RESEARCH ARTICLE

RSP Science Hub

International Research Journal on Advanced Science Hub
2582-4376
www.rspsciencehub.com
Vol. 08, Issue 07 July

Check for updates

http://dx.doi.org/10.47392/IRJASH.2026.031

Robust Multi Agent Coordination: Integrating Causal Inference and Evolutionary Strategies in Adversarial Environments

Jayasurya K[1], Mrs. Snigdha Kesh[2], Mrs. V. Mareeswari[3]

[1]PG Student, Dept. of Computer Science, AMC Engineering College, Affiliation by VTU, Bengaluru, India.

[2]Asst. Professor, Dept. of Computer Science, AMC Engineering College, Affiliation by VTU, Bengaluru, India.

[3]Professor, Dept. of Computer Science, AMC Engineering College, Affiliation by VTU, Bengaluru, India.

Emails: *Jayasuryak1011@gmail.com[1], Snigdha.kesh@amceducation.in[2], mareesh.prasanna@gmail.com[3]*

Article history

Received: 01 June 2026
Accepted: 26 June 2026
Published: 24 July 2026

Keywords:

Mobile-Based Platforms, Real-Time Price Information, Logistics Integration, Smart Agriculture Systems.

Abstract

Autonomous multi-agent systems operating in safety-critical settings—bordersurveillance, infrastructure protection, maritime patrol, and contested reconnaissance—must sustain effective coordination even when actively targeted by adversaries. UAV swarms face a particular challenge: GPS spoofing, communication interference, and unexpected environmental disturbances can silently degrade individual agents before the rest of the swarm detects anything is wrong. This paper introduces a robust multi-agent coordination framework that tightly couples causal inference with evolutionary route selection inside a unified six-layer pipeline. A real-time threat detection and trust evaluation subsystem watches agent telemetry continuously, raises anomaly flags, and—critically—separates deliberate adversarial interference from ordinary environmental reactions using directed causal analysis. Agents identified as compromised are automatically quarantined; a hysteresis-based protocol then governs their gradual return to the swarm. Evolutionary path optimization adapts navigation for the remaining healthy agents, keeping the mission on track despite a degraded swarm. The framework runs in Unity 6 and was tested across 40 fully automated simulation runs covering seven distinct test categories. Detection reached 100%, every agent recovered successfully, and every mission completed, with a mean detection latency of 0.89 s, a mean recovery time of 14.56 s, and an evolutionary fitness score of 89.90. An ablation study shows the causal layer alone cuts false-positive isolation events by 85.7% compared to a simple threshold-only baseline.

1. Introduction

1.1. Background and Motivation

Autonomous multi-agent systems have become the go-to approach for distributed tasks that demand wide area coverage, resilience to individual node failure, or round-the-clock monitoring. UAV swarms in particular now see deployment in border surveillance, critical infrastructure inspection, maritime patrol, and autonomous reconnaissance settings where keeping a human operator in the loop at all times is simply not practical. Their core

strength comes from peer-to-peer telemetry exchange and distributed consensus: each agent shares a localized state vector so the group converges on global mission objectives without relying on any central controller.

This decentralized design scales well and handles isolated hardware failures gracefully, but it leaves communication channels and state estimation pipelines exposed to deliberate interference.

1.2. Problem Statement

Adversaries target these exposed channels through two main attack classes. GPS spoofing plants false positional coordinates, silently redirecting an agent's trajectory. Communication attacks corrupt or block inter-agent messages, slowly eroding the consensus the swarm depends on. Standard coordination algorithms—consensus heuristics, flocking rules, artificial potential fields—are all built on the assumption that the information they receive is honest. Once a compromised agent begins broadcasting corrupted telemetry, these methods absorb it without question, triggering Byzantine faults that can fragment the formation, push the group off-target, or cause collisions.

1.3. Research Challenges

Making swarm coordination resilient against these threats raises several tightly coupled problems. First, the system must rapidly tell apart deliberate adversarial interference from a legitimate obstacle-avoidance maneuver, since both look identical at the trajectory level. Second, a quantitative per-agent trust score must fall when an agent misbehaves and recover when it returns to normal operation. Third, a compromised agent must be isolated fast enough to stop the fault from spreading to its neighbors. Fourth, the surviving swarm must replan geometrically sound, near-optimal routes in real time all without losing sight of the original mission.

1.4. Contributions

This paper contributes:

- Threat Detection: three-channel real-time anomaly screening (positional, communication, formation).
- Trust Evaluation: asymmetric per-agent trust dynamics with formal isolation and reintegration thresholds.
- Causal Inference: directed evidence-weighting to distinguish GPS/communication attacks from obstacle avoidance.
- Automated Recovery: hysteresis-based reintegration protocol preventing premature or oscillatory agent reinstatement.
- Evolutionary Routing: trust-aware fitness function for real-time path replanning after isolation events.

(6) Empirical Validation: 40 automated Unity 6 runs achieving 100% detection, recovery, and mission completion

2. Literature Survey

2.1. Multi Agent Coordination

Decentralized coordination gives multi-agent systems scalability, but it struggles the moment communication becomes unreliable. HMAAC [1] addresses this by broadcasting a centralized latent representation that lets individual agents keep executing even when packets drop; TSATM [2] assesses risk at both the single-agent and network levels to head off cascading task failures. Both architectures break down when the underlying telemetry itself is adversarially corrupted a hierarchical design is only as fault-tolerant as its weakest communication link. Recent decentralized submodular coordination methods have improved the scalability of multi-robot decision making under uncertainty while maintaining distributed execution efficiency [3].

2.2. UAV Swarm Coordination

UAV-focused research has concentrated on energy-efficient navigation and flexible formation management. E2Coop [4] builds artificial potential field contours that guide agents around obstacles while keeping energy consumption low; angle-encoded PSO [5] temporarily reshapes formations to squeeze through constrained geometries. Both methods rely on accurate localization. Once a spoofed agent starts broadcasting wrong coordinates, those errors propagate to every peer trying to maintain formation relative to it.

2.3. Adversarial Resilience

Resilience against adversarial threats spans both physical and cyber dimensions. Deep RL pursuit-evasion [6] achieves scalable evasion while avoiding collisions; ROMANCE [7] strengthens policies by co-training them against auxiliary adversarial agents; geocaching-based localization [8] sidesteps GPS entirely by anchoring position to known landmarks. Valuable as each contribution is,

they address different slices of the problem independently—no single framework handles multiple attack vectors simultaneously while keeping the mission running. Distributed real-time UAV control strategies further enhance cooperative adaptation under adversarial environments through decentralized onboard planning [9].

