USWF-DESIGN-001

Unified Space-Weather & Non-Gravitational Force Modeling System

Production Design Document — Rev.18

Implements: USWF-SPEC-001 (Rev.13 – Requirements & SLOs)
Supersedes: USWF-DESIGN-001 (Rev.17)

Document Class: Production-Build Technical Design
Length Target: ~50 pages equivalent
Audience: Engineering teams, technical leadership, program management
Use: Implementation blueprint, design review artifact, engineering onboarding, contracting input

---

Table of Contents

---

1. Executive Summary

The USWF system provides real-time **force environment estimation** for spacecraft, including:

- Atmospheric drag
- Solar radiation pressure (SRP)
- Albedo / IR forces
- Empirical small forces

It augments traditional orbit determination (OD) workflows by adding:

- Physics-based force models with **formal uncertainty and covariance**
- **Maneuver detection** and **cause attribution** (burn vs space weather vs other)
- **Historical replay** of environment state (EnvRecords) for ≥10 years
- **APIs and dashboards** for flight dynamics, analysis, and operations teams

This design document translates the high-level requirements and SLOs in **USWF-SPEC-001 (Rev.13)** into a concrete, implementable production system: compute and storage architecture, ETL pipelines, API contracts, model lifecycles, performance budgets, monitoring and alerting, security, disaster recovery, and a high-level cost model.

---

2. System Goals & Use-Cases

2.1 Core Goal

Provide **trusted, fast, reproducible** force environment estimates with quantified uncertainty that improve:

- Orbit prediction quality
- Conjunction assessment and response
- Root cause understanding of anomalies
- Long-term, forensically reliable analysis

2.2 Primary Use-Cases

Use-Case	Value
Drag Forecasting	Plan station-keeping, fuel expenditure, orbital lifetime
Maneuver Attribution	Distinguish operator burns from storms or modeling error
Conjunction Response	Faster, risk-informed decisions using environment-aware forces
Post-Event Forensics	Replay environmental conditions for specific dates and times
OD Calibration	Feed forces + covariance directly into OD filters
Science / Research	Long-term, high-quality environment dataset for analysis

2.3 Key Guarantees (from Spec)

- **Latency:** Real-time responses with **p99 < 60 s**
- **Covariance:** Force covariance matrix **Σ_a must be positive-definite**
- **Attribution:** **≥80% precision** in the HIGH-confidence attribution class
- **History:** **≥10-year** reproducible environment archive

---

3. High-Level Architecture

```
  ┌──────────────────────────┐ │ External Space Weather │ │ NOAA/SWPC/GOES/OMNI │
  └─────────────┬────────────┘ │ (Ingest ETL) │ ▼ ┌───────────────┐ │ Raw Data Lake│
S3/Parquet └─────────┬────────┘ │ ETL Normalize/QC ▼ ┌─────────────────────┐ │ Time-Series
DB │ Postgres/Timescale └──────────┬──────────┘ │ (Model Compute Layer) │
  ┌────────────────────┐ │ Force Engine │ ← Drag/SRP/Albedo/Small Forces
  └────────┬───────────┘ │ + Covariance Engine + Maneuver Detection + Attribution
Engine │ ▼ ┌──────────────────────────┐ │ EnvRecord Generation │
  └──────────┬───────────────┘ │ ┌──────────────────────────┐ │ API + Streaming Outputs │
  └──────────┬───────────────┘ │ ┌──────────────────────────┐ │ Dashboards & Analysis │
  └──────────────────────────┘
```

---

4. Force Modeling Design

4.1 Core Physics Loop

The force engine runs at a configurable timestep (10–60 s), per satellite:

```
for t in timeline: state = get_sat_state(t) space_weather = load_env_inputs(t) a_drag
= drag_model(state, space_weather) a_srp = srp_model(state, space_weather) a_albedo =
albedo_model(state) a_emp = empirical_correction(state, space_weather) a_total =
a_drag + a_srp + a_albedo + a_emp sigma, Sigma = compute_uncertainty_and_covariance(
state, space_weather, a_total ) write_envrecord(t, a_total, sigma, Sigma)
```

Unit convention (forces):
All acceleration vectors (e.g., `a_drag`, `a_srp`, `a_total`) are in **m/s²** internally, in the database, and in the API responses.

