked form.
- **Repro Packs:** provide the lightweight, field-deployable toolset for independent verification.
- **Data Validity Posture:** defines explicit handling rules for absent, invalid, or partial data.

Together, these create a trust layer that is **offline-verifiable, forward-compatible, and immune to obfuscation**.

Why This Matters
Most electrical and telemetry systems today rely on opaque logs, unverifiable averages, or proprietary aggregation software.
SSMEQ replaces this fragility with a transparent chain of trust — every operation is stamped, every policy is declared, and every outcome is regenerable from first principles.

This ensures that:

- **No silent drift** occurs between analysis and display.
- **Cross-site audits** are conclusive, not interpretive.
- **Peer verification** requires no special tooling — just standard hashing and ASCII arithmetic.
- **Historical reproducibility** extends across years, as long as manifests and stamps remain intact.

In short, this section defines the **mathematical ethics** of SSMEQ:
no hidden layers, no arbitrary interpretations — just symbolic truth, traceable from its origin to its outcome.

12.1 — Policy Locks (Immutable Configuration and Trust Core)

The **Policy Lock layer** establishes the unchangeable backbone of every SSMEQ deployment. It ensures that once a system has been commissioned, its parameters, lenses, anchors, and weight definitions remain frozen—anchored by manifest and stamp—and that no operator, integrator, or update can silently alter symbolic behavior.

This is not just a configuration safeguard; it is a philosophical commitment to **truth immutability**: the idea that once symbolic policy has been declared, its lineage and identity become part of an unbreakable chain of reproducibility.

Purpose and Principle
The aim of policy locking is to maintain **perfect determinism across all time and sites**. Every symbolic value in SSMEQ depends on a few key definitions. If these were allowed to drift, results would lose reproducibility and comparability. Hence, SSMEQ mandates that each critical constant be locked and cryptographically tied to its commissioning stamp.

This mechanism parallels the role of physical calibration seals in instruments—once fixed, they can only be reopened under explicit re-certification.

Immutable Elements (Frozen Fields)
The following parameters are locked immediately upon commissioning:

1. **lens_id[]** — the list of declared lenses used for each symbolic channel (V, I, P, f, pf, THD, etc.).
2. **anchors_ref** — reference magnitudes or calibration anchors used by each lens (e.g., V_ref, I_ref, f_ref).
3. **weight_basis** — definition of weighting logic (uniform, magnitude-based, or declared).
4. **gamma|weights[]** — explicit weighting constants or exponents (`w := |m|^gamma`).
5. **tick_ms** — time resolution for sampling or integration.
6. **eps_a, eps_w** — fixed small constants preventing saturation or division errors.
7. **division_policy** — symbolic rule for division near zero ("strict", "meadow", or "soft").
8. **band_table_ref** — declared hysteresis thresholds and rate limits (A^{+}..D).
9. **manifest_id** — the unique identifier for the governing manifest file that defines all the above.

All these are bound into a single immutable record known as the **Policy Lock File**.

Policy Lock Format (ASCII Example)

```
POLICY_LOCK
lens_id=[V_main,I_main,P_total,PF_line]
anchors_ref={V_ref:230.0,I_ref:10.0,PF_ref:1.0}
weight_basis=|m|^gamma
gamma=1.0
tick_ms=1000
eps_a=1e-6
eps_w=1e-12
division_policy=strict
band_table_ref=bands.json
manifest_id=ELECTRICAL_2025.Q4.STABLE
stamp=sha256(manifest_bytes || time_utc)
```

This single file, once stamped, becomes the source of truth for every downstream computation.

Lock Enforcement Logic

At commissioning:

```
hash_policy := sha256(policy_file_bytes)
commissioning_stamp := sha256(inputs_hash || outputs_hash || hash_policy ||
time_utc)
```

During runtime or audit:

```
verify sha256(policy_file_bytes) == recorded hash_policy
```

Any mismatch automatically invalidates the session and halts symbolic accumulation, ensuring that **no modified rulebook** can produce valid stamps.

Benefits

- **Audit Integrity:** Any policy drift is instantly detectable via hash mismatch.
- **Operational Safety:** Once locked, no parameter can shift silently.
- **Cross-Site Consistency:** Identical manifests across facilities yield identical symbolic behavior.
- **Regulatory Compliance:** The locked policy file serves as a tamper-evident configuration baseline.

Conceptual Reflection
In classical systems, calibration drift is an operational inconvenience.
In symbolic systems, it is an existential threat — it compromises reproducibility.
Policy Locks therefore act as both digital calibration seals and ethical contracts, ensuring that the Shunyaya symbolic framework remains incorruptible from within.

12.2 — Stamp Chain (Chronological Integrity and Proof of Continuity)

The **Stamp Chain** is the temporal spine of SSMEQ — a cryptographic ledger that binds every symbolic event, computation, and record into an unbroken sequence of trust.
It ensures that the evolution of data across time remains auditable, tamper-evident, and self-verifiable — even when systems operate offline, across air-gapped sites, or under constrained industrial conditions.

Every symbolic transaction (from a new measurement pool to a commissioning report) produces a **stamp**, and every stamp is linked to the previous one, forming a continuous, chronological thread.

Purpose and Rationale
The Stamp Chain provides **proof of order**, not just proof of content.
In traditional data logging, one may know *what* was recorded, but not whether the order of entries has been altered. The stamp chain guarantees both — content immutability and sequence integrity — through a single deterministic formula:

```
stamp_k := sha256(prev_stamp || sha256(payload_bytes) || time_utc)
```

This definition is minimal, ASCII-only, and platform-agnostic, ensuring every device can produce identical results without specialized cryptography libraries.

Core Concepts

1. **Prev-Linked Integrity**
   Each stamp is chained to its predecessor through `prev_stamp`.
   Breaking or reordering entries immediately invalidates subsequent hashes.
   The first link (genesis) uses `prev_stamp = "0"*64`.
2. **Payload Independence**
   The payload may represent any deterministic data segment — a sensor reading, a policy lock, a commissioning report, or a UI snapshot.
   Because the payload itself is hashed (`sha256(payload_bytes)`), the actual content can remain private or compressed, yet its fingerprint remains verifiable forever.
3. **Temporal Anchoring**
   The UTC time at stamping (`time_utc`) is included in the hash, embedding an irreversible temporal order.
   Optional use of `theta_deg` and `rasi_idx` from the symbolic clock system further converts each stamp into a human-interpretable moment in cyclic time (for example, “degree 126.582 on the daily circle”).

Canonical ASCII Format

```
SSMCLOCK1|iso_utc|rasi_idx|theta_deg|sha256(payload)|chain
```

Optional metadata tail (non-breaking):

```
|kv:algo=sha256;chain_algo=sha256;theta_prec=5;float=ieee75464;time_mode=derived_utc
```

This format inherits from the same foundation used by symbolic stamping across all Shunyaya systems, ensuring universal interoperability.

Daily Roll and Anchoring
For multi-entry sequences within a day, a daily roll anchor can be produced:

```
rollup_D := sha256( ascii(Stamp_1 "|" ... "|" Stamp_n) )
```

Sorted canonically by `(iso_utc, stamp_core, chain)`, this rollup acts as a public "no-later-than" proof — an optional audit marker confirming that all listed stamps existed by that moment in UTC.

Verification Process

At any future time, verification is as simple as:

```
for each stamp_k in chain:
    assert sha256(prev_stamp || sha256(payload_bytes) || time_utc) == recorded_stamp
```

Any missing, reordered, or edited record will break the chain immediately — revealing tampering with mathematical certainty.

Operational Notes

- **Offline Safe:** Works entirely without network or PKI; verification needs no external authority.
- **Compact:** Each stamp is a single line of ASCII; storage cost remains negligible.
- **Human-Readable:** The inclusion of symbolic clock angles (`theta_deg`, `rasi_idx`) makes every stamp auditable even by eye.
- **Algorithm-Flexible:** The kv-tail allows declaration of alternate hash algorithms (sha3_256, blake2b-256) without affecting legacy compatibility.

Example Sequence

```
SSMCLOCK1|2025-11-12T08:05:24Z|4|126.582|sha256(file_A)|f23a9...9b1c
SSMCLOCK1|2025-11-12T08:06:10Z|4|127.562|sha256(file_B)|6b3f0...e26d
SSMCLOCK1|2025-11-12T08:07:02Z|4|128.540|sha256(file_C)|bdc92...1fa0
```

Each subsequent chain entry depends mathematically on all prior ones.
Even a single-byte change in `file_A` invalidates the entire downstream sequence.

Ethical and Practical Strength
The stamp chain embodies **mathematical honesty**: it does not claim authority, it proves sequence.
By fusing UTC time, cyclic symbolic coordinates, and deterministic hashing, it provides a universal chronology for all symbolic computations — traceable, provable, and reproducible for decades.

Benefits

- **Irreversible lineage** across all data and manifests.
- **Tamper-evident archives** for regulatory or scientific records.
- **Unification** of time, data, and proof into one structure.
- **Human and machine readability** in one form.

12.3 — Repro Pack (Reproducibility and Independent Verification Toolkit)

The **Repro Pack** is the operational heart of reproducibility in SSMEQ.

It turns the symbolic framework from a theoretical construct into a **verifiable, portable, independently auditable system** that anyone can reproduce — anywhere, anytime — using only plain ASCII files and the canonical formulas.

Every Repro Pack carries the complete mathematical footprint of a deployment: **inputs, policies, expected outputs, and hashes**. From these, any competent reviewer can regenerate every result, confirm alignment integrity, and reproduce the **commissioning stamp bit-for-bit**, without proprietary tools or closed runtimes.

Purpose and Design Philosophy

The Repro Pack guarantees that symbolic outcomes are **deterministic, open, and long-lived** — resistant to software version drift, hardware changes, or institutional control.

It encapsulates the full symbolic evidence required to rebuild the electrical state from first principles, ensuring the core invariant

```
phi((m,a)) = m
```

remains inviolable through time.

Guiding principles:

- **Minimalism** — Only what is essential to reproduce results is stored.
- **Transparency** — All files are human-readable, ASCII, and self-explanatory.
- **Verification-first** — Each file participates in hashing or checksum logic.
- **Universality** — The structure is reusable across all Shunyaya symbolic domains, not just SSMEQ.

