INDEPENDENT RESEARCH PREPRINT — SUPPLEMENTARY MATERIALS

ARGUS-PD/POD-7K and Geomagnetic Companion Papers

Supplementary Materials & Reproducibility Annex

Christopher I. V. Farmer, LL.M.

Version 1.0.0 · July 2026

This annex was assembled 24 July 2026 from the deposited v1.0.0 reproducibility module; deposited tests re-run at assembly (7/7 passing). It answers one question for the reader of Papers I–III and the geomagnetic companion: where every number comes from.

1. Scope and honest limits

The deposited module is a controlled, non-operational screening model. In its own words: it implements conservation-law and parametric relationships only; it does not implement targeting, firing logic, trajectory optimisation, or a buildable electromagnetic launcher design. All quantities are illustrative screening values for a hypothetical research system; none are as-built figures. The geomagnetic screen uses the official IGRF-14 (epoch 2025.0) degree-1 Gauss coefficients and is an auditable screening model, not a substitute for a full degree-13 propagation, disturbance model, structural model, or controller simulation.

2. Module structure

- argus_pd_repro/argus_model.py — study constants; reference moving-mass budget; per-shot conservation-law relations; ideal magnetic-pressure screen; covariance miss screen; momentum-transfer screen; exact non-dominated set.
- argus_pd_repro/geomagnetic.py — IGRF-14 degree-1 field in ECEF; circular-orbit field/authority screen; rectangular-loop dipole model.
- argus_pd_repro/run_analysis.py — regenerates every table and figure cited by the publication set.
- tests/test_models.py — seven unit tests pinning the reference identities.
- data/ — generated CSVs plus run_provenance.json. figures/ — generated figures.

3. Study constants and reference mass budget

Constant	Value
Accelerator active length	7,000 m
Electrical efficiency η	0.35
Installed reserve fraction	0.20
Reference accounting tranches	6,000
Reference coupling area	0.200 m^2
Magnetic-pressure screen field	16.5 T
Screening distance	0.5 AU (7.47984×10^{10} m)
μ_0	$4\pi\times10^{-7}$ H/m

Reference moving-assembly budget (the total, not the core alone, is accelerated): core 0.150 kg; functional skin 0.015; load spreader 0.020; coupling collar 0.090; guidance & telemetry 0.010; release interface 0.010; other moving 0.005 — total 0.300 kg (validated by test to 10^{-12} kg). Collar areal density: 0.090 kg / 0.200 m^2 = 0.45 kg/m^2.

4. Governing relations (as implemented)

- Kinetic energy $E = \frac{1}{2}mv^2$; momentum $p = mv$.
- Mean force $F = E/L$; mean acceleration $a = v^2/2L$; transit time $t = 2L/v$ (L = 7,000 m).
- Admitted energy = E/η; installed nameplate = admitted × (1 + reserve); accounting tranche = installed / 6,000.
- Ideal magnetic-pressure field $B = \sqrt{(2\mu_0 F/A)}$ — a necessary force-density screen, not a realizable topology model. Required area at a field screen: $A = 2\mu_0 F/B^2$. Circular-equivalent diameter $Ø = \sqrt{(4A/\pi)}$.
- Screening flight time = 0.5 AU / v.
- Covariance miss screen (one-sigma RSS, three terms as deposited): $\sigma\_miss = \sqrt{[(R\cdot\sigma\_angular)^2 + (v\cdot\sigma\_timing)^2 + \sigma\_target^2]}$. Deposited screening defaults: $\sigma\_angular = 1.0\times10^{-9}$ rad; $\sigma\_timing = 1.0\times10^{-4}$ s; σ_target = 100 m.
- Momentum-transfer screen: $\Delta v = \beta\cdot m\cdot v\_rel / M\_asteroid$.
- Rectangular loop: dipole $m = I\cdot A\_enclosed$; resistance, ohmic power, conductor mass from perimeter, conductor area, resistivity (Al defaults).
- IGRF-14 degree-1 screen: dipole coefficients $g_1^0 = -29{,}350.0$ nT, $g_1^1 = -1{,}410.3$ nT, $h_1^1 = 4{,}545.5$ nT (epoch 2025.0); Earth reference radius 6,371.2 km; $\mu = 3.986004418\times10^{14}$ m^3/s^2; Earth rotation 7.2921150×10^{-5} rad/s. Initial conditions as deposited: circular orbit sampled over one period (1,440 samples), argument $u = n\cdot t$ from $u_0 = 0$ at the +X ECI axis, ascending node at RAAN 0°, ECEF frame aligned with ECI at t = 0. Per-sample torque-axis authority $\eta = \sqrt{(1 - (\hat{a}\cdot\hat{b})^2)}$; the screen reports min/mean/max field and mean/RMS/min authority.

5. Test suite

Seven unit tests pin the reference identities (150 GJ; 21.429 MN; 16.410 T vs the 16.5 T screen; 514.286 GJ nameplate; budget closure; frontier membership; screen behaviour). Result at assembly: 7 passed in 0.19s.

