Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Runtime telemetry from evolutionary and static resource policies on a Jetson Nano

Antonio Souto Rodriguez

antonio.rodriguez@icomp.ufam.edu.br

Federal University of Amazonas

José Reginaldo Hughes Carvalho

Federal University of Amazonas

data-descriptor

Keywords: embedded systems, heterogeneous computing, runtime resource management, power and thermal management, evolutionary computation, measurement methodology, research data

Posted Date: November 3rd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11570131/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Runtime telemetry from evolutionary and static resource policies on a Jetson Nano

Antonio Souto Rodriguez[1,*] and José Reginaldo Hughes Carvalho[1]
[1]Institute of Computing, Federal University of Amazonas (ICOMP/UFAM), Manaus, Brazil
[*]Corresponding author: antonio.rodriguez@icomp.ufam.edu.br
ORCID: A.S.R. 0000-0002-2107-5074; J.R.H.C. 0000-0002-3828-7233

Abstract

We describe 108 hours of recorded execution on an NVIDIA Jetson Nano (2 GB): eighteen six-hour missions in which four image-processing workloads alternate in twelve 1 800-s phases. Six missions run a persistent surrogate-assisted evolutionary runtime controller with adaptive objective weights and a thermal supervisor, two under each of three application thermal profiles; twelve run fixed CPU-only, GPU-only and balanced policies on the same schedule. Every mission carries a 2-Hz system trace, one record per finalized camera frame (8.08 million in all), an external power-meter stream, a hot-swap log and a sealed collector event log; controller missions add one record per 800-ms control period (161 723) with requested controls, software read-back and the frozen forecast of each decision, retrospectively associated with recorded outcomes along the selected trajectory. The two missions per arm and profile are chronological repetitions, not randomized replicates. The recordings separate requests, read-back and measured outcomes, were validated channel by channel, and are released with verifiers and the scripts that regenerate every derived table.

Subject terms: embedded systems; heterogeneous computing; runtime resource management; power and thermal management; evolutionary computation; measurement methodology; research data.

Background & Summary

Runtime resource managers for heterogeneous systems-on-chip decide, every few hundred milliseconds, which processor executes the current kernel, with how many threads and under which frequency cap, while the workload and the thermal state change. Their evaluation depends on what the platform reports about itself, and an earlier audit of the controller recorded here showed that a logged request is not the same thing as an applied setting [1]: in all recordings before October 2026 the controller's CPU-frequency caps were never written to the kernel, although every log said they were. Public recordings of such controllers in operation are rare (the controller recorded here is the ERL framework — evolutionary reinforcement learning for runtime resource management — of the first author's doctoral work); recordings in which request, software read-back and measured outcome are kept apart, and in which fixed-policy references were run on the same schedule, are rarer still.

This descriptor documents the campaign recorded after that audit's corrections (the controller evaluation itself is reported in a separate manuscript under consideration, supplied as a related file): eighteen missions of 21 600 s each, recorded between 10 and 23 October 2026 on one device (Fig. 1). Six missions run the evolutionary controller (two chronological repetitions under each of three application thermal profiles); twelve run static policies — four CPU workers with the GPU disabled (CPU_STATIC), one worker with full GPU split (GPU_STATIC) and two workers with half split (BALANCED) — two repetitions each under the inactive and aggressive profiles. All eighteen follow the same workload schedule, so each workload is visited three times per mission at different thermal states and with different histories.

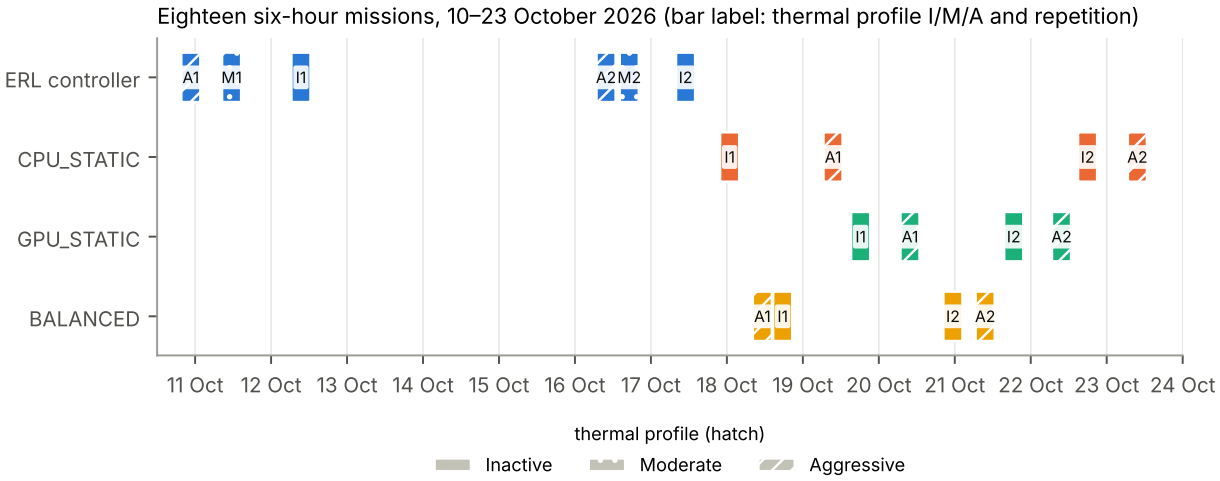


Figure 1: The eighteen missions by start time and arm; the bar label gives the thermal profile (I inactive, M moderate, A aggressive) and the repetition number; hatching encodes the profile. Every bar is one continuous 21 600-s mission of twelve 1 800-s phases.