Canonical Contents

Each Repro Pack contains, at minimum, the following artifacts:

1. **`golden.csv` — Reference Dataset**
   The reference dataset of inputs and expected outputs used during commissioning. Each row contains symbolic and classical values for key dials:

   ```
   timestamp_utc, channel, m, a, band, U, W
   2025-11-12T08:01:00Z, V_main, 229.5, +0.742, A, 0.962, 1.294
   2025-11-12T08:01:00Z, I_main, 9.85,  +0.715, A, 0.880, 1.231
   2025-11-12T08:01:00Z, P_total,2258.3, +0.701, A, 0.913, 1.265
   ```

2. **`policy.json` — Policy Lock Snapshot**
   The frozen policy that governed the above dataset:

   ```
   {
     "lens_id": ["V_main", "I_main", "P_total"],
     "anchors_ref": {"V_ref": 230.0, "I_ref": 10.0, "PF_ref": 1.0},
     "eps_a": 1e-6,
     "eps_w": 1e-12,
     "gamma": 1.0,
     "division_policy": "strict",
     "band_table": "bands.json"
   }
   ```

3. **`expected.csv` — Benchmark Outputs**
   The benchmark outputs generated during commissioning:

   ```
   timestamp_utc, channel, a_pool, band, stamp
   2025-11-12T08:02:00Z, system, +0.813, A, sha256(...)
   ```

4. **`harness.py` (or `harness.txt`) — Reference Harness**
   A ~60-line reference script or pseudocode that recomputes symbolic outputs and verifies order-invariance:

```
# SSMEQ harness (deterministic)
read policy.json
read golden.csv

for each channel:
      clamp_a = max(-1+eps_a, min(+1-eps_a, a))
      U += w * atanh(clamp_a)
      W += w

      a_pool = tanh(U / max(W, eps_w))
      assert abs(a_pool - expected) < tol

   print("PASS: order-invariance confirmed")
```

5. **`commissioning_stamp.txt` — Binding Proof**
   One-line proof that binds the entire Repro Pack:

   ```
   commissioning_stamp := sha256(inputs_hash || outputs_hash ||
   time_utc)
   ```

   This stamp is re-verified during audits to confirm identical results and policy.

Workflow Overview

Step 1 — Validation
A new deployment runs the harness using **local measurements** but the same policy and structure.
The script compares outputs against `expected.csv` using declared tolerances.

Step 2 — Hash and Compare

```
sha256(golden.csv)   == golden_hash
sha256(policy.json)  == policy_hash
sha256(expected.csv) == expected_hash
```

If all comparisons hold, the Repro Pack is intact.

Step 3 — Stamp Reproduction
Using identical inputs and policy, a fresh commissioning stamp is generated:

```
commissioning_stamp := sha256(inputs_hash || outputs_hash || time_utc)
```

A matching stamp confirms that the site is **symbolically consistent** with the original reference; a mismatch demands investigation (policy drift, file changes, or harness modifications).

Extended Optional Files

- **`bands.json`** — Custom band thresholds and hysteresis timings (if different from defaults).
- **`manifest.log`** — Event-level documentation of manifest IDs, versions, and timestamps.
- **`variance.csv`** — Observed deviations between field runs and golden vectors, used for calibration and tuning without altering the locked policy.

Audit Example

A typical verification report might look like:

```
golden_hash:    4f7d2a...  PASS
policy_hash:    16b99e...  PASS
expected_hash:  e1a8f0...  PASS
stamp_match:    TRUE

Summary: Deterministic parity confirmed.
```

This concise log acts as the **audit certificate** for an SSMEQ site or device.

Benefits

- **Reproducibility across environments**
  Any site can independently re-run the harness and reproduce identical results.
- **Immutable trust**
  The commissioning stamp binds the full configuration (inputs + policy + outputs), providing tamper-evidence.
- **Open standard**
  All artifacts are ASCII-based and readable by both humans and machines.
- **Future resilience**
  Even decades later, results remain verifiable without any proprietary tooling, as long as SHA-256 and basic text processing exist.

Philosophical Perspective

The Repro Pack embodies an ethic of **clarity and permanence**:

Truth should not depend on context, tools, or authority.

It turns reproducibility from a slogan into a **living, testable property**.

If the pack reproduces, the truth holds.

12.4 — Independence Note (Observation-Only Principle)

The **Independence Note** defines one of the most important ethical and operational boundaries of the SSMEQ framework:
symbolic mathematics observes and informs, but it never replaces or overrides control logic.

This separation preserves both **scientific integrity** and **engineering safety** — ensuring that all symbolic quantities remain diagnostic and advisory until verified within a validated control envelope.

Core Principle
The symbolic lane `a` exists to express stability, drift, and posture alongside the classical magnitude `m`.
However, the system is explicitly designed so that:

```
phi((m,a)) = m
```

This means that the classical quantity always remains intact. Control systems, actuators, and protection circuits continue to act on `m` (the classical value), not on any transformed or augmented value.
The alignment lane `a` exists purely for monitoring, confidence evaluation, and predictive reasoning — not for direct actuation.

Operational Meaning

1. **Observation-Only Math**
   All symbolic computations are advisory. They describe *how stable* a number is, not *what should be done* about it.
   - Example: `a = -0.82` indicates severe drift, but corrective action must come from the plant's established safety logic or human decision, not automatically from the symbolic layer.
2. **Separation of Concerns**
   - Classical channels (V, I, P, PF, f, etc.) remain the official source of truth for hardware actions.
   - Symbolic channels run in parallel, producing context such as confidence, drift rate, or risk band.
   - No control relay, circuit breaker, or automation script should be directly triggered by `a` without explicit risk analysis and certification.
3. **Feedback Integration (Ethical Pathway)**
   Where symbolic data informs adaptive systems — for example, a scheduler reducing load or a predictive model forecasting failure — such interactions must follow a *bounded influence path*:

```
classical_input → symbolic_observer → advisory_output → human/AI
review → approved action
```

Never the reverse. Symbolic layers must remain logically and ethically insulated from safety-critical command paths.

Engineering Safeguards

- **Manifest Declaration:** Each deployment must explicitly record that symbolic quantities are for observation only.
- **Config-Level Enforcement:** If the framework is embedded in hardware or software, any attempt to use symbolic outputs as direct actuator inputs should be blocked by configuration.
- **Audit Flag:** Periodic audits should confirm that `a` values are used for analysis, dashboards, or alerts only — not for autonomous control loops.

Scientific Context
The independence of observation ensures reproducibility across all domains.
By keeping the symbolic lane passive, any recorded sequence can be replayed and analyzed without entanglement with uncontrolled external reactions.
This maintains the **purity of symbolic mathematics** as a diagnostic discipline and ensures that every result can be independently reproduced without safety dependencies.

Ethical Reflection
Symbolic independence is not a limitation — it is the foundation of trust.
It prevents misapplication of an emerging mathematical framework in live operational contexts until its behavior has been peer-validated and certified.
In practice, this distinction turns SSMEQ into a reliable partner for safety-critical industries: always aware, never intrusive.

Summary

- Symbolic values (`a`) inform but never command.
- Classical quantities (`m`) remain the sole control reference.
- Observation-only status must be preserved by manifest and audit.
- Separation ensures ethical deployment, reproducibility, and long-term trust.

12.5 — Data Validity Posture (Explicit Semantics and Suppression Rules)

The **Data Validity Posture** defines how SSMEQ handles uncertainty, gaps, and corrupted inputs in a transparent, deterministic way.
It introduces a **symbolic taxonomy of data states** — `DATA_VALID`, `DATA_INVALID`, `DATA_ABSENT`, and `QUORUM_LOST` — that ensures clarity and reproducibility even in imperfect or degraded environments.

Rather than silently discarding incomplete data or blending them into averages, SSMEQ marks their semantic state explicitly, propagates that state through the symbolic kernel, and applies controlled suppression or substitution logic as declared in the manifest.

Purpose and Principle
Real-world electrical systems face sensor dropouts, noise, and misreads. Classical analytics often hide these issues behind smoothing or interpolation.
SSMEQ takes the opposite stance: every datum, valid or invalid, must declare its condition — so that audits, analytics, and symbolic alignment remain trustworthy.

This creates a culture of **visible integrity** rather than silent correction.

Declared Data States

State	Meaning	Symbolic Behavior
DATA_VALID	Sensor reading verified and within declared range.	Included in U/W accumulation normally.
DATA_INVALID	Reading out of bounds or inconsistent with physical constraints.	Excluded from accumulation; logged with `a = -1.0`.
DATA_ABSENT	Reading missing, sensor offline, or timestamp gap detected.	U and W frozen; latest `a` held as advisory only.
QUORUM_LOST	Insufficient valid channels to compute a meaningful pool.	`a_pool` suppressed; band state = “UNDEFINED”.

All states are explicitly coded in the symbolic stream, preventing ambiguity or silent bias.

Symbolic Propagation Logic

At runtime:

```
if state == DATA_VALID:
    U += w * atanh(a_c)
    W += w
elif state == DATA_INVALID:
    log("Invalid data excluded"); continue
elif state == DATA_ABSENT:
    hold a_out := previous_a; freeze timestamp
elif state == QUORUM_LOST:
    set a_pool := None; mark band := "UNDEFINED"
```

These deterministic rules ensure consistent behavior across all implementations.

Suppression and Recovery Rules

1. **Suppression (Graceful Silence)**
   When data validity falls below quorum threshold (e.g., <70% valid samples), the pool result is withheld from dashboards and logs:

   ```
   if (N_valid / N_total) < quorum_threshold:
       suppress_output := True
   ```

   This prevents false stability signals.

2. **Recovery (Deterministic Reintegration)**
   Once quorum is restored and continuity is verified:

   ```
   a_pool_recovered := tanh(U / max(W, eps_w))
   band := classify(a_pool_recovered)
   ```

   The band state reappears with continuity trace showing downtime duration.

3. **Manifest Declaration**
   Each deployment defines its own thresholds and policies via:

   ```
   "data_policy": {
       "quorum_threshold": 0.7,
       "max_gap_ms": 5000,
       "invalid_action": "exclude",
       "absent_action": "hold"
   }
   ```

Audit and Visualization

- **Dashboards** mark invalid states in gray or dashed bands instead of hiding them.
- **Repro Packs** include full state timelines, so audits can reconstruct exactly when and why a symbol was suppressed.

- **Stamps** still chain all events, meaning even invalid or missing data produce visible time continuity.

Example fragment:

```
timestamp, a_pool, state, band
2025-11-12T09:01:00Z, +0.812, DATA_VALID, A
2025-11-12T09:02:00Z, -, DATA_ABSENT, UNDEFINED
2025-11-12T09:03:00Z, +0.803, DATA_VALID, A
```

Benefits

- **Complete transparency** — no silent interpolation or erasure.
- **Deterministic traceability** — invalid or absent states are visible and timestamped.
- **Audit resilience** — analysts can replay all logic without guessing intent.
- **Operational clarity** — dashboards clearly show when data is informative and when it isn't.