4.2 Model Release Roadmap

Version	Model Implementation Level
v1.0	NRLMSIS atmospheric drag + Cannonball SRP
v1.1	Box-Wing SRP + Albedo/IR modeling
v1.5	Empirical small-forces + ML-based maneuver-assist
v2.0	Full covariance exposure + real-time SLA at scale

5. Uncertainty & Covariance Subsystem

5.1 Uncertainty Calculation

Drag uncertainty is composed from independent sources:

$$\sigma^2_{\text{drag}} = \sigma^2_{\rho} + \sigma^2_{C_D} + \sigma^2_{A/m} + \sigma^2_{\theta}$$

Implementation:

```
drag_vars = [sigma_rho, sigma_cd, sigma_am, sigma_attitude] sigma_drag =
math.sqrt(sum(v**2 for v in drag_vars))
```

Each σ-term is configured per-satellite or per-orbit regime and validated against ILRS/IGS data as part of system tuning.

5.2 Covariance Matrix Construction & Units

Force covariance between drag, SRP, and empirical terms:

$$\Sigma_a = \begin{bmatrix} \sigma_d^2 & \rho_{ds}\sigma_d\sigma_s & \rho_{de}\sigma_d\sigma_e \\ \rho_{ds}\sigma_d\sigma_s & \sigma_s^2 & \rho_{se}\sigma_s\sigma_e \\ \rho_{de}\sigma_d\sigma_e & \rho_{se}\sigma_s\sigma_e & \sigma_e^2 \end{bmatrix}, \qquad \Sigma_a \succ 0$$

Unit convention:

- All accelerations: **m/s²**
- Covariance matrix **Σ_a** entries: **(m/s²)² = m²/s⁴**

PD enforcement (runtime):

```
try: L = np.linalg.cholesky(Sigma) except np.linalg.LinAlgError: # Regularize
covariance slightly to enforce PD Sigma = Sigma + 1e-9 * np.eye(Sigma.shape[0])
```

This mechanism directly mitigates covariance failure risks in filters and OD systems.

5.3 R-Metric Definition

$$R = \frac{\text{RMS(actual error)}}{\text{RMS(predicted } 1\sigma)}$$

Target:
For operational acceptance, **R must remain in [0.8, 1.2]** for **90 consecutive days** across the reference arc set.

---

6. Attribution Engine Design

6.1 Inputs

The feature vector combines space weather, spacecraft behavior, and event context:

$$X = [Kp, Dst, F10.7, \Delta\rho, \Delta A/m, ECOM\ drift, flux, belt, prox, QC, ...]$$

Where, for example:

- **Kp, Dst, F10.7**: geomagnetic and solar activity indices
- **Δρ**: anomaly in density vs baseline/climatology
- **ΔA/m**: changes in effective area-to-mass ratio (e.g., attitude changes)
- **ECOM drift**: empirical orbit model residual behavior
- **flux, belt**: radiation belt indicators, GOES fluxes
- **prox**: proximity to known planned maneuvers
- **QC**: quality flags from ingest/ETL

Storm-weighting to damp spurious maneuver calls during high activity:

$$w = e^{-\alpha Kp}$$

6.2 Modeling Strategy

The attribution engine is **hybrid**:

1. **Rule-Based Front-End**
    - Handles obvious patterns (e.g., clear, impulsive residual step aligned with known burns).
    - Encodes safety rules (e.g., "don't call a maneuver during extreme Kp without cross-check").
2. **ML Backend (Classifier)**
    - Handles ambiguous patterns where multiple causes are plausible.
    - Initial implementation: Gradient Boosted Trees (e.g., XGBoost, LightGBM).
    - Future v2.0 upgrade path: Sequence-aware models (Transformers/LSTMs) for time series.

Classes can include: `maneuver`, `drag_storm`, `srp_change`, `charging_event`, `model_issue`, `unknown`.