The corpus is intended for four kinds of reuse. First, as a benchmark of controller behaviour under nonstationary workloads: hot-swap cost, state continuity, decision patterns and their consequences can be read directly from the traces. Second, as a measurement-methodology resource: because the same quantity is recorded at several points of the control path (gene, commanded cap, policy read-back, reported current frequency; GPU preference, permission, utilization), the recordings support studies of how far self-reported telemetry can be trusted. Third, as evaluation data for surrogate models of throughput, power and temperature on an edge SoC: the collector freezes the forecast of every control period before actuation and stamps issue, application and outcome times, giving 161 717 scored forecast–outcome pairs (161 649 after excluding the 68 pairs that straddle a recorded phase boundary; 161 625 with a prior same-workload outcome). These are retrospective associations along the selected trajectory, with outcome windows assigned by the collector and settings-change status unknown; they are not verified causal action–response labels. Fourth, as a thermal-management dataset: under the aggressive profile the device is above its GPU-withdrawal threshold for 265–345 min of each mission (73.5–95.9 % of the scheduled 21 600 s, integrating the recorded trace intervals with each sampled state held until the next row), with the device's temperature trajectories, the supervisor's decisions and the resulting throughput all recorded.

Limitations of the resource. The corpus contains no image frames (only per-frame timing and system metrics); no CPU/GPU output-equivalence test (in the reviewed source the Sobel and Gaussian CPU and GPU paths do not compute the same function, so cross-processor timings are recorded implementations, not equal work); no source hash in the run manifests and no deposited build recipe, so the released source snapshot is consistent with, not bound to, the executed binaries, and the controller and static executables differ (two hashes, one configuration digest); no static arm under the moderate profile; no randomization of start temperature (30.0–41.0 °C), ambient conditions or scene content across missions; and only two missions per arm and profile, recorded as chronological repetitions, not matched or randomized replicates. The forecast–outcome pairs are retrospective associations along the selected trajectory with collector-assigned windows. Software read-back is policy state, not physical clock residency; GPU permission is not GPU execution; meter rail attribution and calibration were not independently certified.

Methods

Platform. NVIDIA Jetson Nano Developer Kit (2 GB), L4T 32.7.5 (Linux 4.9.337-tegra), four Cortex-A57 cores with the stock 15-entry operating-point table (102–1 479 MHz) [4] and a 128-core Maxwell GPU. A CSI camera (1 280×720 RG10 at a requested 60 frames/s) is converted on the CPU to 320×240 luma frames; the realized capture rate is a run-level state (40–60 frames/s) and is recorded per frame. Power is measured by an external Lynsyn meter on four rails sampled at the meter's rate and averaged over each host wake (the number of raw samples per wake is recorded); channel 0, which reads ≈5.0 V, is the rail used for the trace power field and the energy table and was placed by the operator at the board's 5-V supply input, but the wiring was not photographed and the attribution and the meter's calibration are not independently certified; channels 1–2 read 1.5–1.6 V at ≤0.01 A and channel 3 reads zero. Temperatures and per-core utilization come from the platform's thermal zones and `tegrastats`.

Controller (six missions). Every 800 ms a generational genetic algorithm (population 8, elite 2, NSGA-II ranking and crowding [6] as selection operators, scalar deployment pick) proposes a genome (m, r, b, g, f, s) — policy mode, target frame rate, power budget, GPU preference, CPU-cap gene from {102, 403.2, 921.6, 1 132.8, 1 479} MHz and GPU work split. Candidates are scored by a workload-indexed surrogate (exact-bucket lookup, k-nearest-neighbour interpolation or a physics prior over the projection (f, g, s, c) with worker count c derived from the mode), weighted by an adaptive weight manager that detects nominal, latency, thermal and power regimes. A scheduler applies the selected genome through `sysfs` and records the software read-back of every written control (the `cpufreq` policy state [5], not physical clock residency); a thermal supervisor withdraws the GPU whenever $\max(T_{\mathrm{CPU}}, T_{\mathrm{GPU}})$ exceeds the profile's WARNING threshold. The supervisor, the search and the surrogate persist across workload changes: at a phase boundary the processing worker is replaced in the running process while the generation counter, the population, the weight manager and the surrogate banks are kept; the recordings show the counter and the observation counts continuing (Technical Validation), which is observed continuity rather than proof that the full population state or any learning benefit was preserved. The three profiles set CAUTION/WARNING/CRITICAL at 75/85/95 °C (inactive — above the device's operating range), 55/62/70 °C (moderate) and 45/50/55 °C (aggressive); `thermal_contracts.csv` lists every threshold.

Static arms (twelve missions). The same launcher runs the same twelve-phase schedule with a fixed policy and no controller: the launcher owns the workload epoch, re-pins the policy after every hot-swap and applies the supervisor's GPU-withdrawal predicate as a static gate. Static arms command no frequency cap. Their manifests declare the same configuration digest and camera configuration as the controller missions, but a different executable; the manifests record no source hash or build flags, so neither build nor effective-configuration equivalence between the two binaries is established beyond the shared digest (the launcher-only difference is reported history).

Workloads and schedule. HistogramEqualization, SobelEdge, HeterogeneousGaussianBlur and MedianFilter, each with a CPU and a GPU implementation, in that order, 1 800 s each, three cycles: twelve phases and eleven hot-swaps per mission. Missions start after an excluded warm-up at generation 0 and $t = 0$. In the reviewed source snapshot the CPU and GPU paths of SobelEdge and HeterogeneousGaussianBlur do not compute the same function (one-dimensional versus 3×3 Sobel; different Gaussian σ); because the snapshot is not bound to the executed binaries this is a property of the inspected code, and cross-processor timings in this corpus should be read as recorded implementations, not equal work.