6. Deposited data tables

6.1 Reference family (iso-energy, 150 GJ)

Case	m (kg)	v (m/s)	E (J)	p (N·s)	a (m/s^2)	B ideal (T)	Installed (J)	0.5 AU (days)
R-DART	0.3	1e+06	1.5e+11	3e+05	7.143e+07	16.41	5.143e+11	0.8657
Q-DART	1.2	5e+05	1.5e+11	6e+05	1.786e+07	16.41	5.143e+11	1.731
V-DART	3.333	3e+05	1.5e+11	1e+06	6.429e+06	16.41	5.143e+11	2.886
M-DART	13.33	1.5e+05	1.5e+11	2e+06	1.607e+06	16.41	5.143e+11	5.772
T-DART	30	1e+05	1.5e+11	3e+06	7.143e+05	16.41	5.143e+11	8.657

6.2 Non-dominated screening frontier (28 points, exact set over the 522-point grid)

m (kg)	v (m/s)	E (J)	p (N·s)	Installed (J)	A@16.5T (m²)	0.5 AU (days)	σ_miss (m)
30	1e+05	1.5e+11	3e+06	5.143e+11	0.1978	8.657	125.3
19.74	1.2e+05	1.421e+11	2.369e+06	4.872e+11	0.1874	7.214	125.5
14.93	1.4e+05	1.463e+11	2.09e+06	5.017e+11	0.193	6.184	125.7
11.29	1.6e+05	1.446e+11	1.807e+06	4.957e+11	0.1907	5.411	125.9
8.544	1.8e+05	1.384e+11	1.538e+06	4.746e+11	0.1825	4.81	126.2
7.431	2e+05	1.486e+11	1.486e+06	5.096e+11	0.196	4.329	126.5
5.621	2.2e+05	1.36e+11	1.237e+06	4.664e+11	0.1794	3.935	126.8
4.889	2.4e+05	1.408e+11	1.173e+06	4.828e+11	0.1857	3.607	127.2
4.252	2.6e+05	1.437e+11	1.106e+06	4.928e+11	0.1896	3.33	127.6
3.699	2.8e+05	1.45e+11	1.036e+06	4.971e+11	0.1912	3.092	128
3.217	3e+05	1.448e+11	9.65e+05	4.963e+11	0.1909	2.886	128.4
2.798	3.2e+05	1.432e+11	8.953e+05	4.911e+11	0.1889	2.705	128.9
2.433	3.4e+05	1.407e+11	8.274e+05	4.822e+11	0.1855	2.546	129.4
2.116	3.6e+05	1.371e+11	7.619e+05	4.702e+11	0.1809	2.405	130
1.841	4e+05	1.473e+11	7.363e+05	5.049e+11	0.1942	2.164	131.1
1.601	4.2e+05	1.412e+11	6.724e+05	4.841e+11	0.1862	2.061	131.8
1.392	4.6e+05	1.473e+11	6.405e+05	5.051e+11	0.1943	1.882	133.1
1.211	4.8e+05	1.395e+11	5.813e+05	4.784e+11	0.184	1.804	133.8
1.053	5.2e+05	1.424e+11	5.477e+05	4.883e+11	0.1878	1.665	135.3
0.9162	5.6e+05	1.437e+11	5.13e+05	4.925e+11	0.1894	1.546	136.9
0.7968	6e+05	1.434e+11	4.781e+05	4.918e+11	0.1892	1.443	138.5
0.693	6.4e+05	1.419e+11	4.435e+05	4.866e+11	0.1872	1.353	140.3
0.6028	7e+05	1.477e+11	4.219e+05	5.063e+11	0.1948	1.237	143.2
0.5243	7.4e+05	1.435e+11	3.88e+05	4.921e+11	0.1893	1.17	145.2
0.456	8e+05	1.459e+11	3.648e+05	5.003e+11	0.1924	1.082	148.3
0.3966	8.6e+05	1.467e+11	3.411e+05	5.028e+11	0.1934	1.007	151.6
0.3449	9.2e+05	1.46e+11	3.173e+05	5.005e+11	0.1925	0.941	155.1
0.3	1e+06	1.5e+11	3e+05	5.143e+11	0.1978	0.8657	160

