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.

Remote UAV: A capability-based architecture with deterministic, evidence-grounded independent verification for PX4 SITL/Gazebo mission simulation

Than Le

ldthan@gmail.com

Thu Dau Mot University https://orcid.org/0000-0002-7551-0756

Tam Vu

Wisdom Research

Research Article

Keywords: Unmanned aerial vehicles, software-in-the- loop simulation, PX4, Gazebo, MAVSDK, runtime verification, evidence-based testing, mission orchestration, deterministic verification, Wilson confidence interval, flight-readiness, LOCAL_NED

Posted Date: November 3rd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11576111/v1

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

Additional Declarations: The authors declare no competing interests.

Remote UAV: A Capability-Based Architecture with Deterministic, Evidence-Grounded Independent Verification for PX4 SITL/Gazebo Mission Simulation

Van-Huy Pham[1], Tam T. Vu[2], Than Le[3], and *Trung Van Nguyen[4]

***Abstract*—Software-in-the-loop (SITL) simulation is widely used to validate unmanned aerial vehicle (UAV) control software before flight on real hardware, but a control module reporting "completed" is not, by itself, quantitative evidence that a flight objective was met. We present *Remote UAV*, a simulation framework built on PX4 SITL, Gazebo, and MAVSDK that enforces a strict separation between *capability execution*, *evidence collection*, and *deterministic, independent verification*. The system exposes four reusable capabilities – TAKEOFF-001, NAV-001, RETURN-HOME-001, and LAND-001 – composed into two interaction modes, continuous Single Flight and multi-waypoint Mission Builder, and verified by a Python verifier that recomputes pass/fail from frozen telemetry rather than trusting module self-reports. We formalize this architecture – capabilities as precondition–execution–postcondition tuples, verification as a pure function over frozen telemetry traces, mission-level correctness as a conjunction of child verdicts with explicit continuity, evidence-integrity, and terminal-state predicates, a four-class taxonomy partitioning module/verifier outcome pairs, and a Bernoulli retry model of simulator flight-readiness – and validate it on two separate evidence layers. On a development-time functional baseline (V7.0 Deployment Edition/Package 1.0.0), a 190-case regression suite passes $190/190$, a 50-flight NAV-001 soak test passes $50/50$, and an illustrative eight-step mission passes with maximum inter-step continuity $0.973\,$m and terminal-home distance $0.771\,$m. On a separately frozen evaluation baseline (v1.6.1), isolated from development to prevent post-hoc patching, the full regression suite passes $3028/3028$ in $152.19\,$s; a comparative campaign crosses capability self-report against the independent verifier over 30 requested mission cycles, of which 25 reach PX4/Gazebo flight-readiness and execute, yielding $100/100$ AGREE-PASS capability outcomes (Wilson 95% CI 86.7–100% per capability) with zero SILENT-FAIL or V-ONLY-LENIENT classifications; a reliability/fault-injection smoke test confirms expected verdicts on normal flights and correctly rejects two injected-fault conditions $(4/4)$; and a longitudinal resource smoke test over 26 consecutive flights shows file-descriptor count constant at 6, zero residual MAVSDK processes, and a position-error distribution of mean $0.094\,$m (SD $0.036\,$m, $p_{95} = 0.133\,$m). Separately, flight-readiness itself is $25/30$ (83.3%) within three initialization attempts but only $13/30$ (43.3%) on the first attempt, and a simple independent-retry Bernoulli model calibrated on the first-attempt rate predicts a three-attempt readiness probability of 81.8%, close to the 83.3% observed – evidence that PX4/Gazebo cold-start readiness, not capability logic, is this system's dominant source of end-to-end variability. We report what remains open just as explicitly: the reliability and longitudinal studies above are smoke-scale qualifications of the measurement protocol, not the $n \geq 30$-per-condition and $\geq 4\,$h campaigns the protocol itself specifies as the confirmatory target. These results indicate that the proposed architecture is a viable foundation for simulation-first regression, soak, and comparative evaluation, and for staged extension toward a real-UAV adapter and a supervised AI planning layer that cannot bypass the independent verifier.**

***Index Terms*—Unmanned aerial vehicles, software-in-the-loop simulation, PX4, Gazebo, MAVSDK, runtime verification, evidence-based testing, mission orchestration, deterministic verification, Wilson confidence interval, flight-readiness, LOCAL_NED.**

I. INTRODUCTION

SIMULATION occupies the layer between control-algorithm design and physical flight test in autonomous UAV development. It lets engineers exercise control logic, vehicle state transitions, trajectories, and fault scenarios at lower cost and lower risk than hardware trials. However, a control module returning a "completed" status is not, on its own, quantitative proof that a UAV reached its objective. Position error, airborne/landed state, completion time, telemetry quality, and continuity across successive steps all require evaluation against independently measured data, not against the module's own account of its actions.

This observation motivates *Remote UAV*: not merely a control interface for PX4 SITL, but an experimental framework with an explicit, auditable chain of evidence. Every flight action is required to pass through an auditable pipeline – input normalization, capability precondition checks, PX4/Gazebo execution, telemetry recording within a bounded execution window, evidence freezing, and independent re-verification from the frozen dataset – before the system emits a verdict and updates session or mission state. This organizes the project around regression testing, soak testing, and research on the reliability of autonomous systems, rather than around a one-off flight demo.

Four classes of engineering problems drove the design. First, takeoff, navigation, return-to-home, and landing needed a single reusable abstraction so that multiple workflows (CLI

Manuscript received XXX XX, XXXX; revised XXX XX, XXXX; accepted XXX XX, XXXX. Date of publication XXX XX, XXXX; date of current version XXX XX, XXXX.

[1] is with the Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam.

[2] is with Wisdom Research, Ho Chi Minh City, Vietnam.

[3] is with Thu Dau Mot University, Thu Dau Mot, Vietnam

[4] is with the Faculty of Information Technology, Van Lang School of Technology, Van Lang University, Ho Chi Minh City, Vietnam (Corresponding Author: trung.nv@vlu.edu.vn).

tests, interactive Single Flight, multi-step Mission Builder) do not re-implement the same behavior. Second, execution result and evaluation result needed to be decoupled, so that a module self-reporting "completed" cannot silently stand in for a quantitative pass. Third, state had to persist correctly across multiple flights or mission steps without assuming an incorrect starting position. Fourth, the runtime had to remain stable over long sessions, without accumulating callbacks, threads, file descriptors, or unbounded in-memory data.

This paper reports two distinct layers of evidence and keeps them separate rather than pooling them into one statistic. The first is *functional evidence* accumulated while hardening the system toward the V7.0 Deployment Edition/Package 1.0.0 – Single Flight, Mission Builder, soak testing, and resource cleanup (Section VI). The second is a *frozen evaluation campaign* on baseline v1.6.1, in which the evaluation code, verifier, and protocol were isolated on a dedicated branch and frozen (identified by a SHA-256 content hash, not by version number alone) *before* the comparative, reliability, and longitudinal measurements of Section VIII were run, to prevent selecting favorable runs or patching the implementation after observing a failure. The two layers used different regression baselines ($190/190$ and $3028/3028$ respectively) because the evaluation branch carries its own test suite in addition to the product suite; both numbers are reported, never merged into a single "tests passed" figure.

A. Contributions

1) A formal *capability* abstraction – precondition, execution, verification predicates, timeout – that unifies four reusable flight primitives across three interaction surfaces (CLI, Single Flight, Mission Builder), eliminating duplicated per-surface implementations.
2) An *evidence pipeline* and *deterministic verifier* formalism in which the verdict is a pure function of a frozen telemetry trace and a declarative specification, structurally independent of any value the executing module produces about itself.
3) A *mission-level composition* model that goes beyond conjunction of child verdicts, adding explicit *continuity*, *child-evidence-integrity*, and *terminal-state* predicates that catch failures a step-by-step view misses (e.g., a vehicle that lands successfully but far from home).
4) A *bounded-resource* runtime design – fixed-size live-trajectory ring buffers, cursor-based incremental telemetry transport, SQLite-indexed evidence metadata, and an explicit retention policy – that keeps working-set memory and per-poll I/O independent of elapsed session time, enabling long-duration soak testing.
5) *Quantitative, fully-traceable validation* spanning unit/regression tests ($190/190$ functional, $3028/3028$ on the separately frozen v1.6.1 evaluation baseline), capability-level soak tests ($50/50$ NAV-001 flights after a runtime stability fix; $10/10$ repeated TAKEOFF-001), an end-to-end multi-waypoint mission, and a comparative campaign on the frozen baseline crossing capability self-report against the independent verifier over 100 valid capability executions ($100/100$ AGREE-PASS, Wilson 95% CI 86.7–100%), with every verdict backed by retained per-attempt evidence (including retained FAIL cases).

None of these five contributions is novel in isolation: the test-oracle independence principle behind Contribution 2 is a known concern in software testing [6], and online runtime monitoring of UAV mission execution has been demonstrated before [7]. What Section II argues is novel is the combination – an offline, freeze-then-verify pipeline (rather than an online monitor) applied specifically to produce an independently re-auditable evidence record, extended with mission-level composite predicates that no single step's monitor can express, and packaged as a capability abstraction shared across interactive and batch interaction surfaces – together with the quantitative, fully-traceable validation of Contribution 5, now including an executed (not merely specified) comparative campaign on a frozen baseline. Section VIII is explicit about what is still only smoke-scale (reliability and longitudinal resource measurement) rather than substituting engineering polish for missing evidence.