Ethical Context
In symbolic mathematics, truth includes its gaps.
Declaring data absence is as important as reporting its presence.
By embedding validity into the symbolic stream, SSMEQ ensures that every dataset carries its own self-assessment — **honesty as a numerical property**.

Summary

- Data validity is explicit, never implicit.
- Invalid or absent readings are logged, not discarded.
- Quorum-based suppression preserves integrity of pooled results.
- Each deployment's policy is fixed via manifest and verifiable by stamp.

13.0 — Reference Recipes (Portable, Vendor-Neutral Implementation Library)

The Reference Recipes section is the **implementation bridge** of SSMEQ — a compact library of formulas, pooling methods, data formats, and verification tools that any team can lift directly into code, spreadsheets, or firmware.

Here, the abstract symbolic rules of Shunyaya symbolic mathematics — collapse parity, bounded alignment, rapidity pooling, stamps, and manifests — are expressed as **small, copy-ready building blocks**.

Each recipe is:

- Written in plain ASCII
- Independent of any specific vendor, tool, or platform
- Suitable for microcontrollers, PLCs, scripts, or cloud analytics
- Designed to preserve the core invariants:
  - `phi((m,a)) = m` (classical values never altered)
  - `a ∈ (-1,+1)` (bounded alignment)
  - `U += w*atanh(a_c) , W += w , a_pool := tanh(U / max(W, eps_w))`
  - Identical outputs for batch, streaming, and shuffled inputs

Within this section you will find, in a compact, copy-ready form:

- A **lens library** for turning electrical measurements (voltage, current, power factor, THD, unbalance, temperature, etc.) into symbolic alignments.
- **Pooling recipes** that combine multiple alignments using the rapidity kernel in a way that is order-invariant and numerically stable.
- **Band and gating recipes** that map continuous alignment into readable A^{+}…D states with hysteresis and optional environmental gates.
- **CSV and JSON schemas** that define how members, samples, policies, and stamps are represented in plain text.
- A **minimal harness outline** that verifies order-invariance, policy-lock integrity, and commissioning stamps from first principles.

All of these recipes are:

- **Vendor-neutral** — they can be implemented in any language or stack without semantic drift.
- **Testable** — each can be plugged into a Repro Pack or commissioning harness to confirm numerical fidelity.
- **Future-proof** — the reliance on ASCII, explicit formulas, and deterministic rules ensures that implementations remain verifiable even decades from now.

The intent of this section is not to introduce new theory but to provide a **precise, portable cookbook**: any engineer, auditor, or researcher should be able to pick a recipe, implement it line-by-line, and obtain the same symbolic results as any other conforming SSMEQ deployment, anywhere in the world.

13.1 — Lens Library (Electrical, Copy-Ready)

The Lens Library provides **ready-to-use formulas** that convert raw electrical quantities into **bounded symbolic alignments** $a \in (-1,+1)$.

Each lens is:

- **Zero-centric** at a published reference (anchor)
- **Unit-invariant** where applicable
- **Monotone** with respect to the underlying physical stress
- **Clamp-safe** and compatible with the rapidity kernel

All lenses here are **vendor-neutral** and expressed in plain ASCII.

13.1.1 Common Guards and Notation (Copy-Ready)

These constants and helpers are shared by all lenses:

```
# Guards (publish once per deployment)
eps_pos := 1e-12      # positivity floor for ratios/logs
eps_div := 1e-12      # division floor
eps_a   := 1e-6       # alignment clamp epsilon

# Generic clamp and atanh helper
clamp(x, lo, hi) := max(lo, min(hi, x))

a_c := clamp(a_raw, -1+eps_a, +1-eps_a)
u   := atanh(a_c)        # rapidity
```

Zero-centric log contrast template:

```
e_LOG(x, x_ref) := ln( max(x, max(eps_pos, alpha*x_ref)) / x_ref )
```

with alpha ∈ [0, 1e-6] as a tiny relative floor.

13.1.2 Voltage and Current Magnitude Lenses (V, I)

Purpose. Encode deviations from nominal RMS values into zero-centric, unit-invariant contrasts.

Parameters (per channel):

```
V_ref  > 0      # nominal RMS voltage
I_ref  > 0      # nominal RMS current
c_V    > 0      # sensitivity for a_V
c_I    > 0      # sensitivity for a_I
```

Step 1 — Log contrasts:

```
e_V := e_LOG(V_rms, V_ref)
e_I := e_LOG(I_rms, I_ref)
```

Step 2 — Map to bounded alignment lanes:

```
a_V_raw := tanh( c_V * e_V )
a_I_raw := tanh( c_I * e_I )

a_V := clamp(a_V_raw, -1+eps_a, +1-eps_a)
a_I := clamp(a_I_raw, -1+eps_a, +1-eps_a)
```

Properties.

- e_V = 0 and e_I = 0 at V_rms = V_ref, I_rms = I_ref.
- Small deviations ±5% produce near-linear small a_V, a_I.

13.1.3 Power Factor Lens (PF)

Purpose. Turn power factor into a **symbolic corridor** around a published target pf_ref.

Parameters:

```
pf_ref  in (0,1]       # target or nominal PF (often 1.0)
pf_min  in (0, pf_ref] # lower bound to avoid log(0), e.g., 0.05
c_PF    > 0            # sensitivity for PF deviations
```

Steps:

```
pf_clamped := clamp(pf_raw, pf_min, 1.0)
e_pf       := ln( pf_clamped / pf_ref )
a_PF_raw   := tanh( c_PF * e_pf )
a_PF       := clamp(a_PF_raw, -1+eps_a, +1-eps_a)
```

Interpretation.

- a_PF ≈ 0 at pf_raw ≈ pf_ref
- a_PF < 0 when pf_raw < pf_ref ("below target")
- a_PF > 0 allowed when pf_ref < 1.0 and pf_raw exceeds it (if policy permits)

13.1.4 Voltage THD Lens (THDV)

Purpose. Map total harmonic distortion into a bounded index where **low THD → high alignment**, **high THD → low/negative alignment**.

Parameters:

```
THDV_ref > 0      # reference THD (e.g., limit or typical clean value)
c_THDV   > 0      # sensitivity
```

Steps:

1. **Contrast (log tail):**

```
THDV_safe := max(THDV_raw, eps_pos)
e_THDV    := ln( 1 + THDV_safe / THDV_ref )
```

2. **Alignment (invert so high distortion lowers alignment):**

Two equivalent options:

- **Option A (recommended): define a cleanliness score first**

```
e_clean    := -e_THDV              # more THD → more negative
a_THDV_raw := tanh( c_THDV * e_clean )
```

- **Option B (direct form):**

```
a_THDV_raw := tanh( -c_THDV * e_THDV )
```

3. **Clamp:**

```
a_THDV := clamp(a_THDV_raw, -1+eps_a, +1-eps_a)
```

Interpretation.

- a_THDV close to +1 ≈ very clean waveform.
- a_THDV near 0 ≈ around reference level.
- a_THDV drifting toward −1 ≈ heavy distortion.

13.1.5 Voltage Unbalance Lens (unbV)

Purpose. Express three-phase voltage unbalance as a bounded dial.

Let **Unb_V** be the percent or per-unit unbalance metric (e.g., NEMA or IEC definition).

Parameters:

```
Unb_ref > 0      # reference unbalance (typical or limit)
c_unbV  > 0      # sensitivity
```

Steps:

```
Unb_safe := max(Unb_V_raw, 0)
e_unbV   := ln( 1 + Unb_safe / Unb_ref )

# Higher unbalance → lower alignment
a_unbV_raw := tanh( -c_unbV * e_unbV )
```

```
a_unbV     := clamp(a_unbV_raw, -1+eps_a, +1-eps_a)
```

Interpretation.

- a_unbV ≈ +1 when phase voltages are well-balanced.
- a_unbV ≈ 0 around declared tolerance.
- a_unbV → −1 for extreme unbalance.

13.1.6 Thermal Lens (Temperature T)

Purpose. Capture cabinet, transformer, or breaker **thermal headroom** symbolically.

Parameters (per asset):

```
T_ref    # reference temperature (e.g., nominal ambient or operating)
T_max    # critical or nameplate limit
c_T      > 0
```

Contrast and alignment:

One simple, zero-centric form around T_ref with stronger penalty near T_max:

```
delta_T   := T_raw - T_ref
range_T   := max(T_max - T_ref, eps_div)
e_T       := delta_T / range_T             # linearized contrast
a_T_raw   := tanh( c_T * e_T )
a_T       := clamp(a_T_raw, -1+eps_a, +1-eps_a)
```

Alternative (if you want sharper tails close to T_max, log-based):

```
headroom  := max(T_max - T_raw, eps_pos)
e_T       := -ln( headroom / (T_max - T_ref) )    # headroom shrinkage
a_T_raw   := tanh( c_T * e_T )
a_T       := clamp(a_T_raw, -1+eps_a, +1-eps_a)
```

Interpretation.

- a_T near +1 ≈ cool, comfortable.
- a_T near 0 ≈ mid-range.
- a_T → −1 as T_raw approaches or exceeds T_max.

13.1.7 Composite Stress Lens (a_stress from multiple contrasts)

Purpose. Provide a canonical recipe for fusing several per-channel contrasts into a **single stress dial** usable as a_stress (aligned with Section 7.2).

Inputs per tick:

```
a_THDV, a_unbV, a_PF, a_I, a_T   # any subset; all ∈ (-1,+1)
weights w_j >= 0                 # per-dial importance
eps_a, eps_w                     # guards
```

Recipe (copy-ready):

```
U := 0 ; W := 0
for each dial j in DECLARED_DIAL_SET:
    if dial j is valid at this tick:
        a_c := clamp(a_j, -1+eps_a, +1-eps_a)
        U  += w_j * atanh(a_c)
        W  += w_j

a_stress := (W == 0) ? 0 : tanh( U / max(W, eps_w) )
```

This is the canonical **rapidity pooling** recipe reused across SSMEQ.