6.3 Geomagnetic orbit screen (deposited rows)

Axis	Alt (km)	Inc (°)	B min (μT)	B mean (μT)	B max (μT)	η mean	η min
ECI-X	400.0	0.0	24.77	25.26	25.7	0.9847	0.9595
ECI-Y	400.0	0.0	24.77	25.26	25.7	0.9837	0.9532
ECI-Z	400.0	0.0	24.77	25.26	25.7	0.2439	0.1601
ECI-X	400.0	51.6	24.77	31.99	38.83	0.8088	0.5685
ECI-Y	400.0	51.6	24.77	31.99	38.83	0.7552	0.3996
ECI-Z	400.0	51.6	24.77	31.99	38.83	0.8047	0.116
ECI-X	400.0	98.0	24.77	38.07	49.54	0.6391	0.002316
ECI-Y	400.0	98.0	24.77	38.07	49.54	0.9951	0.9883
ECI-Z	400.0	98.0	24.77	38.07	49.54	0.6511	0.1232
ECI-X	800.0	0.0	20.85	21.26	21.64	0.9847	0.959
ECI-Y	800.0	0.0	20.85	21.26	21.64	0.9837	0.9533
ECI-Z	800.0	0.0	20.85	21.26	21.64	0.2439	0.1601
ECI-X	800.0	51.6	20.85	26.94	32.74	0.8084	0.5665
ECI-Y	800.0	51.6	20.85	26.94	32.74	0.7559	0.4
ECI-Z	800.0	51.6	20.85	26.94	32.74	0.8041	0.1117
ECI-X	800.0	98.0	20.85	32.04	41.7	0.6393	0.004253
ECI-Y	800.0	98.0	20.85	32.04	41.7	0.9951	0.9883
ECI-Z	800.0	98.0	20.85	32.04	41.7	0.6508	0.1196
ECI-X	2000.0	0.0	13.11	13.37	13.6	0.9845	0.9572
ECI-Y	2000.0	0.0	13.11	13.37	13.6	0.9839	0.9536
ECI-Z	2000.0	0.0	13.11	13.37	13.6	0.2438	0.1601
ECI-X	2000.0	51.6	13.11	16.97	20.7	0.807	0.5597
ECI-Y	2000.0	51.6	13.11	16.97	20.7	0.7585	0.4016
ECI-Z	2000.0	51.6	13.11	16.97	20.7	0.8025	0.09736
ECI-X	2000.0	98.0	13.11	20.13	26.22	0.64	0.002907

ECI-Y	2000.0	98.0	13.11	20.13	26.22	0.9951	0.9883
ECI-Z	2000.0	98.0	13.11	20.13	26.22	0.6501	0.1069

7. Known v1.0.0 inconsistencies (scheduled for v1.1)

- σ_miss term count: the deposited, tested code computes a three-term RSS; Paper I prints four terms and Paper III five. The deposited code is the arbiter of what v1.0.0 computed; the manuscripts will be aligned in v1.1.
- The orbit-screen initial conditions (u_0 = 0 from +X ECI, RAAN 0°, ECEF aligned at t = 0, 1,440 samples) were previously stated only in code; they are published in §4 of this annex.
- The papers' data/code availability statements predate this annex; v1.1 will point to the deposited package (and, on deposit, its DOI) explicitly.