II. RELATED WORK

A. UAV Software-in-the-Loop Simulation

PX4 and ArduPilot-class software-in-the-loop (SITL) simulators, typically paired with a physics engine such as Gazebo, are the de facto substrate for low-cost UAV algorithm validation before hardware flight. The choice of simulator materially affects what can be validated: Dimmig *et al.* [10] survey 44 aerial-robot simulators and compare 14 of them in depth along fidelity, extensibility, and reproducibility axes, and explicitly note the absence of a single standardized benchmark across frameworks – the same gap this paper flags in Section IX regarding its own evaluation. This paper does not propose a new simulator or physics engine; it takes PX4 SITL and Gazebo as given and asks a different question, namely how to make the *verdicts* produced on top of such a simulator trustworthy and auditable.

RotorS [12] is the closest antecedent for this kind of modularity inside Gazebo: it organizes vehicle dynamics models, sensor models, low-level controllers, state estimators, and evaluation scripts for multirotor MAVs into interchangeable components, so that a researcher can swap one layer (e.g. a controller) without rewriting the others. Remote UAV keeps the same modularity instinct but applies it one layer up the stack: the unit of reuse is not a dynamics or sensor model but a *capability* (Definition 1) shared across interactive and batch surfaces, and the simulator itself – PX4 SITL/Gazebo, accessed through `UAVAdapter` – is treated as a replaceable execution backend behind that abstraction rather than as the object being modularized. Put differently, RotorS modularizes *what flies*; this paper modularizes *how a flight is judged*. The two are complementary rather than competing: a RotorS-style dynamics/sensor stack could sit behind `UAVAdapter` without changing Definitions 1–3 or Proposition 1.

B. The Oracle Problem and Non-Self-Verification

The requirement that a system under test not be allowed to grade its own output is a specific instance of what the software-testing literature calls the *test oracle problem*: given an execution, how does one decide, reliably and independently, whether it succeeded? Barr *et al.* [6] survey oracle construction techniques precisely because an oracle that is not independent of the system under test reintroduces the faults it is meant to catch. Proposition 1 (non-self-verification) is this paper's domain-specific instantiation of that principle for UAV mission execution: the verifier V is constructed to depend only on frozen telemetry and the declarative specification (Φ, τ), never on any status value produced by Exec, which is the structural property the oracle-problem literature argues is necessary (though rarely sufficient on its own) for a trustworthy oracle.

C. Runtime Verification for Robotic and Autonomous Systems

The closest prior work is Zudaire *et al.* [7], who combine temporal task planning with stream runtime verification to monitor, at execution time, whether the explicit and implicit assumptions underlying a UAV's task plan continue to hold, flagging the "silent failure" that occurs when a robot's plan remains nominally on track while an assumption has in fact been violated, and demonstrate the approach on real and simulated UAV flights. This paper shares the same motivating concern – that execution can appear to succeed while quantitatively failing – but differs along three axes: (i) Zudaire *et al.* synthesize *online* monitors from the task plan's own assumption language, whereas the verifier here is an *offline*, two-phase process that first freezes a complete telemetry trace as an immutable, independently re-playable artifact and only then computes a verdict from it (Section III.C), which trades monitor-synthesis sophistication for a simpler, fully re-auditable evidence record; (ii) their evaluation targets assumption-violation detection within a single mission execution, whereas this paper additionally formalizes *mission-level* composite predicates – continuity between steps and terminal-home distance (Section III.E) – that are not statements about any single step's assumptions but about the physical coherence of the step sequence as a whole; and (iii) this paper packages the verified primitives as a reusable *capability* abstraction shared across three different interaction surfaces (CLI, interactive Single Flight, multi-waypoint Mission Builder), rather than a monitor attached to one task-planning pipeline.

More broadly, Luckcuck *et al.* [8] survey formal specification and verification techniques for autonomous robotic systems and highlight verification of the discrete decision-making layer as comparatively mature relative to verification spanning the full decision-to-actuation pipeline – precisely the pipeline this paper's evidence-freezing step is designed to make independently checkable, albeit with a deterministic predicate-based verifier rather than a model-checked specification. In subsequent work on compositional verification, Luckcuck *et al.* [9] propose assume-guarantee contracts for ROS nodes, expressed in a dedicated contract language (RCL) and compiled to runtime monitors by their VANDA tool, demonstrated on an autonomous inspection rover. This is a design-by-contract formalization in the same spirit as the precondition–execution–postcondition capability tuple of Definition 1, but aimed at general ROS node composition via synthesized online monitors, whereas the capability abstraction here is narrower by design – four flight primitives with a fixed predicate structure – in exchange for the simpler, fully offline verifier construction this paper relies on for Proposition 1.

D. Evidence and Provenance for Reproducible Experimentation

Treating every attempt's telemetry, verdict, and intermediate artifacts as an immutable, independently inspectable record connects this paper's evidence pipeline to the broader concern of computational provenance. Dhruv and Dubey [11] argue that software provenance – not just data provenance – must be tracked for computational results to be reproducible, and discuss the practical difficulty of retrofitting provenance capture onto systems that were not designed for it from the start. The evidence schema of Appendix A is this paper's answer to that difficulty for the UAV-simulation setting: provenance (input hashes, canonicalization steps, measured telemetry, verifier output, finalization state) is a first-class, schema-defined output of every attempt rather than an afterthought layered on top of an existing control program.

E. Positioning of This Paper

Relative to this body of work, the contribution of the present paper is not a new physics engine (Section II.A), a new oracle-construction technique in the abstract (Section II.B), or a new monitor-synthesis method (Section II.C), but a concrete architecture and formal specification that (1) applies the oracle-problem principle of non-self-verification to UAV mission execution via an explicitly two-phase freeze-then-verify pipeline rather than an online monitor, (2) extends verification from single-step assumption checking to mission-level composite correctness via continuity and terminal-state predicates, (3) packages the result as a capability abstraction reusable across multiple interaction surfaces, and (4) reports quantitative evidence from an operating implementation, now including an executed comparative campaign on a frozen baseline, rather than a simulation-only proof of concept with no measured agreement/disagreement rate. Sections VI–VII show this architecture validated against unit/regression, soak, and mission-level tests; Section VIII executes the comparative evaluation this positioning claim implies is necessary – quantifying *how much* the independent-verifier design catches that a self-report baseline would miss – and separately reports the reliability and longitudinal resource studies at the smoke scale they were actually run at, rather than at the $n \geq 30/\geq 4\,\mathrm{h}$ scale the measurement protocol specifies as confirmatory.

III. System Model and Formalization

A. Notation

The vehicle state is represented in the local North-East-Down (LOCAL_NED) frame with origin at the session's *Home* position. At continuous time t, the position is

$$p(t) = \big(N(t), E(t), D(t)\big) \in \mathbb{R}^3, \tag{1}$$

where N and E are the horizontal axes and D increases downward. The front end displays altitude using the positive-up convention $h(t) = -D(t)$, which the back end converts back via $D(t) = -h(t)$ before any downstream computation; every internal verification predicate is evaluated in the D-signed frame to avoid a sign-convention bug at the boundary.

Telemetry is sampled at discrete times $t_k = k\,\Delta t$ with sampling rate $f_s = 1/\Delta t$. The deployed recorder uses $f_s = 10\,\text{Hz}$. For an execution window $[t_0, t_f]$, the recorded *telemetry trace* is

$$T = \big\{\, \big(t_k,\, p(t_k),\, v(t_k),\, s(t_k)\big) \;:\; t_0 \le t_k \le t_f \,\big\}, \tag{2}$$

where $v(t_k)$ is measured velocity and $s(t_k)$ is discrete vehicle state (e.g., GROUNDED, AIRBORNE, LANDED). Immediately before verification, the recorder is stopped and one additional *terminal sample* is appended so that the final row of T reflects the state *after* the action completes, rather than depending on the recorder's own polling schedule.

B. Capability Abstraction

Definition 1 (Capability)**.** A capability is a tuple $C = (\text{Pre}, \text{Exec}, \Phi, \tau)$ where $\text{Pre} : X \to \{0, 1\}$ is a precondition over session/runtime state X (e.g., resource gate, current vehicle state), Exec is the control routine that issues setpoints to the UAV adapter, $\Phi = \{\varphi_1, \ldots, \varphi_m\}$ is a finite set of verification predicates over a frozen telemetry trace, and τ is a timeout bound.

