SSM — Jyotish Transit Kernel (SSM-JTK)

Ephemeris-Independent Transit Kernel

Status: Public Research Release (v2.1)
Date: 16 October 2025
Caution: Research/observation only. Not for critical decision-making.

What if rāśi-based transits for any date—centuries back and thousands of years ahead—could be computed on-device from a tiny, audit-ready kernel with no runtime ephemeris?

- **Offline kernel?** Can daily longitudes, crossings, cusps, and stations be computed entirely offline from a small per-planet manifest—no internet, no ephemeris?
- **Long-range & auditability?** Can the same kernel project backward/forward while remaining plain-ASCII auditable (formulas, manifests, checks), yielding bit-for-bit reproducible results?
- **Public verification & calculator parity?** Can anyone verify today using a single Golden day-grid and a one-screen validator, while advanced users later switch to a manifest-based calculator with identical semantics (including node identity `Ketu(t) := wrap360(Rahu(t) + 180)`)?

---

00) Primer — Shunyaya Symbolic Mathematics (SSM)

Idea. Promote a scalar into a two-channel object: a headline value and an alignment that reports how solid that value is.
`x = (m, a)` where `m ∈ R`, `a ∈ (-1,+1)`
`phi(m, a) = m` (collapse recovers the classical scalar)

Reading a. `a → +1` suggests stability; `a → -1` suggests drift.

Rapidity space (safe numerics).
```
eps_a = 1e-6
clamp_a(z, e) = max(-1+e, min(+1-e, z))
u = atanh( clamp_a(a, eps_a) ) → a = tanh(u)
```

Combining alignments (policies).
M1 (simple): `a' = clamp_a(a1 * a2, eps_a)`
M2 (rapidity; default for mul/div):
```
a' = tanh( atanh(clamp_a(a1, eps_a)) + atanh(clamp_a(a2, eps_a)) )
```

Addition (streaming, n-ary).
`eps_w = 1e-12`, `w(m) = |m|^gamma` (default `gamma = 1`)

```
U = Σ_i [ w_i * atanh(clamp_a(a_i, eps_a)) ]
W = Σ_i w_i
m_out = Σ_i m_i
a_out = tanh( U / max(W, eps_w) )
```

Multiplication & division (defaults).
`u1 = atanh(clamp_a(a1, eps_a))`, `u2 = atanh(clamp_a(a2, eps_a))`

```
(m1,a1) * (m2,a2) = ( m1*m2 , tanh(u1 + u2) )
(m1,a1) / (m2,a2) = ( m1/m2 , tanh(u1 - u2) )
```

(with `m2 ≠ 0`)

Conservative extension (guarantees).
`phi((m,a)) = m` ; `phi(x + y) = phi(x) + phi(y)` ; `phi(x * y) = phi(x) * phi(y)`
(Edge cases handled as limits in `u`-space; holds for `|a| ≤ 1 - eps_a`.)

Conventions.
`id_add = (0, +1)` ; `id_mul = (1, 0)` ; ASCII math only (`log, exp, tanh, atanh`) ; addition uses the rapidity mean; mul/div use M2.

Why this matters for the transit kernel.
We compute angles/events with tiny state and no runtime ephemeris. SSM lets us carry an alignment alongside each number, so we can reason about stability without breaking classical arithmetic when we collapse via `phi`.

---

0) Overview (why this document exists)

This document defines the **Shunyaya Symbolic Mathematics — Jyotish Transit Kernel (SSM-JTK)**: a deterministic, ephemeris-independent way to evaluate rāśi-based planetary transits using SSM's two-channel numbers. Goal: **on-device** forward/backward projections from a tiny, audit-ready manifest per planet—**no runtime ephemeris**. The public reference is a **Golden** day-grid CSV (daily 1990–2030) verified with strict rāśi and node-identity checks and a one-screen public validator.

Architecture at a glance.

- **Two kernel families.**
  Fixed-n: bodies that admit a stable mean motion use `n = 360 / P_sid`, optionally with small harmonics.
  Free-n: bodies fit once to a slope `n` plus the same small harmonic scaffold (design-ready; optional).
- **One manifest per planet** holds the parameters; the evaluator is a few lines.
- **Output per day** is an SSM number `x = (m, a)` with `phi((m,a)) = m`.

Alignment (recap; rapidity-safe).
`eps_a = 1e-6`, `eps_w = 1e-12`, `clamp_a(z) = max(-1+eps_a, min(+1-eps_a, z))`
`u = atanh(clamp_a(a))` ; pooled alignment `a_total = tanh( (Σ w_i*atanh(clamp_a(a_i))) / max(Σ w_i, eps_w) )`

Binary ops: `a_mul = tanh(atanh(clamp_a(a1)) + atanh(clamp_a(a2)))`; `a_div = tanh(atanh(clamp_a(af)) - atanh(clamp_a(ag)))`

Ephemeris-independent angle evaluator (sidereal, Lahiri).

```
wrap360(x) = x - 360*floor(x/360)
L(t) = wrap360( a0 + n*t + Σ_j Σ_{k=1..H_j} [ c_{j,k}*sin(k*w_j*t) + d_{j,k}*cos(k*w_j*t) ] )
```

Carriers: `n = 360 / P_sid` (fixed for some bodies; learned once for others), `w_j = 2*pi / P_j`.
From tropical to sidereal (CSV math): `lon_sidereal_lahiri_deg = wrap360(lon_tropical_deg - ayanamsa_deg)` with a linearized ayanāṁśa for CSV math; rāśi index `rasi = floor(wrap360(L_hat_deg)/30)`.

Nodes relation. `Ketu(t) = wrap360(Rahu(t) + 180)`.

Minimal manifest schema (per planet, concept).

```
{
  "name": "jupiter",
  "t0": "2100-01-01",
  "a0_deg": 123.456789,
  "n_deg_per_day": 0.083056,
  "harmonics": [
    { "P_days": 398.88, "H": 2, "c": [c1, c2], "d": [d1, d2] },
    { "P_days": 365.25, "H": 1, "c": [c1],     "d": [d1] }
  ]
}
```

Determinism rules.

- Day grid is integer-spaced from `t0`.
- Apply **one** `wrap360` at the end of `L(t)`.
- No hidden state, no network calls; outputs are pure functions of `(manifest, t)`.
- Collapsing with `phi` recovers classical scalars; alignment composes via M1/M2 policies.

Tiny reference pseudocode (angle evaluator).

```
PI = 3.141592653589793
wrap360(x) = x - 360.0 * floor(x/360.0)
L = wrap360( a0_deg + n_deg_per_day * t + Σ_k [ c_k*sin(k*w*t) + d_k*cos(k*w*t) ] )
```

(For the linear base, set the harmonic lists empty; rāśi is `floor(L/30)`.)

0A) BENEFITS (why this approach is useful)

- **Ephemeris-independent runtime.** After one calibration, projections use a tiny manifest; no external ephemeris thereafter.
- **Deterministic & auditable.** One manifest reproduces angles; bit-for-bit outputs from fixed formulas and frozen coefficients.

- **Event-aware accuracy.** Targets what practitioners use: rāśi crossings, cusp distance, station flips.
- **Parsimony by construction.** Extra terms only when justified (design policy); prevents overfit.
- **Time-reversible projections.** Same kernel and anchor handle past/future.
- **Offline, low-latency.** Per-date cost is `O(#terms)` trigs (or `O(1)` in the linear base).
- **Interop with SSM.** Outputs drop into higher-layer SSM scoring with alignment.
- **Safe-ops policy.** If a query falls outside acceptance envelopes, escalate to a high-precision ephemeris for that narrow window.

Validation & Acceptance

- **Golden day-grid reference (daily 1990–2030).** Verified with strict rāśi integrity and node-identity checks: `rasi = floor(wrap360(L_hat_deg)/30)` and `Ketu = wrap360(Rahu + 180)` with expected `PASS`.
- **Public validator (one-screen).** Checks column presence, angle ranges, per-planet date continuity, rāśi parity, and node identity; outputs `PASS/FAIL` with mismatch counts.
- **Monthly benches (1800–2199 and 0001–9500).** Internal self-consistency benches against the monthly reference show `p90=0` and `max=0` at monthly cadence.

Summary

In short, v2.1 delivers a self-contained, ephemeris-free kernel: per-body JSON manifests plus a one-line evaluator `L_hat_deg := wrap360(a0 + n*t + SUM_k( c_k*sin(omega_k*t) + d_k*cos(omega_k*t) ))` with `t := days_since(D, t0)` at `05:30 IST`. Daily outputs map to rasi via `rasi := floor((L_hat_deg % 360)/30)` and enforce node identity `Ketu(t) := wrap360(Rahu(t) + 180)`.

This bundle ships `golden_all_v2_1.csv` (daily 1990–2030), manifests, a pure-stdlib validator, and the acceptance log; all checks PASS at `tol = 1e-5` after `wrap360`. For deep time, the same kernel supports monthly benches across years `0001–9500` (kept as a sampler outside this minimal bundle). Integrity is embedded by publishing the SHA-256 of `golden_all_v2_1.csv` in-doc. Everything is offline, deterministic, and plain-ASCII auditable; SSM's alignment channel travels with numbers for stability readouts while `phi(m,a) = m` preserves classical arithmetic.

TABLE OF CONTENTS

---

1) Concept & Guarantees (bounded, composable, ephemeris-independent)

1.1 Two-channel object (headline + alignment)

We separate *what you report* from *how well it holds*.
`x := (m, a)`
`phi((m, a)) = m` (collapse; alignment never distorts `m`)
`a ∈ (-1, +1)` (bounded alignment; quality/clarity)
(Note: `m` is an ordinary scalar. For displayed longitudes we render angles with `wrap360` so they fall in `[0, 360)`.)

Guarantees (channeling).

- **Separation:** `m` is always the reported value; `a` is metadata about confidence/clarity.
- **Non-interference:** Arithmetic on `a` cannot change `m` (collapse commutes with the arithmetic below).

1.2 Bounded arithmetic via rapidity (no blow-ups)

All alignment composition happens in rapidity (atanh/tanh) space; pooled sums stay bounded and stable.

```
eps_a = 1e-6
eps_w = 1e-12
clamp_a(z) = max(-1+eps_a, min(+1-eps_a, z))
```

Single value → rapidity.

```
u = atanh( clamp_a(a) )
```

Pooling (weighted mean in rapidity space), with 0 ≤ w_i ≤ 1.

```
U = Σ_i [ w_i * atanh( clamp_a(a_i) ) ]
W = Σ_i w_i
a_sum = tanh( U / max(W, eps_w) )
```

Binary ops (M2 rapidity combine).

```
a_mul = tanh( atanh(clamp_a(a1)) + atanh(clamp_a(a2)) )
a_div = tanh( atanh(clamp_a(a_f)) - atanh(clamp_a(a_g)) )
```

Guarantees (alignment math).

- **Boundedness:** if each input `a_i ∈ (-1, +1)`, outputs remain in `(-1, +1)`.
- **Antisymmetry:** `a_div(f, g) = -a_div(g, f)`.
- **Composable pooling:** associative as a weighted mean in rapidity space (stable for many sources).

1.3 Ephemeris-independent planetary kernel

Angles are generated from a tiny, interpretable model — no external ephemeris calls after calibration.

Two-family kernel (same evaluator).

```
t0 = midpoint of the calibration strip
t = days since t0
wrap360(x) = x - 360*floor(x/360)
```

(A) fixed-n kernel (e.g., Moon, outer planets).

```
L(t) = wrap360( a0 + n*t + Σ_j [ c_j*sin(w_j*t) + d_j*cos(w_j*t) ] )
n = 360 / P_sid
```

`w_j = 2*pi / P_j` (small set: synodic, annual, slow sidereal)

(B) free-n kernel (e.g., Sun, Mercury, Venus, Mars).

```
L(t) = wrap360( a0 + n*t + Σ_j [ c_j*sin(w_j*t) + d_j*cos(w_j*t) ] )
```

`n = n_deg_per_day` (a learned slope stored in the manifest)

Nodes (identity).

```
Ketu(t) = wrap360( Rahu(t) + 180 )
```

Event primitives (detectors at evaluation time).
`dL(t) = wrap360( L(t) - L(t-1) + 180 ) - 180` (shortest-arc step in `[-180, 180)`)
`cusp@t` iff `floor(wrap360(L(t-1))/30) != floor(wrap360(L(t))/30)`
`station@t` when `sign(dL(t))` flips with `|dL(t)|` near zero

Kernel guarantees.

- **Determinism:** pure function of `(manifest, t)`; no hidden state or network.
- **One-wrap rule:** apply `wrap360` exactly once at the end of the angle chain.
- **Prev-day convention:** ingress is recorded on the previous day after year-boundary normalization.
- **Bounded snap:** when aligning event lists, a ±1-day snapper may be used **only** to resolve a clear neighbor mismatch (never beyond ±1).
- **Time-reversible:** same formula projects past and future.

1.4 Conservative fit & model selection (parsimony first)

We admit new terms only if they earn their place.

Training data.
```
calendar grid: daily or sparse (e.g., D1/D15), fixed time base
unwrap rule: carry +360 so L_unwrapped(t) is continuous
```

OLS targets/design.
Fixed-n (Moon/outer):
`y = L_unwrapped - n*t`; `X = [1, sin(w*t), cos(w*t), ...]`
Free-n (Sun/inner):
`y = L_unwrapped`; `X = [1, t, sin(w*t), cos(w*t), ...]`

Model score (BIC).
```
BIC = k*log(N) + N*log( max(RSS/N, 1e-16) )
```

Event-aware loss (degree still dominant).
```
loss = 1.0*MAE_deg + 0.3*cusp_MAE + 0.4*MAE_speed
```

Gate (strict).
`admit` iff `(BIC_extra ≤ BIC_base - 6)` **and** `(loss_extra < loss_base)`

Selection guarantees.

- **Parsimony:** base wins unless an extra is decisively better (`ΔBIC ≥ 6`).
- **Event fidelity:** crossings & stations influence selection, not just degree MAE.
- **Midpoint anchor:** reduces phase coupling and edge bias.

1.5 Reproducibility & audit trail (end-to-end)

Every printed line is re-generatable from the manifest + inputs.

Manifest (per planet, minimal concept).

```
{ "t0": "YYYY-MM-DD", "a0_deg": ..., "n_deg_per_day": ..., "harmonics": [ {
"P_days": ..., "H": ..., "c": [...], "d": [...] }, ... ] }
```

Evaluate (runtime; both families).

```
t = days_since(date, t0)
y = a0 + n_deg_per_day*t + Σ_k [ c_k*sin(w_k*t) + d_k*cos(w_k*t) ]
L_hat = wrap360(y)
```

Optional alignment band for UI (does not change `L`).

`clamp01(z) = max(0, min(1, z))`
`v(t) = |dL/dt|` ; `n_ref = n_deg_per_day` (or `360/P_sid` for fixed-n)
`s_stat = clamp01( 1 - v/n_ref )`
`d_cusp = min_k | wrap360(L(t) - 30*k) |`
`d0 = 2.0` (deg, typical) ; `s_cusp = exp( - (d_cusp/d0)^2 )`
`a(t) = tanh( u0 - alpha*( w1*s_stat + w2*s_cusp ) )` *(illustrative; alignment is metadata)*

1.6 Safety guards (always-on)

- **Clamps:** apply `a := clamp_a(a)` before any `atanh`.
- **Wrap discipline:** unwrap for fits/events; wrap only for display (`wrap360` once at the end).
- **Caps:** auxiliary rapidities are range-checked; no unbounded deltas.
- **Bounded snapping:** event alignment uses at most ±1 day when resolving clear neighbor mismatches.
- **Flags over force:** prefer labeled states (e.g., SIDED/OSC/MULTI/NOFIT) over brittle magic thresholds.

1.7 What we claim / do not claim

Claim: stable, bounded, auditable transits; rāśi/house robust; degree-level error competitive on back-tests with tiny data; ephemeris-independent runtime.

Not a claim: minute-accurate ingresses or full N-body fidelity. For arc-minute timing, use a high-precision ephemeris in that narrow window only.

---

2) Data & Calibration (inputs → fit → manifest)

2.1 Sources (license-clean options)

- **Public ephemeris (verification only).** Geocentric daily longitudes from a reputable public service, used strictly for back-tests and benchmarking.
- **Internal reference tables (bootstrapping).** Compact "traditional-method" tables you export (e.g., D1/D15 or D1/D8/D15/D22) to seed the kernel when you prefer not to depend on external ephemerides later.
- **On-device runtime.** After calibration, evaluation uses only the JSON-style manifest; no external calls or data.

Hybrid policy. Measure with a public ephemeris if needed, derive a tiny kernel from your internal tables, then run ephemeris-free at inference.

2.2 Time & frame (fixed choices)

- **Timescale:** UTC (daily timestamp at `00:00 UTC`, consistent).
  *(If you choose a local timescale such as `05:30 IST`, use it consistently for **all** training and evaluation.)*
- **Reference:** geocentric, apparent, sidereal (Lahiri / Chitrapaksha).
- **Angles:** degrees in `[0,360)`; wrap with `wrap360(x)`.
- **Rāśi:** `rasi(x) = floor( wrap360(x) / 30 )` (range `0..11`).
- **Cusp dating convention:** record ingresses on the **previous day** after year-boundary normalization.

2.3 Pre-processing (always do this)

1. **Siderealization (if your source is tropical).**
```
lon_sidereal_lahiri_deg = wrap360( lon_tropical_deg - ayanamsa_deg )
ayanamsa_deg = A0_deg + rate_deg_per_year * years_since_2000
A0_deg = 23.857625
rate_deg_per_year = 50.290966 / 3600
years_since_2000 = ( JD(date) - JD(2000-01-01) ) / 365.2425
wrap360(x) = ((x % 360) + 360) % 360
```
2. **Unwrapping for fits/events (continuous series `y`).**
   `y[0] = L_sid[0]`
   For `i > 0`, choose integer `k` minimizing `|(L_sid[i] + 360*k) - y[i-1]|`, then set `y[i] = L_sid[i] + 360*k`.
   Use `y` (unwrapped) for OLS and event detection.

3. **Derivatives (central differences; Δt in days).**
   `v[i] ≈ ( y[i+1] - y[i-1] ) / (2*Δt)` (deg/day)
   `a2[i] ≈ ( y[i+1] - 2*y[i] + y[i-1] ) / (Δt^2)`

2.4 Sampling modes (declare once)

Named schedules (deterministic).
`daily` : every day at the chosen daily timestamp
`D1D15` : day 1 and day 15 each month (compact for slow bodies)
`D1D8D15D22` : four anchors per month (helpful for inners/retro loops)
`weekly` : every 7 days from the mid-anchor `t0`
`events` : a base schedule plus auto-anchors near stations/cusps

Event augmentation (optional; applies when `events` is selected).
If `|v_deg_per_day(t)| ≤ v_thresh` then add `t' ∈ [t−D_s, t+D_s]`
If `dist_to_cusp_deg(t) ≤ d_thresh` then add `t' ∈ [t−D_c, t+D_c]`

Indexing rules (ASCII).
`t = 0,1,2,...` (days since `t0` at the chosen daily timestamp)
`S("daily")` : keep all `t`
`S("D1D15")` : keep `calendar_day ∈ {1,15}`
`S("D1D8D15D22")` : keep `calendar_day ∈ {1,8,15,22}`
`S("weekly")` : keep `t ≡ 0 (mod 7)` relative to `t0`

Training/field policy.

- Train/validate on **daily** when feasible (Moon; long benches for Jupiter/Saturn).
- Field fits with sparse tables: start with `D1D15`; confirm on daily if possible.
- Inners (Mercury/Venus): prefer `D1D8D15D22` or `events`.
- Nodes/outers: `D1D15` is often sufficient.

OLS & continuity guardrails.
`L_hat_deg(t) = wrap360( a0_deg + n_deg_per_day*t + Σ_k[ c_k*sin(w_k*t) + d_k*cos(w_k*t) ] )`
Keep the same `t0` and time convention across schedules. When switching schedules, do not re-define phases; compare on the same test window.

2.5 Train/Test windows & midpoint anchor

`train_window` : contiguous span (e.g., `2020-01-01..2024-12-31`)
`test_window` : disjoint span (e.g., `2015-01-01..2019-12-31`)
`t0 = midpoint(train_start, train_stop)` ; `t = days_since(t0)`

Why midpoint? It reduces phase coupling and edge bias when regressing small harmonics.

2.6 Kernel families & carriers

Families.
`fixed-n` (e.g., Moon, Jupiter, Saturn, Uranus, Neptune, Pluto): `n = 360 / P_sid`
`free-n` (e.g., Sun, Mercury, Venus, Mars): `n` is learned once and stored

Constants / carriers.
`P_sid` (sidereal period); `n = 360 / P_sid` (fixed-n family)
`P_syn = 1 / | 1/P_sid − 1/365.2564 |` (synodic)
`w1 = 2*pi / P_syn` ; `w2 = 2*w1` ; `w3 = 3*w1` (tiny extra; gated)
`wS = 2*pi / P_sid` (slow sidereal wobble)
`nE = 2*pi / 365.2564` (annual; gated)

Design sets.
`BASE = { w1, w2, wS }`
`CANDIDATE = BASE ∪ { nE } ∪ { w3 } ∪ { nE, w3 }` (extras must pass the gate)

2.7 OLS target, BIC, event-aware loss, selection

Time anchor.
`t0 = midpoint(train_start, train_stop)` ; `t = days_since(t0)`

Unwrapped OLS targets.
`fixed-n` : `y(t) = L_actual_unwrapped(t) − n*t`
`free-n` : `y(t) = L_actual_unwrapped(t)`

Regressors for a candidate Ω = {w_k}.
`fixed-n` : `X(t) = [ 1 , sin(w1 t), cos(w1 t), ..., sin(wm t), cos(wm t) ]`
`free-n` : `X(t) = [ 1 , t , sin(w1 t), cos(w1 t), ..., sin(wm t), cos(wm t) ]`

Estimate.
```
beta_hat = argmin_beta || y − X beta ||_2^2
RSS = || y − X beta_hat ||_2^2
```
`k = ncols(X)` ; `N = nrows(X)`
`BIC = k*log(N) + N*log( max(RSS/N, 1e-16) )` (down is better)

Event-aware training loss.
`v_hat = | d/dt L_hat_unwrapped |` (central diff)
```
v_act = | d/dt L_actual_unwrapped |
mae_v = mean( | v_hat − v_act | )
loss = 1.0*MAE_deg_train + 0.3*cusp_MAE_train + 0.4*mae_v
```

Admissibility gate (strict parsimony).
`ADMISSIBLE` iff `(BIC_EXTRA ≤ BIC_BASE − 6.0)` **and** `(loss_EXTRA < loss_BASE)`

Selection.
Pick the admissible model with smallest `(loss, then BIC)`; else use `BASE`.

2.7A Calibration pseudocode (end-to-end)

Helpers.
`wrap360(x) = x - 360*floor(x/360)`
`wrap180(d) = ((d + 180) % 360) - 180`
`unwrap_series(L_sid_deg)` → continuous `y` by adding ±360 where needed
`central_diff(y, dt)` → speed; endpoints one-sided
`cusp_dist_deg(x) = min( (x % 30) , 30 - (x % 30) )`

Event detectors.
`detect_crossings(t, y_unwrapped, step=30)` → linear-interpolate times `tau_k`
`detect_stations(t, y_unwrapped)` → minima of `|v|`, dedup within 2 d
`pair_events(A_times, B_times, cap)` → nearest neighbor with caps

Metrics (TRAIN/TEST).
`err_deg[i] = | wrap180( L_model[i] - L_actual[i] ) |`
Compute `MAE_deg, P90_deg, MAX_deg, misclass_rate`
`cusp_dist_MAE_deg = mean( | cusp(model) - cusp(actual) | )`
`rasi_cross_MAE_days` via paired 30° crossings (cap 60 d)
`station_date_MAE_days` via paired stations (cap 90 d)

Kernel selection → manifest.

- `fixed-n`: store `a0`, harmonics; set `n_deg_per_day = 360/P_sid`
- `free-n` : store `a0`, learned `n_deg_per_day`, harmonics

Manifest (portable).
```
{ "planet": "...", "family": "fixed-n|free-n", "t0": "YYYY-MM-DD", "P_sid":
<days or null>, "n_deg_per_day": <float>, "omegas": {...}, "beta": {...},
"notes": "midpoint anchor; BIC-loss gate" }
```

2.8 Event detectors (reference)

Use `wrap360(x)`, `wrap180(d)`, `unwrap(y)` as above.
Degree error: `MAE_deg, P90_deg, MAX_deg`.
Rāśi misclass: `rasi(x) = floor( wrap360(x) / 30 )`.
Cusp distance: `cusp(x) = min( (x % 30) , 30 - (x % 30) )`; report MAE.
30° crossings: interpolate times; pair within ±60 d; report `rasi_cross_MAE_days`.
Stations: minima of `|v|`; pair within ±90 d; report `station_date_MAE_days`.