Recorded streams. Each mission archive holds: `dynamic_adaptation_trace` (2-Hz trace of the harness main thread, 46 fields: phase, controller state, provenance identifiers, the latest finalized

frame's throughput, latencies, power, utilization and temperatures, and the read-back fields added by the audit's correction FIX-E1 [1]: commanded cap, policy read-back, expected cap, cap-verified flag, live current frequency, online-core mask, `sysfs` write failures); `metrics` (one row per finalized camera frame, 42 fields: capture, algorithm, display and composed latencies, per-frame joules, per-core utilization and frequency, the four meter rails held over the algorithm window); `lynsyn` (one row per meter wake: device and host time stamps, current and voltage of each rail); `workload_transitions` (one row per hot-swap: downtime, first new-workload frame latency, frames drained, generation, epoch, surrogate observations and weight-manager regime before and after); `awm_history` and `surrogate_by_workload` (controller missions; header-only in static missions); `events.jsonl` and `manifest.json` (the PilotCollector 1.8.1 event log — CRC32 per line, SHA-256 seal in the manifest — with run and workload events in every mission and, in controller missions, the per-query `prediction_issue`, `action_select`, `action_apply`, `outcome_available`, `score` and `admission` events). Controller missions add `actions` (one record per control period, 111 fields: the genome, the commanded cap, the published runtime controls, twelve read-back fields, the weight-manager regime and the surrogate source, support and forecast of the selected candidate) and `pareto` (every evaluated candidate of every generation, 72 fields, eight rows per generation): three of the six exports are inside their mission archives and the other three — left outside the per-mission folder when the archives were packed, a packaging omission found by a reviewer — are supplied unchanged in `pareto_supplement.zip` since version 2.2, so that the six-run candidate audit (`analysis/candidate_audit.py`) runs from public files. The data dictionary in the record documents every column.

Timing conventions. Five clocks occur and the dictionary maps them. The manifest's `wall_start_ns` (device epoch, UTC) is the authoritative mission start; the trace's `wall_time_ms` is on the same clock, and every `CLOCK_MONOTONIC` stamp of the controller and collector converts to it through the manifest's `mono_start_ns` (agreement with the trace within 1 ms). `mission_elapsed_s` is zero at the release of generation 0, 0.09–0.21 s after that stamp. The metrics `Timestamp` strings and the time stamp in every `run_id` were written from the device's local clock, which a frame-identifier join against the trace shows to be **+01:00 from UTC in all 18 missions** (median offset 3 599.7–3 599.9 s); the configured zone name is not recorded and the dictionary states the observed numeric offset per mission rather than a zone (a previous version of the documentation named the recording site's −04:00 zone; that was wrong). The meter stream carries the host's epoch stamp and the meter's own clock, and only the host stamp is used for integration because the device clock restarts after re-arming. The metrics and meter streams begin 31.0–31.6 s and 28.2–28.3 s before the mission start (camera and host warm-up); the archive totals quoted below include these rows (12 506 of the 8 077 130 frame records) and the dictionary gives the in-mission filter. The composed latency field (`end_to_end_latency_ms`) is a dequeue-to-completion interval that includes input-queue waiting and display; it is neither the processing interval nor a sensor-to-output latency.

Data Records

The corpus is published on Zenodo as `https://doi.org/10.5281/zenodo.23824450` [2] (version 2.2; 39 files, 563 MB compressed, about 3.6 GB unpacked), under CC BY 4.0 for data and documentation and MIT for the authors' code. All versions share the concept record `https://doi.org/10.5281/zenodo.23350633`. The 18 mission archives, the source snapshot and the mission index are byte-identical in versions 2.0 (26 October 2026, 32 files), 2.1 (30 October 2026, 36 files [3]; added the extracted-layout and bundle manifests and the download verifier, renamed the README, and replaced six same-named files) and 2.2 (this descriptor; corrected documentation and tooling, the three population exports and the candidate audit; the exact inventory difference of each version is listed in the record's `README.md`). Version 1 (`https://doi.org/10.5281/zenodo.23350634`, 13 October 2026) is the reproducibility package of the audit letter [1]; its final-mission archive contains the recordings of the three controller missions of 10–12 October in the letter's packaging — the

same recordings as the first three controller missions here, whose archives are the reference copies — together with earlier diagnostic runs that are not part of this campaign.

Each mission is one archive `<run_id>.zip` whose name encodes arm, profile and start time (Table 1); `MISSIONS.csv` indexes the eighteen archives with arm, profile, repetition, wall-clock window, collector status, event-seal hash, archive size and SHA-256. Four further archives hold the analysis scripts (`analysis.zip`: the fail-fast driver `run_all.sh`, the programs that regenerate the derived tables, the collector's seal verifier, the candidate audit and the producers of this descriptor's tables and figures), the frozen derived tables with the replay logs, the reference comparisons, the seal-verification results and the candidate audit (`derived_tables.zip`), the three population exports (`pareto_supplement.zip`) and the reviewed controller source snapshot (`source.zip`: 90 of the 91 files reviewed, one unused third-party header with a proprietary notice omitted, with a per-file manifest of the complete reviewed snapshot and third-party notices for the GPL-licensed meter library). Four manifests and a verifier cover integrity: `SHA256SUMS.txt` (every member of every mission archive, every archive and every top-level file; ZIP-aware), `SHA256SUMS_EXTRACTED.txt` (the same content as extracted paths, for `sha256sum -c`), `SHA256SUMS_ARCHIVES.txt` and `SHA256SUMS_BUNDLES.txt` (the 18 mission archives and the four bundles as uploaded), and `verify_record.py` (standard library only), which checks a download in the archive layout, in the extracted layout, or both at once — when an archive and its extracted copy are both present, both are checked, so a file corrupted after extraction is reported even though the archive is intact — and prints how many checks were made in each representation; `VERIFICATION_COUNTS.txt` lists the expected counts. `DATA_DICTIONARY.md` defines every column with units, clock domains, sentinels and join keys and contains a worked join; `thermal_contracts.csv` gives the compiled thresholds of the three profiles.