Four capability instances are used as reusable building blocks across all interaction surfaces (Table I):

$$\Phi_{\text{TAKEOFF}} = \{\, \text{airborne},\ h \in [h^* - \varepsilon_h,\, h^* + \varepsilon_h], \tag{3}$$

$$\text{drift}_{xy} \le \varepsilon_d,\ \text{transition_valid},\ t_{\text{exec}} \le \tau \,\} \tag{4}$$

$$\Phi_{\text{NAV}} = \{\, e_{\text{pos}} \le \varepsilon_{\text{pos}},\ \text{telemetry_valid},\ t_{\text{exec}} \le \tau \,\} \tag{5}$$

$$\Phi_{\text{RTH}} = \{\, d_{\text{home}} \le \varepsilon_{\text{home}},\ h \approx h_{\text{ret}}, \tag{6}$$

$$\text{airborne},\ \text{transition_valid} \,\} \tag{7}$$

$$\Phi_{\text{LAND}} = \{\, \text{landed},\ h_{\text{ground}} \approx 0,\ \text{drift}_{xy} \le \varepsilon_d, \tag{8}$$

$$\text{transition_valid} \,\} \tag{9}$$

where $e_{\text{pos}} = \|p_{\text{measured}} - p_{\text{target}}\|_2$ is Euclidean position error and $d_{\text{home}} = \|(N, E)_{\text{measured}}\|_2$ is horizontal distance from Home. The deployed configuration uses $\varepsilon_{\text{pos}} = 0.5\,\text{m}$ (`position_tolerance_m`), $\varepsilon_{\text{start}} = 0.75\,\text{m}$ (`start_position_tolerance_m`), and $\tau = 30\,\text{s}$ (`timeout_s`), recorded per attempt alongside `frame = LOCAL_NED`.

C. Deterministic, Independent Verification

Definition 2 (Verifier)**.** Given a frozen trace T and a specification (Φ, τ), the verifier is the pure function

TABLE I
CORE CAPABILITIES AND THEIR ROLE IN THE PIPELINE.

Capability	Role	Typical predicates	Used in
TAKEOFF-001	Climb to target altitude	airborne, altitude, horizontal drift, transition, telemetry, timeout	First step
NAV-001	Fly to target in LOCAL_NED	position error, telemetry, timeout	Every waypoint
RETURN-HOME-001	Return to Home region at return altitude	home distance, altitude, airborne, transition	Before LAND
LAND-001	Land and confirm ground state	landed, ground altitude, drift, transition	Final step

$$V(T, \Phi, \tau) \;=\; \bigwedge_{\varphi \in \Phi} \varphi(T) \;\in\; \{\text{PASS}, \text{FAIL}\}. \tag{10}$$

V is computed by a process with no write access to T and no dependence on any "completed" flag produced by Exec: it is a function solely of (T, Φ, τ).

Proposition 1 (Non-self-verification)**.** Let $r \in \{\text{completed}, \text{error}\}$ be the self-reported outcome of Exec. Then $r = \text{completed} \not\Rightarrow V = \text{PASS}$, and $V = \text{PASS} \not\Rightarrow$ "Exec internally judged itself successful."

Justification. V's domain is (T, Φ, τ), which contains no reference to r; hence r cannot appear in any $\varphi \in \Phi$, and the two propositions are logically independent by construction. The empirical counterpart of this property is observed directly: NAV-001 can report completion while the verifier independently recomputes e_{pos} from telemetry and returns FAIL when $e_{\text{pos}} > \varepsilon_{\text{pos}}$ (Section VI). □

This is the formal counterpart of the system's foundational design principle:

> ***Design Principle.*** *A capability may terminate in a "completed" state and still receive a* FAIL *verdict if telemetry shows the final position exceeds tolerance, the timeout is violated, or the evidence is invalid. The control module never grades its own action.*

D. Session Continuation: RESET vs. CONTINUE

Let $p^{(i)}_{\text{target}}$ be the commanded target of flight i and $p^{(i)}_{\text{actual}}$ its measured terminal position, obtained from *fresh* telemetry rather than cached state. The system anchors the next flight's start as

$$p^{(i+1)}_{\text{start}} \;=\; p^{(i)}_{\text{actual}} \qquad (\text{CONTINUE}), \tag{11}$$

explicitly avoiding the naive anchor $p^{(i+1)}_{\text{start}} := p^{(i)}_{\text{target}}$, which silently assumes zero residual error. We define the measured-start deviation

$$\Delta^{(i)}_{\text{start}} \;=\; \| \, p^{(i)}_{\text{actual}} - p^{(i)}_{\text{target}} \, \|_2, \tag{12}$$

and the backend requests RESET_REQUIRED rather than CONTINUE when $\Delta_{\text{start}}^{(i)}$ (or staleness of the underlying telemetry) makes the starting state insufficiently certain, instead of silently proceeding from an assumption that may no longer hold.

E. Mission Composition

Definition 3 (Mission)**.** A mission is an ordered sequence of capability instances $M = \langle C_1, \ldots, C_n \rangle$ – typically TAKEOFF $\rightarrow$ $\text{WP}_1 \rightarrow \cdots \rightarrow \text{WP}_m \rightarrow$ RTH $\rightarrow$ LAND – where each step produces an attempt and a child verdict $v_i = V(T_i, \Phi_i, \tau_i)$.

Mission-level correctness is *not* simply the conjunction of child verdicts. The deployed mission verifier evaluates

$$V_{\text{mission}}(M) \;=\; \Big(\bigwedge_{i=1}^{n} v_i = \text{PASS} \Big) \wedge \Phi_{\text{seq}} \wedge \Phi_{\text{integ}} \wedge \Phi_{\text{cont}} \wedge \Phi_{\text{term}}, \tag{13}$$

where Φ_{seq} checks step ordering, Φ_{integ} checks completeness of every child evidence artifact, and:

a) Continuity: For consecutive steps, let

$$c_i \;=\; \| \, p_{\text{end}}(T_i) - p_{\text{start}}(T_{i+1}) \, \|_2, \qquad i = 1, \ldots, n-1, \tag{14}$$

$$\Phi_{\text{cont}} \;\equiv\; \max_i c_i \;\leq\; \delta_c. \tag{15}$$

This catches a child-level-invisible failure mode: a mission in which every step individually passes but the orchestration silently resets or mis-anchors between steps, breaking the physical continuity of the flight.

b) Terminal-home distance: With p_{land} the measured landing position and p_{home} the session Home position,

$$d_{\text{term}} \;=\; \| \, p_{\text{land}} - p_{\text{home}} \, \|_2, \qquad \Phi_{\text{term}} \;\equiv\; d_{\text{term}} \leq \delta_{\text{home}}. \tag{16}$$

RETURN-HOME-001 and LAND-001 are deliberately kept as two separate capabilities rather than one combined step, which makes $v_{\text{LAND}} = \text{PASS}$ and Φ_{term} logically independent: a vehicle can land successfully (small h_{ground}, small local drift) while still failing the mission because $d_{\text{term}} > \delta_{\text{home}}$ – i.e., it landed, but not *at Home*.

F. Terminal Settling Condition

An observed failure mode is a capability that stops as soon as its target tolerance envelope is entered while the vehicle retains kinetic energy and continues to drift, so that the terminal telemetry sample read by the verifier can fall outside tolerance even though the control loop judged itself finished. We formalize the fix as an explicit settling predicate evaluated before the terminal sample is taken:

$$\text{settled}(t) \;\equiv\; \forall\, t' \in [t - \Delta t_{\text{settle}},\, t] : \; \|v(t')\|_2 \leq v_{\text{thresh}}, \tag{17}$$

i.e., velocity must remain below v_{thresh} for a dwell window Δt_{settle} immediately preceding the explicit terminal sample. This keeps the execution boundary and the verification boundary consistent without relaxing the verifier's tolerance.

G. Bounded-Resource Runtime Model

Proposition 2 (Bounded working-set memory)**.** Let each active run maintain a live-trajectory ring buffer of fixed capacity N_{rb} and let the system retain metadata for at most $R_{\max}$ recent/active runs. Then the live in-memory working set is bounded by

$$\text{Mem}_{\text{live}} \;\leq\; R_{\max} \cdot \big(N_{\text{rb}} \cdot s_{\text{sample}} + s_{\text{meta}} \big) \;=\; O(1), \tag{18}$$

independent of elapsed session wall-clock time or total flight count.

Justification. By construction, a ring buffer of capacity N_{rb} evicts the oldest sample on every insertion past capacity, so its size never exceeds $N_{\text{rb}} \cdot s_{\text{sample}}$ regardless of run duration; the recent-run cache is explicitly capped at $R_{\max}$ entries by the same argument. The deployed configuration uses $N_{\text{rb}} = 300$ points per run and $R_{\max} = 64$ recent/active runs (Table II), giving a constant bound in contrast to an unbounded append-only live log. □

Complementing Proposition 2, telemetry *transport* to the browser is incremental: given a client-held cursor c, the server returns at most $\min(|\{\text{new samples}\}|, 120)$ points per response, bounding per-poll payload size independent of total trace length, while the *raw* evidence used by the verifier is retained at full 10 Hz fidelity – UI-side bandwidth optimization is kept structurally separate from verification-grade data resolution.

H. Comparative Taxonomy, Readiness Model, and Statistical Reporting

Section VIII reports a campaign that crosses capability self-report against the independent verifier and separately measures simulator initialization behavior. We formalize both constructs here so that Section VIII only needs to instantiate them with measured counts.