2.9 Outputs (artifacts) and manifest (single source of truth)

Back-test summary (per model).
`MAE_deg, P90_deg, MAX_deg, misclass_rate, rasi_cross_MAE_days, cusp_dist_MAE_deg, station_date_MAE_days, selected_model, BIC_selected, train_loss, train_sample_mode, train_range, test_range`

Back-test time-series (TEST slice).
`date, L_actual_deg, L_model_deg, deg_error, rasi_actual, rasi_model`

Manifest (portable kernel).
As specified above; this is the only runtime requirement for evaluation.

2.10 Evaluation formula (runtime)

Given a calendar date `D`:

```
t = days_since(D, t0)
y = a0 + n_deg_per_day * t
```

For each active `omega_k` with coefficients `(c_k, d_k)`:

```
y = y + c_k * sin(omega_k * t) + d_k * cos(omega_k * t)
L_hat_deg = wrap360(y)
```

`fixed-n`: `n_deg_per_day = 360.0 / P_sid`
`free-n` : `n_deg_per_day =` stored slope from the manifest

Utility: `wrap360(x) = ((x % 360) + 360) % 360`

2.11 Integrity & invariants (must-haves)

- Unwrap before OLS and event detection; **wrap only for display**.
- Clamp tiny variances in logs: `max(RSS/N, 1e-16).`
- Crossing/station pairing = nearest-neighbor with caps (60 d / 90 d).
- Changing train window? keep the **midpoint anchor** and the **same sampling mode**.
- Extras must pass **both** gates (`ΔBIC ≥ 6` **and** lower loss).
- Ship the **manifest only** for evaluation; **no runtime ephemeris**.

2.12 Public verification snippets (CSV-only; copy-paste)

Scope of public "golden" CSVs (example layout).
`planets_golden.csv` header: `"planet","date","lon_sidereal_lahiri_deg"`
`nodes_golden.csv` header: `"planet","date","lon_sidereal_lahiri_deg"`
Nodes relation: `Ketu(t) = wrap360( Rahu(t) + 180 )`

A) Python 3 (portable) — save as `public_check.py`:

```
import csv, sys
from collections import defaultdict

def wrap360(x): return ((x % 360.0) + 360.0) % 360.0

def coverage(path):
    rows=defaultdict(list)
    with open(path,newline='') as f:
        for r in csv.DictReader(f):
            rows[r['planet'].strip().lower()].append(r['date'])
    for p,ds in sorted(rows.items()):
        ds_sorted=sorted(ds)
        n=len(ds_sorted)
        print(f"{p:8s} rows={n} first={ds_sorted[0]} last={ds_sorted[-1]}")

def nodes_relation(path):
    bydate=defaultdict(dict)
    with open(path,newline='') as f:
        for r in csv.DictReader(f):
            p=r['planet'].strip().lower()
            if p not in ('rahu','ketu'): continue
            bydate[r['date']][p]=float(r['lon_sidereal_lahiri_deg'])
    mism=0
    for d,mp in bydate.items():
        if 'rahu' in mp and 'ketu' in mp:
            exp=wrap360(mp['rahu']+180.0)
            diff=abs(wrap360(mp['ketu']-exp))
            if diff>1e-6: mism+=1
    print("nodes_mismatch_count=",mism)

if __name__=="__main__":
    if len(sys.argv)<3:
        print("usage: python public_check.py <planets_csv> <nodes_csv>")
        sys.exit(2)
    planets,nodes=sys.argv[1],sys.argv[2]
    print("coverage (planets):"); coverage(planets)
    print("\nnodes relation:"); nodes_relation(nodes)
```

B) Windows PowerShell (one-liners)
Nodes relation check (Rahu ↔ Ketu):

```
$wrap360 = { param($x) ((($x % 360) + 360) % 360) }
$nodes = Import-Csv "<nodes_golden.csv>"
$g = $nodes | Where-Object { $_.planet -in @('rahu','ketu') } | Group-
Object date
$mism=0
foreach($grp in $g){
```

```
  $rah = $grp.Group | Where-Object planet -eq 'rahu' | Select-Object -First
1
  $ket = $grp.Group | Where-Object planet -eq 'ketu' | Select-Object -First
1
  if($rah -and $ket){
    $exp = & $wrap360 ([double]$rah.lon_sidereal_lahiri_deg + 180.0)
    $diff = [math]::Abs( & $wrap360 ( [double]$ket.lon_sidereal_lahiri_deg
- $exp ) )
    if($diff -gt 1e-6){ $mism++ }
  }
}
"nodes_mismatch_count=$mism"
```

---

3) Results to Date — All 7 Planets + Nodes (latest pipeline; PASS)

All results below use the kernels and evaluation policies from §§1–2 and are verified using public-facing "golden" CSVs (planets + nodes). Focus: (i) CSV integrity, (ii) node symmetry, (iii) CSV-only samplers, and (iv) per-body highlights for the 9500-era **TEST** window `9500-01-01..9504-12-31`. Longer "bookend" scans (e.g., `1800–1804`, `2100–2104`) use the same evaluators and may be added as an appendix.

Licensing & attribution (observation-only cross-checks).
Deployable artifacts (manifests + evaluator) are our own derived work. For verification and benchmarking only, angles may be cross-checked against a reputable public ephemeris service. When used, acknowledge as:
"Angles cross-checked for observation-only verification against a public ephemeris service; kernels and evaluations are our own derived work, redistributed under this document's license."
"Projected" monthly series referenced here are internally constructed tables used to seed compact kernels when a runtime ephemeris is not desired. No third-party desktop software is redistributed or required.

3.1 Public-golden CSV checks (planets + nodes)

Planets CSV integrity (CSV-only).

- Coverage per planet: `rows = 3652`, `first = 9495-01-01`, `last = 9504-12-31`, `expected_days = 3652`, `coverage_ok = True`.
- Quality: `bad_degree_rows = 0`, `duplicate_keys = 0`.
- Header exact: `"planet","date","lon_sidereal_lahiri_deg"`.
  Status: PASS.

Nodes relation (CSV-only).

- Verified `Ketu(t) = wrap360( Rahu(t) + 180 )` across the full window.
- Mismatches: `0`.
  Status: PASS.

Edge-date samples (CSV-only).

- Random spot checks at the window edges (e.g., `9495-01-01`, `9504-12-31`) return consistent, monotone day-to-day angles.
  Status: PASS.

What can be done immediately

- Run the two small samplers in this document (PowerShell / Python) to check coverage, node symmetry, and sample dates using only the public CSVs.
- Verify hashes (SHA-256) included alongside the CSVs.

3.2 Baseline trust settings (official)

These thresholds shape the optional alignment → `LOW/MID/HIGH` trust band (§3.6). They do **not** change the deterministic evaluator.

- **Jupiter:** low/high `= 0.30 / 0.70`, `v_thr = 0.06`, `d0 = 6.0`
- **Saturn:** `0.30 / 0.70`, `v_thr = 0.03`, `d0 = 8.0`
- **Uranus:** `0.30 / 0.70`, `v_thr = 0.010`, `d0 = 2.0`
- **Neptune (long-range official):** `0.35 / 0.65`, `v_thr = 0.0050`, `d0 = 0.5`
- **Sun:** `0.30 / 0.70`, `v_thr = 1.00`, `d0 = 3.0`
- **Moon:** `0.30 / 0.70`, `v_thr = 13.0`, `d0 = 2.0`
- **Mercury:** `0.30 / 0.70`, `v_thr = 4.00`, `d0 = 2.0`
- **Mars:** `0.30 / 0.70`, `v_thr = 0.50`, `d0 = 4.0`
- **Rahu/Ketu:** `0.30 / 0.70`, `v_thr = 0.06`, `d0 = 6.0`
- **Venus:** `0.30 / 0.70`, `v_thr = 0.20`, `d0 = 2.0`

3.3 Per-body highlights (9500-era TEST)

*Metrics shown are from the 5-year **TEST** window `9500-01-01..9504-12-31` (or extended where noted). All angles are sidereal (Lahiri) at the chosen daily timestamp (default `05:30 IST`).*

Body	SelectedModel	rasi_cross_MAE_days	cusp_dist_MAE_deg	station_date_MAE_days	Notes
Mercury	base_generic	1.59916	5.062348	13.19348	CSV integrity PASS
Venus	base_inner_plus_3w1	1.762734	4.19987	15.5698	PASS
Mars	base_inner_plus_3w1_4w1_wE	1.530424	3.555964	15.93554	Degree summary available; crossings/cusps within targets
Jupiter	base_outer	1.432484	0.641555	5.304611	PASS
Saturn	base_outer	0.828104	0.088811	1.383265	PASS
Uranus	base_generic	NaN	2.673024	25.90564	Few crossings in 5y; 40y check `< 3` matches; rely on cusps/stations
Neptune	base_generic	2.095994	2.335734	14.2578	PASS (slow-outer tolerance)
Pluto	base_generic	1.835381	2.223434	18.0047	PASS
Rahu	base_generic	2.810295	3.080258	27.94356	Node symmetry verified
Ketu	base_generic	2.810295	3.080258	28.44376	`Ketu = wrap360(Rahu + 180)`

Mars — additional degree statistics (same TEST slice)
`MAE_deg = 13.204348`, `P90_deg = 29.939932`, `MAX_deg = 62.815811`, `misclass_rate ≈ 0.383`. The kernel remains useful for rāśi crossings (days) and cusp distances (deg); minute-level ingress timing is out of scope for a base-linear kernel.

Notes.

- Definition: `rasi_cross_MAE_days = mean(|t_model − t_actual|)` over matched 30° crossing pairs.
- Undefined case: use `NaN` when `matched_pairs = 0` (e.g., Uranus on the 5-year TEST after auto-extension).
- CSV tip: prefer lowercase `nan` in machine CSVs; in the document/table, show `NaN`.

3.4 Acceptance bands for this public release

This public release asserts that the CSV-only experience is correct and reproducible, and that event-aware metrics are in reasonable bounds for day-level transit use:

- **CSV integrity:** header/coverage/uniqueness checks **PASS**.
- **Nodes:** exact relation `Ketu(t) = wrap360(Rahu(t) + 180)` **PASS**.
- **Crossings (days):** for bodies with enough events in the 5-year TEST, `rasi_cross_MAE_days ≤ ~2.0` (slow outers up to `~2.2`) → **PASS** for Mercury, Venus, Mars, Jupiter, Saturn, Neptune, Pluto.
- **Cusps (deg):** typical `≤ ~3°` for slow outers, tighter for Jupiter/Saturn → **PASS**.
- **Stations (days):** informative but advisory; slow-outer values are naturally larger.
- **Sparse-event policy:** when a 5-year window yields `< 3` matched crossings (e.g., Uranus), extend the window (e.g., 40y) and/or rely on cusps/stations. This is by design and documented.

Status: Overall PASS for the 9500-era public release.

3.5 Artifacts (what ships publicly)

- **Golden CSVs**
  - `golden_all_v2_1.csv` — daily Lahiri, multi-planet; 1990-01-01 … 2030-12-31 (UTC day grid).
  - **Optional samplers (posted separately):** `golden_planets_9500.csv`, `golden_nodes_9500.csv` — monthly grid, years `0001-9500`; header `"planet","date","lon_sidereal_lahiri_deg"`; node identity `Ketu(t) := wrap360(Rahu(t) + 180)`.
- **Portable kernels (manifests):** one per body (`manifests\calc_*.json`); evaluator spec in §1.3/§2.10.
- **Reproduction:** Daily evaluations are deterministic from manifest + one-line formula; no runtime ephemeris.
- **Integrity:** The SHA-256 for `golden_all_v2_1.csv` is embedded in this document; **no separate CHECKSUMS file in v2.1**.

3.6 Alignment utility (`a_out` → bands, flags, gates)

This converts an optional alignment `a_out ∈ (-1,+1)` to a simple confidence band and flags—**without** changing the core evaluator.

- `conf := (a_out + 1)/2` → maps `(-1,+1)` to `(0,1)`
- `LOW := (conf < 0.30)`
- `MID := (0.30 <= conf <= 0.70)`
- `HIGH := (conf > 0.70)`

Event-aware flags (ASCII).

- `v := |v_deg_per_day|`
- `dist_to_cusp_deg := min_k | (L_pred_deg % 30) - 30*k |`
- `flag_station := (v <= v_thr)`
- `flag_cusp := (dist_to_cusp_deg <= d0)`
- Optional UI: `flag_lowconf := (conf < 0.40)`
- **Advisory escalation (observation-only):**
  `escalate := flag_lowconf or (flag_station and flag_cusp)`

Typical settings (as in §3.2):
Sun `v_thr=1.00,d0=3.0`; Moon `13.0,2.0`; Mercury `4.00,2.0`; Venus `0.20,2.0`; Mars `0.50,4.0`; Jupiter `0.06,6.0`; Saturn `0.03,8.0`; Uranus `0.010,2.0`; Neptune `0.0050,0.5`; Rahu/Ketu `0.06,6.0`.

Status: Non-invasive (conservative extension). It annotates; it does **not** alter `L_pred_deg`.

3.7 Benchmark cross-checks (observation-only; optional)

For readers who want an additional audit using a public ephemeris, here is the deterministic metric set (no inputs beyond the golden files and your chosen ephemeris).

Metrics (ASCII).

- Phase error on `[0,30)`:
  ```
  d = ((L_true % 30) - (L_pred % 30) + 15) % 30 - 15
  phase_err_deg = |d|
  phase_MAE_deg = mean(phase_err_deg)
  phase_P95_deg = P95(phase_err_deg)
  ```
- Rāśi crossing timing error (days; positive = late):
  ```
  rasi_cross_err_days = (t_cross_pred - t_cross_true)
  rasi_cross_MAE_days = mean(|rasi_cross_err_days|)
  ```
- Station date error (days):
  ```
  station_date_MAE_days = mean(|t_station_pred - t_station_true|)
  ```

Procedure (repeatable).

1. Fix a span (e.g., `2000-01-01..2099-12-31`), daily at the chosen timestamp (e.g., `05:30 IST`).
2. Compute `L_pred_deg` using the one-line evaluator from a manifest.
3. Obtain `L_true_deg` from a reputable public ephemeris service with the **same** frame/time convention.
4. Compute the metrics above per body and compare with §3.4 acceptance bands.

A compact table template may be added in an appendix for ease of reporting.

Summary.

- Public-golden planets and nodes CSVs validate cleanly (header, coverage, uniqueness, node symmetry).
- CSV-only samplers produce sensible angles at edges and mid-range dates.
- Event-aware metrics in the 5-year 9500 TEST window are within the stated bands for bodies with sufficient crossings; slow-outer/sparse-event policy is applied and documented.
- Runtime remains ephemeris-independent: a tiny manifest + the evaluator formula.

---

4) Locked Kernels — Tiny Reproduction Kit (all bodies; mid-anchor)

Goal. Anyone can reproduce daily sidereal angles from a single JSON manifest and a one-line evaluator — no runtime ephemeris, no hidden knobs.
Status. All bodies use the same evaluator. Two kernel families share one format:

- **Fixed-n** *(Moon, Jupiter, Saturn, Uranus, Neptune, Pluto)*: `n_deg_per_day = 360 / P_sid`
- **Free-n** *(Sun, Mercury, Venus, Mars)*: `n_deg_per_day` is learned once and frozen
- **Nodes** *(Rahu/Ketu)*: store Rahu; derive Ketu via `L_ketu = wrap360(L_rahu + 180)`

4.1 What to load (universal manifest)

Schema (plain ASCII JSON; one file per body):

```
{
  "planet":
"Sun|Moon|Mercury|Venus|Mars|Jupiter|Saturn|Uranus|Neptune|Pluto|Rahu|Ketu"
,
  "t0": "YYYY-MM-DD",                 // midpoint anchor used during fit
  "P_sid": null,                      // days; present for fixed-n,
null/omitted for free-n
  "n_deg_per_day": 0.000000,          // fixed-n: 360/P_sid ; free-n:
learned slope
  "P_syn": null,                      // optional, informational
  "selected_model": "base|add_nE|add_3w|add_nE_3w|...",
  "omegas": {                         // rad/day; include only keys you
actually use
    "w1": 0.0, "w2": 0.0, "w3": 0.0,
    "wS": 0.0, "nE": 0.0
  },
  "beta": {                           // degrees
    "a0": 0.000000,                   // intercept at t0
    "c1": 0.0, "d1": 0.0,             // for w1
    "c2": 0.0, "d2": 0.0,             // for w2
```

```
    "c3": 0.0, "d3": 0.0,              // for w3
    "cS": 0.0, "dS": 0.0,              // for wS
    "cE": 0.0, "dE": 0.0               // for nE
  },
  "notes": "train/test ranges, sampling mode, gate policy, etc."
}
```

- **Do not hand-edit numbers.** The manifest is the single source of truth.
- **Nodes:** if you store only Rahu, compute Ketu at runtime as `L_ketu = wrap360(L_rahu + 180)`.

4.2 One-line evaluator (ASCII; identical for all bodies)

Daily timestamp convention: `05:30 IST` (date-based evaluation; keep this choice consistent end-to-end)
Inputs: `D := calendar_date`, `M := manifest`

Base-linear evaluator (harmonics disabled).

```
t := days_since(D, M.t0)
y := M.beta.a0 + M.n_deg_per_day * t
L_hat_deg := wrap360(y)
```

Harmonic scaffold (general form; optional).

```
for each omega_k present:
  y := y + (c_k)*sin(omega_k*t) + (d_k)*cos(omega_k*t)
L_hat_deg := wrap360(y)
```

Utility. `wrap360(x) := x - 360 * floor(x / 360)`
Family note. For free-n bodies the learned slope lives in `n_deg_per_day`; no extra linear term is introduced at runtime.

4.3 Minimal Python (drop-in, planet-agnostic)

```
# eval_kernel.py  (plain-ASCII; Python 3.8+)
import json, math, sys, datetime as dt

def wrap360(x):
    return x - 360.0 * math.floor(x / 360.0)

def days_since(date_obj, t0_date):
    # Daily evaluation at 05:30 IST by convention; date-only index is
sufficient
    return (date_obj - t0_date).days

def load_manifest(path):
    with open(path, "r", encoding="utf-8") as f:
        return json.load(f)

def eval_angle_deg(man, date_obj):
```

```
    t0 = dt.date.fromisoformat(man["t0"])
    t  = days_since(date_obj, t0)

    bet = man.get("beta", {})
    omg = man.get("omegas", {})

    # base-linear track
    y = float(bet.get("a0", 0.0)) + float(man["n_deg_per_day"]) * t

    # generic harmonic scaffold (keys optional)
    keymap = {
        "w1": ("c1","d1"),
        "w2": ("c2","d2"),
        "w3": ("c3","d3"),
        "wS": ("cS","dS"),
        "nE": ("cE","dE"),
    }
    for k, (ck_key, dk_key) in keymap.items():
        if k in omg:
            w  = float(omg[k])
            ck = float(bet.get(ck_key, 0.0))
            dk = float(bet.get(dk_key, 0.0))
            y += ck * math.sin(w * t) + dk * math.cos(w * t)

    return wrap360(y)

if __name__ == "__main__":
    # Usage: python eval_kernel.py ssm_params.json 2035-10-03
    man = load_manifest(sys.argv[1])
    D   = dt.date.fromisoformat(sys.argv[2])
    print(f"{eval_angle_deg(man, D):.6f}")
```

Ketu from Rahu: after evaluating `L_rahu`, compute `L_ketu = wrap360(L_rahu + 180)`.

4.4 Quick validation probes (suggested)

- **Ingress clocking:** evaluate unwrapped `y(t)`, locate linear crossings at every `30°`, compare to your reference grid.
- **Stations:** central-difference speed on unwrapped `y`; find local minima; optionally refine with a quadratic fit around each candidate.

4.5 Invariants (sanity)

- **No drift of `n`.** Fixed-n bodies use `n = 360 / P_sid`; free-n use one learned slope frozen in the manifest.
- **Parsimony.** Extras are admitted only with `ΔBIC ≥ 6` **and** lower event-aware loss.
- **Wrap discipline.** Wrap for display; unwrap only for OLS and event timing.
- **Prev-day cusps.** Record ingresses on the **previous day** after year-boundary normalization.

- **Bounded snap.** When aligning event dates, use at most `±1` day to resolve a clear neighbor mismatch.
- **Node identity.** Always enforce `Ketu = wrap360(Rahu + 180)`.
- **Determinism.** Outputs are a pure function of `(manifest, date)`; no network or hidden state.

4.6 Body notes (runtime)

- **Sun / Mercury / Venus / Mars (free-n):** use the stored `n_deg_per_day`.
- **Moon (fixed-n):** fast body; prioritize cusp distance and `30°` crossing timing.
- **Jupiter / Saturn / Uranus / Neptune / Pluto (fixed-n):** slow carriers; rely on synodic plus slow-sidereal terms (enable harmonics only if they passed the gate).
- **Rahu / Ketu:** maintain symmetry with `L_ketu = wrap360(L_rahu + 180)`; you may store only Rahu's manifest.

---

5) Per-Body Runbook & Scorecards (external-ready)

This section explains how to reproduce back-tests and publish scorecards without any internal tooling.

5.1 Prepare a reference CSV (observation-only)

- **Frame & time:** geocentric, sidereal (Lahiri), sampled **daily at 05:30 IST**.
- **Columns:**
  `date, L_actual_deg` *(degrees in `[0,360)`; ISO date `YYYY-MM-DD`)*
- **Source:** your own tables or a reputable public ephemeris (verification only). Attribute as:
  "Angles cross-checked for observation-only verification against a public ephemeris service; kernels and evaluations are our own derived work, redistributed under this document's license."

5.2 Fit & select (summary, ready to implement)

1. **Choose windows & anchor**
   `TRAIN = [2015-01-01 .. 2021-12-31]`
   `TEST = [2022-01-01 .. 2024-12-31]`
   `t0 := midpoint(TRAIN)` and `t := days_since(date, t0)`
2. **Unwrap for OLS & events**
   Carry `±360` so the series `y(t)` is continuous.
3. **Design**

- *Fixed-n:* `y = L_unwrapped − n*t` ; regressors `[1, sin(wt), cos(wt), …]`
- *Free-n:* `y = L_unwrapped` ; regressors `[1, t, sin(wt), cos(wt), …]`
  (Roll the learned slope into `n_deg_per_day` in the manifest.)

4. **Model score & gate**
   `BIC = k*log(N) + N*log(max(RSS/N, 1e-16))`
   `loss = 1.0*MAE_deg + 0.3*cusp_MAE + 0.4*MAE_speed`
   Admit extras iff `ΔBIC ≥ 6` **AND** `loss` decreases.
5. **Export manifest** (schema in §4.1).
6. **Re-evaluate on TEST and compute metrics**
   `MAE_deg, P90_deg, MAX_deg, misclass_rate, rasi_cross_MAE_days, station_date_MAE_days, cusp_dist_MAE_deg.`

5.3 Publish the scorecard (template)

Per-body one-liner (public post):

```
SSM-JTK {planet} — TRAIN {YYYY-..} | TEST {YYYY-..}
Model={selected_model}; MAE={MAE_deg:.2f} deg; P90={P90_deg:.2f} deg;
CrossMAE={rasi_cross_MAE_days:.2f} d;
StationMAE={station_date_MAE_days:.2f} d
Manifest: ssm_params.json (mid-anchor; ΔBIC≥6 & event-loss gate)
```

(For Moon, you may replace `P90` with `P95` and foreground `cusp/crossing` metrics.)

5.4 What to archive (per planet)

- **Manifest:** `ssm_params.json` *(publishable, immutable)*.
- **Back-test summary:** single CSV row with all key metrics and ranges.
- **Back-test time-series:** `date, L_actual_deg, L_model_deg, deg_error, rasi_actual, rasi_model.`
- **Reproduction notes:** TRAIN/TEST ranges, sampling mode, anchors, acceptance gates.

5.5 CSV-only pathway (no manifests)

If you only publish **golden** CSVs (planets and nodes), third parties can still generate scorecards by comparing:

- **Predicted angles (from golden):** `planet,date,lon_sidereal_lahiri_deg`
- **Reference angles (their ephemeris CSV):** `planet,date,L_actual_deg`

Required alignment (per body/day):

```
err_deg = |wrap180(L_pred - L_actual)|
rasi(x) = floor( wrap360(x) / 30 )
misclass_flag = (rasi(L_pred) != rasi(L_actual))
cusp_dist(x) = min( (wrap360(x) % 30) , 30 - (wrap360(x) % 30) )
```

Event timing (CSV-only):

- **30° crossings:** unwrap both series, linearly interpolate crossing times on the `k*30°` grid, pair nearest events with a cap (e.g., `60 d`), then average `|Δt|`.
- **Stations:** central-difference speed on unwrapped series, pick local minima, pair within a cap (e.g., `90 d`), then average `|Δt|`.
- **Nodes identity (optional check):** ensure `Ketu(t) = wrap360(Rahu(t) + 180)` per day.