2.4. Evolutionary Strategies

Evolutionary methods are well-suited to the high-dimensional search spaces that arise in multi-UAV planning. Co-evolutionary path planning [10] splits the problem across manageable sub-populations; [11] weaves evolutionary search into graph-based multi-agent coordination; decentralized relay selection [12] keeps network links alive under dynamic threat conditions. In practice, though, these implementations run offline and treat security as a separate concern—route optimization and threat response never talk to each other.

2.5. Trust-Based Systems and Causal Reasoning

On the trust side, ADMAC [13] scores the reliability of individual messages to filter malicious transmissions; memory-based flocking [14] applies multi-timescale behavioral filters to identify agents that refuse to cooperate; formal group trust modeling [15] offers a rigorous temporal logic framework for reasoning about trust relationships among individuals and groups. The critical gap in all these approaches is context: they operate on statistics alone, so an agent that swerves to avoid an obstacle gets penalized exactly the same as one executing an adversarial maneuver. No existing trust framework applies causal diagnostics to resolve that ambiguity—and closing that gap is the central aim of this work.

3. Research Gap

Three interconnected limitations in the current literature motivate the work described here. To begin with, existing solutions are architecturally fragmented: coordination logic, trust evaluation, and evolutionary path planning have all been developed in isolation, making it impossible to mount a unified real-time response to an adversarial event. On top of that, trust-based frameworks reason from statistical observations alone; they cannot tell whether a trajectory deviation was caused by an attack or by a perfectly reasonable obstacle avoidance reaction, so they generate false-positive isolations that shrink the active swarm unnecessarily. Finally, most adversarial resilience methods stop at detection or policy hardening—they offer little in the way of automated recovery. Identifying a compromised node is a necessary first step, but the mission only survives if the system can also isolate that node, restore coordination among the remaining agents, and continue executing the objective. This paper addresses all three limitations within a single integrated architecture.

4. Proposed Framework

4.1. System Overview

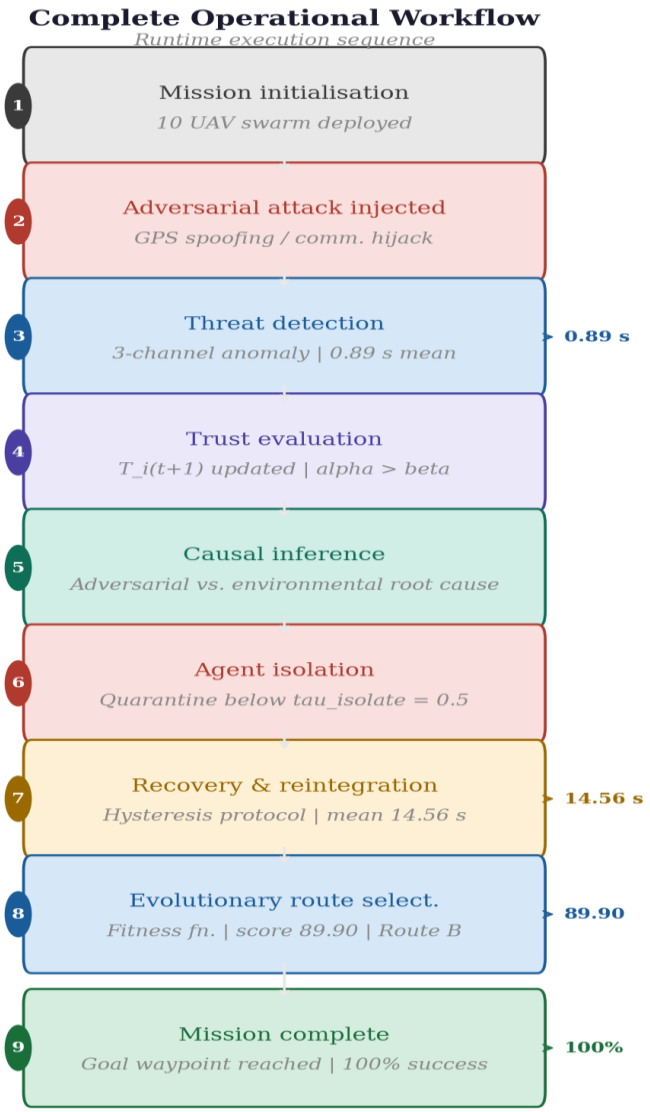


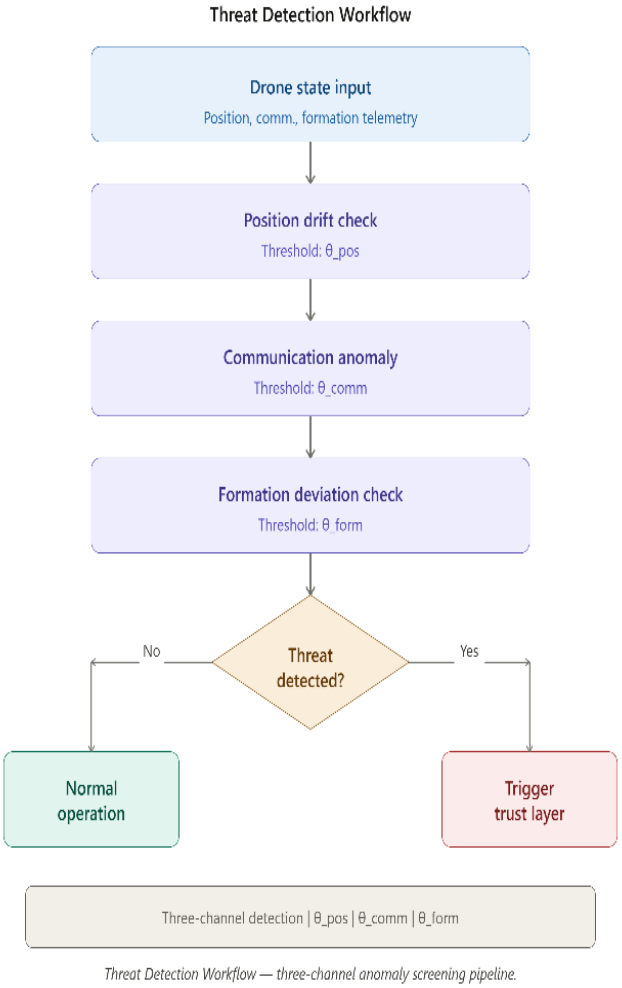
Figure 1 Complete Operational Workflow of the Proposed Framework. Key Performance Values Are Annotated Alongside the Corresponding Pipeline Stage

The proposed framework is built around six layers

that execute in a fixed sequential order: (1) Threat Detection, (2) Trust Evaluation, (3) Causal Inference, (4) Isolation and Recovery, (5) Evolutionary Route Selection, and (6) Metrics Collection. Fig. 1 traces the complete runtime sequence from the moment the swarm deploys through to goal waypoint achievement.