6.3 Training Pipeline

The training pipeline is structured as a repeatable ML lifecycle:

1. **Data Selection**
    - Extract anomaly windows from EnvRecords, OD residuals, and logbooks.
    - Include representative coverage of storms, quiet periods, known burns, mis-modeled arcs, and instrument issues.
2. **Data Cleaning**
    - Exclude corrupted/partial records (e.g., missing key feeds).
    - Enforce basic QC using `qc_flag` from the data dictionary.
3. **Feature Generation**
    - Temporal windows (e.g., ±6–12 hours around events).
    - Rolling features: mean, std, max, slope over multiple time scales.
    - "Shape" features: peak magnitude, duration, skewness, rise/decay times.
    - Encoded metadata: orbit regime (LEO/MEO/GEO), satellite type, etc.
4. **Human Labeling Workflow**
    - Provide an internal labeling UI showing:
        - Residual time-series
        - Kp/Dst/F10.7, solar wind parameters
        - EnvRecords around the event
        - Known maneuver logs (if available)
    - Each event is labeled by **two independent experts**.
    - Track **Cohen's κ** for inter-rater agreement; target $\kappa > 0.7$.
5. **Consensus Labels & Conflict Resolution**
    - If annotators disagree, route to a third SME or resolution process.
    - Maintain provenance of all label changes.
6. **Train/Val/Test Splits**

   - Split by **time block**, not by individual samples, to avoid temporal leakage.
   - Example: 60% train, 20% validation, 20% test by contiguous time segments.
7. **Model Training** (`train.py`)
   - Use cross-validation on the training set.
   - Use class weighting or focal loss to handle class imbalance.
   - Log feature importance and model diagnostics.
8. **Evaluation**
   - Compute **precision, recall, F1** per class.
   - Focus metric: HIGH-confidence class precision ≥80%.
   - Calibration assessment (e.g., reliability curves for predicted probabilities).
9. **Shadow Deployment**
   - Deploy trained model in **shadow mode** for ≥90 days.
   - Compare model output with human labels and future confirmed events.
10. **Promotion Criteria**
    - Meets or exceeds accuracy/precision thresholds.
    - Stable performance across different solar and geomagnetic conditions.
    - Reviewed and approved by domain experts.

6.4 Attribution Model Lifecycle (Training, Versioning, Deployment)

- **Model Registry**
  - Each trained model stored with:
    - Unique ID (e.g., `attrib-gb-v1.3.2`).
    - Training dataset IDs and versions.
    - Metrics per class, date of training, and config hash.
- **Retraining Triggers**
  - Degradation of production metrics (e.g., HIGH-precision <80%).
  - Major changes in solar activity regime.
  - Accumulation of significant new labeled data (e.g., +100–200 events).
  - New satellites with substantially different behavior.
- **Deployment Flow**
  1. Train → evaluate → human review.
  2. Deploy as **shadow model** alongside current production model.
  3. Run for ≥90 days and compare direct outputs.
  4. Promote via configuration toggle when criteria met.
- **Rollback Strategy**
  - Maintain the last known good model in registry.
  - Promotion is reversible by configuration (e.g., feature flag or version pin).

7. Data Ingestion & ETL Pipelines

7.1 Sources & Protocols

Source	Data	Format	Method
NOAA/SWPC	Kp, Dst, F10.7, indices	JSON/CSV	HTTPS pull
GOES	X-ray, particle flux, radiation	GRIB/NetCDF	HTTPS/FTP
OMNI2	L1 solar wind, IMF	CDF	Daily bulk ingest
NRLMSIS / JB2008	Atmospheric density inputs	Model interface	Python bindings
Satellite Meta	A/m, Cd, attitude, ID	CSV/DB	REST/DB integration

7.2 ETL Steps

Core ETL pipeline:

```
def etl_step(): raw = fetch_raw() validated = validate_schema(raw) filled =
gap_fill_interpolate(validated) normalized = normalize_units(filled)
store_timeseries(normalized) log_ingest_status()
```

7.3 Retry & Fallback Logic

- Maximum of **5 retries** per feed with exponential backoff.
- Where possible, use **mirror endpoints** or alternate mirrors.
- If critical feeds are down beyond retry window:
  - Use **last-known values** plus climatology where safe.
  - Mark output EnvRecords with appropriate **degraded QC flags** (see below).