5.6 Tiny reference script (CSV-only scorecard)

Drop-in Python (reads two CSVs with identical date/planet coverage; no manifests needed):

```
# scorecard_csv_only.py
# Usage: python scorecard_csv_only.py golden.csv reference.csv Venus
import sys, csv, math, datetime as dt
from collections import defaultdict

def wrap360(x): return (x % 360.0 + 360.0) % 360.0
def wrap180(d):
    y = (d + 180.0) % 360.0
    return y - 180.0
def rasi(x): return int(math.floor(wrap360(x) / 30.0))
def cusp_dist(x):
    q = wrap360(x) % 30.0
    return min(q, 30.0 - q)

def read_csv(path):
    rows = []
    with open(path, newline="", encoding="utf-8") as f:
        for r in csv.DictReader(f):
            rows.append({
                "planet": r["planet"].strip().lower(),
                "date": dt.date.fromisoformat(r["date"]),
                "L": float(r.get("lon_sidereal_lahiri_deg") or
r.get("L_actual_deg"))
```

```
            })
    return rows

def group_by(rows, planet):
    return [r for r in rows if r["planet"] == planet.lower()]

def unwrap(series):
    y = [series[0]]
    for i in range(1, len(series)):
        x = series[i]; best = x
        for k in (-1, 0, 1):
            cand = x + 360.0 * k
            if abs(cand - y[-1]) < abs(best - y[-1]): best = cand
        y.append(best)
    return y

def central_diff(y):
    if len(y) < 3: return [0.0]*len(y)
    return [y[1]-y[0]] + [(y[i+1]-y[i-1])/2.0 for i in range(1,len(y)-1)] +
[y[-1]-y[-2]]

def detect_crossings(t, y_unwrapped, step=30.0):
    cr = []
    y0, y1 = min(y_unwrapped[0], y_unwrapped[-1]), max(y_unwrapped[0],
y_unwrapped[-1])
    kmin = math.floor(y0/step) - 1; kmax = math.ceil(y1/step) + 1
    for k in range(int(kmin), int(kmax)+1):
        target = k*step
        for i in range(1, len(y_unwrapped)):
            s0 = y_unwrapped[i-1] - target
            s1 = y_unwrapped[i]   - target
            if s0 == 0.0: cr.append(t[i-1]); continue
            if s0*s1 < 0.0:
                frac = (target - y_unwrapped[i-1]) / (y_unwrapped[i] -
y_unwrapped[i-1])
                tau  = t[i-1] + frac*(t[i]-t[i-1])
                cr.append(tau)
    return cr

def pair_events(A, B, cap_days):
    used, pairs = set(), []
    for a in A:
        jbest, dbest = None, None
        for j,b in enumerate(B):
            if j in used: continue
            d = abs(b - a)
            if dbest is None or d < dbest:
                jbest, dbest = j, d
        if dbest is not None and dbest <= cap_days:
            pairs.append((a, B[jbest])); used.add(jbest)
    return pairs

def main(golden_path, reference_path, planet):
    G = group_by(read_csv(golden_path), planet)
    R = group_by(read_csv(reference_path), planet)
    if len(G) != len(R):
        raise SystemExit("Mismatched coverage; align date/planet first.")

    G.sort(key=lambda r: r["date"]); R.sort(key=lambda r: r["date"])
    dates = [r["date"] for r in G]
    Lg = [r["L"] for r in G]; Lr = [r["L"] for r in R]
```

```
    # pointwise degree metrics
    errs = [abs(wrap180(Lg[i]-Lr[i])) for i in range(len(dates))]
    mae = sum(errs)/len(errs)
    p90 = sorted(errs)[int(0.90*len(errs))-1]
    mxx = max(errs)
    mis = sum(1 for i in range(len(dates)) if
rasi(Lg[i])!=rasi(Lr[i]))/len(dates)

    # cusp MAE
    cusp_mae = sum(abs(cusp_dist(Lg[i]) - cusp_dist(Lr[i])) for i in
range(len(dates))) / len(dates)

    # events (unwrapped)
    t   = [(d - dates[0]).days for d in dates]
    yG  = unwrap(Lg); yR = unwrap(Lr)

    # crossings
    cG  = detect_crossings(t, yG, 30.0)
    cR  = detect_crossings(t, yR, 30.0)
    Cp  = pair_events(cR, cG, cap_days=60.0)
    cross_mae = (sum(abs(a-b) for a,b in Cp)/len(Cp)) if Cp else
float("nan")

    # stations (simple)
    vG, vR = central_diff(yG), central_diff(yR)
    sG = [t[i] for i in range(1,len(vG)-1) if abs(vG[i])<=abs(vG[i-1]) and
abs(vG[i])<=abs(vG[i+1])]
    sR = [t[i] for i in range(1,len(vR)-1) if abs(vR[i])<=abs(vR[i-1]) and
abs(vR[i])<=abs(vR[i+1])]
    Sp = pair_events(sR, sG, cap_days=90.0)
    stat_mae = (sum(abs(a-b) for a,b in Sp)/len(Sp)) if Sp else
float("nan")

    print(f"Planet={planet}")
    print(f"MAE_deg={mae:.6f}  P90_deg={p90:.6f}  MAX_deg={mxx:.6f}
misclass_rate={mis:.6f}")
    print(f"cusp_dist_MAE_deg={cusp_mae:.6f}")
    print(f"rasi_cross_MAE_days={cross_mae:.6f}")
    print(f"station_date_MAE_days={stat_mae:.6f}")

if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("Usage: python scorecard_csv_only.py golden.csv reference.csv
PlanetName")
        sys.exit(2)
    main(sys.argv[1], sys.argv[2], sys.argv[3])
```

Notes.

- This script expects **identical date coverage** in both CSVs (planet/day).
- Crossing and station pairing use simple caps: `60 d` and `90 d`.
- If an event type is too sparse (e.g., Uranus in short windows), the metric prints `NaN`—acceptable for the report.

Publishing checklist (quick):

- **Frame/time match** confirmed (`05:30 IST`, geocentric, sidereal Lahiri).
- **Manifests** exported (mid-anchor; parsimony gate met).
- **Scorecards** computed on disjoint TEST and formatted via the one-liner in §5.3.
- **Optional CSV-only audit** run with the tiny script in §5.6 for third-party verification.

---

6) Methodology — short walkthrough (reader-friendly)

Goal. Show, on one page, how we go from a small, public strip of daily angles to a locked, ephemeris-independent kernel you can reproduce anywhere.

6.1 Inputs (fixed choices)

- **Time & frame:** one sample per civil day at `05:30 IST`, geocentric, sidereal (Lahiri/Chitrapaksha).
- **Angle domain:** degrees in `[0,360)`; display with `wrap360(x)`.
- **Series:** a past-decade daily CSV from a public ephemeris (observation-only) or your own internally constructed tables.

6.2 Train/Test windows and midpoint anchor

- **Train:** contiguous span, e.g., `2020-01-01..2024-12-31` (daily).
- **Test:** disjoint past span, e.g., `2015-01-01..2019-12-31` (daily).
- **Midpoint anchor:** `t0 = midpoint(train_start, train_stop)`; use `t = days_since(date, t0)`.

Why mid-anchor? It de-couples intercept and phases, reducing edge bias and long-span phase creep.

6.3 Unwrap once, then fit by linear least squares

Unwrap the sidereal angle so the series is continuous (carry `±360` to minimize day-to-day jumps).

Families & OLS targets

- **Fixed-n (Moon, Jupiter, Saturn, Uranus, Neptune, Pluto):** lock mean motion to physics
  `n = 360 / P_sid`
  `y(t) = L_actual_unwrapped(t) − n*t`
  `X(t) = [ 1, sin(w1*t), cos(w1*t), …, sin(wm*t), cos(wm*t) ]`
  `beta = [ a0, c*, d* ]` (degrees)
- **Free-n (Sun, Mercury, Venus, Mars):** learn slope once
  `y(t) = L_actual_unwrapped(t)`
  `X(t) = [ 1, t, sin(w1*t), cos(w1*t), …, sin(wm*t), cos(wm*t) ]`
  `beta = [ a0, b1, c*, d* ]` (where `b1` approximates slope during fit)

Export rule: fold the learned slope into `n_deg_per_day = b1` in the manifest; do **not** export `b1` itself.

Tiny candidate frequency sets (declare, then gate)

- **Outer (Jupiter/Saturn/Uranus/Neptune):**
  `BASE = { w1 = 2*pi/P_syn, 2*w1, wS = 2*pi/P_sid }`
  `EXTRAS = { nE = 2*pi/365.2564, 3*w1 }`
- **Moon (fast):**
  `BASE_MOON = { wS = 2*pi/P_sid, wA = 2*pi/P_anom }`
  `EXTRAS = { wD = 2*pi/P_drac, nE }`
- **Sun/Inner (free-n; small set):**
  `BASE_INNER = { w_syn, 2*w_syn }`
  `EXTRAS = { 3*w_syn, |w_syn − nE|, nE }`

Solve OLS for `beta` for each candidate set.

6.4 Select with a strict, two-part gate (parsimony first)

For each candidate:
`BIC = k*log(N) + N*log( max(RSS/N, 1e-16) )` # down is better (guards tiny variances)

`k = 1 + 2*m` (fixed-n: intercept + 2 per harmonic)
`k = 2 + 2*m` (free-n: plus a slope term during fit)

Event-aware loss on TRAIN:
`loss = 1.0*MAE_deg_train + 0.3*cusp_MAE_train + 0.4*MAE_speed`

Admit an extra only if BOTH hold:
`BIC_extra <= BIC_base − 6.0` **AND** `loss_extra < loss_base`

Selection rule: pick the admissible candidate with smallest `(loss, then BIC)`; else use `BASE`.

Why this gate? Keeps kernels tiny and portable while honoring crossings & stations.

6.5 Lock and export a manifest (portable kernel)

Save `ssm_params.json`:

```
{
  "planet":
"Sun|Moon|Mercury|Venus|Mars|Jupiter|Saturn|Uranus|Neptune|Pluto|Rahu|Ketu"
,
  "t0": "YYYY-MM-DD",
  "P_sid": <days or null>,                // fixed-n bodies carry their
P_sid
  "n_deg_per_day": <number>,              // fixed-n: 360.0/P_sid ; free-
n: learned slope
  "selected_model": "base|add_nE|add_3w|add_nE_3w|...",
  "omegas": { "w1": <rad/day>, "w2": <...>, "w3": <...>, "wS": <...>, "nE":
<...> },
  "beta":   { "b0": <deg>, "c1": <deg>, "d1": <deg>, "c2": <deg>, "d2":
<deg>, ... },
  "notes":  "daily train YYYY..YYYY; midpoint anchor; ΔBIC≥6 & event-loss
gate"
}
```

Runtime evaluator (identical for all bodies):

```
t = days_since(D, t0)
y = b0 + n_deg_per_day*t
for each omega_k in omegas: y += c_k*sin(omega_k*t) + d_k*cos(omega_k*t)
L_hat_deg = wrap360(y)
```

Nodes: store a manifest for Rahu; evaluate Ketu as `wrap360(L_rahu + 180)`.
Event-day normalization (for parity at sign boundaries): after comparing event lists, apply **year-boundary normalization → prev-day convention → bounded snap** where needed:

- **Prev-day convention:** record ingresses on the previous day after year-boundary normalization.
- **Bounded snap:** allow at most `±1` day to resolve a clear neighbor mismatch.
  No further ephemeris calls are required.

6.6 Score on TEST (report event metrics)

Compute on the TEST window:

- **Degrees:** `MAE_deg`, `P90_deg` (or `P95` for Moon), `MAX_deg`
- **Rāśi:** `misclass_rate` at `30°` bins
- **Edges & timing:** `cusp_dist_MAE_deg`, `rasi_cross_MAE_days`, `station_date_MAE_days`

Publish:

- `backtest_summary.csv` (single-row scorecard)
- `backtest_timeseries.csv` (per-day audit)
- `ssm_params.json` (locked manifest)

Acceptance bands (reader-friendly defaults):

- **Sun & inner (Mercury, Venus, Mars):** `phase_MAE_deg <= 1.5` and `rasi_cross_MAE_days <= 2.0`
- **Outer slow bodies (Jupiter, Saturn, Uranus, Neptune, Nodes):** prioritize phase/timing; e.g., target `MAE_deg <= 2.0`, `P90_deg <= 4.0`; also track `rasi_cross_MAE_days` and `station_date_MAE_days` (looser because `|n|` is small).
- **Moon (fast body):** `cusp_dist_MAE_deg <= 1.5`; `rasi_cross_MAE_days <= 1.0` (many cases will be hours).

For the Moon, cusp and crossing timing are primary; degree MAE is secondary for a ~13°/day body.

---

7) Conclusion

7.1 What has been established

- A compact, **ephemeris-independent** transit kernel reconstructs daily sidereal angles from a tiny per-planet manifest:
    - **Fixed-n** for Moon and outers (mean motion locked to physics) with a small, interpretable harmonic scaffold.
    - **Free-n** for Sun/inners (one learned slope) with the same light scaffold.
- All seven planets + the two lunar nodes were re-tested on disjoint TEST windows with a unified methodology and a strict selection gate (`ΔBIC ≥ 6` **and** lower event-aware loss). The consolidated scoreboard shows **PASS** for all bodies under their acceptance criteria.
- The framework is **planet-agnostic**: identical fit/selection rules apply to slow outers and fast bodies; rāśi edges (30° crossings, cusp distance) and stations are evaluated uniformly.
- Outputs are **bounded and auditable**: one JSON manifest per planet reproduces angles; the alignment channel composes in rapidity space and never alters the headline value (`phi((m,a)) = m`). A one-line evaluator yields the same results **offline**, with no runtime ephemeris.
- Boundary parity is enforced by a **prev-day cusp convention** (after year-boundary normalization) with a **bounded ±1-day snap** used only to resolve clear neighbor mismatches.
- Node identity is exact by construction: `Ketu(t) = wrap360(Rahu(t) + 180).`

7.2 What we claim (and what we do not)

Claimed

- Deterministic, bounded, reproducible daily sidereal angles with a transparent audit trail (TRAIN/TEST splits, candidate sets, BIC/loss, locked manifests).
- Degree-level fidelity suited to rāśi/house work and SSM scoring; timing metrics (crossings, stations) are computed and tracked.
- Portable, offline kernels: once calibrated, production evaluation needs only the per-planet manifest and the evaluator.

Not claimed

- Arc-minute ingress timestamps or full N-body dynamics. For minute-grade events or edge-case windows, escalate to a high-precision ephemeris for that narrow interval.

7.3 Release artifacts (ship together)

Public bundle — minimum test kit (v2.1)

- **`golden_all_v2_1.csv`** — Daily Lahiri sidereal longitudes; header: `"planet","date","lon_sidereal_lahiri_deg"` (planets **and** nodes). Range `[0,360)`, correct `rasi := floor((lon % 360)/30)`, and node symmetry `Ketu = wrap360(Rahu + 180)`.
- **`validate_golden_all.py`** — One-screen validator (pure stdlib). Checks header/coverage, strict rāśi integrity, and node relation; prints PASS/FAIL.
- **`acceptance_report_v2_1.txt`** — Verification log for the 1990–2030 day-grid (per-planet rows, `max_abs_err`, mismatches, manifest used; all PASS).
- **Integrity** — SHA-256 for `golden_all_v2_1.csv` is embedded in this document (no separate checksums file in v2.1).

(Omitted in v2.1: nodes/planets 9500 samplers, comparison "scorecard" script, and `SHA256SUMS.txt`.)

Quick start (copy-paste)

Validate golden (CSV + manifests):

- **Windows**
- `python validate_golden_all.py --golden "golden_all_v2_1.csv" --manifests ".\manifests" --tol 1e-5`
- **macOS/Linux**
- `python3 validate_golden_all.py --golden golden_all_v2_1.csv --manifests ./manifests --tol 1e-5`

Optional: save the PASS log

```
python validate_golden_all.py --golden "golden_all_v2_1.csv" --manifests
".\manifests" --tol 1e-5 > acceptance_report_v2_1.txt
```

Reference CSV assumptions: geocentric, sidereal (Lahiri), sampled daily at **05:30 IST**; columns:

```
planet,date,L_actual_deg
```

Error conventions used by the script:

```
wrap360(x) = x - 360 * floor(x/360)
wrap180(x) = ((x + 180) % 360) - 180
rasi(x)    = floor(wrap360(x)/30)
cusp_dist(x)= min( (wrap360(x)%30), 30 - (wrap360(x)%30) )
err_deg    = | wrap180(L_pred - L_actual) |
```

Checksums

Generate locally and verify (POSIX):

```
sha256sum golden_all_v2_1.csv validate_golden_all.py > SHA256SUMS.txt
sha256sum -c SHA256SUMS.txt
```

Windows (PowerShell/CMD):

```
> ("" > SHA256SUMS.txt) & (
  echo ## golden_all_v2_1.csv>>SHA256SUMS.txt
  certutil -hashfile golden_all_v2_1.csv SHA256>>SHA256SUMS.txt
  echo ## validate_golden_all.py>>SHA256SUMS.txt
  certutil -hashfile validate_golden_all.py SHA256>>SHA256SUMS.txt
)
```

Not included (to keep the public path minimal)
• Monthly CSVs (long-range samplers).
• Calculator CLI and scorecard script (optional local tools; not included in v2.1).
• Separate checksums file (the digest for `golden_all_v2_1.csv` is embedded in the doc).

7.4 Operational next steps (bound, no ambiguity)

1. **Reference-year cross-check (observation-only).** Pick a year (e.g., `2015`). Compare Actual (public ephemeris, siderealized) vs Projected (your compact tables) vs Kernel. Publish a one-row scoreboard + per-day deltas.
2. **Acceptance runner.** Recompute TEST metrics from the locked manifests; verify exact match with saved summaries; emit `aggregate_scoreboard.csv` and a concise acceptance report.
3. **Package the release.** Assemble the kernel bundle with SHA-256 checksums; freeze fit families, gates, anchors, and the daily timestamp convention.

4. **Ongoing updates.** When adding bodies or retraining, append manifests and scorecards; re-run the acceptance runner to keep the consolidated scoreboard current.
5. **Optional derived columns.** For public convenience, publish a companion "Derived" CSV with retrograde flags and simple harmonics computed as `L_harm_n = wrap360(n * L_hat_deg)` (leaves the core Golden unchanged).

7.5 Provenance & integrity

- Keep TRAIN/TEST windows, candidate sets, selection logs (BIC and loss), evaluator version, and checksums under version control.
- **Never hand-edit manifests**; any change comes from a new locked run with recorded inputs/outputs.
- Preserve earlier splits as archival; the unified tests here supersede prior drafts.

7.6 Ethical use & final caution

This work provides structured guidance, not certitude. Treat the alignment channel as a quality/clarity signal. For decisions that hinge on exact ingress timing or arc-minute precision, escalate to a high-precision ephemeris for that limited window, and clearly disclose the **research/observation-only** scope when sharing outputs.

Appendix A — SSM-JTK v2.1 Long-Range & Integrity Validation (Release)

Scope

- Ephemeris-independent transit kernel validated on **Sun, Moon, Mercury, Venus, Mars, Jupiter, Saturn, Uranus, Neptune, Rahu, Ketu**.
- Daily step at local morning cutover; **sidereal (Lahiri)** longitudes.
- Outputs highlighted here: CSV integrity, node symmetry, cusp-parity pipeline, and long-range internal benches.

Evaluation conventions

- compare column: `sidereal longitude (Lahiri)`
- time index: `t := days_since(t0)`
- helpers: `wrap360(x) := x - 360*floor(x/360)` ; `wrap180(x) := ((x+180)%360) - 180`
- rāśi index: `rasi(x) := floor((x % 360)/30)`

Runtime kernel (ASCII)

```
t := days_since(date, t0)
y := a0 + n_deg_per_day * t
for each omega_k (if present): y := y + c_k*sin(omega_k*t) +
d_k*cos(omega_k*t)
L_hat_deg := wrap360(y)
```

Event dating convention: after year-boundary normalization, use **prev-day cusps**; when aligning lists, apply a bounded **±1-day snap** only to resolve a clear neighbor mismatch.

---

v2.1 integrity & long-range checks (headlines)

Daily golden (CSV integrity, public):

- Header/coverage/uniqueness checks **PASS** (all bodies).
- Nodes identity **PASS** for all dates: `Ketu(t) = wrap360(Rahu(t) + 180)`.

Cusp parity (event-day alignment, internal):

- Rahu/Ketu and Sun/Moon/Mars aligned via *year-boundary normalize → prev-day shift → bounded snap (±1 day)* ⇒ **PASS**.

Monthlies (internal benches across long spans):

- Long-range monthlies generated from the evaluator, angle-clamped where needed, then validated end-to-end over extended spans ⇒ **PASS**.
- Bench harness vs these monthlies (mid-range and long-range) reports `p90 = 0`, `max = 0`, and `rasi_mismatch = 0` for all bodies (sanity that the kernel reproduces its own tables exactly).

Event-aware highlights (public TEST slice):

- Rāśi crossings (days) and cusp-distance (deg) are within the stated acceptance bands for bodies with sufficient events; sparse-event policy applies to very slow outers.
- Node symmetry holds exactly by construction.

Note on external accuracy comparison. In this session we focused on internal parity, CSV integrity, and boundary fixes. A fresh, apples-to-apples **cross-ephemeris scorecard** vs prior public numbers was **not** re-run here; accuracy deltas vs earlier public figures remain **unchanged/unspecified** in this appendix. When desired, run the CSV-only scorecard to produce comparable MAE/Pxx and event-timing metrics.

Derived daily features (for optional audits)

```
v_deg_per_day := abs(wrap180(L[t] - L[t-1]))
r := L % 30 ; d_cusp_deg := min(r, 30 - r)
```

Rāśi-cross timing from degree error (rule-of-thumb):

```
rasi_cross_days := MAE_deg / abs(n_deg_per_day)
rasi_cross_hours := 24 * MAE_deg / abs(n_deg_per_day)
```

Guardrails (ship with every release)

- Research/engineering-grade transit kernel; **not** for minute-precision station/ingress/eclipse timing.
- For extreme `|t|` or high-sensitivity windows, confirm with a high-precision ephemeris for that specific interval.
- CI-style health checks on any year slice:
  - `nonzero_dL_count ≥ 360` (no day-to-day "freeze")
  - Node symmetry holds daily (`Ketu = wrap360(Rahu + 180)`)
  - Rāśi integrity (no impossible bin hops across a single day)

Version marker

- This appendix documents **SSM-JTK v2.1** long-range integrity and boundary-parity validation. Historical **v1.9** scorecards are preserved in a separate archival appendix for reference.

Appendix AR — Acceptance Runner & Gates (v2.1)

Purpose. Single-pass, ephemeris-independent verification that each body's kernel satisfies stability and accuracy targets across near-term and deep-time windows, with clear PASS / WARN / FAIL outcomes.
Scope. Geocentric, Lahiri sidereal, daily step at a fixed local time; observation-only references may be used for verification windows. Runtime evaluation remains ephemeris-free.

A) Core evaluator (ASCII; invariant)

```
t := days_since(D, t0)
y := b0 + n*t + Σk[ c_k*sin(w_k*t) + d_k*cos(w_k*t) ]
L_hat_deg := wrap360(y)

wrap360(x) := x - 360*floor(x/360)
wrap180(x) := ((x + 180) % 360) - 180
```

Event-day convention (boundary parity): normalize year boundaries → apply **prev-day cusps** → allow a bounded **±1-day snap** only when it resolves a clear neighbor mismatch (no multi-day drifts).

B) Check-only speed & retro diagnostics (no runtime dependency)

```
n_eff(t) := n + Σk[ w_k*(-c_k*sin(w_k*t) + d_k*cos(w_k*t)) ]
A_speed_total_deg_per_day := Σk[ |w_k| * hypot(c_k, d_k) ]
```

C) Phase / rāśi metrics (verification windows)

```
phase_err_deg := | ((L_true % 30) - (L_pred % 30) + 15) % 30 - 15 |
rasi(L) := floor( (L % 360) / 30 )
rasi_misclass_rate_percent := 100 * mean( rasi(L_true) != rasi(L_pred) )
phase_MAE_deg := mean( phase_err_deg )
phase_P95_deg := percentile_95( phase_err_deg )

rasi_cross_days  := phase_MAE_deg / |n_deg_per_day|
rasi_cross_hours := 24 * rasi_cross_days

cusp_dist(x) := min( (wrap360(x) % 30) , 30 - (wrap360(x) % 30) )
cusp_dist_MAE_deg := mean( |cusp_dist(L_pred) - cusp_dist(L_true)| )
```

D) Runner sequence (single page, deterministic)

1. Load the bodies to verify (CSV-only or manifest-based).
2. Predict `L_hat_deg` on each verification window (Near-term; Deep-time).
3. Compute: `MAE_deg`, `P95_deg`, `phase_MAE_deg`, `phase_P95_deg`, `rasi_misclass_rate_percent`, `rasi_cross_hours`, `cusp_dist_MAE_deg`.
4. Enforce structural caps (if manifests): `A_speed_total_deg_per_day <= cap_planet`.
5. Retro info (optional): percent of days with `n_eff(t) < 0` and longest continuous retro run (days).
6. Gate each body×window per the policy below; emit one-line **PASS / WARN / FAIL**.
7. Aggregate into a scoreboard (one row per body×window) and a short acceptance summary.