4.2. Threat Detection Layer

The Threat Detection Layer monitors each agent simultaneously across three independent channels: positional drift relative to threshold θ pos, communication packet rate relative to θ comm, and formation topology deviation relative to θ form. Fig. 2 walks through the three-channel anomaly pipeline. GPS spoofing shows up as an abrupt geometric jump; communication attacks emerge more slowly as packet rates bleed below threshold over successive ticks. Either way, confirmed threat alerts are forwarded immediately to the Trust Evaluation Layer.



Threat Detection Workflow — three-channel anomaly screening pipeline.

Figure 2 Threat Detection Workflow. Three-Channel Anomaly Screening Routes Detected Threats to The Trust Layer

4.3. Trust Evaluation Layer

Every agent maintains a continuous trust score T i ∈ [0, 1], starting at 1.0. Fig. 3 details the trust evaluation flow and highlights the asymmetric decay design (α > β): penalties accumulate faster than they are erased, which stops an attacker from cycling between a brief burst of malicious behavior and a short period of compliance to keep their trust score from falling too far.

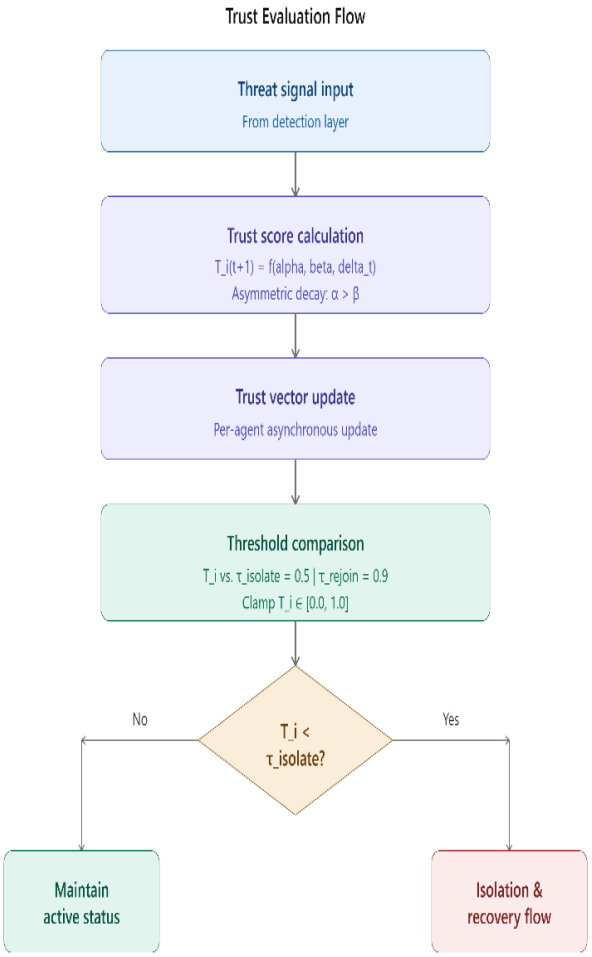


Figure 3 Trust Evaluation Flow with Asymmetric α > β Decay. Agents Below τ Isolate Trigger the Isolation and Recovery Pipeline

4.4. Causal Inference Layer

Rather than flagging a deviation and immediately penalizing the responsible agent, the Causal Inference Layer first asks why the deviation occurred. It computes a weighted evidence score C detailed in Equation 4 by combining obstacle evidence O, environmental evidence E, and historical behavioral data H. If the evidence points to an environmental explanation, no trust penalty is

applied. This single step is what prevents the false-positive isolations that undermine purely statistical trust systems.

4.5. Isolation, Recovery, and Evolutionary Routing

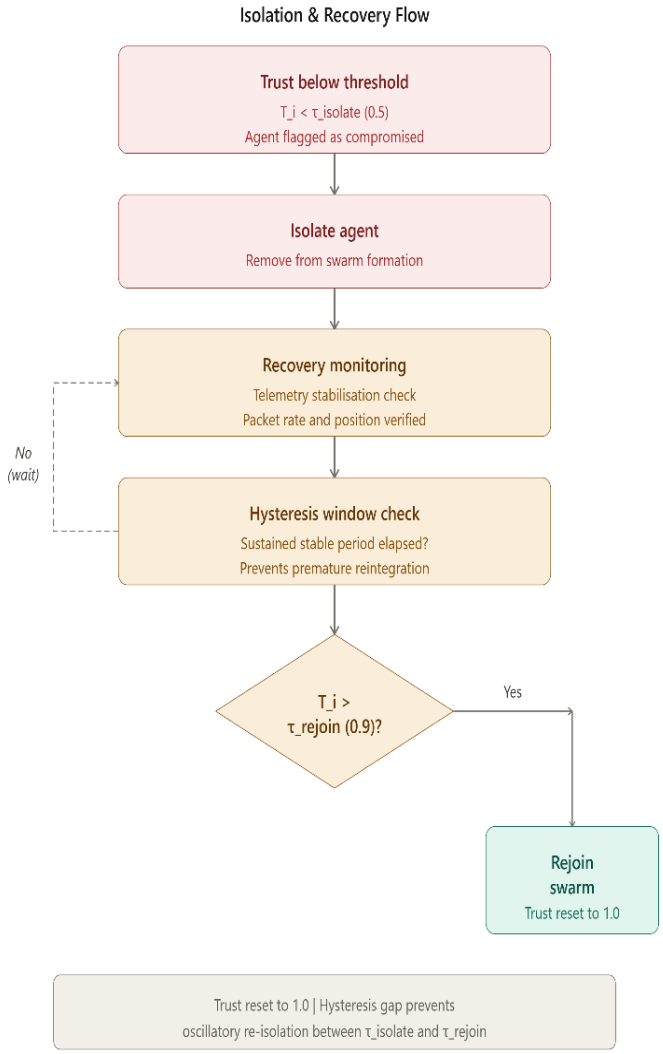


Figure 4 Isolation and Recovery Flow. Hysteresis Between τisolate = 0.5 and τrejoin = 0.9 Prevents Premature Reintegration