13.1.8 Minimal Lens Table (Profile Snippet)

A typical lens configuration in a manifest may look like:

```
"lens_library": {
  "V_main":  { "type": "LOG",   "ref": 230.0,   "c": 4.0 },
  "I_main":  { "type": "LOG",   "ref": 10.0,    "c": 3.0 },
  "PF":      { "type": "PF",    "ref": 1.0,     "pf_min": 0.05, "c": 5.0 },
  "THDV":    { "type": "THDV",  "ref": 0.03,    "c": 3.0 },
  "UNB_V":   { "type": "UNB_V", "ref": 0.02,    "c": 3.0 },
  "TEMP":    { "type": "TEMP",  "T_ref": 40.0,  "T_max": 80.0, "c": 3.0 }
}
```

Each implementation maps these entries to the formulas defined above, preserving:

- **bounded alignment** (−1,+1)
- **zero-centric references**
- **deterministic pooling via U/W**

13.2 — Pooling Recipes (Uniform, Weighted, and Stream-Compatible Fusion)

The Pooling Recipes define how multiple symbolic readings `a_i` from sensors, phases, or time slices are fused into a single representative alignment value `a_pool`.

Unlike classical arithmetic means that collapse meaning through averaging, SSMEQ pooling preserves both magnitude integrity and directional context by operating in rapidity space through the canonical `U/W` kernel.

This guarantees **order invariance**, **numerical stability**, and **semantic consistency** across every data collection method — batch, stream, or shuffled.

Core Pooling Kernel

The universal pooling kernel of SSMEQ is expressed as:

```
# Per-sample:
a_c := clamp(a, -1 + eps_a, +1 - eps_a)
u   := atanh(a_c)

# Pool:
U += w * u
W += w
a_pool := (W == 0) ? 0 : tanh( U / max(W, eps_w) )
```

where:

- `a` = symbolic input from lens output
- `a_c` = clamped alignment in `(-1,+1)` using `eps_a > 0`
- `w` = declared or computed weight for the input
- `U,W` = running accumulators for numerator and denominator
- `eps_w` = small constant to avoid division by zero (default `1e-9`)

This equation ensures that alignment values combine **hyperbolically** rather than linearly, maintaining boundedness and directional coherence.

Pooling Modes

1. Uniform Weighting (default)

```
U := 0 ; W := 0
for each valid a_i:
    a_c := clamp(a_i, -1+eps_a, +1-eps_a)
    U   += atanh(a_c)
    W   += 1
a_pool := (W == 0) ? 0 : tanh( U / max(W, eps_w) )
```

Used when all sources (sensors or channels) are equally reliable.
Example: three-phase voltage alignment pooling.

---

2. Scale Weighting

```
# x_i: scale or magnitude (e.g., kW, kVA, capacity)
S := sum_over_valid(i)( x_i )

U := 0 ; W := 0
```

```
for each valid a_i:
    w_i := x_i / max(S, eps_w)
    a_c := clamp(a_i, -1+eps_a, +1-eps_a)
    U   += w_i * atanh(a_c)
    W   += w_i
a_pool := (W == 0) ? 0 : tanh( U / max(W, eps_w) )
```

Useful when channels represent magnitudes of differing significance — e.g., power-weighted harmonics or load-proportional sensors.

---

3. Declared Weighting

```
U := 0 ; W := 0
for each valid a_i:
    w_i := w_declared_i          # from manifest/policy
    a_c := clamp(a_i, -1+eps_a, +1-eps_a)
    U   += w_i * atanh(a_c)
    W   += w_i
a_pool := (W == 0) ? 0 : tanh( U / max(W, eps_w) )
```

Weights are explicitly specified in the manifest (e.g., `weight_basis: "declared"` or `channel_priority`).
This is common in certified plant systems where reliability levels differ.

Stream-Compatible Form

The pooling process can operate continuously in streaming environments without reinitialization:

```
def update_pool(a_new, w_new, U_prev, W_prev):
    a_c   = clamp(a_new, -1+eps_a, +1-eps_a)
    U_next = U_prev + w_new * atanh(a_c)
    W_next = W_prev + w_new
    a_pool = (W_next == 0) ? 0 : tanh( U_next / max(W_next, eps_w) )
    return (U_next, W_next, a_pool)
```

This form guarantees that live systems and offline computations converge identically:

```
batch == streaming == shuffled
```

given the same inputs and weights.

Order-Invariance Verification

A simple shuffle test verifies kernel determinism:

```
import random, math
```

```
def pool(A):
    U = W = 0.0
    for a in A:
        a_c = max(-1+1e-6, min(+1-1e-6, a))
        U  += math.atanh(a_c)
        W  += 1.0
    return math.tanh(U / W)

A = [+0.25, +0.50, -0.20, +0.35]
ref = pool(A)

for i in range(10):
    random.shuffle(A)
    chk = pool(A)
    print(abs(chk - ref))
```

Expected: all differences are within floating tolerance, confirming the pooling kernel is **effectively commutative and associative** under the tanh–atanh mapping.

Specialized Extensions

1. **Time-Decay Pooling (for drift-sensitive metrics)**

```
U += w * exp(-λ t) * atanh(a_c)
W += w * exp(-λ t)
```

Introduces recency emphasis; older samples fade by exponential decay rate `λ`.

---

2. **Harmonic Pooling (frequency-weighted)**

```
w_i := 1 / order_i^2
```

Applies when combining harmonic distortion components (e.g., 3rd, 5th, 7th order).

---

3. **Reliability Pooling (confidence-weighted)**

```
w_i := r_i / sum(r_j)
```

where `r_i` is sensor reliability from calibration logs or trust scores.

Practical Example

```
import math

a_V = [+0.20, +0.10, -0.05]
U = W = 0.0
```

```
eps_a = 1e-6
eps_w = 1e-9

for a in a_V:
    a_c = max(-1+eps_a, min(+1-eps_a, a))
    U  += math.atanh(a_c)
    W  += 1.0

a_pool = math.tanh(U / max(W, eps_w))
print(a_pool)  # ≈ +0.086
```

This indicates a mild positive alignment (close to reference) despite small local deviations — a more truthful aggregate than a plain arithmetic mean.

Benefits

- **Stable under mixing**: extreme values saturate smoothly.
- **Order invariant**: identical results regardless of sensor sequence.
- **Weight transparent**: policies define, not hide, contribution rules.
- **Universally portable**: can run in Python, C, PLC logic, spreadsheet, or embedded code.

Ethical Reflection

Pooling represents **symbolic fairness**: every data source contributes according to its manifest weight, without dominance or distortion.

It ensures that truth in measurement emerges from **balanced composition**, not arbitrary hierarchy — a principle that extends beyond engineering into all domains of systemic alignment.

13.3 — Band Tables (A^+…D) and Environmental Gating (Normative, Copy-Ready)

The Band Tables define the universal classification system that translates symbolic alignment values $a \in (-1,+1)$ into intuitive stability bands (A^+, A, B, C, D).

They are the **semantic surface** of SSMEQ: a shared language so that "A^+" means the same posture in every plant, lab, and fleet, independent of vendor or UI.

Environmental gating then provides a **situational dampener** that can soften or tighten decisions without ever altering the underlying truth a or the stored bytes.

Purpose and Principle

- **Bands**: map continuous alignment into discrete health states with **hysteresis** (enter / clear / hold).
- **Hysteresis**: prevents band flapping when `a` hovers near a threshold.
- **Environmental gate** `g_t`: scales the **decision input only** (`a_env := g_t * a`) so policies can adapt to storms, maintenance, or grid emergencies without touching data.

All band and gate behavior must be **manifest-declared**, **deterministic**, and **auditable**.

Canonical Band Table (Starter Defaults, Copy-Ready)

These thresholds are portable defaults; sites may tune them but must publish any changes.

Band	Meaning	Enter Condition	Clear Condition
A^+	Ideal alignment, near perfect	`a ≥ +0.90`	`a ≥ +0.85`
A	Healthy, within tolerance	`+0.60 ≤ a < +0.90`	`+0.55 ≤ a < +0.90`
B	Mild deviation, monitor	`+0.25 ≤ a < +0.60`	`+0.20 ≤ a < +0.60`
C	Significant drift, review	`−0.20 ≤ a < +0.25`	`−0.25 ≤ a < +0.30`
D	Unstable / critical	`a < −0.20`	`a < −0.25`

- **Hysteresis gap**: enter vs clear differ by ≈ `0.05` in `a`.
- Same principles apply if sites choose a slightly different ladder (e.g., A^+/A/B/C/D thresholds customized per dial).

Band Assignment Logic (Per Tick, No Gate)

Core logic (before any gate):

```
function band_from_a(a):
  if a >= 0.90: return "A+"
  if a >= 0.60: return "A"
  if a >= 0.25: return "B"
  if a >= -0.20: return "C"
  return "D"
```

Hysteresis is implemented via separate **enter/clear** thresholds and optional **hold times** (see Sections 5.1–5.2):

```
# Inputs: a_t, band_{t-1}, thresholds_enter[], thresholds_clear[]
band_candidate := band_from_a(a_t)

# Upgrade logic: only move up when enter condition held long enough.
# Downgrade logic: only move down when clear condition held long enough.
# Time guards: t_hold_enter, t_hold_clear per band.
```

A simple conceptual pattern:

```
if band_candidate is "higher" than band_{t-1}:
    require a_t >= enter_threshold[new_band] for t_hold_enter[new_band]
if band_candidate is "lower" than band_{t-1}:
    require a_t <= clear_threshold[old_band] for t_hold_clear[old_band]
```

This is consistent with the **ladder_hysteresis** patterns used elsewhere in the spec.

Time-in-Band and Transition Tracking

Each dial or pool maintains:

- `time_in_band[band]` — cumulative ms in the current band.
- `transition_log` — list of `(old_band, new_band, time_utc, stamp)`.

Example logic (conceptual):

```
if band_t == band_{t-1}:
    time_in_band[band_t] += tick_ms
else:
    time_in_band[band_t] := 0
    emit_transition_stamp(old=band_{t-1}, new=band_t, time_utc)
```

Transition stamp (aligned with stamp chain rules):

```
transition_stamp := sha256(prev_stamp || band_old || band_new ||
time_utc_iso8601)
prev_stamp := transition_stamp
```

This gives investigators a crisp history of **when** and **how long** each posture was held.

Environmental Gating (Summary of Normative Behavior)

Environmental gating introduces a **decision surrogate**:

```
a_env := g_t * a
```

with the following **non-negotiable rules** (matching Section 8.2):

1. **Do not modify the dial itself.**
   - Storage, pooling, CSVs, and stamps remain on `a` (or `a_pool`), not on `a_env`.
   - Collapse parity remains inviolable: `phi((m,a)) = m`.
2. **Gate only on the band/decision path.**
   - Band tests and actions use `a_env`.
   - Analytics, long-term statistics, and Repro Packs use the ungated `a`.
3. **Gate range:** `g_t ∈ [0,1]`.
   - `g_t = 1` ⇒ no damping (neutral).

- `g_t < 1` ⇒ effectively "softens" the decision input.