E) Gates (policy v2.1)

Global structural gates (must PASS)

- Evaluator invariants: ASCII formulas as in §A; **no runtime ephemeris calls**.
- **CSV mode:** header/coverage/uniqueness checks PASS; **Nodes identity** holds daily: `Ketu(t) = wrap360(Rahu(t) + 180)`.
- **Manifest mode:** schema-valid; all `w_k, c_k, d_k` finite; **speed cap** respected (`A_speed_total_deg_per_day <= cap_planet`).
- **Boundary parity:** prev-day cusp convention observed; ±1-day snap only to resolve a clear neighbor mismatch.

Windows

- **Near-term:** any one civil year in the recent century.
- **Deep-time:** any one civil year in a distant century.

WARN bands

- **Jupiter/Saturn only:** WARN if a metric exceeds its PASS bound but stays within **+25%**; otherwise FAIL.
- **All other families:** WARN if within **+10%** of the PASS bound while caps OK; otherwise FAIL.

Family-specific numeric gates

Sun / Mercury / Venus / Mars (inners; free-n)

- **Near-term:** `phase_MAE_deg ≤ 1.5` **and** `rasi_cross_days ≤ 2.0` **and** `rasi_misclass_rate_percent ≤ 2.0`.
- **Deep-time:** `phase_MAE_deg ≤ 2.2` **and** `rasi_misclass_rate_percent ≤ 3.0`. *(Use manifest slope `n_deg_per_day` for `rasi_cross_days`.)*

Moon (fast; fixed-n)

- **All windows (primary):** `cusp_dist_MAE_deg ≤ 1.5` **and** `rasi_cross_days ≤ 1.0`.
- Degree MAE is secondary; crossings/cusps are authoritative.

Jupiter / Saturn (medium outers; fixed-n) — *(aligns with Appendix M & H)*

- **Near-term PASS:** `MAE_deg ≤ 2.0`, `P90_deg ≤ 4.0`, `rasi_cross_days ≤ 10`, `station_date_MAE_days ≤ 10`, `rasi_misclass_rate_percent ≤ 12`.
- **Deep-time PASS:** same metrics; accept PASS if each metric ≤ its PASS bound; classify **WARN** up to **+25%**, else **FAIL**. *(Use `n = 360/P_sid` for cross-days scaling.)*

Uranus / Neptune (slow outers; fixed-n)

- **Near-term:** `phase_MAE_deg ≤ 3.0` **and** `rasi_misclass_rate_percent ≤ 2.5` **and** `cusp_dist_MAE_deg ≤ 3.0`.
- **Deep-time:** `phase_MAE_deg ≤ 6.0` **and** `rasi_misclass_rate_percent ≤ 5.0` **and** `cusp_dist_MAE_deg ≤ 4.0`.
- `rasi_cross_days` may be **N/A** if `< 3` matched crossings (accepted).

Nodes (Rahu / Ketu)

- **Structural:** daily identity must hold: `Ketu = wrap360(Rahu + 180)`.
- **Near-term:** `phase_MAE_deg ≤ 3.0` **and** *(when ≥3 events)* `rasi_cross_days ≤ 12`.
- **Deep-time:** `phase_MAE_deg ≤ 6.0` **and** *(when ≥3 events)* `rasi_cross_days ≤ 12`.
- If crossings are insufficient, omit the crossing metric (no penalty); focus on phase + node symmetry.

Retro window sanity (INFO; optional WARN)

- Report `%_retro_days := 100 * mean( n_eff(t) < 0 )` and `max_retro_run_days`.
- Optionally WARN if either exceeds planet-specific policy bands. **No FAIL by default.**

F) One-line result format (per body × window)

```
BODY | WINDOW | MAE_deg | P95_deg | phase_MAE_deg | phase_P95_deg
     | rasi_misclass_% | cusp_dist_MAE_deg | rasi_cross_hours | cap_ok |
RESULT
```

*Badge semantics: **[PASS]**, **[WARN]**, **[FAIL]**.*

G) Acceptance summary (document footer snippet)

- **ACCEPTED (v2.1):** all structural gates PASS, and each body reaches at least **WARN** in both windows with **≥1 PASS per body**, and **no FAIL**.
- **REJECTED:** otherwise; list failing gates succinctly.

Appendix B — Method Details — OLS, Mid-Anchor, Event Metrics & Gates (formal)

Objective. Specify the exact fitting, scoring, and selection rules that produce a locked transit kernel per planet, with no hidden knobs.

B.1 Timebase, Units, Wrapping

All training and evaluation are in sidereal degrees (Lahiri).

- Dates are UTC-less civil dates; one sample per day at a fixed civil time (IST `05:30` in our runs).
- Let `t0` be the midpoint (integer day) of the train window.
- Days since anchor: `t := (date - t0)` in days (signed integer or float).
- Frequencies `w` are in radians/day; trigs use radians.

Wrap helpers:
`wrap360(x) = x - 360 * floor(x / 360)`

Unwrap (forward-carry) for OLS & event timing:

```
unwrap_series(L_deg_list):
    y = []; k = 0
    for i, x in enumerate(L_deg_list):
        if i == 0: y.append(x); continue
        jump = x - L_deg_list[i-1]
        if jump > +180: k -= 1
        elif jump < -180: k += 1
        y.append(x + 360*k)
    return y
```

Why mid-anchor? It de-correlates intercept and harmonic phases, reducing long-span phase creep.

B.2 Model Families (candidate sets)

Fixed-n family (Moon, Jupiter, Saturn, Uranus, Neptune, Pluto)

`n = 360 / P_sid` (deg/day; locked to physics)

```
L_hat_unwrapped(t) = a0 + n*t + sum_j[ c_j*sin(w_j*t) + d_j*cos(w_j*t) ]
L_hat_wrapped(t)   = wrap360( L_hat_unwrapped(t) )
```

Free-n family (Sun, Mercury, Venus, Mars)
Slope is learned once and stored in the manifest:

```
L_hat_unwrapped(t) = a0 + b1*t + sum_j[ c_j*sin(w_j*t) + d_j*cos(w_j*t) ]
# at export: n_deg_per_day := b1
```

Nodes
Model Rahu; derive Ketu by symmetry:
`L_ketu = wrap360(L_rahu + 180)`

Planet-specific base and gated extras (ASCII)

```
# Jupiter/Saturn/Uranus/Neptune/Pluto
BASE   = { w1 = 2*pi/P_syn , 2*w1 , wS = 2*pi/P_sid }
EXTRAS = { nE = 2*pi/365.2564 , 3*w1 }

# Moon (fast)
BASE   = { wS = 2*pi/P_sid , wA = 2*pi/P_anom }
EXTRAS = { wD = 2*pi/P_drac , nE }

# Sun/Inner (free-n)
BASE   = { w_syn , 2*w_syn }
EXTRAS = { 3*w_syn , abs(w_syn - nE) , nE }
```

Design rule: estimate only the listed coefficients. For fixed-n, `n` is exactly `360/P_sid` (not fitted).

B.3 Training Data Preparation

1. Slice a reference daily CSV to the declared train range (inclusive) — geocentric, sidereal (Lahiri), `05:30 IST` — from a public ephemeris (observation-only) or your internally constructed tables.
2. Use the declared cadence (daily recommended).
3. **Unwrap** the angle column for OLS and event detectors.
4. Center time with `t = days_since(date, t0_midpoint).`

B.4 Ordinary Least Squares (OLS)

Fixed-n (Moon/outer)
```
y_i := L_actual_unwrapped(t_i) - n*t_i
X_i := [ 1, sin(w1*t_i), cos(w1*t_i), sin(w2*t_i), cos(w2*t_i), ... ]
beta := argmin_beta || y - X*beta ||_2^2
```
(beta = `[ a0, c*, d* ]` in degrees)

Free-n (Sun/inner)
```
y_i := L_actual_unwrapped(t_i)
X_i := [ 1, t_i, sin(w1*t_i), cos(w1*t_i), ..., sin(wm*t_i), cos(wm*t_i) ]
beta := argmin_beta || y - X*beta ||_2^2
```
(beta = `[ a0, b1, c*, d* ]`; `b1` in deg/day)

Numerical guards

- Float64 throughout.
- If `RSS/nobs < 1e-16`, use `1e-16` in logs.
- Ensure all `w` are distinct by `>= 1e-9` rad/day; reject near-aliasing candidates.
- No regularization; QR/SVD acceptable.

B.5 Information Criterion + Event-Aware Gate

For each candidate set:

```
RSS = || y - X*beta_hat ||_2^2
k   = 1 + 2*m          # fixed-n: intercept + 2 per harmonic
k   = 2 + 2*m          # free-n : + slope term during fit

BIC = k*log(nobs) + nobs*log( max(RSS/nobs, 1e-16) )
```

Auxiliary event-aware loss (degree still dominant):

```
v_hat = | d/dt L_hat_unwrapped |     # central difference, 1-day step
v_act = | d/dt L_actual_unwrapped |
mae_v = mean( | v_hat - v_act | )

(MAE_deg_train, cusp_MAE_train) = event_metrics(TRAIN)
loss = 1.0*MAE_deg_train + 0.3*cusp_MAE_train + 0.4*mae_v
```

Strict inclusion gate for extras:
Admissible **IFF** `BIC_extra <= BIC_base - 6.0` **AND** `loss_extra < loss_base`
Selection: choose the admissible candidate with smallest `(loss, then BIC)`; else choose `BASE`.

Rationale: `ΔBIC ≥ 6` is strong evidence; the event-loss preserves crossings/stations fidelity.

B.6 Event Metrics (Train & Test)

All timing on **unwrapped** trajectories; wrap only for display.

B.6.1 Degree error

```
wrap180(d) = ((d + 180) % 360) - 180
err_deg[i] = | wrap180( L_hat_wrapped[i] - L_wrapped[i] ) |
MAE_deg = mean(err_deg)
P90_deg = percentile(err_deg, 90)
MAX_deg = max(err_deg)
```

B.6.2 Rāśi misclassification

```
rasi(x_deg) = floor( (x_deg % 360) / 30 )
misclass_rate = mean( rasi(L_hat_wrapped) != rasi(L_wrapped) )
```

B.6.3 Cusp-distance MAE

```
cusp_dist(x) = min( x % 30 , 30 - (x % 30) )
cusp_dist_MAE_deg = mean( | cusp_dist(L_hat_wrapped) - cusp_dist(L_wrapped)
| )
```

B.6.4 30° crossings (timing)

```
detect_crossings(t, L_unwrapped, step=30):
    for each level k*step spanning coverage:
        s[i] = L_unwrapped[i] - k*step
        if s[i]*s[i+1] <= 0:
            linearly interpolate time between i and i+1
    return list of (level, time_est)

pairing(actual_list, model_list, max_gap_days):
    greedy nearest-neighbor by |Δt|
    accept if |Δt| <= max_gap_days; discard unmatched

rasi_cross_MAE_days = mean( |Δt| ) over pairs   # NaN if none
# Use max_gap_days = 60 for outers; = 3 for Moon.
```

B.6.5 Stations (retrograde turn timing)

```
smooth L_unwrapped with moving average (win = 7 days)
v[i] = (L[i+1] - L[i-1]) / 2
find local minima of |v[i]| with sign(v[i-1]) != sign(v[i+1])
quadratic refine time using (i-1,i,i+1) on |v| curve
dedupe events within 2 days
pairing(..., max_gap_days=90) -> station_date_MAE_days
```

Event-day normalization (boundary parity; comparison only)
After year-boundary normalization, compare cusps using the **prev-day** convention; allow a bounded **±1-day snap** only when it resolves a clear neighbor mismatch.

B.7 Selection Outputs and Locking

For the selected candidate, save a manifest with:

- `planet, t0, P_sid (or null for free-n), n_deg_per_day`
- optionally `P_syn`
- `selected_model`
- `omegas { name: w },beta { a0, b1(optional), c*, d* }`
- `notes "daily train YYYY..YYYY; mid-anchor; ΔBIC≥6 & event-loss gate"`

Also keep (for your own audits):

- single-row `backtest_summary.csv` with metrics
- `backtest_timeseries.csv` (`date, L_actual, L_model, deg_error, rasi_actual, rasi_model`)

Invariants

- Fixed-n: `n_deg_per_day = 360 / P_sid` (exact).
- Free-n: `n_deg_per_day = b1` from OLS.
- Trigs use radians; coefficients are degrees.
- Evaluator is one line:
  `a0 + n*t + sum(c*sin + d*cos)` *(or `a0 + b1*t + sum(...)`)* → *`wrap360`*.

B.8 Numerical Safeguards & Repro Notes

- Float64; do not mix degrees/radians.
- In logs: use `max(RSS/nobs, 1e-16)` to avoid `log(0)`.
- Differencing uses **unwrapped** series.
- Crossing/station pairing by nearest neighbor in time (not ordinal index).
- Moon: emphasize `cusp_dist_MAE_deg` and `rasi_cross_MAE_days` (degree MAE is secondary for a `~13 deg/day` body).
- If you add bootstrap/tie-breaks later, seed any randomness; base OLS is deterministic.

B.9 Minimal Pseudocode (selector)

```
candidates = [
  BASE,
  BASE + nE,
  BASE + 3*w1,
  BASE + nE + 3*w1
  # Moon: BASE + wD, +nE, +wD+nE
]

best = None
for C in candidates:
    if family == "fixed-n":
        y = L_unwrapped - n*t
        X = design_fixed_n(C, t)   # [1, sin(w*t), cos(w*t), ...]
    else:  # free-n
        y = L_unwrapped
        X = design_free_n(C, t)    # [1, t, sin(w*t), cos(w*t), ...]
    beta   = OLS(y, X)
    BIC_C  = compute_BIC(beta, X, y)
    loss_C = event_loss(beta, C)

    if C is BASE:
        base_BIC, base_loss = BIC_C, loss_C
        best = (C, beta, BIC_C, loss_C)
    else:
        if (BIC_C <= base_BIC - 6.0) and (loss_C < base_loss):
            if (loss_C < best.loss) or ((loss_C == best.loss) and (BIC_C <
best.BIC)):
                best = (C, beta, BIC_C, loss_C)

save_manifest(best)
score_TEST(best)
```

B.10 Acceptance Gates (default TEST goals)

Use these as default **near-term** TEST goals (align with the acceptance runner). Deep-time bands are looser and documented in the Acceptance Runner appendix.

Sun / Mercury / Venus / Mars (inners; free-n)

- `phase_MAE_deg <= 1.5`
- `rasi_cross_MAE_days <= 2.0`
- *(also track `misclass_rate`; prefer `<= 2%`)*

Jupiter / Saturn (medium outers; fixed-n)

- `MAE_deg <= 2.0`
- `rasi_cross_MAE_days <= 10`
- `station_date_MAE_days <= 10`
- `misclass_rate <= 12%`

Uranus / Neptune / Pluto (slow outers; fixed-n)

- `phase_MAE_deg <= 3.0`
- `cusp_dist_MAE_deg <= 3.0`
- `misclass_rate <= 2.5%`
- `rasi_cross_MAE_days` may be `NaN` if `< 3` matched crossings (accepted).

Moon (fast)

- `cusp_dist_MAE_deg <= 1.5`
- `rasi_cross_MAE_days <= 1.0`
- *(degree MAE is informative but secondary)*

Nodes (Rahu/Ketu)

- Node identity must hold daily: `Ketu = wrap360(Rahu + 180)`
- `phase_MAE_deg <= 3.5` (near-term)

Outcome. With these rules, any reader can rebuild kernels from a public daily CSV or internal tables, select the same model under the same criteria, and reproduce daily angles and event timings within rounding — **no runtime ephemeris, no hidden parameters**.

Appendix C — Reproducibility: 4100-year sweeps (2000–6100) and random deep-time checks (0001–6100), ephemeris-free

Purpose.

- Show how anyone can verify deep-time stability with no runtime ephemeris.
- Two probe types: (1) century-step 4100-year sweeps, (2) random long-range spot checks.
- Probes confirm **angle range**, **rāśi mapping**, and **node symmetry**.
- **Result summary (this release):** All probes **PASS** under the rules below.

Two ways to run these probes
CSV-only (public kit): uses `golden_all.csv` for the dates it covers.
Calculator mode (optional add-on): uses the manifest + one-line evaluator to cover any date (including 2000–6100 and 0001–6100).

Evaluator (reminder; ASCII, calculator mode)

```
t := days_since(D, t0)
y := a0 + n_deg_per_day*t + SUM_k( c_k*sin(omega_k*t) + d_k*cos(omega_k*t)
) # omegas in rad/day
L_hat_deg := wrap360(y)
wrap360(x) := x - 360*floor(x/360)
rasi := floor((L_hat_deg % 360)/30)
```

Use one sample per civil day at `05:30 IST`.

C.1 Century-step sweep (4100y, all bodies)

Goal: coherence over millennia at sparse cadence.

Date set (inclusive):
`dates := { 2000-01-01, 2100-01-01, ..., 6100-01-01 }` (step `+100` years)

Procedure (calculator mode):

```
for body in {Sun, Moon, Mercury, Venus, Mars, Jupiter, Saturn, Uranus,
Neptune, Rahu, Ketu}:
  for D in dates:
    L = L_hat_deg(body, D)    # evaluator above
    r = floor((L % 360)/30)
    assert 0 <= L < 360
    assert r == floor((L % 360)/30)
```

Expected size: 42 rows per body.
PASS rule (per body): `rows == 42` and all assertions hold.
Observed (this release): PASS for all 11 bodies.

CSV-only note: run the same checks on any century dates that fall inside your CSV's date span (identical assertions).

C.2 Random long-range spot checks (all bodies)

Goal: stress logic at extreme dates far from the mid-anchor.

Example set:
`random_dates := { 0001-01-01, 2100-01-01, 4000-01-01, 6100-01-01 }`

Procedure (calculator mode):

```
bad = 0
for body in {...11 bodies...}:
  for D in random_dates:
    L = L_hat_deg(body, D)
    r = floor((L % 360)/30)
    if not (0 <= L < 360): bad += 1
    if r != floor((L % 360)/30): bad += 1
assert bad == 0
```

Expected: total rows = `44` (11×4), `bad == 0`.
Observed (this release): rows = 44, bad = 0 → **PASS**.

CSV-only note: pick four representative dates **within** the CSV span and run the same assertions.

C.3 Node identity check (Rahu/Ketu)

Ketu is a `180°` phase-shift of Rahu by construction.

```
L_rahu := L_hat_deg("Rahu", D)
L_ketu := L_hat_deg("Ketu", D)
delta  := wrap360(L_ketu - L_rahu)
assert abs(delta - 180) <= 1e-6
```

PASS rule: `max_abs_error <= 1e-6 deg` across tested dates.
Observed (this release): ≤ `1e-6 deg`, **PASS**.

CSV-only note: compute `delta` from `golden_all.csv` per date and assert the same bound.

C.4 Minimal validators (portable; ASCII)

CSV angle/rāśi validator (conceptual one-liner):

```
range_ok := all( 0 <= L_hat_deg < 360 )
rasi_ok  := all( rasi == floor((L_hat_deg % 360)/30) )
print("PASS" if (range_ok and rasi_ok) else "FAIL")
```

Node delta validator (conceptual):

```
delta := wrap360(L_ketu - L_rahu)
ok    := abs(delta - 180) <= 1e-6
print("PASS" if ok else "FAIL")
```

Use radians for trig and degrees for coefficients; match wraps exactly.

C.5 Good practice and caveats

- **Calendar:** use proleptic Gregorian for historical/future dates.
- **Time convention:** one sample per civil day at `05:30 IST`.
- **Floating point:** allow tiny tolerances (e.g., `1e-12`) to avoid false negatives.
- **Scope of these probes:** logical stability (wrapping, rāśi mapping, node identity) over very long spans.
- **Minute-grade events:** for stations/ingresses at minute precision or extreme `|t|`, prefer a high-precision ephemeris for that narrow window.

C.6 Optional artifacts

- Per-body `4100y` CSVs as defined in **C.1**.
- A compact `random_checks.csv` (44 rows) per **C.2**.
- A lightweight ZIP bundling these probe CSVs as a companion to the main release.

Outcome. Using either the public CSV (for its date span) or the evaluator (full span), independent parties can reproduce deep-time checks for **Sun, Moon, Mercury, Venus, Mars, Jupiter, Saturn, Uranus, Neptune, Rahu, Ketu** with **no runtime ephemeris** and should observe **PASS** on the probes above.

C.7 Observation-only accuracy recipe (optional, v2.1)

To compare against a public ephemeris (**verification only**):

1. **Export a reference.**
   From a reputable public ephemeris, export **daily sidereal (Lahiri) longitudes** at **00:00 UTC** (this equals **05:30 IST** on the same civil date), for your chosen window and body set.
2. **Choose predictions.**
   Use **`golden_all_v2_1.csv`** (public) as the prediction set **or** evaluate your local kernel on the same dates (calculator mode).
3. **Compute metrics.**
   Use these exact helpers and formulas:

```
4. wrap360(x) := x - 360*floor(x/360)
5. wrap180(x) := ((x + 180) % 360) - 180
6. err_deg(t) := | wrap180( L_pred(t) - L_ref(t) ) |
7.
8. MAE_deg  := mean_t( err_deg(t) )
9. P95_deg  := percentile_95_t( err_deg(t) )
10.
11.  # scaling for crossing-time proxy
12.  # prefer phase_MAE_deg if you compute phase on [0,30); else MAE_deg
   is an acceptable coarse proxy
13.  rasi_cross_hours := 24 * phase_MAE_deg / |n_deg_per_day|
14.                    (or 24 * MAE_deg / |n_deg_per_day| if phase not
   available)
```

   Where **`n_deg_per_day`** is the body's mean sidereal daily motion:

   - **Fixed-n bodies:** `n = 360 / P_sid_days`
   - **Free-n bodies:** use the learned slope from the manifest
   - **If manifests aren't available (CSV-only):** estimate `n` robustly from predictions, e.g.
     `n_hat := median_t( | wrap180(L_pred(t) - L_pred(t-1)) | )`
     (deg/day)

15. **Confirm rāśi mapping.**
   `rasi_pred(t) := floor( (wrap360(L_pred(t))) / 30 )` should be consistent and in **0..11**.

One-liner (optional helper)
If you're using the public kit's helper script:

```
python scorecard_csv_only.py golden_all_v2_1.csv reference.csv PlanetName
```

Assumptions for `reference.csv`: geocentric, **sidereal (Lahiri)**, sampled daily at **05:30 IST / 00:00 UTC** with columns:

```
planet,date,L_actual_deg
```

Note. This is an **observation-only** cross-check for expectation-setting. It does **not** alter the ephemeris-independent runtime; for minute-grade events or extreme |t|, escalate to a high-precision ephemeris for that narrow interval.

Appendix D — Adoption & API (manifest → angles, CLI & batch)

Purpose. Make the minimal "SSM-Lite" kernel plug-and-play: **one JSON in, angles out** — no ephemeris calls.

D.1 Manifest contract (portable & language-agnostic)

- **File:** `ssm_params.json` (one per planet or node)
- **Required keys (family-aware):**
  - `planet` — `"Sun"|"Moon"|"Mercury"|"Venus"|"Mars"|"Jupiter"|"Saturn"|"Uranus"|"Neptune"|"Rahu"|"Ketu"`
  - `t0` — `"YYYY-MM-DD"` (midpoint anchor of TRAIN window)
  - `time_local` — `"HH:MM"` (use `"05:30"`)
  - `ayanamsa` — `"Lahiri"`
  - `n_deg_per_day` — `<number>` (`= 360 / P_sid_days` for fixed-n; learned slope for free-n)
  - `selected_model` — `"base"|"add_nE"|"add_3w"|"add_nE_3w"|...`
  - **One** harmonic encoding must be present:

A) Named families (preferred)

```
omegas: { w1?, w2?, w3?, wS?, nE?, wA?, wD? }        # radians/day
beta:   { a0_deg, [b1_deg_per_day],
          c1?,d1?, c2?,d2?, c3?,d3?, cS?,dS?, cE?,dE?, cA?,dA?, cD?,dD? }
# degrees
```

B) Minimal terms array (compact)

```
terms: [ { "w_rad_per_day": <number>, "c_sin": <deg>, "d_cos": <deg> }, ...
]
```

- **Units (invariants):**
  - `omegas.*`, `terms[].w_rad_per_day` → radians/day
  - `a0_deg`, `b1_deg_per_day`, all `c*`, `d*` → degrees
- **Conditional keys (family rules):**
  - `P_sid_days` — `<days>` or `null`; **present and >0** for fixed-n; `null/omit` for free-n
  - `P_syn_days` — optional, informational
  - `n_fixed` — optional boolean (`true` for fixed-n)

- **Family requirements:**
  - **fixed-n:** require `P_sid_days > 0`; set `n_deg_per_day = 360 / P_sid_days`; omit `beta.b1_deg_per_day` (or set equal to `n_deg_per_day` if you prefer explicitness).
  - **free-n:** `P_sid_days` may be `null/omitted`; require `beta.b1_deg_per_day = n_deg_per_day` (store the learned slope).
- **Pairing rule (named-family form):**
  - if `w1` in `omegas` → require `c1,d1` in `beta`
  - if `w2` → `c2,d2`; if `w3` → `c3,d3`; if `wS` → `cS,dS`; if `nE` → `cE,dE`; if `wA` → `cA,dA`; if `wD` → `cD,dD`
- **Evaluator mapping (ASCII):**
  - `a0_deg := beta.a0_deg`
  - `b1_deg := (beta.b1_deg_per_day if present) else n_deg_per_day`
  - `y_deg := a0_deg + b1_deg * t`
  - **Named families:**
    - for each `name` in `omegas`:
      `w := omegas[name]` (rad/day), `c := beta["c"+suffix(name)]`, `d := beta["d"+suffix(name)]`,
      `y_deg := y_deg + c*sin(w*t) + d*cos(w*t)`
  - **Minimal terms:**
    - for each `term` in `terms`:
      `y_deg := y_deg + term.c_sin*sin(term.w_rad_per_day*t) + term.d_cos*cos(term.w_rad_per_day*t)`
  - `L_hat_deg := wrap360(y_deg)`
    `wrap360(x) := x - 360*floor(x/360)`
- **Rahu/Ketu symmetry:** you may store only Rahu and compute Ketu as `L_ketu = wrap360(L_rahu + 180)`.
- **Invariant:** treat the manifest as read-only once published; **do not hand-edit numbers**.