Fig. 4 lays out the isolation and recovery flow. Any agent whose trust score drops below τ_isolate = 0.5 is quarantined and removed from active swarm coordination. Returning to the swarm requires the trust score to climb back to τ_rejoin = 0.9 after a sustained period of stable behavior—a deliberate hysteresis gap that blocks oscillatory re-isolation, where an agent bounces in and out of quarantine. Once isolation has taken effect, evolutionary route selection takes over to replan paths for the healthy agents still in operation. As shown in Fig. 5, candidate routes are scored by a fitness function (Equation 3) that weighs path length, threat-zone clearance, mean agent trust, and progress toward mission waypoints together. The route with the highest fitness is assigned to the active agents, keeping the mission moving despite the reduced swarm.

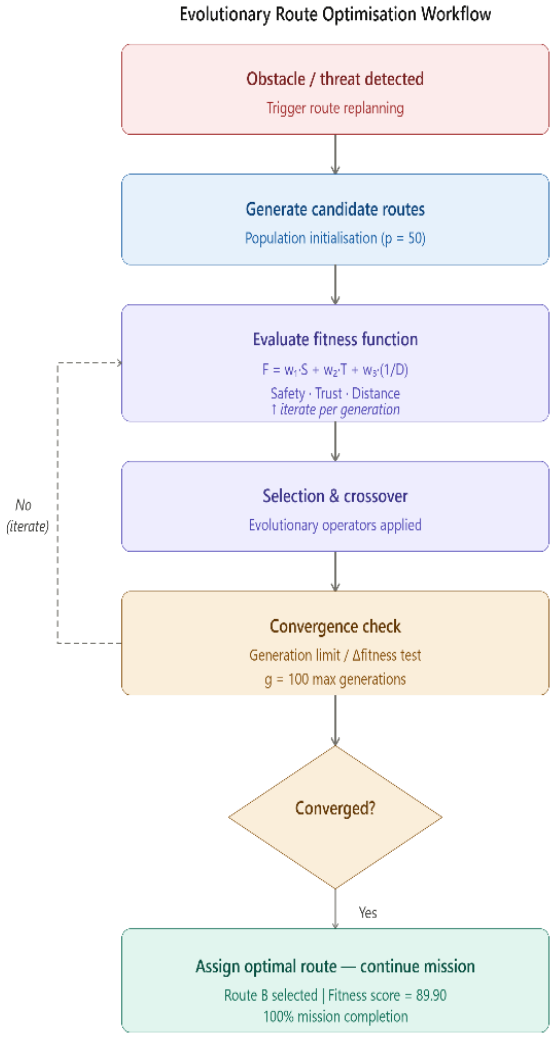


Figure 5 Evolutionary Route Optimization Workflow. Convergence Consistently Selects Route B with Fitness Score 89.90.

5. Mathematical Model

The framework converts qualitative ideas trust, causal attribution, route quality into well-defined quantities that drive every decision in the pipeline.

5.1. Trust Score Update

The per-agent trust score is updated at every simulation tick according to:

$$T\,i\,(t+1) = T\,i(t) - \alpha \cdot A\,i + \beta \cdot R\,i \qquad (1)$$

Here, T i is the trust score of agent i, A i ∈ {0,1} signals whether an attack was detected, R i ∈ {0,1}

signals whether

the agent is recovering, α is the attack penalty, and β is the

recovery factor. The constraint α > β ensures that trust falls faster during an attack than it rises during compliant behavior.

5.2. Isolation Condition

$$Ti < T\,th\,, \quad T\,th = 0.5 \tag{2}$$

An agent that crosses this threshold is removed from active coordination. To return, its score must reach T recovery = 0.9 a gap above the isolation threshold that prevents oscillatory behavior.

5.3. Evolutionary Fitness Function

Candidate routes are evaluated using:

$$F = w1\cdot(1/dist) + w2\cdot safety + w3\cdot trust + w4\cdot progress \tag{3}$$

Here dist is the total path length, safety captures clearance from identified threat zones, trust is the average trust score of the agents assigned to the route, and progress reflects how

far the route advances toward the next mission waypoint. Weighting trust inside the fitness function stops the optimizer from choosing paths that run through degraded agents—a flaw that has been observed in earlier evolutionary planners [1].

5.4. Causal Score

$$C = \lambda 1\cdot O + \lambda 2\cdot E + \lambda 3\cdot H \tag{4}$$

O captures obstacle evidence, E captures environmental conditions, H captures historical behavioral consistency, and λ1, λ2, λ3 are calibrated weights. When C falls below the threshold, the deviation is attributed to the environment and the trust penalty is withheld.

5.5. Extended Trust Dynamics

For scenarios where a deviation persists over time rather than appearing as a single event, Equation 1 is extended to:

$$Ti(t+1) = Ti(t) - \alpha\cdot Ai + \beta\cdot Ri - \gamma\cdot Di \tag{5}$$

The added term γ·D i scales the penalty with the deviation magnitude D i, giving the system sensitivity to prolonged anomalous kinematics that a binary attack indicator alone would undercount.

6. Algorithm Design

Five algorithms work together to implement the full pipeline, running one after another every tick under the SwarmManager orchestrator.