7.4 Data Dictionary (Canonical Fields, Units, QC Semantics)

Canonical Fields & Units (selected)

Field	Description	Units
time	Timestamp	UTC ISO-8601

Field	Description	Units
F10_7	Solar radio flux	sfu
Kp	Planetary K-index	dimensionless (0–9)
Dst	Disturbance Storm Time index	nT
rho	Atmospheric density	kg/m^3
v_sw	Solar wind speed	km/s
Bz	IMF southward component	nT
a_drag	Drag acceleration vector	m/s^2
a_srp	SRP acceleration vector	m/s^2
a_albedo	Albedo/IR acceleration vector	m/s^2
a_empirical	Empirical acceleration vector	m/s^2
Sigma	Force covariance matrix (Σ_a)	m^2/s^4 (entries)
qc_flag	Data quality flag	enum
env_version_id	Environment version ID	string

QC Flag Semantics

qc_flag	Meaning
GOOD	All key feeds nominal; no significant gaps
SUSPECT	Some inputs degraded but usable; caution advised
GAP_FILLED	One or more feeds filled via interpolation or climatology
MISSING	Inputs insufficient; no reliable EnvRecord for this time

Gap-Fill Policy & Method

- **Gaps ≤ 30 minutes:**
    - Method: **linear interpolation** in time for numeric fields.
    - Applies to indices like Kp, Dst, F10.7, v_sw, Bz when safe.

- **Gaps ≤ 6 hours:**
  - Linear interpolation plus **climatology blending**, meaning:
    - Interpolate between last-known and next-known values,
    - Weight towards climatology as gap duration increases.
- **Gaps > 6 hours:**
  - Use **climatology only** where possible, or mark field/MET as `MISSING`.
  - EnvRecords may still be generated but flagged with `GAP_FILLED` or `MISSING` depending on impact.

env_version_id

- A deterministic ID representing the environment configuration used to compute an EnvRecord.
- Construct from:
  - Model versions (e.g., `nrlmsis-v1.0`, `jb2008-v1.1`, `srp-boxwing-v1.1`)
  - Data source versions
  - Time bucket
- Suggested implementation: truncated SHA-256 of a canonical version string, e.g.:
  - `env-2025-11-07T14Z-3f9a2c1b`

---

8. Database + Storage Architecture

8.1 Hot Storage (Timescale/Postgres)

Use **Postgres with TimescaleDB** for time-series and transactional needs.

```
CREATE TABLE env_records ( timestamp TIMESTAMPTZ PRIMARY KEY, rho REAL, F10_7 REAL,
Kp REAL, Dst REAL, a_drag REAL[3], -- m/s^2 a_srp REAL[3], -- m/s^2 a_albedo REAL[3],
-- m/s^2 a_empirical REAL[3], -- m/s^2 Sigma JSONB, -- covariance, m^2/s^4 qc_flag
TEXT, env_version_id TEXT );
```

Timescale hypertable configuration is recommended for efficient time-window queries.

8.2 Cold Archive (S3 + Parquet)

- **Storage:** S3 (or equivalent object store).
- **Format:** Parquet (columnar).
- **Compression:** Zstd (configurable level).
- **Partitioning:** `year=YYYY/month=MM/day=DD`.

Retention:

- Hot DB: 2–3 years of rolling EnvRecords.
- Cold archive: ≥10 years for replay and audit.

---

9. API Specification / Versioning

9.1 Example Endpoints

Environment Query

```
GET /v1/environment?time=2025-11-07T14:00:00Z&sat_id=SAT123
```

Example response:

```
{ "time": "2025-11-07T14:00:00Z", "sat_id": "SAT123", "forces": { "drag": [ax_d,
ay_d, az_d], // m/s^2 "srp": [ax_s, ay_s, az_s], // m/s^2 "albedo": [ax_a, ay_a,
az_a], // m/s^2 "empirical":[ax_e, ay_e, az_e] // m/s^2 }, "covariance": { "sigma_d":
..., "sigma_s": ..., "sigma_e": ..., "rho_ds": ..., "rho_de": ..., "rho_se": ... },
"uncertainty": { "drag_sigma": ..., "srp_sigma": ..., "albedo_sigma": ...,
"empirical_sigma": ... }, "qc_flag": "GOOD", "env_version_id": "env-2025-11-07T14Z-
3f9a2c1b", "quality": "nominal" }
```