8. Appendix — deposited code listings (verbatim)

argus_pd_repro/argus_model.py

```
"""Controlled, non-operational screening model for the ARGUS-PD/POD-7K papers.

The module implements conservation-law and parametric relationships only.  It
does not implement targeting, firing logic, trajectory optimisation, or a
buildable electromagnetic launcher design.
"""

from __future__ import annotations

from dataclasses import asdict, dataclass
import math
from typing import Iterable



MU0 = 4.0e-7 * math.pi
AU_M = 149_597_870_700.0



@dataclass(frozen=True)
class StudyConstants:
    accelerator_length_m: float = 7_000.0
    electrical_efficiency: float = 0.35
    installed_reserve_fraction: float = 0.20
    reference_tranches: int = 6_000
    reference_coupling_area_m2: float = 0.200
    field_screen_t: float = 16.5
    screening_distance_m: float = 0.5 * AU_M



@dataclass(frozen=True)
class MassBudget:
    """Reference moving-assembly budget.

    The total, not the core alone, is the mass accelerated through the full
    launcher.  Values are explicit study allocations, not as-built masses.
    """
```

```
    core_kg: float = 0.150
    functional_skin_kg: float = 0.015
    load_spreader_kg: float = 0.020
    coupling_collar_kg: float = 0.090
    guidance_telemetry_kg: float = 0.010
    release_interface_kg: float = 0.010
    other_moving_kg: float = 0.005

    @property
    def total_kg(self) -> float:
        return sum(asdict(self).values())

    @property
    def collar_areal_density_kg_m2(self) -> float:
        return self.coupling_collar_kg / 0.200

    def validate(self, expected_total_kg: float = 0.300, tolerance: float = 1e-12) -> None:
        if abs(self.total_kg - expected_total_kg) > tolerance:
            raise ValueError(
                f"Moving-mass budget is {self.total_kg:.9f} kg, "
                f"not {expected_total_kg:.9f} kg"
            )


@dataclass(frozen=True)
class ShotResult:
    total_moving_mass_kg: float
    velocity_m_s: float
    kinetic_energy_j: float
    momentum_n_s: float
    mean_acceleration_m_s2: float
    transit_time_s: float
    mean_force_n: float
    ideal_field_t: float
    admitted_energy_j: float
    installed_nameplate_j: float
    accounting_tranche_j: float
    mean_mechanical_power_w: float
    exit_mechanical_power_w: float
    mean_admitted_power_w: float
    exit_admitted_power_w: float
    screening_flight_time_s: float


def kinetic_energy(mass_kg: float, velocity_m_s: float) -> float:
    return 0.5 * mass_kg * velocity_m_s**2


def momentum(mass_kg: float, velocity_m_s: float) -> float:
    return mass_kg * velocity_m_s


def ideal_maxwell_field(force_n: float, coupling_area_m2: float) -> float:
    """Equivalent field from the ideal magnetic-pressure ceiling.

    This is a necessary force-density screen, not a realizable topology model.
    """

    if force_n < 0 or coupling_area_m2 <= 0:
        raise ValueError("Force must be non-negative and area must be positive")
    return math.sqrt(2.0 * MU0 * force_n / coupling_area_m2)
```

```
def coupling_area_required(force_n: float, field_t: float) -> float:
    if force_n < 0 or field_t <= 0:
        raise ValueError("Force must be non-negative and field must be positive")
    return 2.0 * MU0 * force_n / field_t**2



def circular_equivalent_diameter(area_m2: float) -> float:
    if area_m2 <= 0:
        raise ValueError("Area must be positive")
    return math.sqrt(4.0 * area_m2 / math.pi)



def shot(
    total_moving_mass_kg: float,
    velocity_m_s: float,
    constants: StudyConstants = StudyConstants(),
    coupling_area_m2: float | None = None,
) -> ShotResult:
    if total_moving_mass_kg <= 0 or velocity_m_s <= 0:
        raise ValueError("Mass and velocity must be positive")
    area = coupling_area_m2 or constants.reference_coupling_area_m2
    energy = kinetic_energy(total_moving_mass_kg, velocity_m_s)
    force = energy / constants.accelerator_length_m
    acceleration = velocity_m_s**2 / (2.0 * constants.accelerator_length_m)
    transit = 2.0 * constants.accelerator_length_m / velocity_m_s
    admitted = energy / constants.electrical_efficiency
    installed = admitted * (1.0 + constants.installed_reserve_fraction)
    return ShotResult(
        total_moving_mass_kg=total_moving_mass_kg,
        velocity_m_s=velocity_m_s,
        kinetic_energy_j=energy,
        momentum_n_s=momentum(total_moving_mass_kg, velocity_m_s),
        mean_acceleration_m_s2=acceleration,
        transit_time_s=transit,
        mean_force_n=force,
        ideal_field_t=ideal_maxwell_field(force, area),
        admitted_energy_j=admitted,
        installed_nameplate_j=installed,
        accounting_tranche_j=installed / constants.reference_tranches,
        mean_mechanical_power_w=energy / transit,
        exit_mechanical_power_w=force * velocity_m_s,
        mean_admitted_power_w=admitted / transit,
        exit_admitted_power_w=force * velocity_m_s / constants.electrical_efficiency,
        screening_flight_time_s=constants.screening_distance_m / velocity_m_s,
    )



def miss_distance_rss(
    range_m: float,
    angular_sigma_rad: float,
    velocity_m_s: float,
    timing_sigma_s: float,
    target_state_sigma_m: float = 0.0,
) -> float:
    """One-sigma root-sum-square screening miss distance.

    The expression is deliberately a covariance screen, not a targeting model.
    """

    terms = (
        range_m * angular_sigma_rad,
```

```
        velocity_m_s * timing_sigma_s,
        target_state_sigma_m,
    )
    return math.sqrt(sum(value * value for value in terms))



def asteroid_delta_v(
    projectile_mass_kg: float,
    relative_velocity_m_s: float,
    asteroid_mass_kg: float,
    beta: float = 1.0,
) -> float:
    if asteroid_mass_kg <= 0 or beta < 0:
        raise ValueError("Asteroid mass must be positive and beta non-negative")
    return beta * projectile_mass_kg * relative_velocity_m_s / asteroid_mass_kg



def platform_recoil_delta_v(
    projectile_mass_kg: float, velocity_m_s: float, platform_mass_kg: float
) -> float:
    if platform_mass_kg <= 0:
        raise ValueError("Platform mass must be positive")
    return projectile_mass_kg * velocity_m_s / platform_mass_kg



def regular_polygon_area_from_vertex_diameter(vertex_diameter_m: float, sides: int = 32) -> float:
    if vertex_diameter_m <= 0 or sides < 3:
        raise ValueError("Invalid polygon")
    circumradius = vertex_diameter_m / 2.0
    return 0.5 * sides * circumradius**2 * math.sin(2.0 * math.pi / sides)



def uniform_core_length(
    core_mass_kg: float,
    density_kg_m3: float,
    vertex_diameter_m: float = 0.050,
    sides: int = 32,
) -> float:
    if core_mass_kg <= 0 or density_kg_m3 <= 0:
        raise ValueError("Mass and density must be positive")
    area = regular_polygon_area_from_vertex_diameter(vertex_diameter_m, sides)
    return core_mass_kg / (density_kg_m3 * area)



def nondominated(
    rows: Iterable[dict[str, float]],
    maximize: tuple[str, ...],
    minimize: tuple[str, ...],
) -> list[dict[str, float]]:
    """Return the exact non-dominated set for a modest screening grid."""


    items = list(rows)
    front: list[dict[str, float]] = []
    for i, candidate in enumerate(items):
        dominated = False
        for j, challenger in enumerate(items):
            if i == j:
                continue
            no_worse = all(challenger[k] >= candidate[k] for k in maximize)
            no_worse &= all(challenger[k] <= candidate[k] for k in minimize)
            strictly_better = any(challenger[k] > candidate[k] for k in maximize)
            strictly_better |= any(challenger[k] < candidate[k] for k in minimize)
            if no_worse and strictly_better:
```

```
                dominated = True
                break
        if not dominated:
            front.append(candidate)
    return front
```