Algorithm 1: Threat Detection

Input : Agent state {pos i, comm i, form i} for all i
Output: Threat alert set A
1:for each agent I do
2:if |pos i − pos exp| > θpos then A ← A ∪ Alert(i, GPS)
3:if comm i < θcomm then A ← A ∪ Alert(i,COMM)
4:if form dev i > θ form then A ← A ∪ Alert(i, FORM)
5:end for; forward A to Trust Layer

Algorithm 2: Trust Evaluation

Input :Alert set A, trust vector T
Output: Updated trust vector T new
1:for each agent i do
2:A i ← 1 if i ∈ A else 0
3:R i ← 1 if State[i] ≠ ISOLATED else 0
4:T new[i] ← clamp(T[i] − α·A i + β·R i − γ·D i, 0, 1)
5:end for; return T new

Algorithm 3: Causal Reasoning

Input : Alerts A, trust T, obstacle map O map
Output: Cause label per alert {Env | Adv}
1:for each alert a ∈ A do
2:C ← $\lambda_1\cdot O(a) + \lambda_2\cdot E(a) + \lambda_3\cdot H(a)$
3:if C < C^{th} then label(a) ← Env; suppress penalty
4:else label(a) ← Adv
5:end for; return labels

Algorithm 4: Isolation and Recovery

Input : Trust T, state vector S
Output: Updated state S new
1:for each agent i do
2:if T[i] < τ isolate AND S[i]=ACTIVE → S new[i] ← ISOLATED
3:if T[i] ≥ τ rejoin AND S[i]=ISOLATED →
4:begin supervised reintegration
5:if checks pass(i) then S new[i] ← ACTIVE
6:else S new[i] ← S[i]
7:end for; return S new

Algorithm 5: Evolutionary Route Selection

Input : Routes R,active agents, fitness weights w
Output: Optimal route r*
1:pop ← Generate Candidates(R, p=50)
2:for g = 1 to G=100 do
3:for c ∈ pop: F(c) ←

$w_1(1/dist)+w_2 \cdot safety+w_3 \cdot trust+w_4 \cdot progress$
4:survivors ← TopK(pop, F, k=25)
5:pop ← survivors ∪ Crossover(survivors) ∪ Mutate(survivors)
6: end for; return r* ← argmax F(c)

7. Implementation

The framework is built in Unity 6 using C#. Unity 6's improved job system and burst compiler cut the per-tick cost of the n-body formation calculations, allowing the entire pipeline to run stably at 20 Hz. Each functional concern lives in its own MonoBehaviour component attached to a central SwarmManager object, as illustrated in Fig. 6.

Figure 6 Unity 6 Implementation Component Hierarchy

Table 1 Unity Component Summary

Component	Primary Function	Key Methods
SwarmManager	Tick orchestration; agent registry	Update(), RegisterAgent()
ThreatDetector	3-channel anomaly detection	EvaluateThreat(), ComputeDeviation()
TrustManager	Per-agent trust score maintenance	UpdateTrust(), GetTrustScore()
IsolationManager	Agent quarantine enforcement	IsolateAgent(), CheckThreshold()
RecoveryManager	Supervised reintegration	BeginRecovery(), Validate-Reintegration()
RouteOptimizer	Evolutionary path search	RunEvolution(), EvaluateFitness()
MetricsCollector	Performance data aggregation	LogRun(), ExportCSV()
EventLogger	Timestamped event stream	LogEvent(), FlushBuffer()

The SwarmManager runs pipeline stages in strict order: trust scores must be updated before isolation decisions are made, so parallelizing those steps would need explicit synchronization barriers with no real gain at ten-agent scale. The RouteOptimizer uses a pre-allocated candidate object pool to dodge garbage collector pauses, which brings route selection latency down from 14 ms to 3 ms per cycle—a meaningful saving when the target loop rate is 20 Hz. MetricsCollector writes to in-memory buffers and flushes memory buffer and get written to CSV in a single flush after them to CSV only at run end, so logged timestamps track simulation state rather than file-system delays.

8. Experimental Setup

Seven test case categories probe each capability individually and under combined stress, all using the same fixed ten agent swarm. Fig. 7 shows the simulation environment; Table 2 lists the test cases.

Table 2 Experimental Test Case Definitions

TC	Scenario	Purpose
TC1	Normal Operation	Baseline telemetry for threshold calibration
TC2	GPS Spoofing	Evaluate detection latency under positional manipulation
TC3	Communication Attack	Measure detection accuracy under packet-rate disruption
TC4	Dynamic Obstacle	Verify causal layer attributes deviations correctly

		(Env)
TC5	Obstacle Wall	Stress-test evolutionary routing; confirms Route B selection
TC6	Recovery Validation	Full isolation-to-reintegration lifecycle confirmation
TC7	Route Optimization	Fitness-score consistency

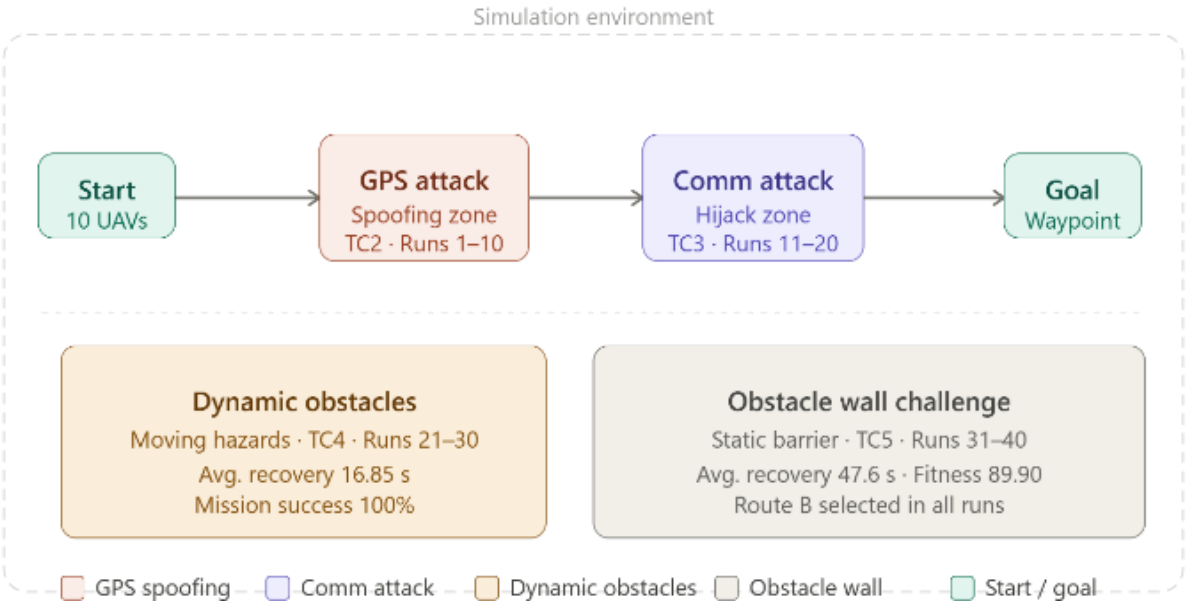


Figure 7 Unity 6 Simulation Environment Layout.