Typical composition (per Section 8.2):

```
g_t := compose(
  g_weather,  # storms
  g_grid,     # system emergency
  g_maint,    # maintenance window
  g_operator  # explicit temporary override
)
# e.g. product or min over factors, declared in manifest.
```

Each factor lies in `[0,1]` and is manifest-described.

Decision substitution (per tick):

```
a_env := g_t * a          # used ONLY for banding and routing

band_new := ladder_hysteresis(
               a_env, band_old,
               th_enter[], th_clear[], t_hold[]
            )
```

The envelope records both:

```
{
  "dial_id": "a_THDV_pool",
  "value": 0.606,        // raw a
  "value_env": 0.424,    // g_t * a
  "gate": 0.70,
  "band": "C",
  ...
}
```

Manifest Example (band + gate, Copy-Ready)

```
{
  "bands": {
    "A+": { "enter": 0.90, "clear": 0.85, "hold_s": 0 },
    "A":  { "enter": 0.60, "clear": 0.55, "hold_s": 0 },
    "B":  { "enter": 0.25, "clear": 0.20, "hold_s": 300 },
    "C":  { "enter": -0.20, "clear": -0.25, "hold_s": 600 },
    "D":  { "enter": -1.00, "clear": -1.00, "hold_s": 0 }
  },
  "gate_policy": {
    "mode": "min",           // or "product"
    "factors": ["weather","grid","maint","operator"],
    "defaults": {
      "weather": 1.0,
      "grid":    1.0,
      "maint":   1.0,
      "operator":1.0
    },
    "g_min_vis": 0.5,        // UI never hides more than this
    "g_min_band": { "D": 0.9 }
  }
}
```

- `hold_s` can be 0 for purely "symbolic" bands, larger for operational bands.
- `g_min_band["D"] = 0.9` ensures critical band D is barely damped.

Commissioning Checks (Bands + Gate)

1. **Band neutrality**: With `g_t = 1` for all ticks, band sequences and stamps match ungated runs bit-for-bit.
2. **Gate monotonicity**: Reducing `g_t` must **not** cause earlier band entry than with a larger `g_t`.
3. **Critical floor**: If `g_min_band["D"] > 0`, verify that D-band decisions respect the floor (no over-damping of criticals).
4. **Hysteresis stability**: Synthetic sweeps across boundaries demonstrate no flapping; holds respect `t_hold`.
5. **Audit continuity**: Envelopes include `value`, `value_env`, and `gate`; recomputing `value_env` matches logs.

Operator & Ethical View

- Operators see **both** `a` and `gate` in the same card:
  - e.g., `a = +0.64`, `gate = 0.7`, `a_env = +0.45`, band = B.
- A "Replay with gate=1" tool lets them see what would have occurred without damping.
- Gates are **temporary** and **reason-tagged** (e.g. `storm: YELLOW`, `planned_maintenance`).

Bands thus convert continuous symbolic truth into calm, discrete awareness, while environmental gating lets policy adapt without ever rewriting reality.

13.4 — CSV / JSON Schemas (Minimal, Portable Data Structures)

The **CSV and JSON Schemas** form the data backbone of SSMEQ — lightweight, interoperable formats designed for quick implementation, zero ambiguity, and seamless exchange across devices, labs, or cloud platforms.
Each file is minimal by intent: human-readable, reproducible, and designed for checksum integrity. Together, they enable complete **symbolic reproducibility** with no dependency on proprietary formats or external parsing libraries.

Philosophy and Design Rules

- **Plain ASCII only:** Every schema must be representable and editable in a basic text editor.
- **Explicit headers:** No hidden columns or inferred meanings — every field is declared in-line.
- **Stable naming:** Field names are fixed and identical across all domains.
- **Auditability:** Each row or record can carry a checksum or stamp field to verify lineage.
- **Zero ambiguity:** No null inference — missing values are explicit (`DATA_ABSENT`, `NA`, or equivalent).

---

1. members.csv — Declared Channels and Lenses

```
channel_id, lens_id, ref_value, unit, weight, active
V_main, a_dV, 230.0, V, 1.0, TRUE
I_main, a_I, 10.0, A, 1.0, TRUE
PF_sys, a_PF, 1.0, -, 1.0, TRUE
THDV, a_THDV, 5.0, %, 1.0, TRUE
THDI, a_THDI, 8.0, %, 1.0, TRUE
T_main, a_T, 45.0, °C, 0.8, TRUE
```

This table defines **what** the system measures and **how** it should be symbolically interpreted. Each lens entry connects a physical channel to its symbolic formula and declared reference.

---

2. samples.csv — Time-Series Measurements

```
timestamp_utc, channel_id, m, a, state, band
2025-11-12T10:01:00Z, V_main, 229.6, +0.015, DATA_VALID, A
2025-11-12T10:01:00Z, I_main, 9.84, +0.012, DATA_VALID, A
2025-11-12T10:01:00Z, PF_sys, 0.97, +0.701, DATA_VALID, A+
2025-11-12T10:02:00Z, THDV, 5.6, -0.122, DATA_VALID, B
```

Each row records a symbolic reading at a fixed timestamp.
The `state` field ensures integrity even when readings are missing or invalid.

---

3. policy.json — Policy Locks and Operational Parameters

```
{
  "lens_id": ["a_dV","a_I","a_PF","a_THDV","a_THDI"],
  "anchors_ref": {"V_ref":230.0,"I_ref":10.0,"PF_ref":1.0},
  "eps_a": 1e-6,
  "eps_w": 1e-9,
  "quorum_threshold": 0.7,
  "division_policy": "strict",
  "weight_basis": "declared",
  "band_table": "bands.json",
  "environment_gate": {"mode":"exp_decay","k_env":0.5,"EnvRef":1.0}
```

```
}
```

This file defines all mathematical constants and thresholds that govern the runtime behavior of SSMEQ.
It serves as the **contract of truth** — ensuring that every device, cloud instance, or auditor operates with identical rules.

---

4. expected.csv — Benchmark Results for Repro Packs

```
timestamp_utc, channel_id, a_pool, band, stamp
2025-11-12T10:01:00Z, system, +0.812, A, sha256(…)
2025-11-12T10:05:00Z, system, +0.768, A, sha256(…)
```

Generated once during commissioning, this file provides the baseline symbolic results against which later runs are verified.

---

5. stamps.csv — Symbolic Integrity Chain

```
index, time_utc, stamp, prev_stamp, event
0, 2025-11-12T10:00:00Z, sha256(a1b2...), NULL, INIT
1, 2025-11-12T10:01:00Z, sha256(3d5c...), sha256(a1b2...), MEASURE
2, 2025-11-12T10:02:00Z, sha256(5e7f...), sha256(3d5c...), UPDATE
```

This file encodes the **cryptographic continuity** of symbolic time — verifying that no data has been altered, deleted, or reordered.
Each stamp binds to its predecessor and event context, producing an immutable audit trail.

Optional Supporting Schemas

- **bands.json:** defines thresholds and hysteresis for A^{+}…D classification.
- **manifest.log:** running record of manifest IDs, timestamps, and commissioning stamps.
- **variance.csv:** optional file for comparing expected vs. observed deviations (`Δa = a_observed - a_expected`).

Implementation Best Practices

- Maintain **consistent UTC timestamps** across all CSVs.
- Hash all files in sequence (`sha256(file_bytes)`) before generating a commissioning stamp.
- Use **comma** separators only (no tabs or semicolons).
- Retain all files under a single manifest folder to preserve lineage.

Benefits

- **Human readability:** all data viewable in any text editor or spreadsheet.
- **Machine portability:** identical field names across systems enable direct API ingestion.
- **Audit simplicity:** anyone can recompute hashes and verify chain integrity.
- **Long-term resilience:** data remains usable even decades later, without decoding tools.

Philosophical Reflection
In SSMEQ, the CSV and JSON schemas are not mere containers — they are the **digital scriptures of truth**.
They ensure that meaning, measurement, and integrity travel together, immune to platform decay or semantic drift.
Through these simple formats, mathematics itself becomes self-verifying and immortal.

13.5 — Harness Outline (Deterministic Order-Invariance Verifier)

The Harness Outline is the capstone of the SSMEQ reference library.
It is the **portable, 60-line proof mechanism** that demonstrates the three foundational invariants of symbolic mathematics:

1. **Order-invariance**
   (batch == stream == shuffle)
2. **Policy-lock fidelity**
   (no silent drift in eps_a, eps_w, weights, bands, or lens definitions)
3. **Deterministic parity**
   (expected alignments in *expected.csv* match recomputed values exactly)

This harness is intentionally language-agnostic.
Whether implemented in Python, C, JavaScript, or a microcontroller, it must always produce identical outputs when operating on the same canonical CSV/JSON files.

Purpose and Structure

The harness does **not** collect or interpret live electrical data.

Instead, it:

1. Loads a frozen **policy.json**
2. Reads symbolic baseline inputs from **golden.csv**
3. Recomputes alignment pooling via the canonical U/W kernel
4. Compares results with **expected.csv**
5. Shuffles inputs to validate order-invariance
6. Emits a **commissioning stamp** to prove mathematical identity

Its goal is to allow any engineer, auditor, or deployment pipeline to independently verify symbolic correctness in under one minute.

Canonical Pseudo-Code (Copy-Ready)

```
# ------------------------------------------------
# SSMEQ Harness — Deterministic Parity Verifier
# ------------------------------------------------
import json, csv, random, math, hashlib

# --- Step 1: Load Policy --------------------------------
policy = json.load(open("policy.json"))
eps_a = policy["eps_a"]
eps_w = policy["eps_w"]

# --- Step 2: Read Golden Data ---------------------------
records = []
with open("golden.csv") as f:
    for row in csv.DictReader(f):
        a = float(row["a"])
        a_c = max(-1+eps_a, min(+1-eps_a, a))
        w  = float(row.get("weight", 1.0))
        records.append((a_c, w))

# --- Step 3: Pool Function ------------------------------
def pool(data):
    U = W = 0.0
    for (a, w) in data:
        U += w * math.atanh(a)
        W += w
    return math.tanh(U / max(W, eps_w))

# --- Step 4: Deterministic Test -------------------------
base = pool(records)
for i in range(10):
    random.shuffle(records)
    test = pool(records)
    assert abs(test - base) < 1e-9
print("PASS: order-invariance confirmed, a_pool =", round(base, 6))

# --- Step 5: Policy Lock Check --------------------------
policy_hash = hashlib.sha256(open("policy.json","rb").read()).hexdigest()
print("policy_hash =", policy_hash)

# --- Step 6: Commissioning Stamp ------------------------
inputs_hash  = hashlib.sha256(open("golden.csv","rb").read()).hexdigest()
outputs_hash = hashlib.sha256(open("expected.csv","rb").read()).hexdigest()
```

```
stamp = hashlib.sha256((inputs_hash + outputs_hash).encode()).hexdigest()
print("commissioning_stamp =", stamp)
```

This reference script must reproduce identical a_pool, policy_hash, and commissioning_stamp values across all platforms.