argus_pd_repro/geomagnetic.py

```
"""Low-degree IGRF-14 orbit screen for the companion attitude-control paper.

The calculation uses the official 2025.0 degree-1 Gauss coefficients.  It is
an auditable screening model, not a substitute for a full degree-13 IGRF
propagation, geomagnetic-disturbance model, structural model, or controller
simulation.
"""

from __future__ import annotations

import math
import numpy as np



EARTH_REFERENCE_RADIUS_M = 6_371_200.0
EARTH_MU_M3_S2 = 3.986004418e14
EARTH_ROTATION_RAD_S = 7.2921150e-5

# IGRF-14, epoch 2025.0, degree 1, nT.
G10_NT = -29_350.0
G11_NT = -1_410.3
H11_NT = 4_545.5



def _rz(angle_rad: float) -> np.ndarray:
    c, s = math.cos(angle_rad), math.sin(angle_rad)
    return np.array([[c, -s, 0.0], [s, c, 0.0], [0.0, 0.0, 1.0]])



def igrf14_degree1_ecef(position_ecef_m: np.ndarray) -> np.ndarray:
    x, y, z = (float(v) for v in position_ecef_m)
    r = math.sqrt(x * x + y * y + z * z)
    theta = math.acos(z / r)
    phi = math.atan2(y, x)
    st, ct = math.sin(theta), math.cos(theta)
    sp, cp = math.sin(phi), math.cos(phi)
    scale = (EARTH_REFERENCE_RADIUS_M / r) ** 3
    common = G10_NT * ct + G11_NT * st * cp + H11_NT * st * sp
    b_r = 2.0 * scale * common
    b_theta = scale * (G10_NT * st - G11_NT * ct * cp - H11_NT * ct * sp)
    b_phi = scale * (G11_NT * sp - H11_NT * cp)

    e_r = np.array([st * cp, st * sp, ct])
    e_theta = np.array([ct * cp, ct * sp, -st])
    e_phi = np.array([-sp, cp, 0.0])
    return (b_r * e_r + b_theta * e_theta + b_phi * e_phi) * 1e-9



def circular_orbit_screen(
    altitude_m: float,
    inclination_deg: float,
    samples: int = 1440,
```

```
    torque_axis_eci: np.ndarray | None = None,
) -> dict[str, float]:
    radius = EARTH_REFERENCE_RADIUS_M + altitude_m
    mean_motion = math.sqrt(EARTH_MU_M3_S2 / radius**3)
    period_s = 2.0 * math.pi / mean_motion
    inc = math.radians(inclination_deg)
    axis = np.asarray(
        torque_axis_eci if torque_axis_eci is not None else np.array([0.0, 0.0, 1.0]),
        dtype=float,
    )
    axis /= np.linalg.norm(axis)


    magnitudes: list[float] = []
    authority: list[float] = []
    for k in range(samples):
        t = period_s * k / samples
        u = mean_motion * t
        position_eci = radius * np.array(
            [math.cos(u), math.sin(u) * math.cos(inc), math.sin(u) * math.sin(inc)]
        )
        ecef_from_eci = _rz(-EARTH_ROTATION_RAD_S * t)
        position_ecef = ecef_from_eci @ position_eci
        field_ecef = igrf14_degree1_ecef(position_ecef)
        field_eci = ecef_from_eci.T @ field_ecef
        magnitude = float(np.linalg.norm(field_eci))
        b_hat = field_eci / magnitude
        eta = math.sqrt(max(0.0, 1.0 - float(np.dot(axis, b_hat)) ** 2))
        magnitudes.append(magnitude)
        authority.append(eta)


    b = np.asarray(magnitudes)
    a = np.asarray(authority)
    return {
        "altitude_km": altitude_m / 1000.0,
        "inclination_deg": inclination_deg,
        "period_min": period_s / 60.0,
        "field_min_uT": float(b.min() * 1e6),
        "field_mean_uT": float(b.mean() * 1e6),
        "field_max_uT": float(b.max() * 1e6),
        "axis_authority_mean": float(a.mean()),
        "axis_authority_rms": float(math.sqrt(np.mean(a * a))),
        "axis_authority_min": float(a.min()),
    }



def rectangular_loop(
    length_m: float,
    width_m: float,
    current_a: float,
    conductor_area_m2: float = 1.0e-4,
    resistivity_ohm_m: float = 2.82e-8,
    density_kg_m3: float = 2700.0,
) -> dict[str, float]:
    enclosed_area = length_m * width_m
    perimeter = 2.0 * (length_m + width_m)
    resistance = resistivity_ohm_m * perimeter / conductor_area_m2
    return {
        "length_m": length_m,
        "width_m": width_m,
        "enclosed_area_m2": enclosed_area,
        "perimeter_m": perimeter,
        "current_a": current_a,
        "dipole_a_m2": current_a * enclosed_area,
        "resistance_ohm": resistance,
        "ohmic_power_w": current_a**2 * resistance,
        "conductor_mass_kg": density_kg_m3 * perimeter * conductor_area_m2,
        "current_density_a_m2": current_a / conductor_area_m2,
```

```
    }
```