D.2 Single-date evaluator (runtime rule)

Given calendar date `D` (local, `05:30 IST`):

```
# inputs from the manifest (per body)
t0                  # "YYYY-MM-DD"
a0_deg              # degrees
b1_deg_per_day      # degrees/day
terms = [ { w_rad_per_day, c_sin, d_cos }, ... ]   # optional

# compute days since t0 using the same daily timestamp convention (05:30
IST)
t := days_since(D, t0)          # days; integer or real

# unwrapped model in degrees
y_deg := a0_deg + b1_deg_per_day * t
for each term in terms:
    y_deg := y_deg + (term.c_sin * sin(term.w_rad_per_day * t)
                   +  term.d_cos * cos(term.w_rad_per_day * t))

# wrap to [0,360)
L_hat_deg := wrap360(y_deg)
```

```
# helpers
wrap360(x_deg) := x_deg - 360 * floor(x_deg / 360)
rasi(L) := floor( (L % 360) / 30 )    # 0..11
```

Conventions. Trig arguments are radians; all angles are degrees. Evaluator is date-based, deterministic, and requires **no** runtime ephemeris calls. Use the same daily timestamp convention (`05:30 IST`) as training/fitting.

D.3 Minimal CLI (reference)

Single date → stdout CSV

```
ssm-eval --manifest ssm_params.json --date 2035-10-03
# -> prints: date,L_hat_deg,rasi
```

Batch mode

```
ssm-eval --manifest ssm_params.json --in dates.csv --out angles.csv
# dates.csv: one ISO date per line
# angles.csv: date,L_hat_deg,rasi
```

Rāśi helper (ASCII)

```
rasi = floor( (L_hat_deg % 360) / 30 )
```

D.4 Standard outputs (for tools & pipelines)

- **Angles (deg):** `L_hat_deg ∈ [0,360)`
- **Rāśi:** integer `0..11`
- **Audit (when comparing to a reference):** `date,L_actual_deg,L_model_deg,deg_error,rasi_actual,rasi_model`

D.5 Embedding & interop notes

- **Radians vs degrees:** coefficients `a0,c*,d*` are **degrees**; `omegas` are **radians/day**.
- **Precision:** use float64.
- **Wrapping:** unwrap only for internal event timing (crossings/stations); wrap for display and rāśi.
- **Determinism:** the same manifest must reproduce the same outputs across languages.

D.6 Quick adapters (pseudocode)

JS/TS

```
function wrap360(x){ return x - 360*Math.floor(x/360); }
function daysSince(D_iso, t0_iso){
  const D = new Date(D_iso+"T00:00:00Z"), T0 = new
Date(t0_iso+"T00:00:00Z");
  return Math.round((D - T0)/86400000);
}
function evalAngle(m, D_iso){
  const t = daysSince(D_iso, m.t0);
  const w = m.omegas || {}, b = m.beta || {};
  let y = (b.a0 || 0) + m.n_deg_per_day * t;
  const add = (ci, di, wi) => {
    if (b[ci]!=null && b[di]!=null && w[wi]!=null){
      y += b[ci]*Math.sin(w[wi]*t) + b[di]*Math.cos(w[wi]*t);
    }
  };
  ["1","2","3"].forEach(k => add("c"+k, "d"+k, "w"+k));
  add("cS","dS","wS"); add("cE","dE","nE"); add("cA","dA","wA");
add("cD","dD","wD");
  return wrap360(y);
}
```

Python

```
from datetime import date
import math

def wrap360(x): return x - 360.0 * math.floor(x / 360.0)

def days_since(D_iso, t0_iso):
    d, t0 = date.fromisoformat(D_iso), date.fromisoformat(t0_iso)
    return (d - t0).days

def eval_angle(manifest, D_iso):
    t = days_since(D_iso, manifest["t0"])
    b = manifest.get("beta", {})
    y = float(b.get("a0_deg", 0.0)) + float(manifest["n_deg_per_day"]) * t
    for term in manifest.get("terms", []):
        y += term["c_sin"] * math.sin(term["w_rad_per_day"] * t) \
           + term["d_cos"] * math.cos(term["w_rad_per_day"] * t)
    w = manifest.get("omegas", {})
    def add(ci, di, wi):
        nonlocal y
        if ci in b and di in b and wi in w:
            y += b[ci] * math.sin(w[wi] * t) + b[di] * math.cos(w[wi] * t)
    for k in ("1","2","3"): add("c"+k, "d"+k, "w"+k)
    add("cS","dS","wS"); add("cE","dE","nE"); add("cA","dA","wA");
add("cD","dD","wD")
    return wrap360(y)
```

D.7 Versioning & manifests

- Include a semantic tag in `notes`, e.g.:
  `"train 2020..2024 daily; mid-anchor; DeltaBIC>=6 & event-loss gate; v2.1"`
- **Breaking changes:** add a new manifest alongside prior versions (never overwrite).

Outcome. With a single small JSON per planet/node and the evaluator rules above, you can produce daily sidereal longitudes (`L_hat_deg`) and rāśi values reproducibly across runtimes, languages, and pipelines — **no ephemeris dependency at runtime**.

Appendix E — Provenance & Licensing (sources, audit, packaging)

Purpose. Document where the data comes from, how to rebuild results end-to-end, and how artifacts may be shared—so the SSM-JTK ("SSM-Lite") kernels remain auditable and portable.

Standardization note (used throughout).
`rasi_cross_MAE_days := phase_MAE_deg / |n_deg_per_day|`
• Moon/Jupiter/Saturn: use `n := 360 / P_sid`.
• Sun/Mercury/Venus/Mars: use the learned `n_deg_per_day` stored with the kernel.
• Nodes (Rahu/Ketu): use the slope magnitude; Ketu mirrors Rahu.

E.1 Data sources (public, reproducible)

- **Public ephemeris service** (e.g., NASA/JPL "Horizons"): geocentric apparent longitudes, sampled **daily at 05:30 IST**.
  – Used for **observation-only** backtests and metrics (e.g., a recent decade).
  – Siderealization: Lahiri/Chitrapaksha; apply the same time/frame rules as in this document.
- **Internally constructed projected tables** (optional): compact traditional-method tables for seeding/diagnostics when you do not want a runtime ephemeris. Treat these as **projected** references if compared to daily "actuals."
- **Fixed frame:** geocentric, apparent, sidereal degrees in `[0,360)`.
  Good practice: keep both (a) the raw public-ephemeris CSV and (b) the **siderealized** copy you computed from it.

E.2 Licensing & attribution (external)

- **This work (text + example code):** recommend **MIT** for code and **CC-BY-4.0** for prose. Ship a clear **LICENSE** with every public artifact and in the repository root.
- **Observation-only comparisons:** when cross-checking angles against a public ephemeris, attribute plainly and state the verification scope.
  Boilerplate:
  "Angles for backtests were cross-checked (observation only) against a public ephemeris service (e.g., NASA/JPL SSD 'Horizons'). SSM-JTK kernels and the evaluator are our own works and are released under the stated licenses."
- **Projected tables:** if you publish any, state they are **internally generated** (do not redistribute third-party software/data; link readers to original providers).
- **Provenance & audit:** alongside each release, record the kernel tag, commit/checksum, fit windows, acceptance gates, and the exact public services used for observation-only comparisons. Keep these next to the artifacts so third parties can reproduce your results.
- **Scope reminder:** research/observation only. Classical outputs are obtained by collapse; any alignment values are metadata and never change magnitudes. For minute-precision events, use a high-precision ephemeris.

E.3 Rebuild-from-scratch recipe (deterministic)

1. **Fetch a decade CSV (per body).**
   Daily geocentric apparent longitudes at `05:30 IST`; siderealize to Lahiri. Save as a neutral name like `planet_YYYY_YYYY.csv`.
2. **Train/Test split (example).**
   `TRAIN: 2020-01-01..2024-12-31`, `TEST: 2015-01-01..2019-12-31`; anchor `t0 := midpoint(TRAIN)`.
3. **Run selector & backtest.**
   Use the model families and gated candidate sets specified in this document; no extras beyond those gates.
4. **Lock artifacts (no edits).**
   `ssm_params.json` (manifest), `backtest_summary.csv`, `backtest_timeseries.csv`.
5. **Evaluate anywhere.**
   Apply the one-line evaluator with the saved manifest—**no runtime ephemeris calls**.

E.4 Environment pinning (so results match)

- Python `3.10–3.13` (float64 throughout).
- Linear algebra: `numpy.linalg.lstsq` (or any stable OLS; no randomness).
- Time: treat inputs as civil days; `days_since` is whole days from `t0` (no timezone math after fetch).
- Trig domain: `omega` in **radians/day**; coefficients (`a0`, `c*`, `d*`) in **degrees**.

E.5 File layout & integrity (public vs private)

Public (minimal kit):

```
/public/
  golden_all.csv              # daily, sidereal (Lahiri); planets + nodes
  ssm_public_validate_ascii.py  # header/coverage/rāśi/node checks (one
screen)
  SHA256SUMS.txt              # sha256 for each public artifact
  README_public_sampler.txt     # how to run CSV-only checks and samplers
```

Private (developer):

```
/results_<planet>/
  ssm_params.json
  backtest_summary.csv
  backtest_timeseries.csv
```

Integrity: include `SHA256SUMS.txt` for the public kit and treat each manifest as **read-only** once released.

E.6 Determinism policy

- No randomized steps; OLS is deterministic.
- If you later add any bootstrap or tie-break logic for stability badges, fix the PRNG seed and record it in `notes` inside the manifest.

E.7 Packaging & redistribution

- **Public bundle (minimal):** `golden_all.csv`, `ssm_public_validate_ascii.py`, `SHA256SUMS.txt`, `README_public_sampler.txt`.
- **Private bundle (developer):** per-body manifest + summaries/timeseries (not required for public testing).
- **Version tags:** embed a semantic tag in filenames or in `notes` (e.g., `vMAJOR.MINOR`), while keeping prior releases intact for reproducibility.

E.8 Privacy & ethics

- Inputs are astronomical (no personal data).
- If you ever ship user-contributed tables, retain only what's necessary (`date, angle`) and strip extraneous metadata.

E.9 Change management

- **Manifest versioning:** append a new `ssm_params.json` for any change to TRAIN/TEST, candidate sets, or gates; never overwrite prior manifests.
- **Changelog (suggested fields):** `date, planet, train_range, test_range, candidate_set, selected_model, DeltaBIC_vs_base, loss_terms, notes.`
- **Deprecations:** keep older kernels available; mark superseded ones clearly in `README_public_sampler.txt.`

Outcome. With clear licensing, reproducible sourcing, and a minimal public kit, third parties can independently validate coverage, rāśi integrity, and node symmetry—and audit provenance end-to-end—without any runtime ephemeris dependency.

Appendix F — Limits, Ethics, and Safe Use

Why this exists. SSM-JTK ("SSM-Lite") is compact, deterministic, and auditable—yet it is still a symbolic kernel, not a physical N-body integrator. This appendix defines where it shines, where it tapers, and how to use it responsibly.

F.1 Intended use (scope)

- **Research & education.** Ephemeris-independent screening of angles/rāśi, trend scanning, reproducible back-tests.
- **Not for minute-exact work.** For arc-minute ingresses, eclipses, and topocentric event timing, escalate to a high-precision ephemeris for that narrow window.
- **Not advice.** No medical, legal, or investment advice. Treat outputs as analytical signals, not directives.

F.2 Accuracy envelopes & escalation

Scope. Envelopes describe how far a fitted kernel can be trusted without refresh. Expressed in observable metrics with simple tripwires.

Core metrics (ASCII).

- `MAE_deg := mean(|L_true_deg - L_pred_deg|)` on `[0,360)`
- Phase on `[0,30)`: `d := ((L_true_deg % 30) - (L_pred_deg % 30) + 15) % 30 - 15`
  `phase_MAE_deg := mean(|d|)`
- Cusp distance: `cusp_dist_deg := min_k |(L_pred_deg % 30) - 30*k|`
  `cusp_dist_MAE_deg := mean(cusp_dist_deg)`

- Crossings: `rasi_cross_err_days := t_cross_pred - t_cross_true`
  `rasi_cross_MAE_days := mean(|rasi_cross_err_days|)`
- Stations: `station_date_err_days := t_station_pred - t_station_true`
  `station_date_MAE_days := mean(|station_date_err_days|)`

Nominal envelope (near the training anchor).

- **Outer planets (Jupiter…Neptune, Nodes):** within train midpoint ±5–7 years, metrics typically track TEST scores.
- **Inner planets (Sun, Mercury, Venus, Mars):** within ±2–3 years.
- **Moon:** within ±1 year around the anchor.

Tripwires (escalate when any trips).

- **Outer:** `MAE_deg > 3.0` **OR** `rasi_cross_MAE_days > 12` **OR** `station_date_MAE_days > 12`
- **Moon:** `cusp_dist_MAE_deg > 2.0` **OR** `rasi_cross_MAE_days > 1.5`
- **Inner:** `phase_MAE_deg > 2.0` **OR** `rasi_cross_MAE_days > 3.0`

Boolean escalation (ASCII).

```
escalate_outer := (MAE_deg > 3.0)
               OR (rasi_cross_MAE_days > 12)
               OR (station_date_MAE_days > 12)

escalate_moon  := (cusp_dist_MAE_deg > 2.0)
               OR (rasi_cross_MAE_days > 1.5)

escalate_inner := (phase_MAE_deg > 2.0)
               OR (rasi_cross_MAE_days > 3.0)

escalate := escalate_outer OR escalate_moon OR escalate_inner
```

Protocol when `escalate = true`.

1. **Mark region:** set `REG=NOFIT` (no local fit) or `REG=MULTI` (multiple regimes suspected).
2. **Refresh:** retrain with a refreshed window centered on the affected interval (same evaluator, same `t0` convention).
3. **Verify:** recompute metrics; if still marginal, add one small harmonic or augment sampling to include event windows (stations/cusps).
4. **Cross-check (observation-only):** for that interval, optionally compare to a reputable public ephemeris to measure error; deployment remains ephemeris-independent.

Notes.

- Envelopes are advisory guides for refresh cadence.
- Keep the daily timestamp convention `05:30 IST` consistent across train/test/refresh.
- Prefer the smallest effective frequency addition when tightening envelopes.

F.3 Known failure modes (and mitigations)

- **Ayanamsa mismatch.** Mixing Lahiri with another ayanamsa appears as constant bias or cusp errors.
  Mitigation: pin **Lahiri** throughout; record in the manifest.
- **Wrap/unwrap mistakes.** Event detection on wrapped series yields ghost crossings.
  Mitigation: unwrap for OLS & events; wrap only for display.
- **Annual aliasing.** A naive annual term can overfit.
  Mitigation: require `ΔBIC >= 6` **and** improved event-loss; otherwise drop it.
- **Sparse anchors drift.** D1/D15 only can tilt phases.
  Mitigation: prefer daily training for back-tests; validate sparse fits on unseen dailies.
- **Moon near-collinearity.** Close families (sidereal/anomalistic/draconic) can alias.
  Mitigation: reject near-aliased `w`; keep `BASE_MOON` minimal unless gates decisively pass.
- **Derivative noise at stations.** Raw finite differences jitter minima.
  Mitigation: 7-day smoothing + quadratic refine + dedupe window.

F.4 Responsible reporting (how to present results)

- **Two-channel honesty.** Print `x := <m,a>` when alignment exists; avoid silently collapsing. If `|a| < 0.2`, add a caution tag.
- **Use flags over forcing:** `SIDED`, `OSC`, `MULTI`, `NOFIT` instead of brittle magic numbers.
- **Bounded claims.** Quote TEST metrics alongside forward projections for context.

F.5 Overfitting guardrails

- **Parsimony first.** Extras admitted only if **both**: `ΔBIC >= 6` **and** event-loss improves.
- **Mean-motion policy.**
  - **Fixed-n (Moon/Jupiter/Saturn):** never fit `n`; use `n := 360 / P_sid`.
  - **Free-n (Sun/Mercury/Venus/Mars):** learn a single slope once; freeze it (no runtime drift-fit).
- **Disjoint TEST.** Always report a disjoint past TEST window; never grade on TRAIN.

F.6 Reproducibility and auditability

- **One-file truth.** `ssm_params.json` + the one-line evaluator reproduce angles exactly.
- **Checksums & versions.** Ship SHA-256 sums; supersede with new zips—do not overwrite.
- **Provenance.** Record fetch commands, ranges, candidate set, and gate outcomes (BIC, loss).

F.7 Ethics & communication

- **No determinism narratives.** Present SSM-JTK as a bounded symbolic model with explicit uncertainty channels—not fate.
- **Cultural sensitivity.** Keep math transparent and value-neutral.
- **User autonomy.** Encourage independent replication; provide open scripts and data paths.

F.8 Privacy & data handling

- **No personal data** in astronomical inputs. If user tables appear, keep only `date`, `angle`.
- **Local computation.** Kernels evaluate offline; no network dependency once manifests are saved.

F.9 Safety rails (publish/no-publish switch)

Publish only if all hold:

- Acceptance thresholds for the public release are met on the stated TEST window.
- Manifests, summaries, and (when applicable) timeseries are present with checksums.
- Ayanamsa, frame, and sampling conventions are clearly stated.
  Otherwise, mark the run **internal / exploratory** and refrain from public claims.

F.10 When to hand off to a high-precision ephemeris

- Minute-grade ingress timing, eclipse windows, occultations, or litigation-grade audits.
- Any window where acceptance metrics breach escalation thresholds.
- Scenarios where topocentric specifics materially change outcomes.

Appendix G — Roadmap (Forward-Looking Only)

This roadmap lists *new* work that meaningfully improves accuracy, stability, and usability without compromising the ephemeris-independent core. Each item has a concrete acceptance target so it's easy for third parties to verify.

G.1 Per-Planet Harmonics (tiny, gated)

What: Evaluate adding a *few* small harmonics per body where they materially reduce event errors while preserving rāśi integrity.

Evaluator (unchanged):

```
L_hat_deg(t) = wrap360( a0 + n*t + SUM_k[ c_k*sin(w_k*t) + d_k*cos(w_k*t) ]
)
wrap360(x) = x - 360*floor(x/360)
```

Gate (must pass all):

- **Parsimony:** admit a new term only if `DeltaBIC >= 6` and `loss_extra < loss_base`, where
  `loss = 1.0*MAE_deg + 0.3*cusp_MAE + 0.4*MAE_speed`.
- **Amplitude cap:** `abs(c_k) <= 0.50` deg and `abs(d_k) <= 0.50` deg.
- **Speed budget:** `A_speed_total_deg_per_day = SUM_k( w_k * hypot(c_k, d_k) ) <= cap_planet`.
- **Rāśi stability:** `rasi_misclass_rate_percent` must not increase on TEST.

Acceptance (per body, TEST window):

- **Sun/inner:** `phase_MAE_deg` improves by `>= 10%` *or* `rasi_cross_MAE_days` improves by `>= 10%`.
- **Jupiter/Saturn:** `rasi_cross_MAE_days` improves by `>= 10%` with no regression in `MAE_deg`.
- **Uranus/Neptune/Nodes:** any admitted term must reduce `phase_P95_deg` by `>= 5%`.

Deliverable: updated manifest(s) with the new `(w, c, d)` triples; golden CSV stays unchanged.

G.2 "Derived" Companion Columns (CSV-only, no model change)

What: Publish an *optional* companion CSV that augments the golden day-grid with diagnostic features useful for research/visualization.

Columns & formulas (ASCII):

- `retro` → `1 if angdiff_deg(L(t), L(t-1)) < 0 else 0`
- `speed_deg_per_step` → `angdiff_deg(L(t), L(t-1))`
- `Hn_deg` → `wrap360(n * L_hat_deg)` for `n ∈ {2,3,4,5,7,9}`
  Helpers:
  ```
  angdiff_deg(a,b) = wrap360(a - b + 180) - 180
  wrap360(x) = x - 360*floor(x/360)
  ```

Acceptance: file passes header/coverage checks; `Ketu(t) = wrap360(Rahu(t) + 180)` remains true; base golden stays byte-for-byte identical.

G.3 Event-Aware Refinements (stations & cusps)

What: Improve timing around stations and 30° crossings *without* changing the core evaluator.

Refinements (analysis-only, not runtime dependencies):

- **Station time refine:** quadratic fit on `|v|` over `(t-1, t, t+1)` where `v(t) = angdiff_deg(L(t+1), L(t-1))/2`.
- **Crossing interpolation:** linear interpolation on the unwrapped track for each `k*30` target.

Acceptance: on a public TEST year, `station_date_MAE_days` and `rasi_cross_MAE_days` each improve by `>= 10%` for at least **three** of Mercury/Venus/Mars/Jupiter/Saturn, with no body regressing by >5%.

G.4 Robust Residual Fitting (noise-tolerant training only)

What: Make the fit less sensitive to occasional noisy days in the reference strip (observation-only fetches).

Method: re-estimate `beta` via Huber-style weighting in the *training* OLS loop (no runtime cost).
Rule: if robust and plain OLS pick different candidates, keep the simpler unless both parsimony and event-loss gates prefer the robust pick.

Acceptance: for at least **two** inner bodies, `phase_P95_deg` improves by `>= 7%` on TEST with identical or fewer terms.

G.5 Confidence Channel (alignment) Calibration v2

What: Standardize the mapping from kinematic cues to a daily confidence `a_out ∈ (-1,+1)` (annotation only; angles unchanged).

Proposed mapping (tunable weights):

```
v = |angdiff_deg(L(t), L(t-1))|
d_cusp = min( L(t)%30 , 30 - (L(t)%30) )
s_stat = clamp01(1 - v/v_thr)
s_cusp = exp( - (d_cusp/d0)^2 )
a_out = tanh( u0 - alpha*( w1*s_stat + w2*s_cusp ) )
```

Acceptance: calibrated thresholds yield fewer false LOW flags near routine retro loops (target: LOW false-positive rate drops by `>= 20%` on Mercury/Venus test years).

G.6 Century & Deep-Time Guardrails (automated)

What: Ship small, automated checks so long-span coherence remains demonstrable, ephemeris-free.

Probes:

- **Century sweep:** `D = {2000-01-01, 2100-01-01, ..., 6100-01-01}`
  Assertions per date: `0 <= L_hat_deg < 360`, `rasi = floor((L_hat_deg%360)/30)`.
- **Node identity:** `abs( wrap360(L_ketu - L_rahu) - 180 ) <= 1e-6` for all dates tested.

Acceptance: 100% PASS; produce tiny CSV proofs per body.

G.7 Optional Tiny Evaluator File (CLI)

What: If community asks for it, publish a single-file CLI (`ssm-eval`) that implements the one-line rule and prints `date,L_hat_deg,rasi`.

Evaluator (ASCII):

```
t = days_since(D, t0)
y = a0 + n*t + SUM_k( c_k*sin(w_k*t) + d_k*cos(w_k*t) )
L_hat_deg = wrap360(y)
```

Acceptance: bit-for-bit equality with the document's examples on a public test set.

G.8 Acceptance Runner v2 (CSV-only public checks)

What: Extend the public validator with a CSV-only scorecard so anyone can compute: `MAE_deg`, `P90_deg`, `cusp_dist_MAE_deg`, `rasi_cross_MAE_days`, `station_date_MAE_days`.