Verification Outputs (Expected Form)

A valid harness run always prints:

```
PASS: order-invariance confirmed, a_pool = +0.813002
policy_hash = 16b99eac...f3b7
commissioning_stamp = 7bcd91a2...5ff9
```

These hashes can be compared directly with commissioning logs to confirm that:

✔ symbolic arithmetic remains unchanged
✔ manifests have not drifted
✔ pooling behavior is mathematically identical

Recommended Add-Ons

To strengthen verification:

- **Tolerance-drift test:**
  increment a_i by ±0.01 and confirm smooth tanh behavior.
- **Weight-perturbation test:**
  verify stability under small weight variations (< 1%).
- **Batch vs. stream test:**
  compute a_pool incrementally and confirm exact parity.
- **Manifest integrity log:**
  store SHA-256 hashes of all input files in `verify.log` for archival proof.

Benefits

- **Reproducible in < 1 minute**
- **Cross-language identical results**
- **No hidden scaling or approximations**
- **Future-proof: remains valid decades later**
- **Auditable from plain ASCII CSV/JSON files**

Philosophical Reflection

The harness is more than a script.
It is the **ritual of mathematical truth** in SSMEQ.

Every time it prints:

"PASS: order-invariance confirmed"

it re-establishes the integrity of symbolic arithmetic —
proving that truth, when expressed properly, does not change with tools, time, or interpretation.

13.6 — Residual Test Pack (r_P Focused, Copy-Ready)

Purpose. The Residual Test Pack is the small, concrete truth table for the power-identity residual

```
r_P := e_P - (e_V + e_I + e_pf)
```

It provides a handful of tiny numeric cases that any lab can run to confirm that its implementation of `r_P` behaves correctly:

- Near-zero residual when measurements are internally consistent
- Small residual for mild CT/PT bias
- Large residual for wiring / sign errors
- Large residual for pathological low–pf cases (and why `pf_min` matters)

This pack is designed to integrate cleanly with the Repro Pack and harness flows described earlier.

Core Residual Definition (Recap, Copy-Ready)

For this test pack, assume the following references and guards:

- `V_ref := 230.0`
- `I_ref := 10.0`
- `pf_ref := 1.0`
- `pf_min := 0.05`
- `eps_pos := 1e-12`

Log contrasts (copy-ready):

```
e_V := ln( max(V_meas, eps_pos) / V_ref )
e_I := ln( max(|I_meas|, eps_pos) / I_ref )
```

```
pf_eff := max(pf_meas, pf_min)
e_pf := ln( pf_eff / pf_ref )

P_ref := V_ref * I_ref * pf_ref
e_P := ln( max(|P_meas|, eps_pos) / P_ref )
```

Residual:

```
r_P := e_P - (e_V + e_I + e_pf)
```

Default classification rule (recommended):

- If `|r_P| <= tol_rP` → `status := "PASS"`
- Else → `status := "ALARM"`

with `tol_rP` typically in the range `0.02 .. 0.05` (log-domain).

Canonical Test Cases (Numeric Mini-Table)

All rows use:

- `V_ref = 230.0`, `I_ref = 10.0`, `pf_ref = 1.0`, `pf_min = 0.05`
- `P_meas` is in watts, usually `P_meas ≈ V * I * pf` except where deliberately corrupted.

Values shown are rounded to 4 decimal places (implementations may keep higher precision).

Canonical Test Cases (Numeric Mini-Set, Copy-Ready)

All cases use:

- `V_ref = 230.0`
- `I_ref = 10.0`
- `pf_ref = 1.0`
- `pf_min = 0.05`
- `tol_rP = 0.05`

Case C1 — Baseline (no error)

- Description: Baseline, internally consistent measurements
- Inputs:
  - `V_meas = 230.0`
  - `I_meas = 10.0`
  - `pf_meas = 0.99`
  - `P_meas ≈ 230 * 10 * 0.99 ≈ 2277`
- Contrasts (rounded):
  - `e_V ≈ 0.0000`
  - `e_I ≈ 0.0000`
  - `e_pf ≈ -0.0101`
  - `e_P ≈ -0.0101`

- Residual:
  - `r_P ≈ 0.0000`
  - `|r_P| <= tol_rP` → **status := "PASS"**

Case C2 — CT ratio +2 % on current

- Description: Mild CT ratio error on current (+2 %)
- Inputs:
  - `V_meas = 230.0`
  - `I_meas = 10.2`
  - `pf_meas = 0.99`
  - `P_meas ≈ 230 * 10 * 0.99 ≈ 2277` (P not corrected for CT bias)
- Contrasts (rounded):
  - `e_V ≈ 0.0000`
  - `e_I ≈ 0.0198`
  - `e_pf ≈ -0.0101`
  - `e_P ≈ -0.0101`
- Residual:
  - `r_P ≈ -0.0198`
  - `|r_P| <= tol_rP` → **status := "PASS"** (mild bias visible if tol_rP tightened)

Case C3 — Sign flip on current and power

- Description: Serious wiring / sign error (I and P negative, pf negative)
- Inputs:
  - `V_meas = 230.0`
  - `I_meas = -10.0`
  - `pf_meas = -0.99`
  - `P_meas ≈ -230 * 10 * 0.99 ≈ -2277`
- Contrasts (rounded):
  - `e_V ≈ 0.0000`
  - `e_I ≈ 0.0000` (uses `abs(I_meas)`)
  - `e_pf ≈ -2.9957` (after applying `pf_min`)
  - `e_P ≈ -0.0101`
- Residual:
  - `r_P ≈ 2.9857`
  - `|r_P| >> tol_rP` → **status := "ALARM"`** (wiring / sign inconsistency)

Case C4 — pf near zero, physically consistent P

- Description: Very low power factor, but P consistent with V and I
- Inputs:
  - `V_meas = 230.0`
  - `I_meas = 10.0`
  - `pf_meas = 0.02`
  - `P_meas ≈ 230 * 10 * 0.02 ≈ 46`
- PF floor handling:
  - `pf_eff = max(pf_meas, pf_min) = 0.05`

- Contrasts (rounded):
  - `e_V ≈ 0.0000`
  - `e_I ≈ 0.0000`
  - `e_pf ≈ ln(0.05 / 1.0) ≈ -2.9957`
  - `e_P ≈ ln(46 / 2300) ≈ -3.9120`
- Residual:
  - `r_P ≈ -0.9163`
  - `|r_P| >> tol_rP` → **status := "ALARM"**` (extreme low-pf regime, investigate)

Case C5 — pf near zero, inconsistent P

- Description: Reported pf very low, but power closer to 0.5 instead of 0.02
- Inputs:
  - `V_meas = 230.0`
  - `I_meas = 10.0`
  - `pf_meas = 0.02`
  - `P_meas ≈ 230 * 10 * 0.50 ≈ 1150`
- PF floor handling:
  - `pf_eff = max(pf_meas, pf_min) = 0.05`
- Contrasts (rounded):
  - `e_V ≈ 0.0000`
  - `e_I ≈ 0.0000`
  - `e_pf ≈ -2.9957`
  - `e_P ≈ ln(1150 / 2300) ≈ -0.6931`
- Residual:
  - `r_P ≈ 2.3026`
  - `|r_P| >> tol_rP` → **status := "ALARM"**` (severe mismatch between pf and P)

These five residual cases (C1…C5) form the canonical **Residual Test Pack** for `r_P`. Any conformant implementation should reproduce the same `r_P` values (within rounding tolerance) and the same `PASS` / `ALARM` status.

`*` In C4 and C5 the PF lens uses `pf_eff := max(pf_meas, pf_min) = 0.05`, so `e_pf = ln(0.05 / 1.0) ≈ -2.9957`

while `e_P` still uses the **actual** `P_meas`, which exposes the tension between low–pf operation and the floor `pf_min`.

Interpretation patterns (copy-ready):

- **C1 (Baseline).** `r_P ≈ 0` → internal consistency; should be `PASS`.
- **C2 (CT +2%).** Small `|r_P|` shows mild CT ratio bias; visible if `tol_rP` is tightened (for example `0.01`), but passes `0.05`.
- **C3 (Sign flip).** Very large `|r_P|` → wrong sign / quadrants; immediate `ALARM`.
- **C4 (pf ≈ 0, P consistent).** Large negative residual: physics of extremely low pf plus `pf_min` choice. Treat as "investigate", never silently normalized away.
- **C5 (pf ≈ 0, P inconsistent).** Very large positive residual: strong inconsistency between reported pf and power; clear `ALARM`.

Copy-Ready Residual Table for Repro Packs

These rows can be dropped into a residual-focused CSV as part of a Repro Pack:

```
case_id, V_meas, I_meas, pf_meas, P_meas, V_ref, I_ref, pf_ref, pf_min,
tol_rP, expected_rP, expected_status
C1, 230.0, 10.0, 0.99, 2277.0, 230.0, 10.0, 1.0, 0.05, 0.05, 0.0000, PASS
C2, 230.0, 10.2, 0.99, 2277.0, 230.0, 10.0, 1.0, 0.05, 0.05,-0.0198, PASS
C3, 230.0,-10.0,-0.99,-2277.0, 230.0, 10.0, 1.0, 0.05, 0.05, 2.9857, ALARM
C4, 230.0, 10.0, 0.02,   46.0, 230.0, 10.0, 1.0, 0.05, 0.05,-0.9163, ALARM
C5, 230.0, 10.0, 0.02, 1150.0, 230.0, 10.0, 1.0, 0.05, 0.05, 2.3026, ALARM
```

Numeric values may be stored at higher precision; the key invariants are:

- Recomputing `r_P` with the declared formulas yields the same sign and approximate magnitude.
- `expected_status` matches the simple rule based on `tol_rP`.