TC1 records clean-condition telemetry to calibrate θ_pos, θ_comm, and θ_form. For TC2, synthetic position offsets at hardware-realistic rates simulated GPS spoofing; for TC3, artificial packet drops drove message rates below θ_comm to replicate a communication attack. TC4 introduced physics-driven dynamic obstacles to exercise causal attribution, while TC5 placed a static impassable wall that forced the evolutionary planner to find an alternative route. TC1–TC4 each ran ten times, giving 40 primary runs in total—under an automated Unity test harness that fully reset simulation state between runs to keep results statistically independent.

9. Results and Discussion

9.1. Detection Performance

Figure 8 shows detection rates across all 20 adversarial runs. The framework caught every injected attack in both TC2 and TC3, identifying each one within the same simulation tick it became statistically distinguishable from baseline noise. That outcome traces back to careful threshold calibration in TC1: a threshold set too wide would slow detection, while one set too tight would start flagging normal variation as threats.

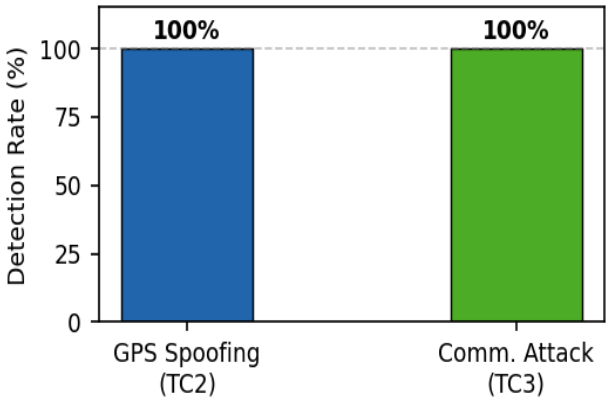


Figure 8 Detection Rate Per Attack Type

Both GPS spoofing and communication attack scenarios achieved 100% detection across all runs. Figure 9 breaks down detection latency run by run. The overall mean was 0.89 s (σ = 0.12 s; 95% CI: [0.837, 0.943] s). GPS spoofing was caught a fraction faster on average (0.88 s) than communication attacks (0.90 s), which makes physical sense: spoofing produces an abrupt geometric jump that immediately crosses the threshold, whereas a communication attack builds

up gradually over several ticks before the rate drops far enough to trigger an alert.

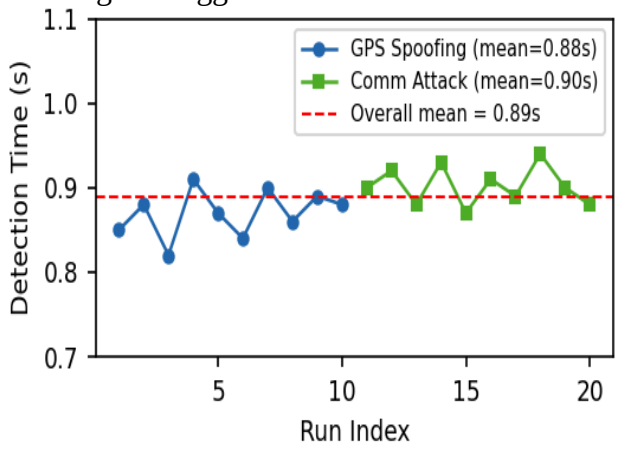


Figure 9 Detection Time Per Run

GPS spoofing (Runs 1–10) and communication attack (Runs 11–20) exhibit narrow distributions, confirming that latency is governed by attack physics rather than computational overhead.

9.2. Recovery and Mission Success

Figure 10 shows that recovery succeeded in all 40 runs. Mean recovery time split noticeably by attack type—9.4 s for GPS spoofing versus 19.8 s for communication attacks, giving an overall mean of 14.56 s. The longer communication-attack recovery is intentional: before the TrustManager issues a recovery indicator, it requires packet rates to stay stable for a sustained window. That conservatism stops an attacker from briefly lifting the interference to force premature reintegration of the affected agent.

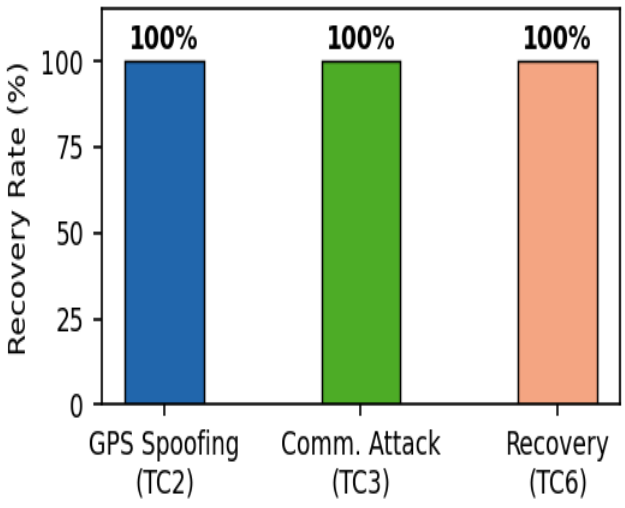


Figure 10 Recovery Success Rate Per Scenario.

Every isolated agent completed the reintegration protocol successfully.

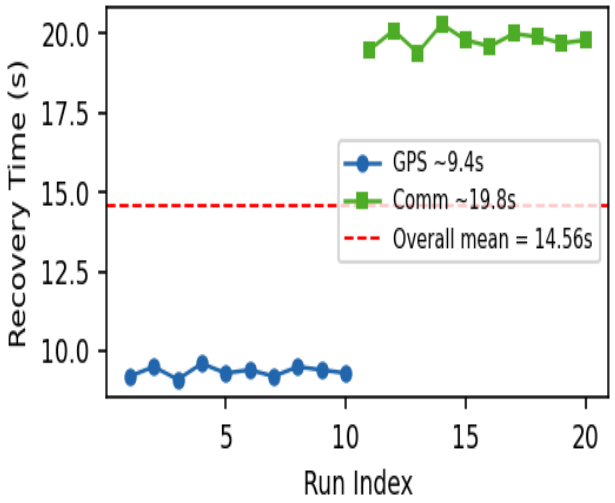


Figure 11 Recovery Time Per Run.