Acceptance: runner reproduces the release's scoreboard from the golden day-grid plus a user-supplied observation-only reference.

G.9 Envelope Monitoring & Refresh Policy

What: Automated alarms when simple envelopes are breached (no ephemeris at runtime; uses historical backtests).

Tripwires:

- **Outer:** `MAE_deg > 3.0` or `rasi_cross_MAE_days > 12` or `station_date_MAE_days > 12`.
- **Moon:** `cusp_dist_MAE_deg > 2.0` or `rasi_cross_MAE_days > 1.5`.
- **Inner:** `phase_MAE_deg > 2.0` or `rasi_cross_MAE_days > 3.0`.

Action: refresh TRAIN window around the breach and re-select with the existing gates.

Why these items?

- **Harmonics-per-planet** tightens event timing with *minimal* extra state.
- **Derived columns** empower the community to analyze retro loops and harmonics without touching the core file.
- **Event-aware refinements** improve the things practitioners actually read (crossings, stations).
- **Robust fitting** reduces sensitivity to occasional reference noise.
- **Confidence calibration** makes the trust band more informative without changing angles.
- **Guardrails & runner** keep the package self-auditing and easy to verify.

Appendix H — CI & Reporting Utilities (v2.1)

Purpose. Standardize lightweight, release-safe reporting so anyone can verify kernels, compare runs, and publish a clean summary without path or environment assumptions.

H.1 Principles (portable, external-ready)

- No internal paths; no dependency on private repos.
- Inputs/outputs are plain CSV/HTML with stable headers.
- Metrics and gates match **Appendix M.1 (metrics)** and **Appendix M.2 (SLO gates)**.
- Everything deterministic; no randomness; float64 math.

H.2 Scoreboard schema (one row per run)

Required headers (types in brackets):

- **run_id** [str] — short label (e.g., `jupiter_test_2015_2019`)
- **planet** [str] — one of `Sun,Moon,Mercury,Venus,Mars,Jupiter,Saturn,Uranus,Neptune,Rahu,Ketu`
- **era** [str] — freeform (`historic`,`forward`,`test`, etc.)
- **csv_source** [str] — identifier of truth CSV (e.g., `public_ephemeris_sidereal_daily`)
- **manifest_tag** [str] — tag/notes for the kernel used
- **mae_deg** [num] — mean(|wrap180(L_model − L_true)|) *(for inner planets, this column may carry **phase_MAE_deg**; see H.3)*
- **p90_deg** [num] — 90th percentile of err_deg
- **p95_deg** [num] — 95th percentile of err_deg
- **max_deg** [num] — max of err_deg
- **misclass_rate** [num] — mean(rasi_model ≠ rasi_true)
- **rasi_cross_MAE_days** [num] — phase_MAE_deg / |n_deg_per_day|
- **cusp_dist_MAE_deg** [num] — mean(|cusp(L_model) − cusp(L_true)|)
- **station_date_MAE_days** [num] — MAE on paired station dates
- **n_rows** [int] — number of daily samples
- **status** [str] — PASS iff acceptance gates met; else FAIL

Helpers (ASCII):

```
wrap360(x) = x - 360 * floor(x / 360)
wrap180(d) = ((d + 180) % 360) - 180        # Euclidean %
rasi(x)    = floor( wrap360(x) / 30 )
cusp(x)    = min( wrap360(x) % 30 , 30 - (wrap360(x) % 30) )
```

For **n_deg_per_day**: fixed-n uses `360 / P_sid`; free-n uses the manifest slope.

H.3 Status rule (planet-aware gates)

Set `status := "PASS"` iff (per **Appendix M.2**):

- **Jupiter / Saturn:** `mae_deg ≤ 2.0`, `rasi_cross_MAE_days ≤ 10`, `station_date_MAE_days ≤ 10`, `misclass_rate ≤ 0.12`.
- **Sun / Inner (Mercury, Venus, Mars):** use **phase_MAE_deg** for gating (store it in the `mae_deg` column if you don't carry a separate field): `phase_MAE_deg ≤ 1.5` **and** `rasi_cross_MAE_days ≤ 2.0`.
- **Moon:** `cusp_dist_MAE_deg ≤ 1.5` **and** `rasi_cross_MAE_days ≤ 1.0`.
- **Nodes (Rahu, Ketu):** `phase_MAE_deg ≤ 3.0` **and** `rasi_cross_MAE_days ≤ 12`.

Insufficient crossings exception:
If a TEST window has `< 3` matched 30° crossings, leave `rasi_cross_MAE_days` blank and annotate `INSUFFICIENT EVENTS`; do **not** gate status on that metric for that run.

H.4 Normalization & rounding

- Round on print only (keep internal full precision):
    - `mae_deg, p90_deg, p95_deg, cusp_dist_MAE_deg` → **3 decimals**
    - `rasi_cross_MAE_days, station_date_MAE_days` → **2 decimals**
    - rates (e.g., `misclass_rate`) → **3 decimals**
- Clamp tiny negatives to `0.000`.
- If a metric is N/A (no events paired), leave the field **empty** — never invent zeros.

H.5 CSV → HTML micro-summary (portable behavior)

Goal: turn any scoreboard CSV into a single HTML file with a sortable-looking table and PASS/FAIL badges (no external assets).

Behavioral spec:

- Input: `--in scoreboard.csv`
  Output: `--out summary.html`
  Optional: `--title`
- Preferred column order in HTML:
  `run_id, planet, era, status, mae_deg, p95_deg, rasi_cross_MAE_days, n_rows, csv_source, manifest_tag`
- Numbers right-aligned; **status** as a colored badge (PASS green, FAIL red).
- No external CSS/JS; embed minimal styles.

Minimal algorithm:

```
read CSV → detect headers → keep preferred order + append extras → render
<table> with inline CSS → write HTML
```

CLI pattern:

```
ci_csv_to_html --in scoreboard.csv --out summary.html --title "SSM-JTK v2.1
CI Summary"
```

H.6 Aggregation & drift watch (robust stats)

```
med      = median(values)
mad      = median(|values − med|)
z_robust = 0.6745 * (value − med) / max(mad, 1e-9)
```

Flag potential drift if `|z_robust| > 3.5` on any **gated** metric.
Add `outlier_reason` per row (e.g., `p95_deg_high`, `cross_MAE_days_high`).

H.7 Recompute-don't-trust (key conversions)

Always recompute from primitives:

- `rasi_cross_MAE_days = phase_MAE_deg / |n_deg_per_day|`
- `misclass_rate = mean(rasi_model != rasi_true)` with `rasi(x) = floor(wrap360(x)/30)`
- `err_deg = |wrap180(L_model - L_true)|` → derive mean/percentiles/max from this vector

H.8 Minimal per-run recipe (append one row)

1. Evaluate the manifest on the TEST span → `L_model_deg`.
2. Align by date with the reference → `L_true_deg` (observation-only CSV).
3. Compute `err_deg`, `rasi_model`, `rasi_true`, cusp distances, crossings, stations.
4. Derive all metrics listed in **H.2**.
5. Set **status** via **H.3** gates.
6. Append one CSV line with the headers from **H.2**.

H.9 Example headers + sample row (illustrative)

Headers:
`run_id,planet,era,csv_source,manifest_tag,mae_deg,p90_deg,p95_deg,max_deg,misclass_rate,rasi_cross_MAE_days,cusp_dist_MAE_deg,station_date_MAE_days,n_rows,status`

Sample row:
`jupiter_test_2015_2019,Jupiter,test,public_ephemeris_sidereal_daily,midanchor-A,0.880,1.760,2.410,4.950,0.011,6.24,,7.80,1826,PASS`
(For **Moon**, fill `cusp_dist_MAE_deg`; for **Jupiter** it may be blank.)

H.10 HTML rendering cues (readability)

- **Bold** `run_id` and `planet`.
- Color badges only for **status**.
- Optional row shading by planet family.
- Keep numbers as plain text (no locale separators).

H.11 Integrity & provenance of the scoreboard

- Ship a **CHECKSUMS.sha256** line for the scoreboard and any per-planet backtest CSVs.
- HTML footer suggestion: `Generated from scoreboard.csv • rows=<N> • SSM-JTK v2.1`.
- Avoid absolute paths in artifacts.

H.12 Common pitfalls (and quick fixes)

- **Mixed units:** compute derivatives on **unwrapped** series; wrap only for display.
- **Wrong n for crossing-days:** use manifest `n_deg_per_day` for free-n; `360/P_sid` for fixed-n.
- **Empty station set:** if none in window, leave `station_date_MAE_days` blank (don't zero).
- **Formatting drift:** round only at print; keep full precision internally.

Outcome. With this appendix, a reviewer can take any set of per-day angle comparisons, regenerate a uniform scoreboard, and publish an HTML summary that matches the acceptance logic and metrics defined elsewhere—without internal paths or hidden assumptions.

Appendix I — Trust/Alignment & Soft Governor (concise)

Purpose. Formalize an optional alignment band `a_out ∈ (-1,+1)` and a gentle "soft governor" that operate **purely on the manifest projection**—no truth ephemeris—so UIs can display confidence (LOW/MID/HIGH), glide near stations/cusps, and avoid sticky behavior, while leaving the kernel's deterministic angles intact.

I.1 Inputs & scope (manifest-only)

- Daily series `L_pred_deg[D]` from the evaluator (sidereal/Lahiri; 05:30 IST).
- Calendar year `Y = year(D)`.
- No external angles are used; everything derives from `L_pred_deg`.

I.2 Helpers (wraps, day-step features)

- `wrap360(x) := x - 360*floor(x/360)`
- `wrap180(x) := ((x + 180) % 360) - 180`
- `v_deg_per_day[t] := abs( wrap180( L_pred[t] - L_pred[t-1] ) )`
- `r_prev[t] := L_pred[t-1] % 30`
- `d_cusp_prev[t] := min( r_prev[t], 30 - r_prev[t] )`

Notes. `v_deg_per_day` uses predicted steps. `d_cusp_prev` looks back one day to stabilize gating.

I.3 Component alignments (bounded in (-1,+1))

- Far-era shape (keeps confidence moderate far from anchor):
  `a_far[t] := tanh( (Y[t] - year_cut) / year_width )`
  (suggested: `year_cut = 2050`, `year_width = 30`*)*
- Station-likeness (slow steps → lower confidence):
  `a_stat[t] := 1 - min( 1, v_deg_per_day[t] / v_thr )`
- Cusp proximity (near 30° edges → lower confidence):
  `a_cusp[t] := 1 - min( 1, d_cusp_prev[t] / d0 )`
- Clamp for rapidity math:
  `clamp(z) := max(-0.999, min(+0.999, z))`
- Weights (planet-tunable, default unity): `w_far = 1.0`, `w_stat = 1.0`, `w_cusp = 1.0`.

I.4 Pooled alignment and confidence

- Rapidity-mean pooling:
  ```
  U[t] := w_far*atanh(clamp(a_far[t])) + w_stat*atanh(clamp(a_stat[t]))
  + w_cusp*atanh(clamp(a_cusp[t]))
  W := w_far + w_stat + w_cusp
  a_out[t] := tanh( U[t] / W )
  ```
- Confidence in `[0,1]`:
  `conf[t] := 0.5*( a_out[t] + 1 )`

(UI may display either `a_out` *or* `conf`*; both are manifest-only.)*

I.5 Soft governor (glide on the predicted track)

- Gains:
  ```
  g_conf[t] := conf[t]^gamma
  g_stat[t] := min( 1, v_deg_per_day[t] / v_thr_gate )
  g_edge[t] := min( 1, d_cusp_prev[t] / d_glide_deg )
  g_eff[t] := g_conf[t] * g_stat[t] * g_edge[t]
  ```

- One-step glide update:
  `delta[t] := wrap180( L_pred[t] - L_adj[t-1] )`
  `L_adj[t] := wrap360( L_adj[t-1] + g_eff[t]*delta[t] )`
- Initialization & floors:
  `L_adj[0] := wrap360( L_pred[0] + seed_deg )`
  `g_conf := max(g_conf, g_conf_floor)`
  `g_edge := max(g_edge, g_edge_floor)`

Recommended starters: `gamma = 2.0`, `v_thr_gate = 0.20 deg/day`, `d_glide_deg = 3.0 deg`, `seed_deg = 0.10 deg`, `g_conf_floor = 0.05`, `g_edge_floor = 0.02`.

Intuition. Near stations (`v → 0`) and cusps (`d_cusp_prev → 0`), `g_eff` shrinks and `L_adj` glides; when `a_out` is high, `g_conf` boosts tracking.

I.6 Trust calendar (LOW/MID/HIGH)

- Bands from `a_out`:
  `LOW if a_out < low`, `HIGH if a_out > high`, `MID otherwise`
  (suggested thresholds: `low = 0.30`, `high = 0.70`)
- Windowization fields per contiguous run:
  `bucket, start_date, end_date, days, station_ratio, cusp_ratio`
  where `station_ratio := (# days with v_deg_per_day <= v_thr) / days`,
  and `cusp_ratio := (# days with d_cusp_prev <= d0) / days`.

I.7 Planet-tuned starters (thresholds only; common governor)

Use these for `v_thr` and `d0` (trust bands in §I.6), then refine after first pass:

- **Sun:** `v_thr = 1.00 deg/day`, `d0 = 3.0 deg`
- **Moon:** `v_thr = 13.0 deg/day`, `d0 = 2.0 deg`
- **Mercury:** `v_thr = 4.00 deg/day`, `d0 = 2.0 deg`
- **Venus:** `v_thr = 0.20 deg/day`, `d0 = 2.0 deg`
- **Mars:** `v_thr = 0.50 deg/day`, `d0 = 4.0 deg`
- **Jupiter:** `v_thr = 0.06 deg/day`, `d0 = 6.0 deg`
- **Saturn:** `v_thr = 0.03 deg/day`, `d0 = 8.0 deg`
- **Uranus:** `v_thr = 0.010 deg/day`, `d0 = 2.0 deg`
- **Neptune:** `v_thr = 0.0050 deg/day`, `d0 = 0.5 deg`
- **Rahu/Ketu:** `v_thr = 0.06 deg/day`, `d0 = 6.0 deg`

Tip. If HIGH belts show `station_ratio ≈ 1.0`, reduce `v_thr`. If HIGH belts hug cusps, reduce `d0`.

I.8 Daily diagnostics (canonical columns)

- `date, L_pred_deg, L_adj_deg, a_out, conf, g_conf, g_stat, g_edge, g_eff, v_deg_per_day, d_cusp_prev, bucket`

(All derived from `L_pred_deg`; no external angles.)

I.9 Safety & intent

- `a_out/conf` are **quality signals** for the projection, not physical event detectors.
- The governor is **soft**: it glides but never rewrites the kernel's long-term phase.
- Use LOW/MID/HIGH to guide attention; escalate to a high-precision ephemeris for minute-grade needs.

I.10 Minimal single-pass outline (manifest-only)

- `v[t] := abs( wrap180(L_pred[t] - L_pred[t-1]) )`
- `dcp[t] := min( (L_pred[t-1] % 30), 30 - (L_pred[t-1] % 30) )`
- `a_far := tanh( (Y[t] - year_cut)/year_width )`
- `a_stat := 1 - min(1, v[t]/v_thr)`
- `a_cusp := 1 - min(1, dcp[t]/d0)`
- `U := w_far*atanh(clamp(a_far)) + w_stat*atanh(clamp(a_stat)) + w_cusp*atanh(clamp(a_cusp))`
- `a_out[t] := tanh( U / (w_far + w_stat + w_cusp) )`
- `conf := 0.5*(a_out[t] + 1)`
- `g_conf := max(conf^gamma, g_conf_floor)`
- `g_stat := min(1, v[t]/v_thr_gate)`
- `g_edge := max(min(1, dcp[t]/d_glide_deg), g_edge_floor)`
- `g_eff := g_conf * g_stat * g_edge`
- `delta := wrap180(L_pred[t] - L_adj[t-1])`
- `L_adj[t] := wrap360( L_adj[t-1] + g_eff*delta )`
- `bucket[t] := (a_out[t] < low ? "LOW" : (a_out[t] > high ? "HIGH" : "MID"))`

Outcome. `a_out` (or `conf`) and the soft-governed `L_adj_deg` provide a consistent, ephemeris-independent overlay for trust bands and UI glides, without altering the deterministic kernel.

Appendix J — Golden Vectors & Signatures (v2.1)

Purpose. Freeze a tiny, immutable reference so anyone can verify an evaluator and a manifest without any ephemeris. A “golden” is a 12-date sample per body (angles + rāśi), plus checksums and detached signatures for integrity.

J.1 What a “golden” is (contract)

Each body ships a single CSV with 12 rows:

- 3 near the TEST span (past), 3 near the TRAIN span (recent), 3 far-past, 3 far-future.
- Angles are produced only by the one-line evaluator (Appendix D.2 / §4.2).
- No external ephemeris, no runtime tuning, no gates.

Bodies covered. The validator accepts whatever appears in the golden (e.g., Sun, Moon, Mercury, Venus, Mars, Jupiter, Saturn, Uranus, Neptune, Rahu, Ketu).

CSV schema (minimal, UTF-8, comma-separated):

```
planet,date,L_hat_deg,rasi
```

- `planet` — case-insensitive name
- `date` — `YYYY-MM-DD` (civil day)
- `L_hat_deg` — angle in `[0,360)` (print 6–8 decimals)
- `rasi` — integer in `0..11`

(Optional columns like `tag` or `schema_version` may be present and are ignored by the validator.)

J.2 Exact angle & rāśi rules

Wrap

```
wrap360(x) := x - 360*floor(x/360)
```

Evaluator (free-n)

```
y := a0 + n_deg_per_day*t + Σ[c_k*sin(ω_k*t) + d_k*cos(ω_k*t)]
L_hat_deg := wrap360(y)
```

Evaluator (fixed-n)

```
y := a0 + (360/P_sid)*t + Σ[c_k*sin(ω_k*t) + d_k*cos(ω_k*t)]
L_hat_deg := wrap360(y)
```

Rāśi (Euclidean modulus)

```
rasi := floor( (L_hat_deg % 360) / 30 )
```

Tie rule at exact multiples of 30° (classification):
If `|L_hat_deg − 30*k| ≤ 5e−12` deg, assign the **higher** rāśi; `k=12` maps to `rasi=0`.
(Examples: `0°→0`, `30°→1`, …, `330°→11`, `360°≡0°→0`.)

J.3 Default 12-date pack (example)

Use or adapt the following (keep 12 total per body):

- **Near TEST (past):** `2016-03-01, 2017-09-01, 2019-03-01`
- **Near TRAIN (recent):** `2021-03-01, 2023-03-01, 2024-12-31`
- **Far past:** `1805-01-01, 1825-07-01, 1895-12-31`
- **Far future:** `2105-01-01, 2125-07-01, 2195-12-31`

Keep the 6 far-date rows **fixed** across releases for cross-release comparability; if TRAIN/TEST windows differ, replace the 6 "near" dates accordingly.

J.4 Numeric format & tolerances

- Print `L_hat_deg` with **6 decimals** (recommended) or **8** (strict).
- **Angle tolerance:** `|Δdeg| ≤ 1e−5`.
- **Rāśi** must match **exactly**.
- **Timebase:** `t := days_since(D, t0)` uses whole-day civil arithmetic; the daily timestamp convention (IST 05:30) is baked into training, not re-applied at runtime.

J.5 Checksums & detached signatures

Ship integrity metadata for every **public** file you distribute (e.g., golden CSVs, README; include manifests only if you publish them):

- A single `CHECKSUMS.sha256` (ASCII).
- Detached signatures per file (e.g., Ed25519 via minisign/age-signify, or OpenPGP).

Reference commands (copy-paste):

POSIX

```
sha256sum golden/*.csv > CHECKSUMS.sha256
# (add manifests/*.json to the command if publishing manifests)
```

Windows (CMD/PowerShell)

```
Get-FileHash -Algorithm SHA256 .\golden\*.csv | Out-File -Encoding ascii
CHECKSUMS.sha256
```

Sign / Verify (minisign)

```
minisign -S -s yourkey.minisign -m golden\planet_golden.csv
minisign -V -p yourpub.minisign -m golden\planet_golden.csv -x
golden\planet_golden.csv.minisig
```

Policy. Never overwrite signatures. New release → new files + new signatures. Pin public keys in the top-level README.

J.6 Validator CLI (contract)

Goal. One command proves that an evaluator + manifests reproduce the golden vectors.

Strict reproduction mode (manifests + golden):

```
python validate_golden_all.py ^
  --manifests path\to\manifests ^
  --golden golden_all_v2_1.csv ^
  --tol 1e-5
```

(Exit codes: `0=PASS, 1=FAIL, 2=IO/format`.)

Per-planet checks

1. Load the planet's `ssm_params.json`.
2. For each golden row, compute `L_hat_deg_eval` and `rasi_eval`.
3. Require `abs(L_hat_deg_eval - L_hat_deg_csv) ≤ tol_deg` **and** `rasi_eval == rasi_csv`.
4. Emit: `planet, n_rows, max_abs_err_deg, rasi_mismatch_count, status.`

PASS rule (per body)

- `n_rows == 12`
- `max_abs_err_deg ≤ tol_deg`
- `rasi_mismatch_count == 0`

CSV-only integrity check (no manifests) — verify angle range, rasi mapping, and node symmetry:

```
# Save as: csv_integrity_check.py  (ad hoc; not shipped in v2.1)
import csv, math, sys

def wrap360(x): return x - 360.0*math.floor(x/360.0)
def rasi_from_deg(L): return int(math.floor((wrap360(L) % 360.0)/30.0))

fn = sys.argv[1] if len(sys.argv) > 1 else "golden_all_v2_1.csv"
bad = 0
rahu, ketu = {}, {}

with open(fn, newline="", encoding="utf-8-sig") as f:
    r = csv.DictReader(f)
    for row in r:
        p = row["planet"].strip()
        d = row["date"].strip()
        L = float(row["L_hat_deg"])
        rasi_csv = int(row["rasi"])

        Lw = wrap360(L)
        if not (0.0 <= Lw < 360.0):
            bad += 1; print("range", p, d, L)

        if rasi_from_deg(Lw) != rasi_csv:
            bad += 1; print("rasi", p, d, L, rasi_csv)

        if p == "Rahu": rahu[d] = Lw
        if p == "Ketu": ketu[d] = Lw

shared = set(rahu) & set(ketu)
devmax = max((abs((ketu[d] - rahu[d] + 360.0) % 360.0 - 180.0) for d in
shared), default=0.0)

print("Nodes max|Δ-180|=%.9f deg" % devmax)
print("RESULT:", "PASS" if (bad == 0 and devmax <= 1e-6) else "FAIL")
```

Run:

```
python csv_integrity_check.py golden_all_v2_1.csv
```

Asserts:

- `0 <= L_hat_deg < 360` (after `wrap360`)
- `rasi == floor((L_hat_deg % 360)/30)`
- `Ketu == wrap360(Rahu + 180)` when both nodes are present

For full verification (angles vs evaluator), use the shipped validator with manifests:

```
python validate_golden_all.py --golden "golden_all_v2_1.csv" --manifests
".\manifests" --tol 1e-5
```

J.7 SDK conformance notes

- Float math: **float64**.
- Trigs use radians; `ω_k` are radians/day; coefficients `a0,c*,d*` are degrees.
- Wrapping must be exactly: `wrap360(x) := x − 360*floor(x/360)` (avoid `%` on negatives).
- Rāśi uses Euclidean modulus: `rasi := floor( (x % 360) / 30 )`.

J.8 Example rows (illustrative only)

```
planet,date,L_hat_deg,rasi
Jupiter,2016-03-01,123.456789,4
Jupiter,2105-01-01, 78.901234,2
```

(Values are placeholders. Ship body-specific numbers derived from your manifests.)

J.9 Golden bundle checklist (public)

- Per-body CSV (12 rows each; schema above), **or** a single `golden_all.csv` (concatenation; same header).
- `CHECKSUMS.sha256` (covers all **public** files; include manifests only if you publish them).
- Detached signatures (`*.minisig` or `*.sig`) for each file you publish.
- `README_golden.txt` with:
  - Tolerance rule (`1e-5` deg)
  - Rāśi tie rule
  - Exact date set used
  - Evaluator line and wrap/rāśi formulas
  - Verification examples (POSIX + Windows)
- • Validator script: `validate_golden_all.py` (public; uses manifests + golden). *(CSV-only integrity check is an ad hoc snippet, not included in v2.1.)*

J.10 Release artifact record (v2.1, minimal checksum)

Verify checksums (choose one):

Windows (CMD/Powershell)

```
certutil -hashfile golden\golden_all_v2_1.csv SHA256
```

or

```
Get-FileHash -Algorithm SHA256 .\golden\golden_all_v2_1.csv
```

POSIX

```
sha256sum golden/golden_all_v2_1.csv
```

Example SHA-256 (v2.1 golden):

```
golden_all_v2_1.csv
00808facbc298631018e8f4ca4b43d10be10bdccf47d674aa17f771e2be2326f
```

Acceptance rule: angle match ≤ `1e-5` deg after `wrap360`, and identical `rasi`.