argus_pd_repro/run_analysis.py

```
"""Generate the tables and figures cited by the revised publication set."""

from __future__ import annotations

import csv
from dataclasses import asdict
import json
import math
from pathlib import Path
import sys

import matplotlib.pyplot as plt
import numpy as np

ROOT = Path(__file__).resolve().parents[1]
sys.path.insert(0, str(ROOT))

from argus_pd_repro.argus_model import (  # noqa: E402
    MassBudget,
    StudyConstants,
    circular_equivalent_diameter,
    coupling_area_required,
    miss_distance_rss,
    nondominated,
    shot,
    uniform_core_length,
)
from argus_pd_repro.geomagnetic import circular_orbit_screen, rectangular_loop  # noqa: E402



DATA = ROOT / "data"
FIGURES = ROOT / "figures"
DATA.mkdir(exist_ok=True)
FIGURES.mkdir(exist_ok=True)



def write_csv(name: str, rows: list[dict]) -> None:
    if not rows:
        raise ValueError(f"No rows for {name}")
    with (DATA / name).open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(handle, fieldnames=list(rows[0].keys()))
        writer.writeheader()
        writer.writerows(rows)



def reference_family() -> list[dict]:
    constants = StudyConstants()
    cases = [
        ("R-DART", 0.300, 1_000_000.0),
        ("Q-DART", 1.200, 500_000.0),
        ("V-DART", 10.0 / 3.0, 300_000.0),
        ("M-DART", 40.0 / 3.0, 150_000.0),
        ("T-DART", 30.000, 100_000.0),
    ]
    rows: list[dict] = []
    for label, mass, velocity in cases:
        result = shot(mass, velocity, constants)
```

```
        row = {"case": label, **asdict(result)}
        row["coupling_area_at_16_5_t_m2"] = coupling_area_required(
            result.mean_force_n, constants.field_screen_t
        )
        row["coupling_diameter_at_16_5_t_m"] = circular_equivalent_diameter(
            row["coupling_area_at_16_5_t_m2"]
        )
        row["screening_flight_time_days"] = result.screening_flight_time_s / 86_400.0
        rows.append(row)
    return rows



def mass_budget_rows() -> list[dict]:
    budget = MassBudget()
    budget.validate()
    rows = [{"component": key, "mass_kg": value} for key, value in asdict(budget).items()]
    rows.append({"component": "TOTAL MOVING ASSEMBLY", "mass_kg": budget.total_kg})
    rows.append(
        {
            "component": "Illustrative full-diamond-equivalent core length (m)",
            "mass_kg": uniform_core_length(budget.core_kg, 3510.0),
        }
    )
    rows.append(
        {
            "component": "Collar areal density (kg/m2)",
            "mass_kg": budget.collar_areal_density_kg_m2,
        }
    )
    return rows



def screening_grid() -> tuple[list[dict], list[dict]]:
    constants = StudyConstants()
    masses = np.geomspace(0.30, 30.0, 34)
    velocities = np.linspace(100_000.0, 1_000_000.0, 46)
    rows: list[dict] = []
    for mass in masses:
        for velocity in velocities:
            result = shot(float(mass), float(velocity), constants)
            if result.kinetic_energy_j > 150.0e9 * (1.0 + 1e-12):
                continue
            area = coupling_area_required(result.mean_force_n, constants.field_screen_t)
            miss = miss_distance_rss(
                constants.screening_distance_m,
                angular_sigma_rad=1.0e-9,
                velocity_m_s=velocity,
                timing_sigma_s=1.0e-4,
                target_state_sigma_m=100.0,
            )
            rows.append(
                {
                    "total_moving_mass_kg": float(mass),
                    "velocity_m_s": float(velocity),
                    "kinetic_energy_j": result.kinetic_energy_j,
                    "momentum_n_s": result.momentum_n_s,
                    "installed_nameplate_j": result.installed_nameplate_j,
                    "required_area_m2_at_16_5_t": area,
                    "coupling_diameter_m": circular_equivalent_diameter(area),
                    "flight_time_days_at_0_5_au": result.screening_flight_time_s / 86_400.0,
                    "screening_miss_sigma_m": miss,
                }
            )
    # Performance frontier under the explicit 150 GJ mechanical ceiling.
    # Energy, coupling area and uncertainty remain reported attributes; adding
    # them as minimisation objectives would make nearly every low-energy point
    # trivially non-dominated and would obscure the speed-momentum trade.
```

```
    front = nondominated(
        rows,
        maximize=("velocity_m_s", "momentum_n_s"),
        minimize=(),
    )
    front.sort(key=lambda row: row["velocity_m_s"])
    return rows, front



def geomagnetic_rows() -> list[dict]:
    rows: list[dict] = []
    axes = {
        "ECI-X": np.array([1.0, 0.0, 0.0]),
        "ECI-Y": np.array([0.0, 1.0, 0.0]),
        "ECI-Z": np.array([0.0, 0.0, 1.0]),
    }
    for altitude_km in (400.0, 800.0, 2_000.0):
        for inclination in (0.0, 51.6, 98.0):
            for axis_name, axis in axes.items():
                row = circular_orbit_screen(
                    altitude_km * 1000.0, inclination, torque_axis_eci=axis
                )
                rows.append({"torque_axis": axis_name, **row})
    return rows



def loop_rows() -> list[dict]:
    return [
        rectangular_loop(1_000.0, 100.0, current)
        for current in (1.0, 10.0, 100.0, 100_000.0)
    ]



def make_figures(family: list[dict], front: list[dict], geomag: list[dict]) -> None:
    plt.rcParams.update({"font.size": 9, "figure.dpi": 160})


    fig, ax = plt.subplots(figsize=(6.3, 4.0))
    ax.plot(
        [r["velocity_m_s"] / 1000 for r in family],
        [r["momentum_n_s"] / 1e6 for r in family],
        marker="o",
        color="#1F4D78",
    )
    for row in family:
        ax.annotate(
            row["case"],
            (row["velocity_m_s"] / 1000, row["momentum_n_s"] / 1e6),
            xytext=(4, 4),
            textcoords="offset points",
        )
    ax.set(xlabel="Release speed (km/s)", ylabel="Direct momentum (MN s)")
    ax.set_title("150 GJ iso-energy family: response time versus momentum")
    ax.grid(alpha=0.25)
    fig.tight_layout()
    fig.savefig(FIGURES / "iso_energy_trade.png")
    fig.savefig(FIGURES / "iso_energy_trade.pdf")
    plt.close(fig)


    fig, ax = plt.subplots(figsize=(6.3, 4.0))
    ax.scatter(
        [r["velocity_m_s"] / 1000 for r in front],
        [r["momentum_n_s"] / 1e6 for r in front],
        c=[r["installed_nameplate_j"] / 1e9 for r in front],
        cmap="viridis",
```

```
    s=18,
)
ax.set(xlabel="Release speed (km/s)", ylabel="Direct momentum (MN s)")
ax.set_title("Non-dominated screening set (colour: installed GJ)")
ax.grid(alpha=0.25)
fig.tight_layout()
fig.savefig(FIGURES / "pareto_front.png")
fig.savefig(FIGURES / "pareto_front.pdf")
plt.close(fig)


subset = [
    row
    for row in geomag
    if row["torque_axis"] == "ECI-Z" and row["altitude_km"] == 400.0
]
fig, ax = plt.subplots(figsize=(6.3, 4.0))
x = [str(row["inclination_deg"]) for row in subset]
ax.bar(x, [row["axis_authority_mean"] for row in subset], color="#2E74B5")
ax.set(
    xlabel="Orbit inclination (deg)",
    ylabel="Mean attainable-axis factor",
    ylim=(0.0, 1.0),
)
ax.set_title("IGRF-14 degree-1 orbital authority screen, 400 km")
ax.grid(axis="y", alpha=0.25)
fig.tight_layout()
fig.savefig(FIGURES / "geomagnetic_authority_screen.png")
fig.savefig(FIGURES / "geomagnetic_authority_screen.pdf")
plt.close(fig)


labels = [
    "Weighed moving interface",
    "Representative coupling cell",
    "Storage and switching module",
    "Physical multi-cell string",
    "Unarmed deployment demonstrator",
    "Release and impact validation",
    "Target-specific civil review",
]
fig, ax = plt.subplots(figsize=(6.3, 5.6))
ax.set_xlim(0, 1)
ax.set_ylim(0, 1)
ax.axis("off")
y_positions = np.linspace(0.92, 0.08, len(labels))
for idx, (label, y) in enumerate(zip(labels, y_positions, strict=True)):
    ax.text(
        0.5,
        y,
        label,
        ha="center",
        va="center",
        color="#15324A",
        fontweight="bold",
        bbox={
            "boxstyle": "round,pad=0.45",
            "facecolor": "#F4F6F9",
            "edgecolor": "#2E74B5",
            "linewidth": 1.2,
        },
    )
    if idx < len(labels) - 1:
        ax.annotate(
            "",
            xy=(0.5, y_positions[idx + 1] + 0.035),
            xytext=(0.5, y - 0.035),
            arrowprops={"arrowstyle": "-|>", "color": "#2E74B5", "lw": 1.2},
        )
ax.set_title(
```

```
        "Evidence-gated development sequence",
        color="#1F4D78",
        fontweight="bold",
        pad=12,
    )
    fig.tight_layout()
    fig.savefig(FIGURES / "evidence_gate_sequence.png")
    fig.savefig(FIGURES / "evidence_gate_sequence.pdf")
    plt.close(fig)



def main() -> None:
    family = reference_family()
    mass = mass_budget_rows()
    grid, front = screening_grid()
    geomag = geomagnetic_rows()
    loops = loop_rows()
    write_csv("reference_family.csv", family)
    write_csv("reference_mass_budget.csv", mass)
    write_csv("design_space_grid.csv", grid)
    write_csv("pareto_front.csv", front)
    write_csv("geomagnetic_orbit_screen.csv", geomag)
    write_csv("geomagnetic_loop_screen.csv", loops)
    make_figures(family, front, geomag)


    provenance = {
        "model_scope": "conservation-law and parametric screening only",
        "generated_files": sorted(p.name for p in DATA.glob("*.csv"))
        + sorted(p.name for p in FIGURES.glob("*")),
        "controlled_constants": asdict(StudyConstants()),
        "mass_budget_total_kg": MassBudget().total_kg,
        "igrf_scope": "IGRF-14 2025.0 degree-1 Gauss coefficients; not full degree-13",
    }
    (DATA / "run_provenance.json").write_text(
        json.dumps(provenance, indent=2) + "\n", encoding="utf-8"
    )



if __name__ == "__main__":
    main()
```