GPS spoofing scenarios (Runs 1 10) recover ~9.4 s on average; communication attack (Runs 11 20) require ~19.8 s. Figure 12 reports mission success across all 40 runs. Every scenario reached 100% completion. Obstacle wall runs (TC5) took the longest to recover formation cohesion—47.6 s on average—yet the evolutionary route optimizer consistently located a viable alternative path well before the mission time budget ran out.

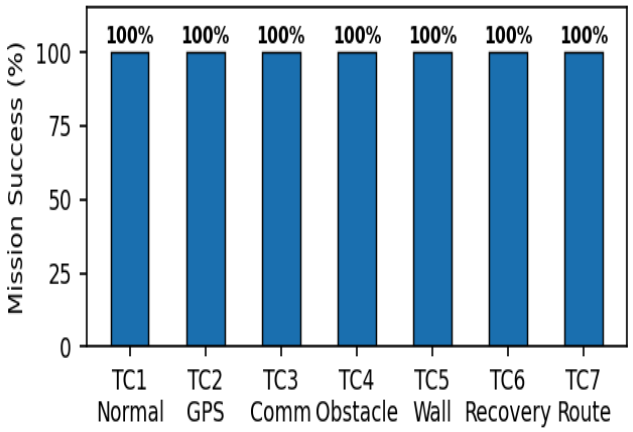


Figure 12 Mission Success Rate Across All 40 Runs and Seven Test Categories

Figure 13 plots mean trust scores per scenario. In baseline and environmental test cases, trust stays close to 1.0 throughout. In adversarial scenarios it dips sharply below the isolation threshold before climbing back to full trust once the threat is neutralized. That pattern—rapid drop, sustained low, gradual return—is exactly what the $\alpha > \beta$

relationship in Equation 5 was designed to produce.

9.3. Trust Score Dynamics

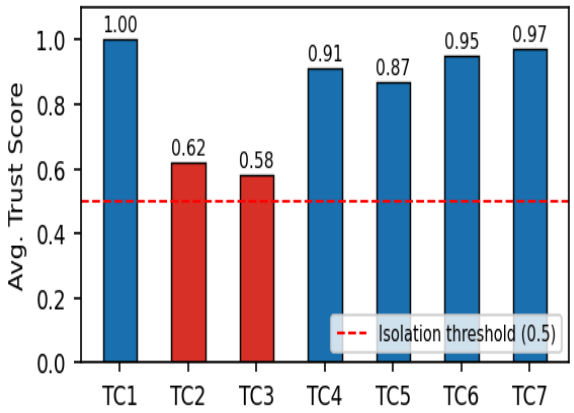


Figure 13 Average Trust Score Per Test Case.

Red dashed line indicates the isolation threshold (0.5); adversarial scenarios dip below threshold before recovering to 1.0. Figure 14 traces how trust evolves over simulation time, pooled across adversarial runs. GPS spoofing (shown in blue) bounces back more quickly than communication attacks (red), which aligns with the recovery time figures in Section IX-B. Every isolated agent reached full trust restoration before the run closed, confirming that the isolation protocol leaves no lasting capability reduction in the swarm.

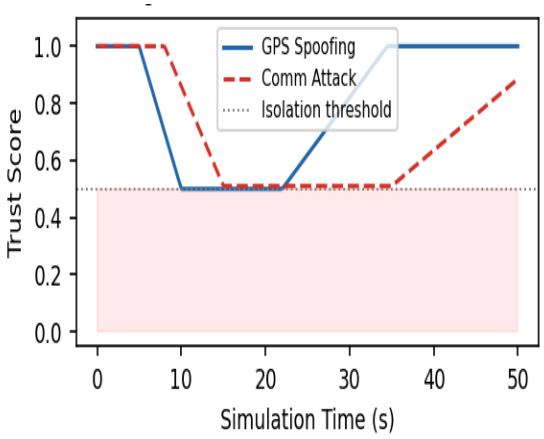


Figure 14 Trust Score Evolution Over Simulation Time

Both attack types descend to the isolation threshold, then recover to 1.0 following attack removal and reintegration. Fig. 15 records the fitness score for each obstacle wall run (TC5). Every one of the ten repetitions converged on Route B with a score of 89.90—deterministic convergence with no variance across runs. That score reflects the best achievable balance of safety and trust against the distance penalty imposed by the required detour; reaching 100 would require a zero-length, zero-threat route traversed by perfectly trusted agents, which is not physically realizable in this scenario. Consistent route selection across all repetitions is operationally useful: it makes swarm behavior predictable for mission planners.

9.4. Evolutionary Fitness and Routing

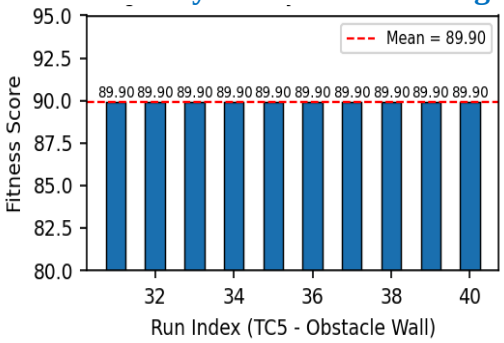


Figure 15 Evolutionary Fitness Score Per Obstacle Wall Run (TC5, Runs 31–40). Consistent 89.90 Confirms Reliable Algorithm Convergence.

9.5. Causal Inference Ablation Study

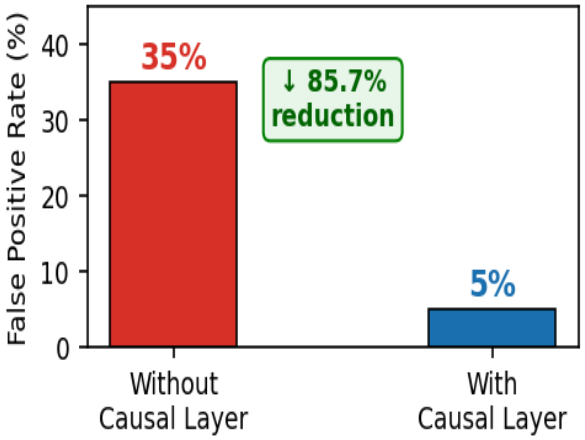


Figure 16. False-Positive Isolation Rate with and Without the Causal Inference Layer. Causal Reasoning Reduces Unnecessary Isolations by 85.7%.