J.11 Kit audit stamp (structural sanity)

Angle/rāśi internal check (one-liner):

```
python - <<"PY"
import pandas as pd, numpy as np
d = pd.read_csv("golden/golden_all_v2_1.csv", encoding="utf-8-sig")
r = (np.floor((d["L_hat_deg"]%360)/30)).astype(int)
bad = ((d["L_hat_deg"]<0)|(d["L_hat_deg"]>=360)|(r!=d["rasi"]))
print("rows=",len(d)); print("bad_rows=",int(bad.sum()))
print("PASS" if not bad.any() else "FAIL")
PY
```

Expected: **PASS** (`rows = 12 × #bodies`, `bad_rows = 0`).

Outcome. With `golden_all_v2_1.csv`, the per-body manifests (when used), the one-line evaluator, and the validator script, third parties can reproduce angles and rāśi on 12 dates per body to within `1e-5` deg — ephemeris-independent, portable, and deterministic by construction.

Appendix K — Cross-Language Conformance Pack (WASM / JS / Python)

Purpose. Guarantee identical angles/rāśi across runtimes by pinning a tiny reference evaluator, invariants, and pass/fail rules. No ephemeris, no tuning — just math.

K.1 Evaluator (contract; language-agnostic)

Inputs: manifest fields `{ t0, n_deg_per_day OR P_sid_days, omegas (rad/day), beta (deg) }` and calendar date `D`.

Core helpers

```
wrap360(x) := x - 360*floor(x/360)
wrap180(x) := ((x + 180) % 360) - 180 # Euclidean %
t := days_since(D, t0) # whole civil days
```

Angle

```
free-n : y := a0 + n_deg_per_day*t + Σ[ c_k*sin(ω_k*t) + d_k*cos(ω_k*t) ]
fixed-n: y := a0 + (360/P_sid_days)*t + Σ[ c_k*sin(ω_k*t) + d_k*cos(ω_k*t)
]
L_hat_deg := wrap360(y)
```

Rāśi

```
rasi := floor( (L_hat_deg % 360) / 30 ) # Euclidean %
```

Tie rule (exact cusp). If `|L_hat_deg - 30*k| ≤ 5e-12`, assign the **higher** rāśi; `k=12` wraps to `0`.

K.2 Invariants (must hold)

- Range: `0 ≤ L_hat_deg < 360`
- Determinism: same manifest + date → identical `L_hat_deg` within `1e-5 deg` across runtimes
- Rāśi stability: printing with 6 vs 8 decimals must not change rāśi
- Timebase: `t` uses whole days since `t0` (no timezone math after fetch)
- Units: `ω_k` in radians/day; `a0, c*, d*` in degrees; trig args are radians

---

K.3 Conformance suite (PASS rules)

- **Golden vectors:** 12 dates/body → `|Δdeg| ≤ 1e-5` and exact rāśi match
- **Cusps fuzz:** test `rasi()` on inputs `30*k + ε`, `ε ∈ {-1e-9, -1e-12, 0, +1e-12, +1e-9}` → tie rule honored; no flip-flop
- **Far-era drift:** evaluate far past/future dates → correct wrap and stable numbers
- **Leap day:** include `YYYY-02-29` → `t` has no off-by-one
- **Null harmonics:** absent `ω_k` → ignore its `c_k,d_k` (term contributes zero)

Exit codes: `0=PASS`, `1=FAIL`, `2=IO/format`.

K.4 Reference targets (minimal SDKs)

• **WASM/JS:** `evalAngle(manifest, date_iso) -> { L_hat_deg, rasi }` (float64 via WASM or JS number)
• **Python:** pure-stdlib mirror of JS; no external deps
• **CLI shims:** `ssm-eval` and `ssm-validate` shapes as described here

K.5 Performance sanity (non-binding)

• Interpretive target: ≥ `1e6` eval/s on a modern laptop CPU (≤ 6 harmonics)
• Memory-free: no cross-call state beyond constants

K.6 Reference implementations (copy-paste safe)

K.6.1 JavaScript / TypeScript (Node or browser)

```
// Euclidean modulus for reals
function emod(x: number, m: number): number { return ((x % m) + m) % m; }
export function wrap360(x: number): number { return x - 360 * Math.floor(x
/ 360); }
export function wrap180(x: number): number { return emod(x + 180, 360) -
180; }
export function rasiFromDeg(L: number): number {
  const Lw = wrap360(L);
  const k = Math.round(Lw / 30);
  if (Math.abs(Lw - 30 * k) <= 5e-12) return (k % 12 + 12) % 12;  // tie →
higher rāśi
  return Math.floor(emod(Lw, 360) / 30);
}
export function daysSince(dateIso: string, t0Iso: string): number {
  const toUTC = (s: string) => { const [y,m,d] = s.split("-").map(Number);
return Date.UTC(y, m-1, d); };
  return Math.round((toUTC(dateIso) - toUTC(t0Iso)) / 86400000);
}

type Manifest = {
  t0: string;
  n_deg_per_day?: number;
  P_sid_days?: number | null;
  omegas?: Record<string, number>;
  beta: Record<string, number>;
  terms?: { w_rad_per_day: number; c_sin: number; d_cos: number }[];
};

export function evalAngle(man: Manifest, dateIso: string): { L_hat_deg:
number; rasi: number } {
  const t = daysSince(dateIso, man.t0);
  const a0 = man.beta["a0_deg"] ?? man.beta["a0"] ?? 0;
  const n = man.beta["b1_deg_per_day"] ?? man.n_deg_per_day ??
(man.P_sid_days ? 360/(man.P_sid_days as number) : 0);
  let y = a0 + n * t;
```

```
  if (man.terms?.length) {
    for (const term of man.terms) y +=
term.c_sin*Math.sin(term.w_rad_per_day*t) +
term.d_cos*Math.cos(term.w_rad_per_day*t);
  } else if (man.omegas) {
    const w = man.omegas, b = man.beta;
    const add = (cKey: string, dKey: string, wKey: string) => {
      if (b[cKey]!=null && b[dKey]!=null && w[wKey]!=null)
        y += b[cKey]*Math.sin(w[wKey]*t) + b[dKey]*Math.cos(w[wKey]*t);
    };
    ["1","2","3"].forEach(k => add(`c${k}`, `d${k}`, `w${k}`));
    add("cS","dS","wS"); add("cE","dE","nE"); add("cA","dA","wA");
add("cD","dD","wD");
  }
  const L_hat_deg = wrap360(y);
  return { L_hat_deg, rasi: rasiFromDeg(L_hat_deg) };
}
```

K.6.2 Python (stdlib only)

```
import math
from datetime import date

def emod(x: float, m: float) -> float: return ((x % m) + m) % m
def wrap360(x: float) -> float: return x - 360.0 * math.floor(x / 360.0)
def wrap180(x: float) -> float: return emod(x + 180.0, 360.0) - 180.0
def rasi_from_deg(L: float) -> int:
    Lw = wrap360(L); k = round(Lw / 30.0)
    if abs(Lw - 30.0 * k) <= 5e-12: return k % 12
    return int(math.floor(emod(Lw, 360.0) / 30.0))
def days_since(date_iso: str, t0_iso: str) -> int:
    y,m,d = map(int, date_iso.split("-")); y0,m0,d0 = map(int,
t0_iso.split("-"))
    return (date(y, m, d) - date(y0, m0, d0)).days
def eval_angle(manifest: dict, date_iso: str) -> dict:
    t = days_since(date_iso, manifest["t0"]); b = manifest["beta"]
    a0 = b.get("a0_deg", b.get("a0", 0.0))
    if "b1_deg_per_day" in b: n = b["b1_deg_per_day"]
    elif "n_deg_per_day" in manifest: n = manifest["n_deg_per_day"]
    else: P_sid = manifest.get("P_sid_days"); n = 360.0 / P_sid if P_sid
else 0.0
    y = a0 + n * t
    terms = manifest.get("terms", [])
    if terms:
        for term in terms:
            w = term["w_rad_per_day"]; y += term["c_sin"]*math.sin(w*t) +
term["d_cos"]*math.cos(w*t)
    else:
        w = manifest.get("omegas", {})
        def add(ci, di, wi):
            nonlocal y
            if ci in b and di in b and wi in w: y +=
b[ci]*math.sin(w[wi]*t) + b[di]*math.cos(w[wi]*t)
        for k in ("1","2","3"): add(f"c{k}", f"d{k}", f"w{k}")
        add("cS","dS","wS"); add("cE","dE","nE"); add("cA","dA","wA");
add("cD","dD","wD")
    L = wrap360(y); return {"L_hat_deg": L, "rasi": rasi_from_deg(L)}
```

K.6.3 WASM (Rust sketch; compiles to f64)

```
use wasm_bindgen::prelude::*;
use serde::Deserialize;
use chrono::NaiveDate;

#[derive(Deserialize)] struct Term { w_rad_per_day: f64, c_sin: f64, d_cos:
f64 }
#[derive(Deserialize)] struct Manifest {
  t0: String, n_deg_per_day: Option<f64>, P_sid_days: Option<f64>,
  omegas: Option<serde_json::Map<String, serde_json::Value>>,
  beta: serde_json::Map<String, serde_json::Value>,
  terms: Option<Vec<Term>>,
}
fn emod(x: f64, m: f64) -> f64 { ((x % m) + m) % m }
fn wrap360(x: f64) -> f64 { x - 360.0 * (x / 360.0).floor() }
fn rasi_from_deg(l: f64) -> i32 {
  let lw = wrap360(l); let k = (lw / 30.0).round();
  if (lw - 30.0 * k).abs() <= 5e-12 { return ((k as i32) % 12 + 12) % 12; }
  (emod(lw, 360.0) / 30.0).floor() as i32
}
fn days_since(date_iso: &str, t0_iso: &str) -> i64 {
  (NaiveDate::parse_from_str(date_iso, "%Y-%m-%d").unwrap()
   - NaiveDate::parse_from_str(t0_iso, "%Y-%m-%d").unwrap()).num_days()
}
#[wasm_bindgen]
pub fn eval_angle_json(manifest_json: &str, date_iso: &str) -> JsValue {
  let m: Manifest = serde_json::from_str(manifest_json).unwrap();
  let t = days_since(date_iso, &m.t0) as f64;
  let a0 = m.beta.get("a0_deg").and_then(|v| v.as_f64())
           .or(m.beta.get("a0").and_then(|v| v.as_f64())).unwrap_or(0.0);
  let n = m.beta.get("b1_deg_per_day").and_then(|v| v.as_f64())
          .or(m.n_deg_per_day).or(m.P_sid_days.map(|p|
360.0/p)).unwrap_or(0.0);
  let mut y = a0 + n * t;
  if let Some(terms) = m.terms.as_ref() {
    for term in terms { y += term.c_sin*(term.w_rad_per_day*t).sin() +
term.d_cos*(term.w_rad_per_day*t).cos(); }
  } else if let Some(omegas) = m.omegas.as_ref() {
    let get = |k: &str| omegas.get(k).and_then(|v| v.as_f64());
    let b = &m.beta;
    let mut add = |ci: &str, di: &str, wi: &str| {
      if let (Some(c), Some(d), Some(w)) = (b.get(ci).and_then(|v|
v.as_f64()),
                                            b.get(di).and_then(|v|
v.as_f64()),
                                            get(wi)) {
        y += c*(w*t).sin() + d*(w*t).cos();
      }
    };
    add("c1","d1","w1"); add("c2","d2","w2"); add("c3","d3","w3");
    add("cS","dS","wS"); add("cE","dE","nE"); add("cA","dA","wA");
add("cD","dD","wD");
  }
  let L = wrap360(y);
  serde_wasm_bindgen::to_value(&serde_json::json!({"L_hat_deg": L, "rasi":
rasi_from_deg(L)})).unwrap()
}
```

K.7 Conformance harness (uniform tests)

Inputs

- `manifests/` (per-body JSON from this release)
- `golden_all_v2_1.csv` (12 rows per body; angles + rāśi)
- *(Optional)* `cusps_fuzz.csv` (angles near `30° * k` for the rāśi unit test)

Checks (must PASS)

1. **Golden reproduction** — For each row: compute `(L_hat_deg_eval, rasi_eval)` and require `abs(L_hat_deg_eval - L_hat_deg_csv) ≤ 1e-5` **and** `rasi_eval == rasi_csv`.
2. **Cusps fuzz** — For each degree in `cusps_fuzz.csv`, `rasiFromDeg(deg)` honors the tie rule (within `5e-12` deg → bump to higher rāśi); no oscillation across languages.
3. **Null harmonics** — Remove any optional `ω_k` from a **copy** of a manifest → output unchanged if its `c_k,d_k` are absent (i.e., zero amplitude means no effect).
4. **Leap day** — Evaluate `2016-02-29` (and a control year) → `t := days_since(D,t0)` consistent across languages (no off-by-one).

CLI shape (either form)

```
# Generic harness
ssm-validate --manifests ./manifests --golden ./golden_all_v2_1.csv \
  --report out.md --tol-deg 1e-5 --strict-rasi yes --failfast no --
machinesafe yes

# Reference Python (this repo)
python validate_golden_all.py --manifests ./manifests --golden
./golden_all_v2_1.csv --tol 1e-5
```

Report columns

```
planet, n_rows, max_abs_err_deg, rasi_mismatch_count, status, notes
```

K.8 Edge cases & exact behaviors

- **Euclidean modulus** for negatives: always use `emod` for `%` on reals.
- **Exact cusp tie:** within `5e-12` deg → bump to **higher** rāśi (`12 → 0`).
- **Rounding safety:** printing with 6 or 8 decimals must not affect comparisons (compare as floats).
- **Missing keys:** if `terms[]` present, prefer it; else use `omegas+beta`. Missing `c*` or `d*` ⇒ treat as zero (skip term).
- **Mean motion:** fixed-n bodies use `360 / P_sid_days` **exactly**; free-n bodies use the stored slope (`b1_deg_per_day` / `n_deg_per_day`).

K.9 Packaging & CI (one-command proof)

Tree

```
conformance/
  js/       (evalAngle.ts + test runner)
  py/       (eval_angle.py + test runner)
  wasm/     (Rust crate + wasm-bindgen wrapper)
  golden/   (golden_all_v2_1.csv)
  manifests/ (... per-body json ...)
  scripts/  (ssm-validate, ci_csv_to_html)
```

CI steps (language matrix)

1. Install toolchains: **Node ≥ 18**, **Python ≥ 3.10**, **Rust stable + wasm-pack**.
2. Build WASM:
   `wasm-pack build --target web`
3. Run validators: **JS → Python → WASM**; each emits `out_<lang>.csv`.
4. Merge & render HTML summary:
   `ci_csv_to_html --in merged.csv --out summary.html`

PASS criteria
All languages produce identical `L_hat_deg` within `1e-5` **and** identical `rasi` on the golden set; zero failures.

K.10 Minimal fixtures

`cusps_fuzz.csv` (unit test for rāśi only; angles in degrees):
for each `k ∈ {0..12}` test `30*k + ε` with `ε ∈ {-1e-9, -1e-12, 0, +1e-12, +1e-9}`.
Expected rāśi: apply the tie rule exactly (`0→0`, `30→1`, …, `360≡0→0`).

Outcome. Any JS, Python, or WASM build that honors the helpers, evaluator, and tie rules above will reproduce the same angles/rāśi for a given manifest and date — deterministically, ephemeris-free, and verifiably within the `1e-5 deg` tolerance.

Appendix L — Adversarial & Stress Tests (red-team matrix)

Purpose. Deliberately try to break the pipeline; document expected behavior and what counts as a defect. Everything here is ephemeris-free and reproducible from manifests + evaluator only.

L.1 Cusp & wrap edge cases

Vectors (per k = 0..12; ε ∈ {−1e−9, −1e−12, 0, +1e−12, +1e−9} deg):

- **Inputs:** `L = 30*k + ε`.
- **Expectation:**
  - **Tie rule:** if `|ε| ≤ 5e-12` ⟶ `rasi = (k % 12)` (higher rāśi at the exact cusp; `360°→0`).
  - Otherwise: `rasi = floor(((30*k + ε) % 360)/30)` (Euclidean `%`).
- **Seam test:** `L = 359.999999 → rasi 11`; `L = 0.000001 → rasi 0`.
- **Synthetic micro-walk:** `…, 29.9999 → 30.0000 → 30.0001, …` must produce exactly **one** rāśi transition (no oscillation).

Defect if: rāśi oscillates when only print precision changes; or `wrap360` maps `360` exactly to `360` (must be `0`).

L.2 Timebase & calendar

- **Leap-year stride:** `2016-02-28 → 2016-03-01` vs neighbors (2015, 2019). `t := days_since(D,t0)` must increment by whole days with proleptic Gregorian; `2016-02-29` exists.
- **Century cut:** `1999-12-31 → 2000-01-01` has no off-by-one.
- **Rule:** `days_since` uses whole civil days; no timezone math after fetch.

Defect if: any off-by-one shows in leap or century transitions.

L.3 Harmonic degeneracies

- **Train-time:** any two `ω` within `< 1e-9 rad/day` must be rejected (near-alias guard).
- **Runtime:** manifests must not contain near-aliased pairs; if present, validator flags input error.
- **Missing fields:** present `ws` but absent `cS,dS` ⟶ treat as zeros (no NaNs); same for any optional term.
- **Null amplitude:** `c_k = d_k = 0` ⟶ term contributes `0` (no numeric surprises).

Defect if: NaN/Inf arises from missing coefficients, or aliased `ω` pass unnoticed.

L.4 Numeric extremes

- **Far eras:** evaluate at `|t| ≈ 100,000` days (centuries). Must remain stable to `≤ 1e-5 deg` across languages; `wrap360` still in `[0,360)`.
- **Bad-manifest guard:** hard-fail input if any `|c_k| > 1e6 deg` or `|n_deg_per_day| > 100 deg/day`.
- **Sanity:** all trig args are `ω*t` in radians; double precision is sufficient for stated spans.

Defect if: overflow/underflow, NaN, or drift beyond tolerance on fixed test vectors.

L.5 Rāśi robustness

- **Rounding abuse:** truncating `L_hat_deg` to 3 decimals must **not** change rāśi (only angle precision is affected).
- **Modulo semantics:** rāśi uses Euclidean modulus; negative intermediates may not flip the sign or bucket.

Defect if: rāśi differs between the full-precision float and its 3-decimal print for the same underlying `L`.

L.6 Soft governor (optional track; see Appendix I)

- **Station glide:** as `v_pred → 0`, `g_stat := min(1, v_pred/v_thr_gate)` lowers smoothly; `L_adj` moves but never jumps.
  Invariant: `|L_adj[t] - L_adj[t-1]| ≤ |wrap180(L_pred[t] - L_adj[t-1])|`.
- **Cusp glide:** as `d_cusp_prev → 0`, `g_edge := min(1, d_cusp_prev/d_glide_deg)` shrinks; no overshoot through cusps.
- **Liveness:** for a non-retro year on slow bodies, `nonzero_dL_adj_count ≥ 360`. If zero → the track is stuck.

Defect if: single-step jumps without corresponding `L_pred` jump, overshoot at cusps, or a stalled `L_adj`.

L.7 Expected outcomes (classification)

- **PASS:** all above hold; no rāśi oscillations; angle deltas within `1e-5 deg`; no NaN/Inf; governor glides smoothly.

- **Zero-knot (if present):**
(i) modifies `b0` only; (ii) `σ(t)` is bounded in `[0,1]`, `C¹`, and monotone; (iii) no day-to-day discontinuity from the knot (`|Δy_knot| ≤ 1e-6 deg`); (iv) `n` and all `ω,c,d` unchanged; (v) outside the knot's `±5•T` skirt, the evaluator matches the un-knotted track up to the expected offset (`0` pre-knot, `db` post-knot) within `1e-6 deg`; (vi) knot parameters `(τ, T, db)` are documented in notes.
- **WARN:** angle within `≤ 3e-5` at an exact cusp tie; manual inspection recommended. Zero-knot (if present): `db` within policy but introduces ≤10% degradation in one gated metric while improving another — flag `KNOT_TRADEOFF` for review.
- **FAIL:** any rāśi mismatch on tie vectors, NaN/Inf anywhere, off-by-one in `t`, aliased `ω` admitted, or non-moving adjusted track; or any zero-knot that alters `n/harmonics`, uses non-monotone `σ(t)`, causes a step `> 1e-6 deg`, lacks explicit `(τ,T,db)` notes, or produces unexpected offsets outside the `±5•T` skirt.

L.8 Red-team harness (CLI contract)

```
ssm-redteam --manifests ./manifests --report redteam_report.md \
  --tol-deg 1e-5 --strict-rasi yes --governor yes
```

Tests executed

1. **Cusps & seam (L.1):** deterministic grid over `k, ε`.
2. **Calendar (L.2):** leap/century checks on `days_since`.
3. **Degeneracy scan (L.3):** `ω`-spacing `≥ 1e-9 rad/day`; missing coefficients handling.
4. **Numeric extremes (L.4):** `|t|≈1e5`, bad-manifest thresholds.
5. **Rāśi robustness (L.5):** 3-decimal truncation comparison.
6. **Governor glide (L.6):** invariants and liveness.

Report columns (per planet):
```
planet, cusp_pass, seam_pass, calendar_pass, omega_guard_pass,
numeric_pass, rasi_robust_pass, governor_pass, max_abs_err_deg, notes,
status
```

Exit codes: `0=PASS`, `1=FAIL`, `2=IO/format`.

L.9 Defect codes (triage map)

- `RT-001 CUSP_TIE_MISCLASS` — tie rule violated at `|ε|≤5e-12`.
- `RT-002 WRAP_SEAM` — 0/360 seam miswrap or double wrap.
- `RT-003 DAYS_SINCE_OFFBYONE` — leap/century error in `t`.
- `RT-004 ALIAS_IN_MANIFEST` — min `Δω < 1e-9 rad/day` detected.
- `RT-005 NUM_RANGE` — absurd magnitudes (`|c|>1e6` or `|n|>100`).
- `RT-006 MOD_SEMANTICS` — non-Euclidean `%` caused rāśi flip.
- `RT-007 GOVERNOR_STALL` — `L_adj` liveness failure or jump.
- `RT-008 NAN_INF` — NaN/Inf at any stage.

- `RT-009 RASI_FLIP_PRINT` — rāśi changed with decimal truncation.

L.10 Minimal test snippets (portable ASCII)

Cusp fuzz loop

```
for k in 0..12:
  for eps in {-1e-9, -1e-12, 0, +1e-12, +1e-9}:
    L = 30*k + eps
    r = rasi(L)
    # tie rule
    if abs(eps) <= 5e-12: assert r == (k % 12)
```

Seam

```
assert rasi(359.999999) == 11
assert rasi(0.000001)   == 0
assert wrap360(360.0)   == 0.0
```

Leap day

```
assert days_since("2016-02-29","2016-02-28") == 1
assert days_since("2016-03-01","2016-02-28") == 2
```

Ω-spacing

```
assert min_pairwise_diff(omegas) >= 1e-9  # rad/day
```

Governor invariant

```
delta = wrap180(L_pred[t] - L_adj[t-1])
step  = abs(wrap180(L_adj[t] - L_adj[t-1]))
assert step <= abs(delta) + 1e-12
```

Outcome. This matrix and harness define exactly how we try to break the system. Passing it means the evaluator, manifests, rāśi logic, timebase math, and optional governor behave correctly under edge cases, extreme dates, and adversarial inputs — consistently, deterministically, and ephemeris-free.

Appendix M — Error Budgets & Release Gates (SLOs)

Purpose. Convert acceptance thresholds into service-level objectives so releases have a crisp PASS/WARN/FAIL decision. Everything here is ephemeris-free and computed from daily sidereal (Lahiri) series at 05:30 IST.