Definition 4 (Comparative taxonomy)**.** For an attempt a, let $m(a) \in \{\text{COMPLETED}, \text{FAILED}\}$ be the capability executor's self-reported outcome and $V(a) \in \{\text{PASS}, \text{FAIL}\}$ the independent verifier's verdict (Definition 2). The *taxonomy map* $\kappa : A \rightarrow L$, with codomain $L = \{\text{AGREE-PASS}, \text{AGREE-FAIL}, \text{SILENT-FAIL}, \text{V-ONLY-LENIENT}\}$, sends attempt a to $\kappa(a) = \ell(m(a), V(a))$ via

$$\ell(\text{COMPLETED}, \text{PASS}) = \text{AGREE-PASS} \tag{19}$$

$$\ell(\text{FAILED}, \text{FAIL}) = \text{AGREE-FAIL} \tag{20}$$

$$\ell(\text{COMPLETED}, \text{FAIL}) = \text{SILENT-FAIL} \tag{21}$$

$$\ell(\text{FAILED}, \text{PASS}) = \text{V-ONLY-LENIENT} \tag{22}$$

Because $m(a)$ and $V(a)$ each take exactly one of two values, κ partitions any set of attempts A

into four disjoint classes whose sizes sum to $|A|$; writing $n_\ell = |\{a \in A : \kappa(a) = \ell\}|$ for $\ell \in \{\text{AGREE-PASS}, \text{AGREE-FAIL}, \text{SILENT-FAIL}, \text{V-ONLY-LENIENT}\}$ we define the *silent-failure rate*

$$\rho_{\text{sf}} = \frac{n_{\text{SILENT-FAIL}}}{n_{\text{AGREE-PASS}} + n_{\text{SILENT-FAIL}}} \tag{23}$$

$$= \frac{n_{\text{SILENT-FAIL}}}{|\{a : m(a) = \text{COMPLETED}\}|}, \tag{24}$$

the fraction of self-reported successes the independent verifier overturns – the single number a self-report-only baseline could not, by construction, compute about itself.

Definition 5 (Wilson score interval)**.** For k successes in n independent Bernoulli trials with observed rate $\hat{p} = k/n$ and $z = z_{1-\alpha/2}$ the two-sided standard-normal quantile ($z \approx 1.95996$ for $\alpha = 0.05$), the Wilson score $(1-\alpha)$ confidence interval for the true rate is

$$\text{CI}_{\text{Wilson}}(k, n, z) = \frac{\hat{p} + \frac{z^2}{2n} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}. \tag{25}$$

Unlike the normal (Wald) approximation $\hat{p} \pm z\sqrt{\hat{p}(1-\hat{p})/n}$, the Wilson interval does not collapse to zero width at $\hat{p} \in \{0, 1\}$ and stays within $[0, 1]$ without clipping, which matters directly here because Section VIII reports several rates observed at exactly 100% on n as small as 25–30. Every binomial rate in this paper (capability agreement, flight-readiness, fault-detection) uses Definition 5, not the Wald approximation.

For a continuous measurement x (position error, RSS, latency) over a sample $\{x_1, \ldots, x_n\}$, we report the quintuple $(\bar{x}, s_x, \tilde{x}, x_{p95}, x_{\max})$ with $\bar{x} = \frac{1}{n}\sum_i x_i$ (mean), $s_x = \sqrt{\frac{1}{n-1}\sum_i (x_i - \bar{x})^2}$ (sample standard deviation), $\tilde{x}$ the median, and $x_{p_{95}}$ the 95^{th}-percentile order statistic, since a single mean $\pm$ SD can hide a heavy upper tail that matters for a timeout-bounded system.

Definition 6 (Readiness-within-k model)**.** Model each simulator (re)initialization attempt as an independent Bernoulli trial that reaches PX4/Gazebo flight-readiness with probability θ. Under this idealization, the probability that a cycle reaches readiness within k bounded retries is

$$R(k) = 1 - (1-\theta)^k. \tag{26}$$

Estimating θ by the *first-attempt* readiness rate $\hat{\theta}$ and predicting $R(3)$ gives a falsifiable check of the independence assumption against the *separately measured* within-3-attempts readiness rate (Section VIII.D): if PX4/Gazebo's chance of reaching readiness were systematically higher on a retry than on a cold start (e.g. residual warm state), $R(3)$ would *underestimate* the observed rate; Section VIII.D reports this comparison rather than only the raw counts.

IV. System Architecture

Fig. 1 shows the layered architecture. A user or engineer drives either Single Flight or Mission Builder through a Web UI/HTTP API layer, which holds session or mission state (Home, last verified position, current target, trajectory, progress). The orchestration layer creates an attempt, canonicalizes input, starts the telemetry recorder, invokes capability execution, freezes evidence, calls the verifier, checks the evidence contract, and finalizes the attempt; mission orchestration repeats this per step while additionally maintaining parent-mission evidence for continuity and terminal-state checks.

Capabilities never call MAVSDK directly; they go through a `UAVAdapter` abstraction, currently implemented by `PX4SITLAdapter`. This keeps mission logic independent of the transport layer and is intended to admit a future `PX4RealAdapter` (Section X) without changing capability code. A Simulation Supervisor owns the Docker/PX4/Gazebo lifecycle (the `x500` vehicle model, world, and spawn position), decoupling simulator lifecycle from capability logic and supporting both a desktop mode (Web UI with the Gazebo GUI visible) and a headless mode. Telemetry async-generator ownership is handled explicitly: the iterator is closed immediately after each sample is retrieved, rather than left open, to prevent callback accumulation over long sessions (Section VII).

A. Evidence Pipeline

Fig. 2 shows the ten-stage pipeline executed for every flight action, with "Freeze Evidence" (stage 7) strictly preceding "Independent Verification" (stage 8):

1) The user enters or clicks a target in North, East, Altitude.
2) The backend normalizes the target to LOCAL_NED and checks tolerance/timeout configuration.
3) A resource gate checks storage quota and runtime state before admitting execution.
4) If the vehicle is grounded but the target requires airborne flight, an auto-takeoff bootstrap is triggered.
5) The adapter retrieves fresh telemetry to establish the measured execution start.
6) NAV-001 executes the motion toward the target in PX4 SITL/Gazebo.
7) Telemetry is recorded within the execution window; the Web UI receives a live trajectory feed.
8) The recorder stops, a terminal sample is appended, and evidence is frozen.
9) The verifier recomputes position error, timeout compliance, and telemetry validity, and emits PASS/FAIL.
10) The session updates *Actual* as the anchor for the next flight when CONTINUE is permitted (Section III.D).

Each attempt directory records a bounded, well-typed artifact set: `raw_user_input.json`, `input.json` (canonicalized), `input_validation.json`, `boundary_result.json`, `metadata.json` (scope, scenario, platform, source hash, verdict authority), `events.jsonl`, `telemetry.csv` (measured telemetry within the execution window), `module_output.json`, `verification.json` (verdict and per-predicate checks), `evidence_contract_result.json`,

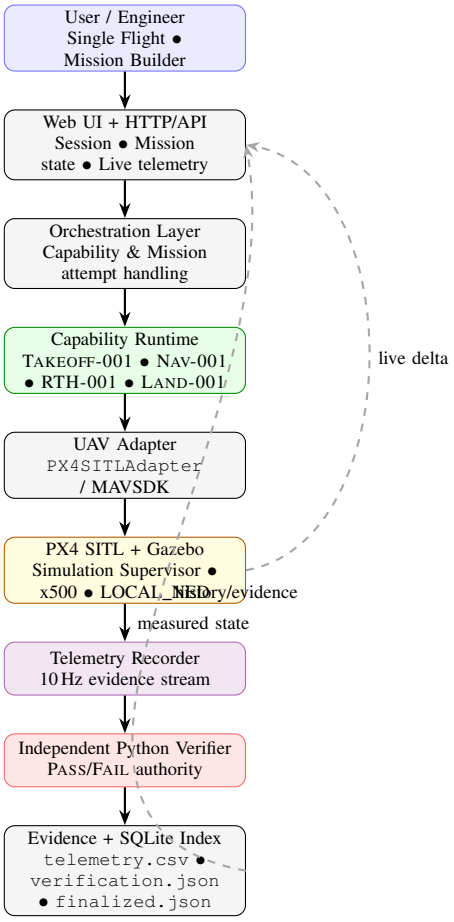


Fig. 1. Remote UAV layered architecture. The control flow runs from the user through orchestration/capability execution down to PX4 SITL/Gazebo; the measurement flow runs back up through telemetry, the independent verifier, evidence storage, and the Web UI.

`runtime_error.json` (if any), and `finalized.json`. Φ_{integ} is precisely the conjunction of existence/well-formedness checks over this artifact set. Metadata is indexed in SQLite so that history views do not require re-parsing every JSON artifact on every request.

V. Experimental Setup

All experiments were executed against the deployed *V7.0 Deployment Edition / Package 1.0.0* of Remote UAV, in SIMULATION_ONLY scope: PX4 SITL and Gazebo under a Dockerized Simulation Supervisor, vehicle model `x500`, coordinate frame LOCAL_NED, telemetry sampling rate $f_s = 10\,\text{Hz}$, position tolerance $\varepsilon_{\text{pos}} = 0.5\,\text{m}$, start-position tolerance $\varepsilon_{\text{start}} = 0.75\,\text{m}$, and per-attempt timeout $\tau = 30\,\text{s}$, as recorded verbatim in each attempt's frozen `metadata.json` (Table II lists the companion resource policy). Four test tiers were used, deliberately separating pure-logic faults from simulator/timing-integration faults:

- **Unit/regression** – schema, state-transition, verifier, evidence-contract, and resource-behavior tests, no simulator required.
- **Capability test** – direct execution against live PX4 SITL/Gazebo.

TABLE II
RESOURCE AND TRANSPORT POLICY, V7.0 BASELINE.