tests/test_models.py

```
from __future__ import annotations

import math
import unittest


from argus_pd_repro.argus_model import (
    MassBudget,
    StudyConstants,
    ideal_maxwell_field,
    miss_distance_rss,
    shot,
    uniform_core_length,
)
from argus_pd_repro.geomagnetic import circular_orbit_screen, rectangular_loop



class ArgusModelTests(unittest.TestCase):
    def test_reference_arithmetic(self) -> None:
        result = shot(0.300, 1_000_000.0)
        self.assertAlmostEqual(result.kinetic_energy_j, 150.0e9, places=3)
        self.assertAlmostEqual(result.momentum_n_s, 300_000.0, places=6)
```

```
        self.assertAlmostEqual(result.mean_acceleration_m_s2, 7.142857142857e7, places=3)
        self.assertAlmostEqual(result.transit_time_s, 0.014, places=12)
        self.assertAlmostEqual(result.mean_force_n, 21.428571428571e6, places=3)
        self.assertAlmostEqual(result.admitted_energy_j, 428.571428571429e9, places=3)
        self.assertAlmostEqual(result.installed_nameplate_j, 514.285714285714e9, places=3)
        self.assertAlmostEqual(result.accounting_tranche_j, 85.714285714286e6, places=3)

    def test_field_screen(self) -> None:
        result = shot(0.300, 1_000_000.0)
        self.assertAlmostEqual(result.ideal_field_t, 16.407, delta=0.005)
        core_field = ideal_maxwell_field(result.mean_force_n, 0.0019509)
        self.assertAlmostEqual(core_field, 166.2, delta=0.3)

    def test_mass_budget_is_total(self) -> None:
        budget = MassBudget()
        budget.validate()
        self.assertAlmostEqual(budget.total_kg, 0.300, places=12)
        self.assertLess(budget.core_kg, budget.total_kg)
        self.assertAlmostEqual(budget.collar_areal_density_kg_m2, 0.45, places=12)

    def test_core_length_uses_core_mass_only(self) -> None:
        length = uniform_core_length(0.150, 3510.0)
        self.assertAlmostEqual(length, 0.0219, delta=0.0002)

    def test_uncertainty_is_monotone(self) -> None:
        a = miss_distance_rss(1e8, 1e-9, 1e5, 1e-4, 10.0)
        b = miss_distance_rss(1e8, 2e-9, 1e5, 1e-4, 10.0)
        self.assertGreater(b, a)

    def test_geomagnetic_field_is_physical(self) -> None:
        row = circular_orbit_screen(400_000.0, 51.6, samples=360)
        self.assertGreater(row["field_min_uT"], 15.0)
        self.assertLess(row["field_max_uT"], 70.0)
        self.assertGreaterEqual(row["axis_authority_min"], 0.0)
        self.assertLessEqual(row["axis_authority_rms"], 1.0)

    def test_rectangular_loop_closes(self) -> None:
        row = rectangular_loop(1_000.0, 100.0, 100.0)
        self.assertAlmostEqual(row["dipole_a_m2"], 1.0e7, places=3)
        self.assertAlmostEqual(row["ohmic_power_w"], 6204.0, delta=5.0)
        self.assertAlmostEqual(row["conductor_mass_kg"], 594.0, delta=1.0)


if __name__ == "__main__":
    unittest.main()
```