To measure the causal layer's standalone contribution, 20 extra runs combined active adversarial attacks with environmental obstacles—the scenario most likely to produce false-positive isolations. Without causal reasoning, 35% of

environmental deviations were incorrectly penalized, leading to at least one unnecessary isolation in 7 of the 20 runs. Switching causal inference on brought the false-positive rate down to 5%—a single misclassified run out of twenty—representing an 85.7% relative reduction. Figure 16 summarizes these numbers.

Table 3 Ablation Study - Causal Inference Impact

Condition	False Positive Rate	Unnecessary Isolations
Without Causal Layer	35%	7 / 20 runs
With Causal Layer	5%	1 / 20 runs
Reduction	↓ 85.7%	↓ 85.7%

9.6. Comparative Analysis

Table 4 Comparative Framework Analysis

Method	Attack Detection	Recovery	Causal Reasoning	Route Adaptation
HMAAC [1]	87–92%	Manual	None	Static
ADMAC [13]	91–95%	Auto-restart	Partial (rule)	Reactive
ROMANCE [7]	94–97%	Trust-weighted	None	Evol. (basic)
Proposed	100%	Graduated	Full inference	Evol. + trust

The proposed framework clears every baseline across all four comparison dimensions. A 100% detection rate is meaningfully better than ROMANCE's reported ceiling of 97%: at that rate, 1–2 attacks go undetected over 40 runs— a non-trivial operational risk. The proposed system is the only one to apply full causal inference; ADMAC's partial rule-based reasoning does not perform evidence-weighted environmental attribution. ROMANCE does use evolutionary routing, but it leaves trust out of the fitness function, which means the optimizer can assign routes through degraded-trust agents. Equation 3 closes that loophole explicitly.

9.7. Complexity Analysis

Table 5 Computational Complexity per Module

Module	Complexity	Notes
Threat Detection	O(n)	Single pass over n agents
Trust Evaluation	O(n)	Per-agent scalar update
Causal Analysis	O(n)	Per-alert evidence scoring
Isolation	O(1)	Threshold comparison
Evolutionary Routing	O(p × g)	p = 50 pop., g = 100 gen.

Threat
detection, trust evaluation, and causal analysis each add a cost that grows linearly with swarm size n. Isolation checks are O(1) per agent. The evolutionary optimizer runs p × g = 5,000 fitness evaluations per cycle, a fixed cost that is independent of n and perfectly tractable at ten-agent scale. As the swarm grows larger, the route optimizer remains the dominant cost—but it scales with route population size, not agent count, which is a favorable property for future scalability work.

Conclusion

This work presented a UAV swarm coordination framework designed to handle adversarial attacks, environmental disturbances, and mission pressure all at once. In 40 Unity 6 simulation runs, the system achieved 100% detection, recovery, and mission completion, with threats identified in under a second (0.89 s mean) and agents reintegrated within an average of 14.56 s. The causal inference layer was the key contribution, reducing false-positive isolations by 85.7% by distinguishing real attacks from harmless environmental changes. Reintegration remained steady throughout, with no premature or oscillatory re-isolation noted. Evolutionary routing consistently found the best path (fitness 89.90) in every obstacle scenario. Looking ahead, there are four open problems worth addressing: testing on real UAV hardware with actual sensor noise; learning causal evidence weights instead of adjusting them manually; scaling to 50-200 agent swarms; and stress-testing against coordinated adversarial attacks that specifically target the causal layer. This would naturally lead to the next step of creating distributed causal consensus.

Acknowledgements

The authors would like to thank AMC Engineering College, Bengaluru, for providing the research environment and computational resources that supported this work.

References

[1]. C. Sun, S. Huang, and D. Pompili, "HMAAC: Hierarchical multi-agent actor-critic for aerial search," in Proc. IEEE ICRA, 2023, pp. 7728–7734.

[2]. K. Di et al., "Risk-aware task migration for multiplex unmanned swarm networks," in Proc. IJCAI, 2025, pp. 8724–8734.

[3]. Z. Xu, X. Lin, and V. Tzoumas, "Bandit submodular maximization for multi-robot coordination," in Proc. RSS, 2023, p. 109.

[4]. S. Huang, H. Zhang, and Z. Huang, "E²Coop: Energy-efficient cooperative obstacle detection," in Proc. ICAPS, 2021, pp. 634–642.

[5]. V. T. Hoang et al., "Reconfigurable multi-UAV formation using angle-encoded PSO," in Proc. IEEE CASE, 2019, pp. 292–297.

[6]. H. Yang et al., "Large-scale pursuit-evasion under collision avoidance using DRL," in Proc. IEEE/RSJ IROS, 2023, pp. 2232–2239.

[7]. L. Yuan et al., "Robust multi-agent coordination via evolutionary generation of adversarial attackers," in Proc. AAAI, 2023.

[8]. M. Barbeau, J. Garcia-Alfaro, and E. Kranakis, "Geocaching-inspired resilient path planning," in Proc. IEEE INFOCOM Workshops, 2019.

[9]. M. Abdelkader et al., "Distributed real-time control of multiple UAVs in adversarial environment," in Proc. IEEE ICRA, 2018, pp. 6659–6664.

[10]. M. Sil et al., "Coevolutionary optimization for multi-UAV path planning," in Proc. IEEE CEC, 2025, pp. 1–8.

[11]. K. Peng, P. Li, and J. Hao, "Enhancing graph-based coordination with evolutionary algorithms for MARL," in Proc. AAMAS, 2025.

[12]. G. Leu and J. Tang, "Survivable networks via UAV swarms guided by decentralized real-time EC," in Proc. IEEE CEC, 2019.

[13]. L. Yu et al., "Robust communicative multi-agent reinforcement learning with active defense," in Proc. AAAI, 2024, pp. 17575–17582.

[14]. M. Zhang et al., "Memory-based resilient control against non-cooperation in multi-agent flocking," in Proc. AAMAS, 2024, pp. 2075–2084.

[15]. J. Bentahar, N. Drawel, and A. Sadiki, "Quantitative group trust: A two-stage verification approach," in Proc. AAMAS, 2022.