Attribution Query

```
GET /v1/attribution?time=2025-11-07T14:00:00Z&sat_id=SAT123
```

Example response:

```
{ "time": "2025-11-07T14:00:00Z", "sat_id": "SAT123", "label": "drag_storm",
"confidence": "HIGH", "probabilities": { "maneuver": 0.10, "drag_storm": 0.82,
"srp_change": 0.03, "charging": 0.01, "other": 0.04 } }
```

Health & Feed Status

```
GET /v1/health GET /v1/feed-status
```

Maneuver Catalog

```
GET /v1/maneuvers?sat_id=SAT123&from=2025-11-01T00:00:00Z&to=2025-11-08T00:00:00Z
```

9.2 Versioning Rules

- **v1.x** is **additive** and backwards compatible.
    - New, optional fields may be added; existing fields not removed or changed.
- Any **breaking change** (field semantics or removal) → new **major version** (v2.x).
- The API should include `X-USWF-API-Version` headers for clarity.

9.3 API ICD / OpenAPI 3.1 Overview

- Provide an OpenAPI 3.1 spec file, e.g. `uswf-api-v1.yaml`, in version control.

- Include schemas for:
  - `EnvRecordResponse`
  - `AttributionResponse`
  - `ErrorResponse`
  - `HealthStatus`
- Standard errors:
  - `400 Bad Request` (invalid parameters)
  - `404 Not Found` (no EnvRecord / attribution for requested time/sat_id)
  - `429 Too Many Requests` (rate limit hit)
  - `503 Service Unavailable` (upstream feeds or core services degraded)

Rate limiting:

- Default: 1000 req/min per API key (configurable).

Authentication:

- OAuth2 / OIDC for internal clients.
- API keys for external, read-only integrations (if allowed by policy).

---

10. Deployment Architecture

Component	Technology
API service	FastAPI (Python)
Compute workers	Python async workers (e.g., Celery/RQ)
Message queue	Redis or RabbitMQ
Orchestration	Kubernetes
Database	Postgres + TimescaleDB
Archive	S3/compatible
CI/CD	GitHub Actions (or GitLab CI equivalent)
IaC	Terraform

Deployment pipeline:

```
dev → staging → shadow-mode → production
```

- **Shadow mode**: new model versions run in parallel with existing production models for ≥90 days.

- Only after evaluation and approval are they promoted to default.

---

11. Monitoring / Logging / Telemetry

Metrics to capture:

- API latency: p50/p90/p99
- EnvRecord generation latency per stage (ingest, force compute, covariance, attribution)
- Covariance PD failure rate (pre-regularization)
- Feed ingest success/failure counts and lags
- Distribution of `qc_flag` and `system_state` (nominal/degraded/limited/offline)
- R-metric over rolling windows and arcs

Tools:

- Prometheus for metrics
- Grafana for dashboards
- Structured logging (e.g., JSON logs) with correlation IDs for requests

Alerting examples:

- p99 latency > 60 s for `X` minutes
- Missing data fraction > 5% for any critical feed
- PD failures > 0.01% of covariance builds per day

---

12. Performance Optimization & Scaling

Overall SLO: **p99 < 60 s** end-to-end real-time path.

Refined Stage-Level Performance Budget

```
 Ingest (feed fetch + DB read): < 5 s Force Models (drag, SRP, albedo, emp): < 10 s
Maneuver Detection: < 5 s Attribution Inference: < 5 s Residual/OD-related
processing: < 20 s Covariance Calculation: < 10 s API Formatting + Response: < 5 s --
--------------------------------------------- Total Target (p99): < 60 s
```

Scaling strategies:

- Horizontally scale compute workers across nodes.
- Cache solar and density outputs where reuse is common.
- Precompute and cache albedo grids for typical orbits/altitudes.
- Periodically profile with `cProfile` and flame graphs to find hotspots.