9. SHA-256 manifest (reproducibility module)

```
691e759d2f685d4b641c558a0a71ae5ab74f01c782f060a6b1cb69e17f61c2fb  argus_pd_repro/__init__.py
3f2e4c0c4305eb9487ca17227731e407c2ad1741750c506eceba7ea8f7728356  argus_pd_repro/argus_model.py
69e99a67662ea2f069acc0852581221750789ad350509b5d5980fda6ff9f22ba  argus_pd_repro/geomagnetic.py
7a180ad14c193d6741b5b1961661db365c43cc2998eaa1db8d600a097d59e49d  argus_pd_repro/run_analysis.py
e617286afa6e95c366cab69b7180e07d7b6db6cb8efac38db0f6bdf5f5310978  data/design_space_grid.csv
4ad455d34067fa51125b76bb3e1befd14f278dd313fe4234e47fec04dc2f6517  data/geomagnetic_loop_screen.csv
1063a14df9b6230266cdc409345d31d75683cd94c2672d8f21b5d4c3160fb4b1  data/geomagnetic_orbit_screen.csv
8da46fcb85df71e7bbccdb96e1bf290af8da84bc7550ddf628730fc817fd9804  data/pareto_front.csv
a5fe114ee5fcf9c55bc19fc020392b0060d2a74aa950b60fc92717f0a30f8041  data/reference_family.csv
37d44ff614132fed48c83d9d86c2acb2b290e937feb2fd3b8c3dabc8a1774f3e  data/reference_mass_budget.csv
8f07b071e3ddd2b7d65eab49b9ffd6417315313dccf32f685cb78d46682cce73  data/run_provenance.json
e3b79dcfe1cb05bc0ae8016401295999e090382c5afa13168f53369af2fec6c2  tests/test_models.py
```