Totals: 161 723 action records, 766 250 trace rows, 8 077 130 finalized frame records (8 064 624 within the mission windows), 212 783 meter rows, 198 hot-swap records, 1 132 499 collector events and 1 293 784 candidate rows in the six population exports. Table 1 gives the per-mission counts; `derived_tables/descriptor/mission_facts.csv` holds them with the producer script and column definitions.

Technical Validation

Validation was applied per mission and per channel; a passing mission is admitted for the channels that passed, and no blanket "valid run" label is used. The programs that perform every check are in `analysis.zip`; the driver runs every producer of the tables and figures of this descriptor and of the validation checks below — the analysis programs, the seal verifier on all 18 missions, the mission-facts producer and the six-run candidate audit — and compares 32 frozen tables with its output; it was executed on 26 and 28 October 2026 (28-table scope of versions 2.0/2.1, the second time from a fresh download) and on 30 October 2026 in the 32-table scope of version 2.2 (`REPLAY_LOG_*.txt`, `REFERENCE_COMPARISON_*.json` in `derived_tables.zip`). The record's `README.md` maps every table and figure to the command that produces it. Tables 2 and 3 and Fig. 2 summarize the outcome.

Schedule and continuity. Every mission contains twelve phases in the expected order at the scheduled 1 800-s boundaries (recorded hot-swaps deviate from $1800k$ by at most 0.10 s; 0.02 s or less in 15 of the 18 missions) and eleven hot-swaps with workload epoch incremented by one; the algorithm's processed epoch matches the active epoch after every swap. In the six controller missions the generation counter is monotonic with no reset and exactly equal before and after all 66 swaps (counted as equality, not as non-decrease), and `action_epoch` equals `generation`+1 in all 161 723 action records (the dictionary's join keys state this); and the surrogate observation count of a workload is 2 216–2 248 at its second visit and 4 422–4 496 at its third, in all six missions: observed continuity of the logged state, not proof that the complete population was preserved byte for byte or that retention conferred a benefit. Hot-swap downtime is 10.9–24.7 ms median and at most 662 ms.

Table 1: Mission inventory. Actions are 800-ms control-period records (controller missions only); trace rows are 2-Hz records; metric rows are finalized camera frames; meter rows are Lynsyn wakes; events are collector event-log lines; MB is the compressed archive size. Run identifiers carry the prefix `pilot_`; Start is the manifest start in UTC, Launch the `run_id` stamp on the device's local clock (+01:00), 33–43 s before the start.

Mission (run_id suffix)	Arm	Profile	Rep.	Start (UTC)	Launch (local)	Actions	Trace rows	Metric rows	Meter rows	Events	MB
THERMAL_AGGRESSIVE_45C_20261010_203819	ERL	Aggressive	1	10-10 19:38	10-10 20:38	26 951	42 461	369 096	11 655	188 680	42.0
THERMAL_AGGRESSIVE_45C_20261016_074913	ERL	Aggressive	2	10-16 06:49	10-16 07:49	26 945	42 494	396 069	11 686	188 638	50.0
THERMAL_INACTIVE_75C_20261012_072555	ERL	Inactive	1	10-12 06:26	10-12 07:25	26 907	42 550	466 897	11 765	188 372	37.7
THERMAL_INACTIVE_75C_20261017_085639	ERL	Inactive	2	10-17 07:57	10-17 08:56	26 972	42 519	435 813	11 749	188 827	44.6
THERMAL_MODERATE_55C_20261011_093117	ERL	Moderate	1	10-11 08:31	10-11 09:31	26 976	42 566	463 557	11 755	188 855	38.6
THERMAL_MODERATE_55C_20261016_150925	ERL	Moderate	2	10-16 14:09	10-16 15:09	26 972	42 530	453 997	11 788	188 827	37.9
CPU_STATIC_THERMAL_AGGRESSIVE_45C_20261019_073140	CPU_STATIC	Aggressive	1	10-19 06:32	10-19 07:31	–	42 514	452 040	11 821	25	22.4
CPU_STATIC_THERMAL_AGGRESSIVE_45C_20261023_074052	CPU_STATIC	Aggressive	2	10-23 06:41	10-23 07:40	–	42 545	403 033	11 840	25	20.6
CPU_STATIC_THERMAL_INACTIVE_75C_20261017_225210	CPU_STATIC	Inactive	1	10-17 21:52	10-17 22:52	–	42 538	428 607	11 776	25	21.3
CPU_STATIC_THERMAL_INACTIVE_75C_20261022_155716	CPU_STATIC	Inactive	2	10-22 14:57	10-22 15:57	–	42 539	428 916	11 797	25	21.6
GPU_STATIC_THERMAL_AGGRESSIVE_45C_20261020_075035	GPU_STATIC	Aggressive	1	10-20 06:51	10-20 07:50	–	42 601	446 425	11 794	25	21.8
GPU_STATIC_THERMAL_AGGRESSIVE_45C_20261022_074229	GPU_STATIC	Aggressive	2	10-22 06:43	10-22 07:42	–	42 642	465 245	11 981	25	22.9
GPU_STATIC_THERMAL_INACTIVE_75C_20261019_161507	GPU_STATIC	Inactive	1	10-19 15:15	10-19 16:15	–	42 648	513 850	11 903	25	26.1
GPU_STATIC_THERMAL_INACTIVE_75C_20261021_163625	GPU_STATIC	Inactive	2	10-21 15:36	10-21 16:36	–	42 636	512 843	11 910	25	26.0
BALANCED_THERMAL_AGGRESSIVE_45C_20261018_091202	BALANCED	Aggressive	1	10-18 08:12	10-18 09:12	–	42 609	422 997	11 855	25	21.3
BALANCED_THERMAL_AGGRESSIVE_45C_20261021_073557	BALANCED	Aggressive	2	10-21 06:36	10-21 07:35	–	42 607	412 225	11 935	25	20.9
BALANCED_THERMAL_INACTIVE_75C_20261018_153703	BALANCED	Inactive	1	10-18 14:37	10-18 15:37	–	42 636	505 006	11 875	25	25.9
BALANCED_THERMAL_INACTIVE_75C_20261020_212425	BALANCED	Inactive	2	10-20 20:24	10-20 21:24	–	42 615	500 514	11 898	25	25.8