Resource	V7.0 baseline	Purpose
RAM metadata	Up to $R_{\max} = 64$ recent/active runs	Bound in-process state
Live trajectory	Ring buffer, $N_{\text{rb}} = 300$ pts/run	Bound Web-live memory
Telemetry evidence	10 Hz	Preserve verifier fidelity
Telemetry UI batch	≤ 120 new pts/response	Reduce polling payload
Resource polling	≈ 15 s interval	Reduce disk scan/network load
Runs quota	5120 MB; min. free disk 1024 MB	Admission control pre-execution

TABLE III
NAV-001 RUN HISTORY, INTERACTIVE DEMONSTRATION SESSION ($\varepsilon_{\text{pos}} = 0.5$ M, $\tau = 30$ S, FRAME LOCAL_NED).

Run	Target [m]	Actual [m]	e_{pos} [m] / Verdict
1	(11.82, −14.20, 2.00)	(12.06, −14.22, 2.12)	0.250 / PASS
2	(−14.55, −6.08, 2.00)	—	— / FAIL
3	(9.64, −7.24, 2.00)	(9.69, −7.12, 2.05)	0.150 / PASS
4	(12.26, 9.04, 2.00)	(12.28, 9.10, 2.02)	0.070 / PASS
5	(−2.76, −8.86, 2.00)	(−2.63, −8.89, 2.00)	0.096 / PASS
6	(−8.27, −10.13, 2.00)	(−8.32, −10.05, 2.00)	0.105 / PASS

- **Soak test** – repeated flights to observe resident memory (RSS), file descriptors (FDs), and cleanup behavior over time.
- **Mission test** – capabilities chained end-to-end with mission-level verification.

VI. Results

A. *Capability-Level Validation:* NAV-001

Table III reports six consecutive logged NAV-001 attempts from an interactive demonstration session (`scenario_id = INTERACTIVE-DEMO`), transcribed directly from the system's run-history and developer-evidence panel. Five attempts produced a valid terminal telemetry sample and a quantitative position error; one attempt (run 2) is retained as FAIL with no recorded *Actual* position, consistent with the design principle that unsuccessful attempts are kept as evidence rather than discarded. Over the five PASS runs with recorded e_{pos}, the sample mean and standard deviation are

$$\bar{e}_{\text{pos}} = 0.134\,\text{m}, \qquad s_{e_{\text{pos}}} = 0.071\,\text{m} \quad (n = 5),$$

i.e., observed error sits roughly $3.6\times$ inside the configured $0.5\,\text{m}$ tolerance on average, with run 1's $0.250\,\text{m}$ as the largest observed deviation. We report this as a small, fully-enumerated illustrative sample rather than a statistically powered benchmark; it exists to demonstrate that the verifier recomputes error from measured telemetry (so a near-tolerance PASS such as run 1 is visibly closer to the boundary than runs 3–6), not to claim a general error distribution for NAV-001.

B. *Regression, Soak, and Repetition Testing*

Table IV summarizes system-level evidence collected during V7.0 stabilization.

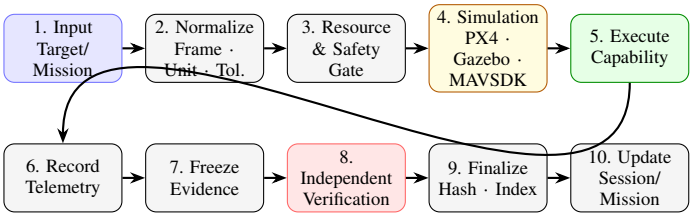


Fig. 2. Ten-stage evidence pipeline executed for every capability attempt. Evidence is frozen (stage 7) strictly before independent verification (stage 8), enforcing Proposition 1.

TABLE IV
SYSTEM-LEVEL TEST EVIDENCE COLLECTED DURING V7.0 STABILIZATION.

Test / evidence	Result	Significance
V7.0 regression suite	190/190 PASS	No regression detected in covered logic
NAV soak, 10 flights	10/10 PASS	Confirms short-horizon continuous flight
NAV soak, 50 flights (post stability fix)	50/50 PASS	No further callback/evidence faults (Sec. VII)
Repeated TAKEOFF, 10 trials	10/10 PASS	Stable repeated takeoff + cleanup
Mission: Takeoff→Nav→RTH→Land	PASS	Confirms two-tier orchestration/verifier
Illustrative mission (logged)	PASS; $c^* = 0.973$ m; $d_{\text{term}} = 0.771$ m	Confirms mission-level checks
Illustrative Single Flight (logged)	PASS; $e_{\text{pos}} = 0.250$ m; 3.44 s	Confirms Actual/Target-based verification
MAVSDK process count post-cleanup	0 residual	Reduced process-leak risk

TABLE V
ILLUSTRATIVE EIGHT-STEP MISSION (HOME AT $(0, 0)$).

Step	Capability / target (N,E,Alt) [m]	Verdict
1	TAKEOFF-001 → Alt 3.0	PASS
2	NAV-001 WP1: (5.0, 0.0, 3.0)	PASS
3	NAV-001 WP2: (15.6, −9.7, 3.0)	PASS
4	NAV-001 WP3: (−6.7, −9.3, 3.0)	PASS
5	NAV-001 WP4: (−10.9, 15.5, 3.0)	PASS
6	NAV-001 WP5: (13.9, 16.4, 3.0)	PASS
7	RETURN-HOME-001 → Alt 3.0	PASS
8	LAND-001	PASS
Mission predicates: seq., child verdicts, integrity, continuity, terminal-land, terminal-home		PASS

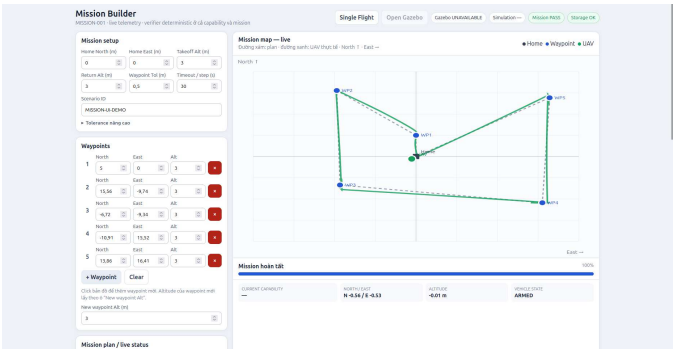


Fig. 3. Illustrative mission reaching MISSION PASS, with planned path and measured trajectory retained on the map and progress at 100%.

C. Mission-Level Validation

Table V transcribes an illustrative eight-step mission – Home at $(0, 0)$, takeoff to 3.0 m, five waypoints, return-home, land – directly from its execution-evidence log. All eight child steps return PASS, and all mission-level predicates (Φ_{seq}, child verdicts, Φ_{integ}, Φ_{cont}, terminal-land-step, terminal-landed-state, Φ_{term}) also return PASS, with

$$c^* = \max_i c_i = 0.973\,\text{m}, \qquad d_{\text{term}} = 0.771\,\text{m}.$$

At the time this mission completed, the resource panel recorded run storage of 551.862 MB against a 5120 MB quota ($\approx 10.8\%$), RAM usage ≈ 62.992 MB, 11 open file descriptors, and 0 residual MAVSDK processes – consistent with the bounded-resource model of Section III and with the system-level evidence of Table IV.

VII. DISCUSSION

A. Telemetry Async-Generator Ownership

During long test sessions the runtime previously logged a growing `''User callback queue slow''` warning with a steadily increasing callback count. Diagnosis traced this to telemetry async generators from which a single sample was read without promptly closing the generator. The fix establishes explicit iterator ownership and calls `aclose()` in a `finally` block after every single-sample read. Post-fix, the number of residual `mavsdk_server` processes after cleanup converges to 0 in recorded tests (Table IV), and file-descriptor count no longer grows linearly with flight count.

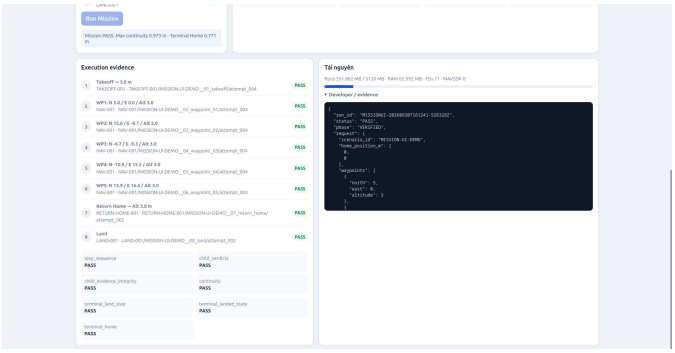


Fig. 4. Execution evidence for the mission of Table V: all eight child steps and all mission-level predicates return PASS.

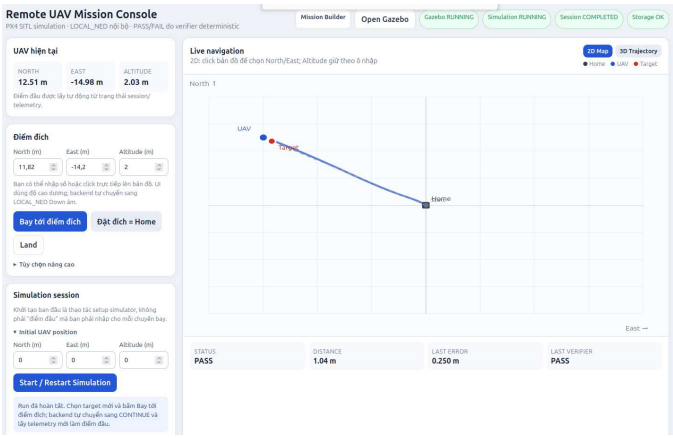


Fig. 5. Single Flight reaching STATUS PASS; LAST VERIFIER PASS; trajectory from Home to target; session state COMPLETED, enabling CONTINUE for the next flight.