Residual Harness Snippet (Copy-Ready)

This pseudo-code extends the harness to test the residual logic:

```
read residual_test_csv

for each row:
  V      := row.V_meas
  I      := row.I_meas
  pf     := row.pf_meas
  P      := row.P_meas

  V_ref  := row.V_ref
  I_ref  := row.I_ref
  pf_ref := row.pf_ref
  pf_min := row.pf_min
  tol_rP := row.tol_rP

  # contrasts
  e_V    := ln( max(V,        eps_pos) / V_ref )
  e_I    := ln( max(abs(I),   eps_pos) / I_ref )
  pf_eff := max(pf, pf_min)
  e_pf   := ln( pf_eff / pf_ref )

  P_ref  := V_ref * I_ref * pf_ref
  e_P    := ln( max(abs(P),   eps_pos) / P_ref )

  r_P    := e_P - (e_V + e_I + e_pf)

  status := (abs(r_P) <= tol_rP) ? "PASS" : "ALARM"

  assert approx_equal(r_P, row.expected_rP, tol=1e-3)
  assert status == row.expected_status
```

If all rows pass these assertions, the residual implementation is certified against the canonical test set.

Deployment Notes (Copy-Ready)

- **Per-site tuning.** Once this pack passes, a site may tighten or relax `tol_rP` (for example `0.02` vs `0.05`), but the qualitative patterns of C1–C5 should remain unchanged.
- **Profile cloning.** Operators can clone C1–C5 with their own anchors (for example `V_ref = 11kV`, `I_ref` per feeder) to create local residual packs, as long as they preserve the same formula and classification rules.
- **Interop expectation.** Any tool that claims SSMEQ compatibility for residuals should be able to ingest this residual test CSV and reproduce `r_P` and `status` exactly, without custom logic per platform.

Interpretation.
If the Residual Test Pack passes, `r_P` is trustworthy as a power-identity check. If it fails, no higher-level analytics can restore symbolic integrity, and the implementation must not be considered conformant until corrected.

14.0 — Annex (Concise, High-Value Reference Section)

The Annex consolidates the symbolic essentials of SSMEQ into compact, ready-to-use references.
It exists as a precision toolkit — a final set of verification notes, safety clauses, and quick-check formulas for practitioners, auditors, or educators who need the **distilled essence** of symbolic mathematics without rereading the full specification.

Each sub-section reaffirms the **ethical, mathematical, and operational backbone** of SSMEQ.
These are the small but foundational rules that guarantee the system's truth, reproducibility, and cross-domain portability.

Equality and Greater-Than Scores

Symbolic comparison replaces binary logic with **bounded contrast**.

Instead of “equal or not equal,” SSMEQ expresses how aligned two quantities are within a common reference frame:

seq:=1−|a1−a2|s_\text{eq} := 1 - \lvert a_1 - a_2 \rvertseq:=1−|a1−a2|
sgt:=tanh (k·(a1−a2))s_\text{gt} := \tanh\big(k \cdot (a_1 - a_2) \big)sgt:=tanh(k·(a1−a2))

where:

- `s_eq ∈ [0, 1]` indicates **degree of equality** (1 = identical).
- `s_gt ∈ (−1, +1)` indicates **directional dominance**:
  positive → $a_1 > a_2$, negative → $a_1 < a_2$.
- `k` is a sensitivity constant (typical `k ≈ 3`).

These symbolic comparators allow systems to evaluate **closeness** or **trend direction** while retaining boundedness and determinism.

Example

- `a1 = +0.72, a2 = +0.69`
- `s_eq = 1 − |0.72 − 0.69| = 0.97`
- `s_gt = tanh( 3 * (0.72 − 0.69) ) ≈ +0.09`

Interpretation: values are ~97% aligned, with a small but positive upward drift.

Division Policy Note (strict | meadow | soft)

Division in electrical metrics (e.g., power / voltage) must never yield infinities or undefined results.
SSMEQ enforces explicit division behavior via manifest declaration:

```
division_policy ∈ { "strict", "meadow", "soft" }
```

- **strict** → classical math; zero-denominator raises an error.
  Used for controlled analytics and environments where invalid inputs must halt the pipeline.
- **meadow** → safe algebraic inversion with total closure:
  `1 / 0 := 0`.
  Useful in symbolic reasoning, proofs, or constrained devices where exceptions are undesirable.
- **soft** → bounded asymptotic response:

  div_soft(x,y):=xy2+εy2\text{div_soft}(x, y) := \frac{x}{\sqrt{y^2 + \varepsilon_y^2}}div_soft(x,y):=y2+εy2x

  with `eps_y > 0`. This ensures graceful degradation near zero without explosion.

Each site chooses one mode via **policy lock**.
Once declared, the rule cannot change without **re-commissioning**, preserving reproducibility and safety.

Zero-Class and Identity Crib

The **Zero-Class** defines invariant behaviors fundamental to symbolic stability.

Rule	Expression	Interpretation
Collapse parity	`phi((m,a)) = m`	Classical magnitude is never altered.
Neutrality	all `a = 0 → a_pool = 0`	Perfect symmetry yields perfect stability.
Clamp saturation	`a ∈ (−1, +1)`	Prevents runaway alignments.
Order-invariance	`pool(A) = pool(shuffle(A))`	Truth independent of data order.
Continuity	`d(a_pool)/d(a_i)` finite ∀ a_i	Smooth behavior; no sudden jumps or cliffs.

These rules form the **symbolic DNA** of every SSMEQ computation.
Any valid implementation must satisfy them under all conformance tests.

Glossary (SSM Family Quick Reference)

Symbol / Term	Definition
`m`	Classical magnitude — the original, unaltered value.
`a`	Alignment lane in (−1, +1); symbolic stability indicator.
`phi((m,a)) = m`	**Collapse parity**: guarantees that observation never modifies classical reality.
U/W Kernel	Hyperbolic accumulator pair (`U`, `W`) defining deterministic pooling: `U += w*atanh(a_c), W += w, a_pool := tanh(U / max(W, eps_w))`.
Lens	Mathematical mapping from physical measurement → symbolic state `a`.
Band	Discrete stability zone (A^+…D) derived from `a` with hysteresis and holds.
Stamp	SHA-256 hash chaining payload and previous stamp to encode event lineage and policy identity.
Manifest	Immutable policy file defining lenses, anchors, epsilons, weights, bands, and timing.

Symbol / Term	Definition
Repro Pack	Minimal bundle (CSV + JSON + harness) enabling independent, bit-for-bit verification.
Data Posture	State declaration such as `DATA_VALID`, `DATA_INVALID`, `DATA_ABSENT`, `QUORUM_LOST`, `DATA_STALE`.
Harness	Deterministic verifier script that confirms order-invariance and policy-lock fidelity.

Closing Reflection

The Annex stands as the **philosophical mirror** of the framework: clarity distilled into precision.

These compact rules, equations, and invariants allow any engineer or auditor to confirm that an SSMEQ implementation is **truthful, complete, and safe** — without proprietary dependency or interpretive drift.

When these principles hold, electrical measurement becomes **symbolic integrity** itself — measurable, reproducible, and ethically transparent.

14.1 — Symbol & Parameter Table (Electrical, Copy-Ready)

This subsection is the single quick-reference crib for all core symbols and parameters used in SSMEQ electrical deployments.
All names are ASCII and must be used exactly as written for interoperability.

Core Quantities (Classical Space)

- **m**
  Classical magnitude (e.g., RMS voltage, RMS current, active power).
  Type: real. Units: per channel (V, A, W, kW, etc.).
- **V_rms, I_rms, P, Q, S, f, T**
  Raw classical measurements as reported by meters or acquisition systems.
  Type: real. Units: volts, amps, watts, vars, VA, Hz, °C (or °F) as declared.
- **V_ref, I_ref, P_ref, f_ref, T_ref, T_max**
  Anchor values used by lenses; typically nominal or contractual targets.
  Type: real. Must be published in the manifest.

Contrasts (e_*) and Lenses

- **e_V**
  Zero-centric contrast for voltage, e.g. `e_V := e_LOG(V_rms, V_ref).`
- **e_I**
  Zero-centric contrast for current, e.g. `e_I := e_LOG(I_rms, I_ref).`
- **e_f**
  Frequency contrast, e.g. `e_f := e_LOG(f, f_ref)` or equivalent lens.
- **e_P, e_Q, e_S**
  Power-space contrasts, e.g. `e_P := e_LOG(P_meas, P_ref)`; used in `r_P`.
- **e_pf**
  Power factor contrast, e.g. `e_pf := ln( pf_eff / pf_ref )` with `pf_eff := max(pf_raw, pf_min).`
- **e_T**
  Thermal contrast, either linearized or log-based headroom contrast (see thermal lens).
- **e_LOG(x, x_ref)**
  Generic zero-centric log contrast, typically:
  `e_LOG(x, x_ref) := ln( max(x, max(eps_pos, alpha*x_ref)) / x_ref ).`

Alignment Lanes (a_*)

All alignments obey `a ∈ (-1,+1)` and must be clamp-safe.

- **a_V, a_I, a_f**
  Symbolic alignments for voltage, current, and frequency.
  General pattern: `a_X_raw := tanh( c_X * e_X )`, `a_X := clamp(a_X_raw, -1+eps_a, +1-eps_a).`
- **a_PF**
  Power factor alignment lane derived from `e_pf`.
- **a_THDV, a_THDI**
  Harmonic distortion alignments for voltage and current.
- **a_unbV, a_unbI**
  Voltage and current unbalance alignments.
- **a_T**
  Thermal alignment (cabinet, busbar, transformer, breaker, etc.).
- **a_stress**
  Composite electrical stress dial (fused from selected a_* dials).
- **a_pool**
  Pooled / aggregate alignment from multiple dials or assets using the rapidity kernel.

Rapidity, Pooling, and Kernel Parameters

- **a_c**
  Clamped alignment: `a_c := clamp(a_raw, -1+eps_a, +1-eps_a).`
- **u**
  Rapidity of one alignment value: `u := atanh(a_c).`
- **U, W**
  Rapidity accumulators:
  `U += w * u`
  `W += w`
- **a_out, a_pool**
  Output / pooled alignment: `a_out := tanh( U / max(W, eps_w) ).`
- **w, w_j, w_declared**
  Weights applied in pooling; can be uniform, scale-based, or declared.
- **eps_a**
  Clamp guard for alignment, typically `eps_a ≈ 1e-6`.
- **eps_w**
  Guard for denominator in pooling, typically `eps_w ≈ 1e-9` or tighter.
- **gamma**
  Exponent for magnitude-based weighting (e.g., `w := |m|^gamma`).

Residuals and Diagnostic Symbols

- **r_P**
  Power-identity residual in contrast space (see Section 4):
  `r_P := e_P - ( e_V + e_I + e_pf )` (canonical form).
- **tol_rP**
  Residual tolerance; typical small positive value (e.g., `0.05`).
  Used to flag `PASS` vs `ALARM` for r_P.
- **r_T, r_other**
  Optional residuals for thermal or custom identities; declared per deployment.