C1 — request versus read-back (controller missions). The commanded CPU cap equals the policy read-back in all 161 723 periods (C1a). Separately, the reported current frequency exceeds the read-back cap in 3 701 periods (2.29 %; C1b), all at the two caps that the scheduler commands outside the operating-point table (460 800 and 1 035 299 kHz), where the reported value is the next table entry (518 400 and 1 036 800 kHz); this is consistent with quantisation to the table, but the driver path that produced it was not independently established. Read-back is software policy state, not physical clock residency. Static missions command no cap: their cap fields are zero and −1, all four cores are online in every row and no `sysfs` write failure is recorded.

C2 — latency composition (all missions). On every finalized frame the composed latency is at least the processing latency and at least the display latency, and the residual is at least −1 ms; frame identifiers are unique and monotonic; the two measurements of the algorithm interval (steady and system clocks) agree to a median of ≤ 0.0002 ms, with isolated outliers of up to 7.9 ms in fewer than 0.002 % of rows. Zero violations in 8 077 130 rows. Finalized records are 52.6–59.1 % of the observed camera frame-identifier span in the controller missions (retention = finalized records / frame-id span), because the pipeline drops the stale frames waiting in its input queue by design; the metrics stream is therefore not a lossless record of every captured frame, and a throughput computed from it is the processed-frame rate (biased towards the frames the pipeline kept, i.e. the freshest), not the capture rate, which `CameraFPS` carries separately. The per-frame checks were run on all archive rows, including the 643–813 pre-mission warm-up rows per mission.

C3 — GPU preference versus permission (all missions). In the controller missions the supervisor denied 0 (inactive), 1 and 40 (moderate) and 8 999 and 7 506 (aggressive) GPU-preferring periods; every denial occurred with $\max(T_{\mathrm{CPU}}, T_{\mathrm{GPU}})$ above the profile's WARNING threshold and no preferring period above it was granted (zero predicate disagreements). In the static missions the gate never enabled the GPU above WARNING; for the two arms that request the GPU (GPU_STATIC, BALANCED) it never withheld it below WARNING, and CPU_STATIC, which does not request the GPU, never enabled it at any temperature. Permission is a scheduler state; aggregate GPU utilization is a sampled quantity that does not identify which frame executed on the GPU; per-frame CUDA kernel time is the only per-frame execution evidence, and all three are recorded separately.

C4 — meter and thermal plausibility. No power or temperature value is missing or non-positive in any trace. The meter stream covers every manifest window on the host clock; host gaps longer than 3 s occur in eleven missions (at most 8.198 s) and are interpolated in the mission-energy table, the supported-interval sensitivity retaining at least 99.829 % of mission time and changing mean power by at most 0.0015 W; device-clock steps are restarts of the meter clock, not host outages (the largest, 11 141 s, in CPU_STATIC inactive repetition 2; 10 721 s in ERL moderate repetition 1; eleven missions have steps above 100 s). Start temperatures span 30.0–41.0 °C and were not controlled; the eight aggressive missions spend 264.7–345.4 min (73.5–95.9 % of the scheduled 21 600 s) above their 50 °C WARNING threshold, integrating the recorded trace intervals with each sampled state held until the next row and clipping to the mission window — an estimate between observations, not exact crossing times; the share of trace rows above the threshold is the same to one decimal. Start temperature and exposure are tabulated per mission in `derived_tables/descriptor/t0_exposure.csv`; within the aggressive missions the two are not monotonically related (the two shortest exposures, 264.7 and 281.2 min, started at 37.5 and 30.0 °C).

Integrity. The collector's verifier (`analysis/verify_run.py`, PilotCollector 1.8.1) reports two facts per mission, kept apart in `derived_tables/seal_verification/seal_verification.csv`: the seal (every line's CRC32, the sequence contiguity and the SHA-256 seal against `manifest.json`) is intact in **18/18** missions with zero dropped events, and the verifier's overall result, which also requires monotone append-order time stamps, per-query stage order and matched sham/reset

Table 2: Per-mission validation, part A: schedule, controller continuity, cap read-back, GPU permission, latency invariants and event seal. Swaps: recorded hot-swaps with workload epoch +1 (of 11); Gen. kept: swaps with the generation counter preserved (controller missions); C1a: periods with read-back equal to the commanded cap (static arms command no cap); C1b: periods with reported current frequency above the cap; GPU denied/pref.: supervisor denials over GPU-preferring periods ("gate": static gate, no preference record); C3 viol.: periods contradicting the WARNING predicate; C2 viol.: finalized frames violating the latency invariants; Seal / verifier: event seal (CRC32, sequence, SHA-256) and the verifier's overall result.