B. Start-State Anchoring

If the next flight rigidly reused the previous flight's *target* as its start, residual error at the end of the previous flight ($\Delta^{(i)}_{\text{start}}$, Section III.D) could cause the start gate to fail. The system instead prioritizes fresh vehicle telemetry and records *expected*, *measured*, and *effective* start positions; when measured-start deviation is large, the backend can require RESET rather than continuing on a stale assumption.

C. Terminal Settling

An observed failure mode – a NAV module halting once inside the tolerance envelope while the vehicle's residual momentum carries it further, so the verifier's terminal sample falls outside tolerance – is addressed by the settling predicate of Section III.F: the system waits for $\text{settled}(t)$ in an inner envelope before taking the explicit terminal sample, aligning the execution boundary with the verification boundary without loosening the verifier itself.

D. Why RETURN-HOME and LAND Are Separate

RETURN-HOME-001 and LAND-001 are kept as two capabilities specifically to avoid conflating "near Home" with "landed." The mission verifier checks terminal-home on final state while LAND-001 independently checks landed state, ground altitude, and drift. This separation matters precisely because a UAV can land successfully far from Home: in that case child LAND may PASS while the mission still FAILS at Φ_{term} – a distinction a single combined capability could not expose.

TABLE VI
EXECUTION ENVIRONMENT, FROZEN `v1.6.1` EVALUATION BASELINE.

Component	Recorded configuration
OS	Ubuntu 24.04.4 LTS, x86_64, kernel 7.0.0-38-generic
CPU / RAM	Intel Core i7-12650H, 16 logical CPUs; 15 GiB RAM, no swap
Python / Docker	Python 3.12.3; Docker 29.8.0
Simulator image	px4io/px4-sitl-gazebo:v1.18.0-beta1-660-g61907962c9
Verifier authority	Independent Python verifier over frozen telemetry/evidence

E. Evidence-First Overhead

An evidence-first design increases file count and I/O relative to a control program that only prints results to a terminal. In exchange, every verdict carries data that can be independently re-checked, and a mission can confirm the integrity of every child result. V7.0 addresses the added cost with a SQLite metadata index, retention policy, cached disk scans, and separation of the live UI buffer from raw verifier-grade evidence (Proposition 2) – a deliberate trade-off favoring reproducibility and auditability over minimizing artifact count.

VIII. COMPARATIVE, RELIABILITY, AND LONGITUDINAL EVALUATION ON A FROZEN BASELINE

Sections VI–VII establish that the architecture works and quantify a small illustrative sample of its behavior on the development-time V7.0 baseline. This section reports a second, methodologically stricter layer of evidence: an evaluation campaign run against a separately *frozen* baseline, identified as `v1.6.1` and content-addressed by the SHA-256 hash of its source archive,

```
0a0439bd0ca672b28d0f17812b0b57b2877364c
b8b22c4746f2ac9d70b967342
```

so that no code change after freezing could silently alter a result the campaign had already produced. Table VI records the execution environment, captured verbatim by the evaluation harness itself rather than transcribed by hand. The evaluation code lives on an isolated branch/worktree (`evaluation/v1.6.1-verification`) that must not be merged into the product branch silently, and the campaign's failure- preservation rule explicitly forbids deleting failed samples, re-running only failures and splicing them in, or adjusting thresholds mid-campaign (Section XII details what of this is independently checkable today).

Before the campaign, the full regression suite was run as a pre-campaign gate (not a flight-reliability metric in itself) and passed **3028/3028** tests in 152.19 s – distinct from, and not merged with, the 190/190 V7.0 functional-baseline count of Table IV: the two baselines differ in scope (the evaluation

TABLE VII
EVALUATION-HARNESS STABILIZATION HISTORY, v1.2→v1.6.1.

Rev.	Observation	Protocol change
v1.2	13/13 NAV PASS, but unbounded host `simulator.log` grew ≈ 0.9 GB in 1.8 min (later ≈ 5.6 GB, exhausting disk)	Stop unbounded stdout mirroring; use Docker-managed (bounded) logging
v1.3	Longitudinal run halted before first flight with PX4 `COMMAND_DENIED`; attempt had no measured start and verifier NOT_RUN	Classify as a *pre-execution infrastructure failure*, not a NAV failure
v1.4	17 targeted tests PASS; full regression 3024 PASS; comparative smoke 16/16 AGREE-PASS; longitudinal smoke 26/26 NAV PASS	Add explicit flight-readiness gate, a stable-window check, and owned-simulator cleanup
v1.6.1	7 targeted tests PASS; 14 simulator tests PASS; reliability smoke completes with correct fault-injection semantics	Freeze: requested-n completion per condition; bound the infrastructure-exclusion retry budget

TABLE VIII
DEVELOPMENT-HISTORY CASE EVIDENCE, BY THE STUDY IT MOTIVATES.

Motivates	Observed issue	Relation to independent verification
§VIII.C (verifier vs. self-report)	Settling/drift: NAV halts inside tolerance while residual momentum carries the vehicle further (Section VII.C)	Documented case where Exec's own "done" judgment and the independently measured terminal position could diverge before the settling fix – the target phenomenon of Prop. 1, not an additional instance of it
§VIII.E (reliability across conditions)	Start-state mismatch across chained flights (Section VII.B)	Motivates sampling across *starting-state* conditions specifically, not just targets
§VIII.F (resource over time)	Callback/FD accumulation over long sessions (Section VII.A)	A resource-leak signature only visible by sampling RSS/FD *over time*, not at isolated points

TABLE IX
COMPARATIVE CAMPAIGN, 25 EVALUABLE CYCLES / 100 CAPABILITY EXECUTIONS (EXECUTED, `v1.6.1`).

Capability	N	Agree-P	Agree-F	Silent-F	V-only
TAKEOFF-001	25	25	0	0	0
NAV-001	25	25	0	0	0
RTH-001	25	25	0	0	0
LAND-001	25	25	0	0	0

Wilson 95% CI (Definition 5) for each capability's AGREE-PASS rate: 86.7–100%.

branch carries its own harness tests in addition to the product suite) and in purpose (confirming no known regression at freeze time, versus functional demonstration during development).

A. Harness Stabilization History

Table VII summarizes the versioned engineering record that led to the frozen protocol, included because it is itself evidence about measurement validity, not only about the product: two of the four entries are failure modes of the *evaluation harness*, discovered and fixed *before* any comparative number was trusted. The v1.2 finding is the harness-level analogue of Proposition 1: a workload can be capability-PASS while being unusable as a benchmark because its own measurement apparatus (unbounded host logging) is consuming the resource the benchmark is supposed to characterize. The v1.3 finding is this paper's second confirmation (after Section VII.B) that a simulator or container reporting "running"/"connected" is not evidence of PX4 flight-readiness – motivating Definition 6 and the explicit readiness gate used from v1.4 onward.

B. Retrospective Case Evidence

Table VIII reframes three issues from Section VII by which of the three studies below they motivate, rather than treating them as closed engineering anecdotes. Only the settling/drift case is, strictly, a documented instance of Proposition 1's target failure mode (a capability's internal notion of "done" diverging from the independently measured outcome); the other two are included because they are the concrete, already-observed instances of exactly the reliability and resource failure classes Sections VIII.E–F measure systematically, not because they are additional silent-failure cases.

C. Comparative Evaluation: Protocol and Executed Results

Protocol. Define a *self-report baseline* B that never inspects telemetry: $B(\text{attempt}) = \text{PASS}$ if Exec returns via its normal completion path, FAIL if Exec raises or exits via its error path; B plays the role of $m(a)$ in Definition 4. The executed campaign (`comparative_20261017T113725Z`) requested 30 mission cycles. Each cycle first passes through the flight-readiness gate of Definition 6 (connection, localization, and armability state, re-tried up to three simulator (re)initializations); a cycle that does not reach readiness within three attempts is logged as a *pre-execution infrastructure exclusion* and contributes *no* sample to the four-capability comparison, by design (Section VIII-B and the v1.3 finding of Table VII motivate keeping this exclusion out of capability N rather than counting it as a capability failure).

Results. Of 30 requested cycles, 25 reached readiness and executed the full TAKEOFF→NAV→RTH→LAND sequence; cycles 1, 5, 10, 18, and 30 were pre-execution exclusions, predominantly with `health_global_position_ok=False` and `health_is_armable=False`. The 25 evaluable cycles yield 100 valid capability executions, tabulated in Table IX by κ (Definition 4). Across 100 valid executions, $\kappa(a) =$ AGREE-PASS for all a: $n_{\text{SILENT-FAIL}} = n_{\text{V-ONLY-LENIENT}} = 0$, so the silent-failure rate of Section III-H is $\rho_{\text{sf}} = 0/100$ on this campaign – the module and the independent verifier never disagreed once execution began. This is the quantitative comparative result Sections II.E and VI previously left as a specified-but-unexecuted protocol: it does *not* show that ρ_{sf} is zero in general (a single frozen baseline and 100 executions is a lower-bound existence demonstration, not a long-run rate estimate), but it does show

TABLE X
FLIGHT-READINESS, COMPARATIVE CAMPAIGN (EXECUTED, `v1.6.1`).