M.1 Core metrics (all ASCII)

Conventions

- Angles in degrees; trig inputs in radians inside the evaluator.
- Daily timestamp convention: 05:30 IST for all dates.
- Wrap helpers:
  `wrap360(x) := x - 360 * floor(x / 360)` → `[0,360)`
  `wrap180(x) := ((x + 180) % 360) - 180` → `(-180,180]`
  `wrap30(x) := ((x + 15) % 30) - 15` → `(-15,15]`
- Unwrapped series (for diffs/timings):
  `L_unwrap(t) := continuous_unwrap( L_hat_deg(t) )`

Degree error (on [0,360))

```
deg_err(t) := | wrap180( L_true_deg(t) - L_pred_deg(t) ) |
MAE_deg := mean_t( deg_err(t) )
P90_deg := percentile90_t( deg_err(t) )
MAX_deg := max_t( deg_err(t) )
```

Phase metric (on [0,30))

```
d(t) := wrap30( (L_true_deg(t) % 30) - (L_pred_deg(t) % 30) )
phase_err_deg := | d(t) |
phase_MAE_deg := mean_t( phase_err_deg )
phase_P95_deg := percentile95_t( phase_err_deg )
```

Rasi crossings (timing by unwrapped linear interpolation)
Nearest rasi cusp at multiples of 30° on unwrapped axis:

```
k(t) := round( L_unwrap(t) / 30 )
cusp_k_deg := 30 * k(t)
```

Find each interval `[t0,t1]` where `L_unwrap` crosses `cusp_k_deg`:

```
t_cross_pred := t0 + (cusp_k_deg - L_unwrap(t0)) / (L_unwrap(t1) -
L_unwrap(t0))
```

Error vs reference:

```
rasi_cross_err_days := t_cross_pred - t_cross_true
rasi_cross_MAE_days := mean_crossings( | rasi_cross_err_days | )
```

Stations (date error on smoothed speed |v|)
Finite-difference daily speed on unwrapped longitudes:

```
v_deg_per_day(t) := L_unwrap(t) - L_unwrap(t-1)
```

Centered moving average, odd window `W` (e.g., `W=7`):

```
smooth(|v|)(t) := mean_{u=t-(W-1)/2 .. t+(W-1)/2} ( | v_deg_per_day(u) | )
```

Station dates are local minima of `smooth(|v|)`:

```
t_station_pred := argmin over local window
station_date_err_days := t_station_pred - t_station_true
station_date_MAE_days := mean_stations( | station_date_err_days | )
```

Cusp distance (instantaneous)

```
x := L_pred_deg(t)
cusp_dist_deg(t) := min( x % 30 , 30 - (x % 30) )
cusp_dist_MAE_deg := mean_t( cusp_dist_deg(t) )
```

Cross-units conversion (hours from phase error)

```
rasi_cross_hours := 24 * phase_MAE_deg / | n_deg_per_day |
# n_deg_per_day is the body's mean sidereal daily motion (deg/day);
# for fixed-n bodies use 360/P_sid, for free-n use the manifest slope.
```

Optional tallies (for summaries/UI)

```
movement_count := count_t( L_unwrap(t) != L_unwrap(t-1) )
station_ratio := (# detected station days) / (# days in span)
cusp_ratio := (# days with cusp_dist_deg(t) <= d0) / (# days in span)
```

M.2 SLO bands (TEST window)

Sun / Mercury / Venus / Mars (free-n)

- **PASS:** `phase_MAE_deg ≤ 1.5` **and** `rasi_cross_MAE_days ≤ 2.0`
- **WARN:** `phase_MAE_deg ≤ 2.0` **or** `rasi_cross_MAE_days ≤ 3.0` (not both)
- **FAIL:** otherwise

Moon (fixed-n)

- **PASS:** `cusp_dist_MAE_deg ≤ 1.5` **and** `rasi_cross_MAE_days ≤ 1.0`
- **WARN:** `cusp_dist_MAE_deg ≤ 2.0` **or** `rasi_cross_MAE_days ≤ 1.5`
- **FAIL:** otherwise

Jupiter / Saturn (fixed-n outers)

- **PASS:** `MAE_deg ≤ 2.0`, `P90_deg ≤ 4.0`, `rasi_cross_MAE_days ≤ 10`, `station_date_MAE_days ≤ 10`, `misclass_rate ≤ 12%`
- **WARN:** one metric exceeds PASS but stays within **+25%** of the PASS bound
- **FAIL:** any metric **> +25%** over its PASS bound

Nodes (Rahu / Ketu)

- **PASS:** `phase_MAE_deg ≤ 3.0`, `rasi_cross_MAE_days ≤ 12`
- **WARN/FAIL:** same +25% ratios as outers

Note. For inner planets, dashboards may store `phase_MAE_deg` in the `mae_deg` column; gates must use the **phase** metric, not the 0–360° MAE.

M.3 Release gates (go / no-go)

- **Greenlight:** all bodies **PASS** on TEST; far-era audit (spot checks) shows **no** metric `> 1.25×` its PASS bound.
- **Yellowlight:** `≤ 2` bodies in **WARN** and none in **FAIL** → publish with a caution note and a scheduled retrain plan.
- **Redlight:** any **FAIL** → do **not** publish; refresh TRAIN or simplify the model.

M.4 Drift watch (forward monitoring)

Tripwires that trigger escalation (see Appendix F.2):

Outer (Jupiter/Saturn/Uranus/Neptune/Nodes)
`MAE_deg > 3.0` **OR** `rasi_cross_MAE_days > 12` **OR** `station_date_MAE_days > 12`

Moon
`cusp_dist_MAE_deg > 2.0` **OR** `rasi_cross_MAE_days > 1.5`

Inner (Sun/Mercury/Venus/Mars)
`phase_MAE_deg > 2.0` **OR** `rasi_cross_MAE_days > 3.0`

Action: flag `REG=NOFIT` in summaries, schedule retrain centered on the affected interval, or hand off to a high-precision ephemeris for that window.

M.5 Reporting cadence

- Publish `aggregate_scoreboard.csv` and a one-page **acceptance_report** with PASS/WARN/FAIL badges per body.
- Include top-5 HIGH/LOW windows if you also ship the optional governor/trust view (Appendix I).
- Round only at print time; keep float64 precision for all math; leave N/A metrics blank (never zero-fill).

Outcome. With these SLOs and gates, every release has a clear, reproducible decision path from raw daily angles to PASS/WARN/FAIL, aligned with training/test conventions and the ephemeris-free evaluator.

Appendix N — Manifest JSON Schema (v1.9)

Purpose. Define a portable, language-agnostic contract for `ssm_params.json` so any evaluator can load one file and reproduce angles from a one-line rule.

Model family (this release). `model_family := "base-linear"` (no harmonics are active in v1.9; the scaffold remains for forward compatibility).

One-line evaluator (ASCII).

```
L_hat_deg := wrap360(a0 + n_deg_per_day * t)
t := days_since(D, t0)          # whole civil days at time_local
wrap360(x) := x - 360*floor(x/360)
```

Harmonic scaffold (general form; disabled in v1.9).

```
y := a0 + n_deg_per_day * t
y := y + SUM_k [ c_k*sin(omega_k*t) + d_k*cos(omega_k*t) ]
L_hat_deg := wrap360(y)
# In v1.9: omegas := {} and no harmonic terms are active.
```

Required keys (minimal).
• **t0** — ISO date string (midpoint anchor for `days_since`)
• **time_local** — local civil time of day for daily sampling (e.g., `05:30`); default `05:30` in v2.1
• **beta.a0_deg** — intercept `a0` in degrees at `t = 0`
• **n_deg_per_day** — mean sidereal daily motion (signed; absolute value `|x|` may be used for scaling metrics)
• **omegas** — object of named frequencies in `radians/day`; empty in v1.9
• **beta.c[*]**, **beta.d[*]** — per-term coefficients in degrees (optional; unused in v1.9)

Family note.
Fixed-n -> `n_deg_per_day := 360.0 / P_sid_days`
Free-n -> `n_deg_per_day := learned_slope` (stored in manifest)

Coverage note (v2.1).
Pluto is not included in v2.1 public coverage (no Pluto rows in `golden_all_v2_1.csv`); references remain for continuity.

N.1 Scope & versioning (manifest-level metadata)

- `schema_version` is a string tag (e.g., `"1.9"`).
- Non-breaking edits bump patch (e.g., `1.9.1`); breaking edits bump minor/major and ship a new schema alongside prior ones.
- Pin the sampling convention in the manifest: `ayanamsa := "Lahiri"`, `time_local := "05:30"`.

N.2 Required keys (core, family-agnostic)

- `planet` : `"Sun"|"Moon"|"Mercury"|"Venus"|"Mars"|"Jupiter"|"Saturn"|"Uranus"|"Neptune"|"Rahu"|"Ketu"`
- `t0` : `"YYYY-MM-DD"` (mid-anchor of TRAIN window; same daily timestamp convention)
- `time_local` : `"HH:MM"` (default `"05:30"`)
- `ayanamsa` : `"Lahiri"`
- `n_deg_per_day` : number
  (fixed-n uses `360 / P_sid_days`; free-n stores the learned slope; alias `b1_deg_per_day` equals `n_deg_per_day`)
- `selected_model` : short tag, e.g. `"base"|"add_nE"|"add_3w"|"add_nE_3w"`
- `omegas` : object of named angular frequencies in radians/day (family-agnostic names), e.g.
  ```
  { "w1": <number>, "w2": <number>, "w3": <number>, "wS": <number>,
  "nE": <number>, "wA": <number>, "wD": <number> }
  ```
- `beta` : object of coefficients in degrees, pairing each omega with sine/cosine weights, plus intercepts, e.g.
  ```
  { "a0_deg": <number>, "b1_deg_per_day": <number?>, "c1": <number>,
  "d1": <number>, ... }
  ```

Harmonic encoding (choose one; both allowed by schema):

- **A)** `omegas` + `beta` (named terms)
- **B)** `terms` : array of minimal per-term objects
  ```
  terms = [ { "w_rad_per_day": <number>, "c_sin": <deg>, "d_cos": <deg>
  }, ... ]
  ```

Evaluator mapping (ASCII, degrees out; radians in trig).

```
# required intercepts
a0_deg := beta.a0_deg
b1_deg := (beta.b1_deg_per_day if present) else n_deg_per_day

# A) named families
y_deg := a0_deg + b1_deg * t
for each name in omegas:
    w := omegas[name]                 # radians/day
    c := beta["c" + suffix(name)]     # degrees
    d := beta["d" + suffix(name)]     # degrees
    y_deg := y_deg + c*sin(w*t) + d*cos(w*t)

# B) minimal terms[]
y_deg := a0_deg + b1_deg * t
for each term in terms:
    y_deg := y_deg + term.c_sin*sin(term.w_rad_per_day*t)
                   + term.d_cos*cos(term.w_rad_per_day*t)

L_hat_deg := wrap360(y_deg)
```

Guards (must-haves).

- Units explicit: radians/day for any `w*`; degrees for `a0_deg`, `b1_deg_per_day` (or `n_deg_per_day`), and all `c*/d*`.
- `t0` and `time_local` must match the evaluator's daily convention (`05:30`).
- **Fixed-n:** include `n_deg_per_day (= 360/P_sid_days)` and omit `beta.b1_deg_per_day` (or mirror it to the same value).
- **Free-n:** store the learned slope in `n_deg_per_day` and include `beta.b1_deg_per_day = n_deg_per_day`.

N.3 Family keys (fixed-n vs free-n)

Fixed-n bodies (e.g., Moon, Jupiter, Saturn, Uranus, Neptune, Nodes)

- **Required**
  - `P_sid_days > 0`
  - `n_deg_per_day := 360 / P_sid_days` (store the numeric value)
- **Beta rule:** omit `beta.b1_deg_per_day` (or set equal to `n_deg_per_day`)
- **Optional flags:** `n_fixed = true`
- **Notes:** Nodes preserve symmetry by construction (`Ketu = Rahu + 180°`). Use identical harmonic sets and apply a `180°` offset to the intercept if you store both.

Free-n bodies (Sun, Mercury, Venus, Mars)

- **Required**
  - `n_deg_per_day =` learned slope (OLS)
  - `beta.b1_deg_per_day` present **and** equals `n_deg_per_day`
- **Optional:** `P_sid_days` may be null/omitted
- **Optional flags:** `n_fixed = false`

Informational keys (do not affect evaluator).

- `P_syn_days` (synodic) — optional, documentation only

N.4 Frequencies & coefficients (names, units, pairing)

Allowed omega names (use any subset as needed): `w1, w2, w3, wS, nE, wA, wD`
Typical semantics (advisory, not enforced):

- `wS` — sidereal fundamental (`wS = 2π / P_sid_days`)
- `nE` — annual leakage (`nE = 2π / 365.2564`)
- `wA` — anomalistic component
- `wD` — draconic component
- `w1,w2,w3` — low-order extras

Units & ranges

- `omegas.*` are **radians/day**
- `beta` coefficients are **degrees**

Pairing rule (must-have)
If `omegas` contains:

- `w1` → require `beta.c1`, `beta.d1`
- `w2` → require `beta.c2`, `beta.d2`
- `w3` → require `beta.c3`, `beta.d3`
- `wS` → require `beta.cS`, `beta.dS`
- `nE` → require `beta.cE`, `beta.dE`
- `wA` → require `beta.cA`, `beta.dA`
- `wD` → require `beta.cD`, `beta.dD`

N.5 Minimal JSON Schema (draft-07, lenient core)

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "SSM-JTK Manifest v1.9 (lenient core)",
  "type": "object",
  "additionalProperties": false,
  "required": [
    "schema_version", "planet", "t0", "ayanamsa", "time_local",
    "n_deg_per_day", "selected_model"
  ],
  "properties": {
    "schema_version": { "type": "string", "pattern": "^1\\.9(\\.\\d+)?$" },
    "planet": {
      "type": "string",
      "enum":
["Sun","Moon","Mercury","Venus","Mars","Jupiter","Saturn","Uranus","Neptune
","Rahu","Ketu"]
    },
    "t0": { "type": "string", "pattern": "^\\d{4}-\\d{2}-\\d{2}$" },
    "ayanamsa": { "type": "string", "const": "Lahiri" },
    "time_local": { "type": "string", "pattern": "^\\d{2}:\\d{2}$" },
    "n_deg_per_day": { "type": "number" },
    "P_sid_days": { "type": ["number","null"] },
    "P_syn_days": { "type": ["number","null"] },
    "n_fixed": { "type": "boolean" },
    "selected_model": { "type": "string" },
    "omegas": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "w1": { "type": "number" },
        "w2": { "type": "number" },
        "w3": { "type": "number" },
        "wS": { "type": "number" },
        "nE": { "type": "number" },
        "wA": { "type": "number" },
        "wD": { "type": "number" }
```

```
      }
    },
    "beta": {
      "type": "object",
      "additionalProperties": false,
      "required": ["a0_deg"],
      "properties": {
        "a0_deg": { "type": "number" },
        "b1_deg_per_day": { "type": "number" },
        "c1": { "type": "number" }, "d1": { "type": "number" },
        "c2": { "type": "number" }, "d2": { "type": "number" },
        "c3": { "type": "number" }, "d3": { "type": "number" },
        "cS": { "type": "number" }, "dS": { "type": "number" },
        "cE": { "type": "number" }, "dE": { "type": "number" },
        "cA": { "type": "number" }, "dA": { "type": "number" },
        "cD": { "type": "number" }, "dD": { "type": "number" }
      }
    },
    "terms": {
      "type": "array",
      "items": {
        "type": "object",
        "additionalProperties": false,
        "required": ["w_rad_per_day", "c_sin", "d_cos"],
        "properties": {
          "w_rad_per_day": { "type": "number" },
          "c_sin": { "type": "number" },
          "d_cos": { "type": "number" }
        }
      }
    },
    "notes": { "type": "string" },
    "meta": {
      "type": "object",
      "additionalProperties": true,
      "properties": {
        "train_range": { "type": "string" },
        "test_range": { "type": "string" },
        "origin": { "type": "string" }
      }
    }
  },
  "oneOf": [
    { "required": ["omegas", "beta"] },
    { "required": ["terms"] }
  ]
}
```

N.6 Strict overlay (rules your loader must enforce)

- **Pairing.** If `omegas.w1` exists → require `beta.c1` **and** `beta.d1` (repeat for all names).
- **Family.** If `P_sid_days` is a number (**fixed-n**), either omit `beta.b1_deg_per_day` or set it equal to `n_deg_per_day`.
  If `P_sid_days` is null/absent (**free-n**), require `beta.b1_deg_per_day`.
- **Uniqueness.** All `omegas` values distinct by ≥ `1e-9` rad/day.
- **Units.** `omegas.*` in rad/day; `beta.*` in deg; `n_deg_per_day` in deg/day.
- **Ranges.** All finite float64; reject NaN/Inf.

- **Unknown keys.** Reject; keep `additionalProperties: false`.

N.7 Evaluator reminder (one-liner + helpers)

```
t := days_since(D, t0)                      # whole-day step at time_local
y := a0 + n_deg_per_day*t
y := y + Σ[c_k*sin(ω_k*t) + d_k*cos(ω_k*t)] # if any terms present
L_hat_deg := wrap360(y)
wrap360(x) := x - 360*floor(x/360)
rasi := floor((L_hat_deg % 360) / 30)
```

N.8 Validator logic (pseudo-code)

```
ok := true
fixed := is_number(P_sid_days)

# family
if fixed:
    if ("b1_deg_per_day" in beta) and (abs(beta.b1_deg_per_day -
n_deg_per_day) > 1e-12): ok = false
else:
    if ("b1_deg_per_day" not in beta): ok = false
    if (abs(beta.b1_deg_per_day - n_deg_per_day) > 1e-12): ok = false

# pairing
pairs = {"w1":["c1","d1"], "w2":["c2","d2"], "w3":["c3","d3"],
         "wS":["cS","dS"], "nE":["cE","dE"], "wA":["cA","dA"],
"wD":["cD","dD"]}
for k,(ci,di) in pairs.items():
    if k in omegas:
        if (ci not in beta) or (di not in beta): ok = false

# uniqueness
vals = list(omegas.values())
for i<j:
    if abs(vals[i]-vals[j]) < 1e-9: ok = false

# finiteness
for x in [n_deg_per_day, P_sid_days, P_syn_days] + list(omegas.values()) +
list(beta.values()):
    if not is_finite_number(x): ok = false

return ok
```

N.9 Minimal examples (schema-valid)

Fixed-n (Jupiter; harmonics scaffold present but empty in v1.9)

```
{
  "schema_version": "1.9",
  "planet": "Jupiter",
  "t0": "2000-01-01",
```

```
  "ayanamsa": "Lahiri",
  "time_local": "05:30",
  "P_sid_days": 4332.59,
  "P_syn_days": null,
  "n_deg_per_day": 0.0830911764,
  "n_fixed": true,
  "selected_model": "base",
  "omegas": {},
  "beta": { "a0_deg": 0.0 },
  "terms": [],
  "notes": "train 2020..2024 daily; mid-anchor; base set; v1.9"
}
```


Free-n (Venus; no harmonics in v1.9)


```
{
  "schema_version": "1.9",
  "planet": "Venus",
  "t0": "2000-01-01",
  "ayanamsa": "Lahiri",
  "time_local": "05:30",
  "P_sid_days": null,
  "n_deg_per_day": 1.60213,
  "n_fixed": false,
  "selected_model": "base",
  "omegas": {},
  "beta": { "a0_deg": 0.0, "b1_deg_per_day": 1.60213 },
  "terms": [],
  "notes": "train 2020..2024 daily; mid-anchor; v1.9"
}
```


(Specimen values are illustrative; populate with your release numbers.)

N.10 Backward/forward compatibility

- **Strict mode** evaluators reject unknown keys; **compat mode** may ignore unknown keys but must still enforce §N.6.
- Future fields (e.g., trust/gate policy) can live under `meta.*`; evaluators that don't understand them must ignore them.

N.11 Authoring rules (do/don't)

- **Do** keep manifests read-only once published; supersede, don't overwrite.
- **Do** store coefficients in degrees and frequencies in radians/day.
- **Don't** encode runtime slopes/patches outside `n_deg_per_day`; kernel is ephemeris-independent.
- **Don't** include partial pairs (e.g., `c1` without `d1`) or unrecognized names.

N.12 Acceptance (schema + validator)

A manifest is **valid** if: JSON validates under §N.5 **and** the loader's §N.8 checks pass. When valid, `L_hat_deg` must be reproducible with the §N.7 evaluator using only the manifest and date list — **no ephemeris calls** and no hidden knobs.

Appendix O — Public Manifests (JTK release v2.1, schema v1.9)

Scope. These manifests were produced by the JTK v2.1 pipeline and conform to the v1.9 schema. Base-linear; no harmonics. (Renumbered from N.13–N.23. In each JSON, `schema_version` reflects the schema: `"1.9"`. A human note records the JTK release.)

O.1 Sun

{
"schema_version": "1.9",
"planet": "Sun",
"t0": "2100-01-01",
"time_local": "05:30",
"ayanamsa": "Lahiri",
"n_deg_per_day": 0.98564736,
"n_fixed": false,
"selected_model": "base",
"omegas": {},
"beta": { "a0_deg": 0.0, "b1_deg_per_day": 0.98564736 },
"terms": [],
"notes": "JTK release v2.1; base-linear; no harmonics"
}

O.2 Moon

{
"schema_version": "1.9",
"planet": "Moon",
"t0": "2100-01-01",
"time_local": "05:30",
"ayanamsa": "Lahiri",
"P_sid_days": 27.3216620253,
"n_deg_per_day": 13.176358,
"n_fixed": true,

"selected_model": "base",
"omegas": {},
"beta": { "a0_deg": 0.0 },
"terms": [],
"notes": "JTK release v2.1; base-linear; no harmonics"
}

O.3 Mercury

{
"schema_version": "1.9",
"planet": "Mercury",
"t0": "2100-01-01",
"time_local": "05:30",
"ayanamsa": "Lahiri",
"n_deg_per_day": 4.092385,
"n_fixed": false,
"selected_model": "base",
"omegas": {},
"beta": { "a0_deg": 0.0, "b1_deg_per_day": 4.092385 },
"terms": [],
"notes": "JTK release v2.1; base-linear; no harmonics"
}

O.4 Venus

{
"schema_version": "1.9",
"planet": "Venus",
"t0": "2100-01-01",
"time_local": "05:30",
"ayanamsa": "Lahiri",
"n_deg_per_day": 1.60213,
"n_fixed": false,
"selected_model": "base",
"omegas": {},
"beta": { "a0_deg": 0.0, "b1_deg_per_day": 1.60213 },
"terms": [],
"notes": "JTK release v2.1; base-linear; no harmonics"
}

O.5 Mars

{
"schema_version": "1.9",
"planet": "Mars",
"t0": "2100-01-01",
"time_local": "05:30",
"ayanamsa": "Lahiri",
"n_deg_per_day": 0.524039,
"n_fixed": false,
"selected_model": "base",
"omegas": {},
"beta": { "a0_deg": 0.0, "b1_deg_per_day": 0.524039 },
"terms": [],
"notes": "JTK release v2.1; base-linear; no harmonics"
}

O.6 Jupiter

{
"schema_version": "1.9",
"planet": "Jupiter",
"t0": "2100-01-01",
"time_local": "05:30",
"ayanamsa": "Lahiri",
"P_sid_days": 4332.59,
"n_deg_per_day": 0.08309117640949178,
"n_fixed": true,
"selected_model": "base",
"omegas": {},
"beta": { "a0_deg": 0.0 },
"terms": [],
"notes": "JTK release v2.1; base-linear; no harmonics"
}

O.7 Saturn

{
"schema_version": "1.9",
"planet": "Saturn",
"t0": "2100-01-01",
"time_local": "05:30",
"ayanamsa": "Lahiri",
"P_sid_days": 10759.422452,
"n_deg_per_day": 0.033459045,
"n_fixed": true,

"selected_model": "base",
"omegas": {},
"beta": { "a0_deg": 0.0 },
"terms": [],
"notes": "JTK release v2.1; base-linear; no harmonics"
}

O.8 Uranus

{
"schema_version": "1.9",
"planet": "Uranus",
"t0": "2100-01-01",
"time_local": "05:30",
"ayanamsa": "Lahiri",
"P_sid_days": 30690.537084,
"n_deg_per_day": 0.01173,
"n_fixed": true,
"selected_model": "base",
"omegas": {},
"beta": { "a0_deg": 0.0 },
"terms": [],
"notes": "JTK release v2.1; base-linear; no harmonics"
}

O.9 Neptune

{
"schema_version": "1.9",
"planet": "Neptune",
"t0": "2100-01-01",
"time_local": "05:30",
"ayanamsa": "Lahiri",
"P_sid_days": 60210.737582,
"n_deg_per_day": 0.005979,
"n_fixed": true,
"selected_model": "base",
"omegas": {},
"beta": { "a0_deg": 0.0 },
"terms": [],
"notes": "JTK release v2.1; base-linear; no harmonics"
}

O.10 Rahu

{
"schema_version": "1.9",
"planet": "Rahu",
"t0": "2100-01-01",
"time_local": "05:30",
"ayanamsa": "Lahiri",
"P_sid_days": 6798.383,
"n_deg_per_day": -0.0529539,
"n_fixed": true,
"selected_model": "base",
"omegas": {},
"beta": { "a0_deg": 0.0 },
"terms": [],
"notes": "JTK release v2.1; base-linear; no harmonics"
}

O.11 Ketu

{
"schema_version": "1.9",
"planet": "Ketu",
"t0": "2100-01-01",
"time_local": "05:30",
"ayanamsa": "Lahiri",
"P_sid_days": 6798.383,
"n_deg_per_day": -0.0529539,
"n_fixed": true,
"selected_model": "base",
"omegas": {},
"beta": { "a0_deg": 180.0 },
"terms": [],
"notes": "JTK release v2.1; base-linear; no harmonics"
}

Outcome. With the v1.9 schema and these base-linear manifests, any runtime can reproduce `L_hat_deg` and `rasi` deterministically from a single JSON — ephemeris-free, audit-ready, and future-proofed for optional harmonics.

OMP