Bands, Gates, and Validity

- **band**
  Stability band label, e.g. `"A+"`, `"A"`, `"B"`, `"C"`, `"D"`.
- **g_t**
  Environment gate at time t; `g_t ∈ [0,1]`.
  Decision surrogate: `a_env := g_t * a_x`.
- **status**
  Data posture label, e.g. `"OK"`, `"DATA_INVALID"`, `"DATA_ABSENT"`, `"QUORUM_LOST"`, `"DATA_STALE"`.
- **N_valid, N_total**
  Counts of valid vs total contributing dials at a given tick.
- **quorum_threshold, N_min_dials**
  Quorum parameters that define when pooled values are allowed to drive actions.

Time, Stamps, and Identifiers

- **tick_ms**
  Core tick duration in milliseconds for dials and pooling.
- **time_utc, timestamp_utc**
  Canonical time representations in UTC (ISO 8601).
- **stamp**
  Hash-chain element, typically:
  `stamp := sha256(prev || sha256(payload_bytes) || time_utc).`
- **manifest_id**
  Identifier for the policy and lens bundle in force.
- **dial_id, pool_id, site_id, asset_id**
  Scope identifiers for dials, pools, and sites.

Policy and Governance Parameters

- **pf_ref**
  Declared power factor target (often `1.0`), per site or asset profile.
- **pf_min, pf_floor**
  Minimum effective power factor to avoid log singularities (e.g., `0.05`).
- **division_policy**
  Division semantics: `"strict"`, `"meadow"`, or `"soft"`; defined in the manifest.
- **validity bounds**
  Symbols such as `V_min_valid`, `V_max_valid`, `f_min_valid`, `f_max_valid`, `stale_after_ms`, `max_skew_ms`, `max_jitter_ms`.
- **bands.json keys**
  For each band label, fields like `enter`, `clear`, `hold`.

This symbol set is the canonical vocabulary of SSMEQ.
Any implementation claiming conformance must use these names and meanings without silent reinterpretation.

14.2 — Startup Defaults & Cold-Start Rules (Copy-Ready)

This subsection defines how SSMEQ behaves at first boot, after reset, or when a new site profile goes live.
The goal is to avoid undefined states, hidden warm-up tricks, or silent “magic numbers.”
Every cold-start rule must be explicit and reproducible.

Cold-Start State for Memory and Pooling

- **Memory dial M**
  - Default: `M_prev := 0`.
  - Interpretation: before any evidence, the symbolic memory is neutral.
  - Rule: do not fabricate history; do not seed M_prev from guessed values.
- **Pooling kernel accumulators**
  - Default: `U := 0`, `W := 0`, `a_pool := 0`.
  - Until sufficient valid inputs arrive, pooled outputs must either remain `0` or be flagged as `"DATA_ABSENT"` according to policy.
- **Band and timers**
  - Default band for each dial: `"A"` or `"A/A0"`-equivalent neutral band as declared in bands.json.
  - `time_in_band := 0` at startup.
  - No retroactive band history may be assumed.

Default Exponents and Sensitivity Parameters

When not explicitly declared in the manifest, the following guidance applies:

- **rho (contrast exponent)**
  - Default: `rho := 1.0`.
  - Rule: any departure from `rho = 1.0` must be explicitly published per lens.
- **c_V, c_I, c_THDV, c_unbV, c_PF, c_T**
  - Default: domain-typical constants as in the lens examples, or site-specific values in the manifest.
  - If unspecified, implementations should not invent values; instead, they must treat the lens as undefined and raise a configuration error.
- **eps_pos, eps_div, eps_a, eps_w**
  - Typical defaults:
    - `eps_pos := 1e-12`
    - `eps_div := 1e-12`
    - `eps_a := 1e-6`
    - `eps_w := 1e-9`
  - Rule: these must be visible in the manifest or documented as global library defaults.

Anchors and pf_ref When Unknown

If explicit anchors are not available at configuration time, SSMEQ follows a safe, governance-first path:

- **Voltage and current anchors**
  - Default:
    - `V_ref :=` nominal system voltage (e.g., 230 V, 400 V, 480 V) chosen from site design documents, not inferred from measurements.

        - `I_ref :=` nominal feeder or asset current (e.g., nameplate or engineering design value).
    - Rule: anchors must not be derived from a short measurement window without governance approval.
- **Power factor reference**
    - If unknown, recommend:
        - `pf_ref := 1.0` for most industrial and commercial sites.
    - If contractual target is lower (e.g., 0.9), then:
        - `pf_ref := contract_target` and `pf_min := 0.05` (or site-declared floor).
- **Thermal anchors**
    - Default guidance when no explicit study exists:
        - `T_ref :=` typical expected operating temperature under normal load.
        - `T_max :=` nameplate or safe design limit.
    - These must be published in the site profile and not tuned by trial and error in production.

Any later change to **V_ref**, **I_ref**, **pf_ref**, or **T_ref/T_max** constitutes an anchor change and must follow the anchor governance rules (version bump, re-commissioning).

Environment Gate and Bands at Startup

- **Environment gate g_t**
    - Default at cold start: `g_0 := 1.0` (no gating).
    - Rule: environment dampers (weather, grid emergency, maintenance) are introduced only after explicit policy configuration; they are not auto-guessed.
- **Band hysteresis**
    - At startup, use `clear` thresholds when initializing band state:
        - Example: initialize as if `a = 0` and apply the standard band selection logic.
    - No preloaded "A+" state is assumed unless justified by commissioning data.

Cold-Start Validity and Posture

- Before first valid tick:
    - Dials must report `status := "DATA_ABSENT"` or remain silent as per policy.
    - No tickets, alerts, or actions are permitted until validity and quorum conditions are satisfied at least once.
- After first valid tick:
    - All future states must be derived from actual observed data and published manifest parameters.
    - If streams become invalid or stale, the system must revert to `"DATA_INVALID"` or `"DATA_ABSENT"` with clear reasons, not to arbitrary defaults.

Copy-Ready Startup Pseudo-Code

```
# Cold-start
M_prev   := 0
U        := 0
W        := 0
a_pool   := 0
band     := band_from_a(0)        # neutral band using clear thresholds
time_in_band := 0
g        := 1.0                   # environment gate

# Each tick T
read raw measurements
apply validity windows
if no valid inputs:
    status := "DATA_ABSENT"
    a_pool := 0
    # no dispatch
else:
    compute e_* contrasts
    compute a_* alignments
    update U, W, a_pool as per rapidity kernel
    update band and time_in_band via hysteresis rules
    emit envelopes with current status and stamp
```

Why These Rules Matter

Startup defaults and cold-start rules ensure that SSMEQ never “cheats” on the first few minutes of data.
There are no hidden warm-up hacks, no invisible tuning, and no silent rewriting of anchors. From the very first tick, symbolic arithmetic is governed by the same manifest-locked rules that will apply for the lifetime of the deployment — making every commissioning, audit, and reproduction exercise straightforward and trustworthy.

15.0 — Unified Electrical Language for a Shared Grid

Electricity is the heartbeat of civilization.
It decides when lights stay on in a hospital, when a motor spins in a factory, when a vaccine freezer holds its line, when a satellite re-orients, when a server rack hums within limits, when a transmission line bends toward collapse, and when the grid holds — or fails.

Yet electrical truth remains fragmented: volts versus amps, frequency drift measured one way by one utility and another by the next, hidden distortions buried in vendor dashboards, “healthy” readings that differ by firmware, and thresholds that change without audit. The result is confusion, cost, downtime, and in many cases — preventable harm.

The goal of this work is to end that fragmentation.

This specification defines a single **symbolic electrical layer** that can travel anywhere — across instruments, substations, fleets, and borders — without changing physics and without hiding responsibility.

The core signals are intentionally simple, and intentionally auditable

- **Contrast dials.**
  Each physical measure (voltage, current, frequency, power factor, unbalance, distortion) is expressed as a zero-centric, unitless contrast such as
  `e_V := ln(V / V_ref)` or `e_I := ln(I / I_ref)`.
  The meaning of "how far from nominal" becomes identical from Mumbai to Munich, from a field cabinet to a spacecraft bus.
- **Alignment dials.**
  Each contrast gains a bounded safety channel in (−1,+1):
  `a_THDV`, `a_THDI`, `a_unbV`, `a_unbI`, `a_df`, `a_dV`, `a_PF`, and `a_T`.
  Negative means "drifting toward under-stress," positive means "toward overload or distortion," near zero means "within declared stability."
- **Fusion.**
  Multiple dials can be merged through the rapidity accumulator
  `a_pool := tanh(Σ w · atanh(a) / Σ w)`,
  yielding one bounded posture of electrical health for machines, feeders, or fleets.

Why this matters

For **operators and regulators** — you can now write safety or quality policy once, symbolically, and enforce it everywhere:
"Escalate if a_pool ≥ +0.75 for > 30 seconds."
That rule is valid in any plant, any nation, any voltage class.

For **procurement and maintenance** — you are no longer trapped behind opaque vendor indices. Require that every device emit `e_V`, `a_THDV`, `a_PF`, and manifest ID. Ask for the lens, anchors, and thresholds that define them. Compliance becomes measurable, portable, and insurable.

For **grids and microgrids** — voltage sag, harmonic drift, unbalance, flicker, and thermal stress stop being five separate problems. They become one language of survivability. Dispatch logic, AI prediction, and human oversight can operate on the same bounded scale, with no translation fights.

For **research and AI safety** — symbolic dials are bounded, unitless, and ethically auditable. They let models learn stress without inheriting hidden bias from proprietary telemetry.

For **finance and continuity** — electrical stability becomes a measurable exposure. When deviation dials are standardized, they become auditable; when auditable, they become contract-grade; when contract-grade, they become insurable.

For humanity

Blackouts. Fires from unnoticed harmonics. Medical equipment failures. Battery explosions. None of these are abstractions; they happen daily.

A universal, open, verifiable **electrical language** is not just an engineering upgrade — it is a **public-safety intervention**.
And it is **open.**
There is no fee to emit `e_V`.
There is no gatekeeper to compute `a_THDV`.
There is no proprietary lock on declaring `V_ref`, `I_ref`, `f_ref`, or `PF_ref`.
Any qualified team — utility, manufacturer, lab, campus, spacecraft, or household — can adopt it now.

Electrical instability is not local.
Grid fatigue is not local.
Energy poverty is not local.
Survivability must be shared.

This specification exists to make that possible — without rewriting physics, without obscuring accountability, and without silencing the truth in the numbers.

OMP