Mission	Swaps	Gen. kept	C1a (%)	C1b	GPU denied/pref.	C3 viol.	C2 viol.	Seal / verifier
ERL A1	11/11	11/11	100	2621	8999/11168	0	0	intact / FAIL (time went backwards at seq 62801)
ERL A2	11/11	11/11	100	203	7506/9941	0	0	intact / PASS
ERL I1	11/11	11/11	100	222	0/12077	0	0	intact / PASS
ERL I2	11/11	11/11	100	265	0/5790	0	0	intact / PASS
ERL M1	11/11	11/11	100	246	1/22419	0	0	intact / PASS
ERL M2	11/11	11/11	100	144	40/18217	0	0	intact / PASS
CPU_STATIC A1	11/11	n/a	n/a (no cap)	n/a	gate	0	0	intact / PASS
CPU_STATIC A2	11/11	n/a	n/a (no cap)	n/a	gate	0	0	intact / PASS
CPU_STATIC I1	11/11	n/a	n/a (no cap)	n/a	gate	0	0	intact / PASS
CPU_STATIC I2	11/11	n/a	n/a (no cap)	n/a	gate	0	0	intact / PASS
GPU_STATIC A1	11/11	n/a	n/a (no cap)	n/a	gate	0	0	intact / PASS
GPU_STATIC A2	11/11	n/a	n/a (no cap)	n/a	gate	0	0	intact / PASS
GPU_STATIC I1	11/11	n/a	n/a (no cap)	n/a	gate	0	0	intact / PASS
GPU_STATIC I2	11/11	n/a	n/a (no cap)	n/a	gate	0	0	intact / PASS
BALANCED A1	11/11	n/a	n/a (no cap)	n/a	gate	0	0	intact / PASS
BALANCED A2	11/11	n/a	n/a (no cap)	n/a	gate	0	0	intact / PASS
BALANCED I1	11/11	n/a	n/a (no cap)	n/a	gate	0	0	intact / PASS
BALANCED I2	11/11	n/a	n/a (no cap)	n/a	gate	0	0	intact / PASS

events, is PASS in **17/18**: aggressive repetition 1 fails it on one 0.270-ms append-order timestamp inversion between a `workload_exit` and the next query's `prediction_issue` (seq 62 801), which leaves within-query order intact. The verifier also counts, for every joined query (161 717), that the terminal `admission` event restates the outcome window with `window_settings_changed` $= 0$ where the `outcome_available` event says 2 (unknown): a collector default on the admission path, not a determination; the outcome event's values govern reuse. `SHA256SUMS.txt` (343 entries) was written after re-verifying the pinned size and hash of each archive; on the complete 2.2 layout `verify_record.py` makes 334 checks inside the archives and 325 on extracted files, and `sha256sum -c SHA256SUMS_EXTRACTED.txt` checks 325 files; all passed on the files uploaded as version 2.2. Integrity of the bytes is distinct from scientific validity of a channel, which the checks above address.

Usage Notes

Reading a mission. Unzip the archive; join streams on `workload_epoch` and, for controller missions, on `action_epoch` (equal to the collector's `query_id` and to `generation`+1 of the actions, population and weight-manager streams) or on the camera frame identifier (`frame_id` in actions and trace, `FrameID` in metrics). Phase k runs from $1800k$ to $1800(k+1)$ s of `mission_elapsed_s`; the visit number of a phase is $v = \lfloor k/4 \rfloor + 1$. To place a metrics row on the epoch time line subtract 3 600 s from its `Timestamp`; the arm of a mission can be enforced on the manifest's executable hash (`d7235178...` controller, `9d05f40d...` static). The data dictionary states the clock domain of every time-stamp column, the units, the sentinels of the static arms and the join keys with their expected cardinalities, and contains a worked join that reproduces the boundary-clean forecast pairing: of 161 717 scored pairs, the 66 whose outcome falls at or after the recorded workload exit and the 2 whose prediction precedes the recorded entry are excluded. Any machine-learning reuse of the pairs should carry the collector's `window_basis = assigned` and `window_settings_changed = 2` (unknown) flags.

Estimands that must not be conflated. Throughput is the sampled reported rate, not a count of equivalent delivered outputs; processing latency is the algorithm interval; the composed latency field includes queueing and differs between arms; sampled power (trace) and mission energy (host-time integral of channel 0 clipped to the manifest window) are different quantities; GPU

Table 3: Per-mission validation, part B: thermal exposure, GPU availability, power and meter coverage. T_0: $\max(T_{CPU}, T_{GPU})$ at mission start; T^{max}: maxima over the mission (°C); min>WARN and % mission: left-hold integral of the trace intervals with $\max(T_{CPU}, T_{GPU})$ above the profile's WARNING threshold (85, 62 and 50 °C), in minutes and as a share of the 21 600-s mission; GPU en.: share of trace rows with the GPU enabled; Power: median of the trace power channel (the meter-integrated mission mean reported in the evaluation manuscript is 0.00–0.27 W lower, because the trace reports the latest finalized frame's window and integration includes idle intervals); Energy: host-time channel-0 integral over the manifest window; Meter: supported host-time coverage; Gaps: host gaps > 3 s (interpolated).