---

13. Degradation & Fault-Tolerance

Pseudo-logic for degradation:

```
system_state = "nominal" if l1_data_missing: sigma_drag *= 1.2 qc_flag = "GAP_FILLED"
system_state = max_state(system_state, "degraded") if goes_down:
charging_terms_enabled = False attribution_confidence_cap = "LOW" system_state =
max_state(system_state, "degraded") if critical_feeds_missing_for_long: system_state
= "limited" if core_services_down: system_state = "offline"
```

Where `max_state` escalates among: `nominal < degraded < limited < offline`.

Principle: **Fail-soft, not fail-silent**. When degraded, the system **increases uncertainties** and flags results instead of returning misleadingly precise data.

---

14. Validation, QA & Benchmarking

Test Scenarios:

- Historical CME storm days
- Extreme beta-angle orbits
- Eclipse transitions and temperature-driven anomalies
- High-drag episodes
- Quiet "golden" orbits for baseline checking

R-Metric Acceptance:

$$R = \frac{\text{RMS(actual error)}}{\text{RMS(predicted } 1\sigma)} \in [0.8, 1.2]$$

Requirement: hold over **90 consecutive days** across the reference case set for a new model version before full promotion.

Automate:

- Daily regression tests comparing new outputs to reference runs
- Threshold checks on differences in key metrics

---

15. Security, Access Control & SRE Runbooks

15.1 Threat Model Summary

Threats considered:

- Tampering or spoofing of upstream space weather data feeds
- Unauthorized access to environment or attribution APIs
- Abuse of attribution outputs in high-stakes operational contexts

- Denial-of-service attacks against real-time endpoints
- Insider misconfiguration or unauthorized model promotion

Mitigations:

- HTTPS with certificate pinning where feasible
- Input validation, schema enforcement
- Auth + RBAC for all non-public endpoints
- Quotas and rate limiting per API key/client
- Strict control over model promotion and configuration changes

15.2 Authentication & Authorization

- Internal services: OAuth2 / OIDC SSO with RBAC roles (`viewer` , `analyst` , `admin` , `ops`).
- External integrations: API keys for read-limited access (if allowed by policy).
- Audit logging:
    - Login/refresh events
    - Configuration changes (including model switches)
    - Administrative actions (e.g., override flags, DR operations)

15.3 Disaster Recovery Runbook (RPO/RTO)

RPO (Recovery Point Objective): 24 hours
RTO (Recovery Time Objective): < 24 hours

Backups:

- Postgres:
    - Nightly full snapshot
    - Hourly WAL archiving
- S3/Parquet:
    - Versioning enabled
    - Lifecycle rules for long-term retention

Recovery Steps (Outline):

1. Declare DR event, freeze config changes.
2. Provision a new DB cluster and restore from last full snapshot.
3. Replay WAL up to last safe timestamp.
4. Reconnect API services and workers to restored DB.
5. Run smoke tests and sample EnvRecord checks against known good results.
6. Resume service with monitoring, document root cause and corrections.

Failover architecture starts as active–passive; multi-region active–active can be considered in future revisions.

---

16. Roadmap, Milestones & Dev-Ops Workflow

16.1 Phase Plan

Phase	Description	Key Outputs
Phase 0	Setup	Hiring, infra bootstrap, basic CI/CD, initial test env
Phase 1	Backbone	Data ingest, ETL, TimescaleDB, minimal API
Phase 2	v1.0	NRLMSIS drag + Cannonball SRP, EnvRecords, basic uncertainty
Phase 3	v1.1	Box-Wing SRP, Albedo/IR, dashboards, SDKs
Phase 4	v1.5	Maneuver detection, Attribution v1, partial covariance
Phase 5	v2.0	Full covariance, real-time SLA, SRE handoff and DR drills

Dev/ops workflow:

- Git-based development (e.g., GitHub Flow)
- CI (lint, tests, security checks)
- Staging environment with automated integration tests
- Shadow-mode deployments for new models
- Promotion after gate reviews

16.2 Cost Model (Cloud + Staffing)

High-level estimate (refined in Phase 0):

- **Total program:** $7.5–10M over ~54 months

Staffing:

- 10–14 FTE core team:
  - Product/Program Manager
  - 2x Physics/Orbit Modeling engineers
  - 2x Backend/Platform engineers
  - 1x ML/Attribution engineer
  - 1x Data/ETL engineer
  - 1x SRE/DevOps engineer
  - 1x QA/Testing engineer
  - 1x UX/Frontend (dashboards)
- Loaded cost ~$150–200k/FTE/year → majority of total budget.