Readiness metric	***k*/30**	**Wilson 95% CI**
Readiness within $\leq$ 3 init. attempts	25/30 (83.3%)	66.4–92.7%
Readiness on first attempt	13/30 (43.3%)	27.4–60.8%
Infrastructure exclusion after 3 attempts	5/30 (16.7%)	7.3–33.6%

the comparative instrumentation itself works end-to-end and produces a defensible, Wilson-bounded number rather than a qualitative claim.

D. Flight-Readiness as a Distinct Evaluation Axis

Table X separates infrastructure readiness from capability correctness, exactly the distinction Section VIII-B's v1.3 finding motivated. Using the first-attempt rate as $\hat{\theta} = 13/30 = 0.4333$ in Definition 6, the independent-retry model predicts $R(3) = 1 - (1 - 0.4333)^3 \approx 81.8\%$ – within 1.5 percentage points of the observed $25/30 = 83.3\%$, and inside the observed interval's width. This close agreement is a useful, if informal, check that successive re-initializations behave close to independently (no strong systematic "warm-up" or "degradation" effect across retries within one cycle); it does not establish independence rigorously, since both rates are drawn from the same 30-cycle campaign rather than separately sampled. The practical reading is unchanged from Section VIII-B: the four flight capabilities agreed with the independent verifier on $100/100$ valid executions, while the infrastructure that must reach flight-ready state *before* any capability executes is the system's dominant source of end-to-end variability (56.7% of requested cycles either needed a retry or never reached readiness).

E. Reliability and Fault-Injection: Protocol and Smoke Results

Protocol. Five conditions probe the normal/fault boundary directly motivated by Table VIII: fresh TAKEOFF to a near target, fresh TAKEOFF to a far target, CONTINUE-chained navigation (stressing the start-state anchoring of Section III.D), injected telemetry dropout, and injected timeout. For the two fault conditions, the *expected* verdict is verifier FAIL – the metric is an *expected-behavior rate*, not an ordinary pass rate, since a `module=completed`, `verifier=FAIL` outcome here is success of the detector, not a capability defect.

Results (smoke scale). Table XI reports the harness-qualification smoke run actually executed. All normal-flight samples returned the expected verdict, and both fault-injection conditions were rejected by the verifier in $4/4$ fault samples, confirming the detector does not merely echo the executor. We report this as a **smoke-scale qualification of the protocol's semantics**, not as the $n \geq 30$-per-condition reliability campaign Section VIII-E (as originally specified) and Table VII's v1.6.1 freeze still call for; running that full campaign on the frozen baseline is the most direct remaining gap toward a statistically powered reliability claim, and we state this as a limitation (Section IX) rather than extrapolate from $n = 2$–4.

TABLE XI
RELIABILITY SMOKE / FAULT-INJECTION QUALIFICATION (EXECUTED, SMOKE SCALE, `v1.6.1`).

Condition	n	**Outcome**
Fresh TAKEOFF, near	2	2/2 PASS (expected)
Fresh TAKEOFF, far	2	2/2 PASS; 1 infra. exclusion
CONTINUE-chained	4	4/4 PASS (expected)
Injected telemetry dropout	2	2/2 verifier FAIL (expected)
Injected timeout	2	2/2 module+verifier FAIL (expected)

TABLE XII
LONGITUDINAL RESOURCE SMOKE, 26 CONSECUTIVE FLIGHTS (EXECUTED, SMOKE SCALE, `v1.6.1`).

Quantity	**Start**	**End / summary**
NAV verdict	–	26/26 PASS
RSS	61.500 MB	72.012 MB (+10.512 MB)
File descriptors	6	6 (no growth)
Residual `mavsdk_server`	0	0
Evidence storage	0.510 MB	0.813 MB (+0.303 MB)
Host `simulator.log`	0 MB	0 MB (bounded-log mode)
API latency, p_{95}	–	$\approx$ 13–35 ms

NAV position error over the 26 flights: $\bar{e}_{\text{pos}} = 0.094$ m, $s = 0.036$ m, $\tilde{e}_{\text{pos}} = 0.102$ m, $e_{\text{pos},p_{95}} = 0.133$ m, $\max = 0.138$ m.

F. Longitudinal Resource Evaluation: Protocol and Smoke Results

Protocol. RSS, open file-descriptor count, residual MAVSDK process count, evidence-storage growth, host simulator-log growth, and end-to-end API latency are sampled at fixed checkpoints over soak runs, with an official target of ≥ 4 h (checkpoints at 0, 60, 120, 240 min), consistent with Proposition 2's pass/fail criterion: a plateau within one ring-buffer/recent-run-cache worth of growth indicates bounded behavior; a monotonically increasing slope with no plateau indicates a leak.

Results (smoke scale). A $\approx$ 3 min smoke run over 26 consecutive NAV-001 flights is reported in Table XII. FD count and residual-process count are flat, and RSS growth over the smoke window is small and plausibly warm-up rather than a leak slope – consistent with, but far too short to confirm, Proposition 2. As with reliability, we report this explicitly as **smoke evidence that the measurement path works** (Table VII's v1.4/v1.6.1 bounded-logging fix is what makes host-log growth 0 MB here, versus ≈ 5.6 GB at v1.2), not as the confirmatory ≥ 4 h, four-checkpoint campaign the protocol specifies; we did not extrapolate a 3-minute sample into a multi-hour memory-stability claim.

G. Synthesis

Three findings from this section combine into a methodologically load- bearing claim. First, once execution begins, module and verifier agreed on every one of 100 valid capability

executions ($\rho_{\mathrm{sf}} = 0/100$, Table IX) – the architecture's central claim held on an executed, frozen-baseline campaign, not only on the $n = 5$–6 illustrative sample of Section VI. Second, the dominant source of end-to-end variability was *not* capability correctness but PX4/Gazebo flight-readiness convergence (43.3% first-attempt, Table X), and this is exactly the distinction that motivated treating readiness as a reportable metric (Definition 6) rather than folding it silently into capability pass/fail. Third, the reliability and longitudinal studies remain honestly at smoke scale: they qualify that the instrumentation and detection semantics work, not that the system has been reliability-tested at the statistically powered scale ($n \geq 30$/condition, ≥ 4 h) the protocol itself specifies as confirmatory. We treat closing that specific gap – not re-deriving the protocol – as the paper's most direct remaining empirical task (Section IX).

IX. Limitations

- The system is SIMULATION_ONLY; it has no real-UAV safety gate and has not been validated against flight hardware. Dynamics, sensor noise, network delay, and hardware failure modes are not inferable from these results.
- Both evidence layers were collected on a single development machine with one OS/Docker/PX4-Gazebo image combination (Table VI); this is not a multi-machine or multi-OS benchmark, nor a controlled comparison against an alternative simulation framework such as RotorS [12].
- The comparative campaign (Section VIII-C) has 25 evaluable cycles rather than 30 execution-complete cycles, because five cycles did not reach flight-readiness; the Wilson intervals reported reflect this realized sample size rather than a post-hoc best-case subsample.
- The reliability/fault-injection and longitudinal resource studies (Sections VIII-E–VIII-F) were executed only at smoke scale ($n = 2$–4 per condition; ≈ 3 min duration), which qualifies the measurement protocol but is explicitly *not* the $n \geq 30$-per-condition, ≥ 4 h-duration campaign the protocol specifies as confirmatory. A short smoke window can rule out a gross leak but cannot characterize slow drift over endurance operation.
- PX4/Gazebo flight-readiness showed material attempt-to-attempt variability (Table X); the readiness-within-k model of Definition 6 assumes approximately independent retries, which the observed agreement between predicted and realized rates supports only informally, on a single 30-cycle campaign.
- LOCAL_NED suits a local simulated area but does not yet fully address geodetic coordinates, geofencing, or large-scale outdoor missions.
- Session state targets a single UAV; multi-UAV/fleet state is not yet a core abstraction.
- The Web console is localhost-first and lacks the authentication/authorization required for public-Internet deployment.
- The Gazebo GUI is a desktop observation tool, not a web-streaming interface for remote users.

X. Future Work

A. Supervised AI Planning Layer

The most consistent extension path places an AI agent *above* the deterministic pipeline rather than letting a language model drive actuators directly. An AI project-manager role could decompose requirements across agents for propulsion, control, mission planning, safety, and verification, and emit a structured mission/capability *proposal*. Critically, such a proposal is still required to pass through the existing input validator, capability runtime, and independent verifier – i.e., it enters the pipeline at the same stage as `raw_user_input.json` in Section IV, with no privileged bypass of V or V_{mission}. Simulation evidence then becomes feedback for the next reasoning iteration.

B. Real-UAV Adapter

Moving to real hardware should add a `PX4RealAdapter` rather than overload `PX4SITLAdapter` with a second role. The real-flight path requires an extended safety predicate set Φ_{safety} – arming authorization, geofence compliance, battery/GPS/EKF health, RC-override availability, emergency stop, command-rate limiting, and operator confirmation – gating Exec in addition to the existing $(\mathrm{Pre}, \Phi, \tau)$. Preserving the `UAVAdapter` abstraction keeps capability logic reusable across both adapters.