Mission	T_0	T^{max}_{CPU}	T^{max}_{GPU}	min>WARN	% mission	GPU en. (%)	Power (W)	Energy (kJ)	Meter (%)	Gaps
ERL A1	40.5	56.5	54.5	329.8	91.6	8.1	5.40	115.7	99.86	6
ERL A2	37.5	56.0	54.0	264.7	73.5	9.1	5.62	120.3	100.00	0
ERL I1	38.0	60.5	57.5	0.0	0.0	44.9	6.38	133.8	100.00	0
ERL I2	39.0	63.0	60.5	0.0	0.0	21.5	6.50	136.2	100.00	0
ERL M1	37.0	62.5	59.0	0.0	0.0	83.1	6.47	133.9	99.83	5
ERL M2	39.5	63.0	60.0	0.5	0.1	67.4	6.21	131.1	100.00	0
CPU_STATIC A1	35.0	58.0	56.0	327.3	90.9	0.0	6.11	131.4	99.90	3
CPU_STATIC A2	32.0	56.0	54.0	345.1	95.9	0.0	6.44	138.5	99.83	5
CPU_STATIC I1	41.0	64.0	61.5	0.0	0.0	0.0	6.57	141.3	99.93	2
CPU_STATIC I2	32.0	57.5	55.5	0.0	0.0	0.0	6.42	137.8	99.90	3
GPU_STATIC A1	33.0	57.0	54.5	342.7	95.2	4.8	6.49	138.8	99.93	2
GPU_STATIC A2	30.0	54.5	52.5	281.2	78.1	21.9	6.17	132.0	99.93	2
GPU_STATIC I1	33.5	61.0	56.5	0.0	0.0	100.0	6.51	140.6	100.00	0
GPU_STATIC I2	34.5	61.0	57.0	0.0	0.0	100.0	6.50	140.9	100.00	0
BALANCED A1	32.0	61.5	58.5	345.4	95.9	4.1	6.45	138.5	99.83	5
BALANCED A2	32.5	58.0	55.5	343.8	95.5	4.5	6.51	139.3	99.90	3
BALANCED I1	36.5	64.0	60.5	0.0	0.0	100.0	6.59	142.9	99.97	1
BALANCED I2	35.5	60.0	55.5	0.0	0.0	100.0	6.52	140.8	100.00	0

enabled is permission, aggregate GPU utilization is a sampled quantity that does not identify which frame ran on the GPU, and per-frame CUDA kernel time is the only per-frame execution evidence. Comparisons across the CPU and GPU paths of SobelEdge and HeterogeneousGaussianBlur compare different implementations.

Replication structure. Two chronological runs per arm and profile are the only replication; they are neither randomized nor matched blocks, and run-level summaries should be reported with their ranges rather than with frame-level significance tests [7], and frames, periods and phases within a mission are autocorrelated (eleven of the 72 controller phases do not decorrelate within 120 s). Missions were recorded on different days with uncontrolled start temperature and scene content; start temperature and the aggressive-profile GPU window are correlated.

Reproducing the derived tables. Place every downloaded file in one directory, move the 18 mission archives into `missions/`, unzip the four bundles in place, run `verify_record.py`, unzip each mission archive inside `missions/` (one folder per `run_id`), run `verify_record.py --require-extracted` (or `sha256sum -c SHA256SUMS_EXTRACTED.txt`) so that the files the analysis will read are the ones checked, and run `bash analysis/run_all.sh` (Python 3.10+, numpy, pandas, matplotlib; about 35 minutes); the driver checks the eighteen mission identities, runs every program into a separate output directory without writing into the mission folders, compares its output with the 32 frozen tables of the expected inventory and prints `REPLAY_COMPLETE` only if all are present and match. Figures are regenerated by the two figure scripts named in the README's command map.

Code availability

All analysis code is in `analysis.zip` of the record (MIT): the validity gate and per-phase statistics (`final6_analysis.py`, `replicate_check.py`), the per-check static-mission reporter (`static_arm_check.py`), the matched comparison (`matched_compare.py`), the boundary-clean forecast evalu-

Figure 2: Per-mission thermal exposure, GPU availability and meter coverage (colour by arm as in Fig. 1). Open and filled circles: temperature at mission start and maximum CPU temperature; dotted lines: the aggressive (50 °C) and moderate (62 °C) WARNING thresholds; the inactive threshold (85 °C) is never reached. GPU-enabled share and minutes above WARNING (left-hold integral) show the supervisor's effect; meter coverage is the share of mission time within host intervals of at most 3 s, with the number of longer gaps in parentheses.

ation (`forecast_baselines.py`, `first75_recurrence.py`), the manifest-clipped mission energy (`mission_energy.py`), the cap-consistency sensitivity (`cap_consistent_sensitivity.py`), the collector's seal verifier and its table (`verify_run.py`, `seal_table.py`), the six-run candidate audit (`candidate_audit.py`), the producers of this descriptor's tables and figures (`descriptor/`), the checksum generator and the download verifier. The controller and benchmark source tree as reviewed is in `source.zip` (authors' files MIT; Lynsyn host library GPL-3.0-or-later); the build recipe of the two campaign executables (compiler, flags, dependency versions) is not deposited because it was not captured at build time, and the run manifests record no source hash, so the snapshot is consistent with, not bound to, the executed binaries; a rebuild from the snapshot would be a new executable, not a reconstruction of the recorded ones. The collector library (PilotCollector 1.8.1) is in version 1 of the record [1].

Data availability

The complete corpus described here is openly available on Zenodo at `https://doi.org/10.5281/zenodo.23824450` (version 2.1; data and documentation CC BY 4.0, authors' code MIT, third-party files under their own notices); versions 2.1 (`https://doi.org/10.5281/zenodo.23820264`) and 2.0 (`https://doi.org/10.5281/zenodo.23676764`) hold the identical mission archives with earlier documentation. No access restrictions apply.