Cloud (order-of-magnitude):

- Compute (Kubernetes nodes): \$5–15k/month
- Storage (DB + S3): \$1–3k/month
- Network/egress: \$0.5–2k/month
- Monitoring + logging services as configured

These values will be refined using specific cloud provider calculators in Phase 0.

---

17. Glossary / Definitions

Term	Definition	Units
EnvRecord	Immutable snapshot of environment & forces at a specific timestamp	—
a_drag, a_srp, a_albedo, a_emp	Acceleration vectors for force components	m/s^2
Σ_a (Sigma_a)	Force acceleration covariance matrix	m^2/s^4 (entries)
PD (Positive-Definite)	Matrix property required for stable filters	—
R-Metric	RMS(actual) / RMS(predicted 1σ)	dimensionless
A/m	Area-to-mass ratio (effective)	m^2/kg
Kp, Dst	Geomagnetic activity indices	dimensionless / nT
QC flag	Data quality status for EnvRecords	enum
SLO	Service Level Objective	—
SRE	Site Reliability Engineering	—

---

18. Appendix (Equations, Schemas, Payloads)

18.1 Core Equations

Total non-gravitational acceleration:

$$a_{total} = a_{drag} + a_{SRP} + a_{albedo/IR} + a_{emp} + a_{small}$$

Drag acceleration:

$$a_{drag} = -\frac{1}{2} C_D \frac{A_{eff}}{m} \rho \|v\| v \quad [\mathrm{m/s^2}]$$

Uncertainty propagation:

$$\sigma^2_{\text{drag}} = \sigma^2_\rho + \sigma^2_{C_D} + \sigma^2_{A/m} + \sigma^2_\theta$$

Covariance:

$$\Sigma_a \succ 0, \quad \Sigma_a \in \mathrm{R}^{3\times 3}, \quad \text{entries in } \mathrm{m}^2/\mathrm{s}^4$$

R-Metric:

$$R = \frac{\text{RMS(actual error)}}{\text{RMS(predicted } 1\sigma)}$$

18.2 Covariance JSON Schema (Sketch)

```
{ "type": "object", "properties": { "sigma_d": {"type": "number", "description":
"drag 1-sigma, m/s^2"}, "sigma_s": {"type": "number", "description": "srp 1-sigma,
m/s^2"}, "sigma_e": {"type": "number", "description": "empirical 1-sigma, m/s^2"},
"rho_ds": {"type": "number", "description": "corr(drag, srp)"}, "rho_de": {"type":
"number", "description": "corr(drag, empirical)"}, "rho_se": {"type": "number",
"description": "corr(srp, empirical)"} }, "required": ["sigma_d", "sigma_s",
"sigma_e"] }
```

---

19. Decision Log (Key Technology & Design Choices)

- **Framework (FastAPI vs Flask):**
  - FastAPI chosen for async support, native type hints, built-in OpenAPI generation, and performance.
- **Time-Series DB (TimescaleDB vs InfluxDB):**
  - Timescale/Postgres chosen for strong SQL semantics, transactional support, and better integration with existing tools.
- **Archive Format (Parquet on S3):**
  - Parquet provides efficient, columnar storage ideal for analytics and long-term retention.
- **Attribution Model Choice (Gradient Boost first):**
  - Gradient boosting is robust, interpretable, and lower ops overhead than deep sequence models.
  - Design explicitly leaves path to Transformers/LSTMs for v2.0.

- **Shadow-Mode Requirement:**
  - Any new model must run in shadow for ≥90 days to ensure stability across diverse regimes before promotion.
- **Unit Standards:**
  - Accelerations in **m/s^2**, covariance entries in **m^2/s^4**, explicitly documented across DB, API, and appendices.
- **Performance Sub-Budget:**
  - Computation budget broken down (force models, detection, attribution, residuals) to give teams clear latency targets.
- **Gap-Fill Method:**
  - **Linear interpolation** mandated for ≤30 min gaps to avoid ambiguity and inconsistent implementations.