C. Multi-UAV / Fleet Extension

A fleet extension replaces the single session with an indexed family of vehicle states $\{x_j(t)\}_{j \in J}$, each with its own `vehicle_id`/`system_id`, state, and evidence stream, plus inter-vehicle predicates Φ_{fleet} (e.g., minimum separation) when coordination is required. Running independent copies of the project per vehicle is explicitly rejected as a strategy, since it cannot produce a single source of truth for fleet state.

D. Secure Remote Deployment

Exposing the interface on a public domain requires an HTTPS reverse proxy, authentication, authorization, rate limiting, audit logging, and a safety gateway in front of the control API. The Gazebo GUI should remain on the simulation/server host; the web client should receive only the telemetry and visualization data it needs, rather than exposing X11/Gazebo processes directly to the Internet.

XI. Conclusion

Remote UAV has grown from a single simulated navigation flow into a reasonably complete architecture for simulation, mission orchestration, and deterministic verification. Its central contribution is not PX4/Gazebo or the Web UI in isolation, but the explicit, formalized boundary between intent, execution, measured telemetry, frozen evidence, and independent verdict (Section III). This boundary preserved its auditability when the system grew from Single Flight to Mission Builder and when V7.0 optimized deployment and resource usage.

Logged results show the Web UI and Gazebo observed in parallel; Single Flight supporting continuous flight anchored

on measured *Actual* position; Mission Builder executing multi-waypoint missions with return-home and land; and child evidence together with mission-level checks displayed explicitly and validated quantitatively ($190/190$ regression, $50/50$ soak, mission continuity 0.973 m, terminal-home 0.771 m). Iterative testing and resource-handling improvements resolved concrete issues – callback accumulation, start-state mismatch, settling at target, and storage growth.

On a separately frozen `v1.6.1` baseline, these qualitative findings became an executed, Wilson-bounded quantitative campaign (Section VIII): $3028/3028$ regression tests; $100/100$ AGREE-PASS capability executions across $25/30$ evaluable mission cycles (95% CI 86.7–100% per capability) with zero silent failures; correct rejection of $4/4$ injected faults; and a 26-flight longitudinal smoke test with flat file-descriptor and residual-process counts. The same campaign showed, with equal explicitness, that PX4/Gazebo flight-readiness convergence (43.3% on the first attempt) is this system's dominant source of end-to-end variability – not the four capabilities' internal logic – and that the reliability and longitudinal studies remain at smoke scale relative to the $n \geq 30/\geq 4$ h campaign the measurement protocol itself specifies as confirmatory. The project now has a foundation sufficient to pursue three directions: closing that specific empirical gap with the already-specified full-scale campaign, increasing simulator-readiness determinism toward real-UAV deployment, and building an AI engineering layer atop the deterministic runtime without surrendering the validator/verifier's authority.

XII. Data and Code Availability

Reproducibility was a design goal of the evidence pipeline itself (Section IV.A), not an afterthought: every attempt already produces the schema-defined, self-contained artifact bundle of Appendix A (canonicalized input, measured telemetry, verifier output, finalization record), and every number reported in Sections VI and VIII was read directly from such a bundle, or from the evaluation repository's own committed result/manifest files, rather than computed ad hoc.

Source code. The orchestration layer, capability runtime, adapters, and verifier are public at https://github.com/wisdombrainscience/auv-aigent, branch evaluation/v1.6.1-verification, which is also where the frozen evaluation code, protocol documents, and the compact result summaries underlying Tables IX–XII (docs/evaluation/results/comparative_primary_30cycles.txt and its accompanying `SHA256SUMS.txt`) are committed. This closes the repository-URL gap of the previous revision of this paper without inventing a placeholder link.

Independent reproducibility check. Beyond hosting the code, the evaluation branch was independently reconstructed from the committed project baseline (commit `1f55ba71...`) *without* the uncommitted development workspace that produced the frozen $3028/3028$ regression figure, and re-run: the reconstructed branch produced 2834 passed tests in 101.29 s for the full branch regression, and 21 passed for the targeted evaluation/simulation subset. We report both regression counts side by side rather than merge them, since they cover deliberately different source scopes (the frozen working copy versus the clean reconstructed branch); the fact that an independent reconstruction from committed history runs and passes at all is itself reproducibility evidence, addressing a concrete instance of the software-provenance concern raised by Dhruv and Dubey [11] for this project rather than only in the abstract.

Still open. Two items remain genuinely unavailable at the time of writing, and we flag them rather than insert placeholder links, since a fabricated DOI would be worse than an honest gap: (1) a frozen *evidence bundle* containing the full per-attempt artifact directories (not only the compact summaries already committed) underlying Tables III, V, and IX–XII, published to an archival repository that issues a persistent identifier ; and (2) the exact PX4 SITL/Gazebo simulator *image*, pinned by tag in Table VI (px4io/px4-sitl-gazebo:v1.18.0-beta1-660-g61907962c9) but not yet mirrored to an archival registry independent of its current upstream host, since SITL behavior is sensitive to simulator and physics-engine version [10]. Once published, this section should also state the license under which the code and evidence bundle are released and a versioned identifier for the exact snapshot any post-submission revision's numbers were computed from.

Acknowledgment

The author thanks the Remote UAV project contributors for the PX4 SITL/Gazebo deployment and test evidence underlying this paper.

References

[1] Remote UAV Project, "README, deployment notes, capability/mission specifications, and build validation," V7.0 Deployment Edition / Package 1.0.0, 2026.

[2] PX4 Autopilot, *PX4 SITL and Vehicle Simulation Documentation*. [Online]. Available: https://docs.px4.io/

[3] Gazebo Sim, *Robot Simulation and World/Model Documentation*. [Online]. Available: https://gazebosim.org/

[4] MAVSDK, *API Documentation for MAVLink/PX4 Control and Telemetry*. [Online]. Available: https://mavsdk.mavlink.io/

[5] Docker, *Container Runtime and Application Deployment Documentation*. [Online]. Available: https://docs.docker.com/

[6] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, pp. 507–525, 2015, doi: 10.1109/TSE.2014.2372785.

[7] S. Zudaire, F. Gorostiaga, C. Sánchez, G. Schneider, and S. Uchitel, "Assumption monitoring using runtime verification for UAV temporal task plan executions," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2021, pp. 6824–6830, doi: 10.1109/ICRA48506.2021.9561671.

[8] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, and M. Fisher, "Formal specification and verification of autonomous robotic systems: A survey," *ACM Comput. Surv.*, vol. 52, no. 5, 2019, doi: 10.1145/3342355.

[9] M. Luckcuck, M. Farrell, A. Ferrando, R. C. Cardoso, L. A. Dennis, and M. Fisher, "A compositional approach to verifying modular robotic systems," arXiv:2208.05507, 2022.

[10] C. A. Dimmig, G. Silano, K. McGuire, C. Gabellieri, W. Hönig, J. Moore, and M. Kobilarov, "Survey of simulators for aerial robots: An overview and in-depth systematic comparisons," *IEEE Robot. Autom. Mag.*, 2024, doi: 10.1109/MRA.2024.3433171.

[11] A. Dhruv and A. Dubey, "Managing software provenance to enhance reproducibility in computational research," *Comput. Sci. Eng.*, 2023, doi: 10.1109/MCSE.2023.3314288.

[12] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "RotorS—A modular Gazebo MAV simulator framework," in *Robot Operating System (ROS): The Complete Reference (Volume 1)*, A. Koubaa, Ed. Cham, Switzerland: Springer International Publishing, 2016, ser. Studies in Computational Intelligence, vol. 625, doi: 10.1007/978-3-319-26054-9_23.

APPENDIX A
EVIDENCE ARTIFACT SCHEMA

An attempt may contain the following artifacts, depending on capability and any errors encountered:

- `raw_user_input.json` – raw input from user/API.
- `input.json` – input canonicalized by task/frame/unit.
- `input_validation.json` – schema/bounds validation result.
- `boundary_result.json` – execution boundary/precondition check result.
- `metadata.json` – scope, scenario, platform, source hash, and verdict authority.
- `events.jsonl` – chronological runtime event sequence.
- `telemetry.csv` – measured telemetry within the execution window.
- `module_output.json` – result or error returned by the capability executor.
- `verification.json` – verdict and per-predicate checks computed by the independent verifier.
- `evidence_contract_result.json` – completeness status of the evidence contract.
- `runtime_error.json` – runtime exception, if any.
- `finalized.json` – marks evidence as finalized.

APPENDIX B
STATE AND STATUS DEFINITIONS

TABLE XIII
RUNTIME STATE/STATUS DEFINITIONS.

State	Meaning
Gazebo RUNNING	Gazebo GUI/backend is active and the simulator can be observed.
Simulation RUNNING	A PX4 SITL simulation session is active.
Session READY	Single Flight state is sufficiently trusted to begin execution.
Session COMPLETED	The most recent run finished; CONTINUE is possible if telemetry/session remain valid.
Session RESET_REQUIRED	The backend is no longer certain of the start state; simulation restart is required.
VERIFYING	Execution has ended and the verifier is evaluating frozen telemetry/evidence.
PASS/FAIL	Verdict from the independent capability or mission verifier, never from the UI alone.
Storage OK	The resource manager has not detected a quota / minimum-free-disk violation under current policy.