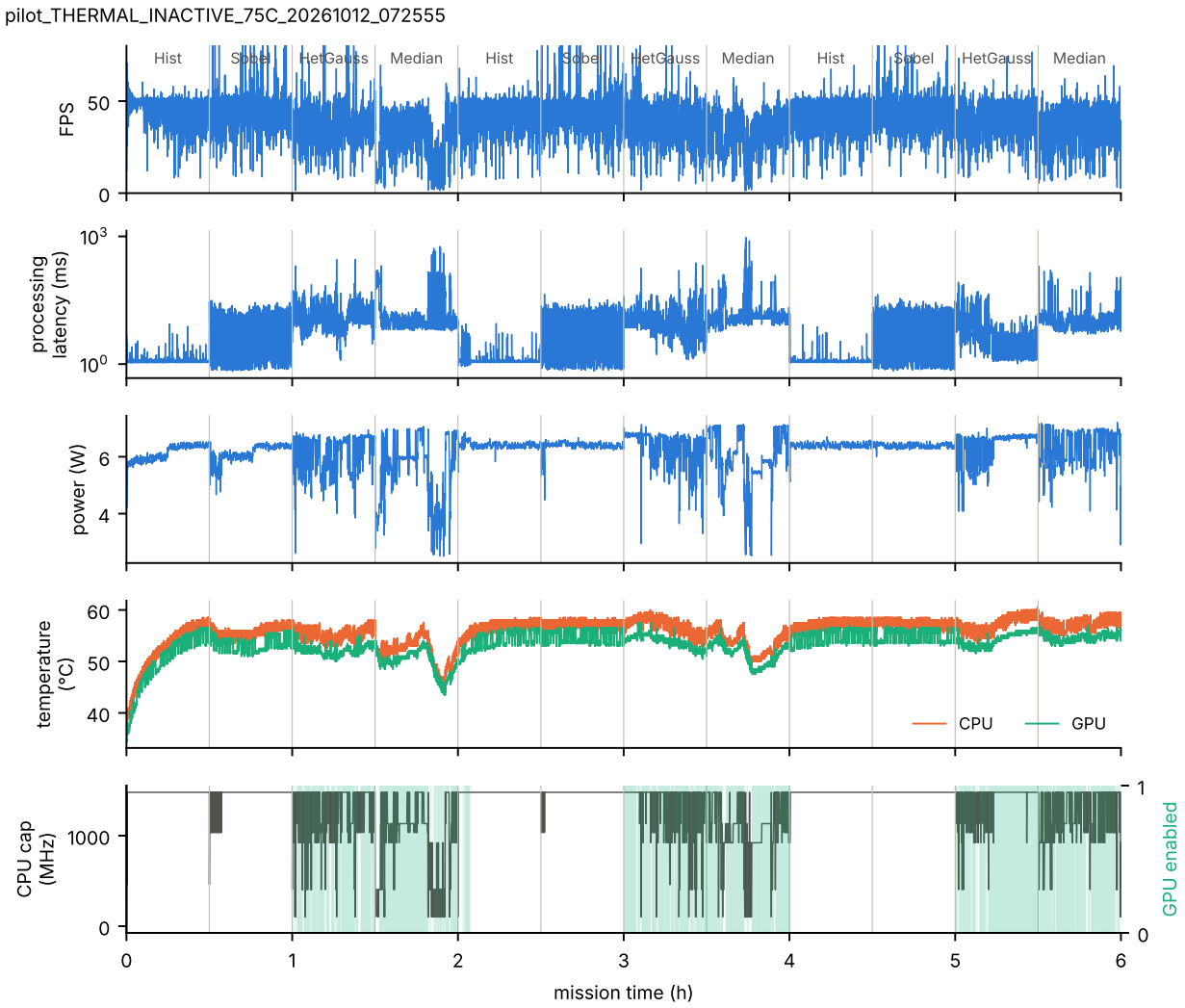


Figure 3: One controller mission (inactive profile, repetition 1) read from its 2-Hz trace: throughput (clipped at 80 frames/s; transient post-swap samples exceed it), processing latency, power, temperatures, commanded CPU cap and GPU-enabled state over the twelve phases, with phase boundaries every 1 800 s. Produced by `analysis/descriptor/descriptor_fig2_timeline.py` from the record.

Funding

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES), PROEX, Finance Code 001, and in part by the Fundação de Amparo à Pesquisa do Estado do Amazonas (FAPEAM) through the POSGRAD project 2024/2025.

Acknowledgements

The authors thank the reviewers of the companion manuscripts whose release checks shaped the verification tooling of the record. Large-language-model assistants were used to help edit the prose of this descriptor and to draft the analysis and verification scripts of the record; the authors reviewed and executed every script, and take full responsibility for the content.

Author contributions

A.S.R.: conceptualization, software, investigation, data curation, formal analysis, visualization, writing – original draft. J.R.H.C.: supervision, methodology, validation, resources, writing – review and editing.

Competing interests

The authors declare no competing interests.

References

[1] Souto Rodriguez, A. & Hughes Carvalho, J. R. Request Is Not Readback: reproducibility package of the audit of a runtime controller's self-reported telemetry on a Jetson Nano (version 1.0). *Zenodo* https://doi.org/10.5281/zenodo.23350634 (2026).

[2] Souto Rodriguez, A. & Hughes Carvalho, J. R. ERL October 2026 campaign — full corpus: six dynamic evolutionary-controller missions and twelve schedule-matched static-reference missions on a Jetson Nano (version 2.2). *Zenodo* https://doi.org/10.5281/zenodo.23824450 (2026).

[3] Souto Rodriguez, A. & Hughes Carvalho, J. R. ERL October 2026 campaign — full corpus (version 2.1). *Zenodo* https://doi.org/10.5281/zenodo.23820264 (2026).

[4] NVIDIA Corporation. Power Management for Jetson Nano and Jetson TX1 Devices. *NVIDIA Jetson Linux Driver Package Developer Guide*, release 32.7.5 (archived). https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-3275/

[5] Wysocki, R. J. CPU Performance Scaling. *The Linux Kernel Documentation*, https://www.kernel.org/doc/html/latest/admin-guide/pm/cpufreq.html.

[6] Deb, K., Pratap, A., Agarwal, S. & Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **6**, 182–197 (2002).

[7] Hoefler, T. & Belli, R. Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results. In *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis (SC '15)*, Article 73, 1–12 (ACM, 2015). https://doi.org/10.1145/2807591.2807644