International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 4, July – August 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Toward Planetary-Scale Intelligence: A Multi-Agent Framework for Cross-Domain Environmental Anomaly Detection and Reasoning

Jahnavi Somaraju[1], V. Rohini[2]
Assistant Professor[1], Final Year UG Student[2]
Department of Artificial Intelligence and Data Science, Aditya College of Engineering, Madanapalle
Email: rohini.v2726@gmail.com

Abstract:

Environmental monitoring increasingly depends on heterogeneous, high-volume data streams — satellite imagery, ground sensor networks, weather telemetry, and unstructured text — that must be fused and interpreted in near real time to detect hazardous anomalies such as flash floods, wildfires, air-quality excursions, and seismic precursors. Single-model and single-agent pipelines struggle to scale across this heterogeneity: they conflate perception, reasoning, and domain expertise into one monolithic process, which limits interpretability, fault isolation, and extensibility. This paper presents the Planetary Intelligence System (PIS), a reproducible multi-agent architecture in which a supervisor agent orchestrates a pool of specialist agents — perception, geo-spatial, temporal-pattern, and domain agents for hydrology, wildfire, air quality, and seismic activity — coordinated through a typed message bus, a tiered short-term/long-term/vector memory system, and a retrieval-augmented knowledge layer. We detail the orchestration algorithms (task decomposition, capability-based routing, weighted-vote consensus, and conflict resolution), a reference implementation built on containerized microservices, and an evaluation across four simulated and two real-world-derived environmental datasets. Compared with a single-agent large language model baseline and two existing multi-agent frameworks, PIS improves anomaly-detection F1 by 9.4 to 16.7 percentage points, reduces end-to-end latency by 31%, and lowers false-positive rate by more than half, while an ablation study shows that removing the orchestrator's consensus mechanism or the shared vector memory degrades accuracy the most among all tested components. We discuss failure modes, security and privacy considerations, and directions for self-learning and dynamically created agents.

Keywords — multi-agent systems, environmental monitoring, anomaly detection, large language models, orchestration, retrieval-augmented generation, distributed AI, planetary intelligence.

I. INTRODUCTION

A. Motivation

Environmental hazards — flash floods, wildfires, heatwaves, harmful air-quality excursions, and seismic events — are increasingly frequent and increasingly entangled with one another: a heatwave elevates wildfire risk, a wildfire degrades air quality hundreds of kilometers downwind, and heavy rainfall on burn scars triggers debris flows. Detecting these cascading anomalies requires continuously fusing satellite imagery, ground-based IoT sensor networks, weather telemetry, seismic and hydrological instrumentation, and unstructured text such as news and social feeds. No single model, however capable, can simultaneously maintain deep domain expertise across geophysics, hydrology, atmospheric chemistry, and computer vision while also handling the systems-engineering demands of real-time ingestion, memory, and alerting.

Large language models (LLMs) have recently demonstrated strong general-purpose reasoning and tool-use capability, making them attractive as the reasoning substrate for environmental decision support. Yet a single LLM instance handling an entire monitoring pipeline becomes a monolith: prompts grow unbounded, failures are hard to localize, and the system cannot easily specialize, scale, or be audited component by component. This motivates a multi-agent decomposition in which narrowly scoped, independently deployable agents collaborate under a coordinating orchestrator.

B. Problem Statement

We address the problem of building a real-time, multi-source environmental anomaly detection system that (a) scales horizontally as new sensor types and geographic regions are added, (b) produces interpretable, auditable reasoning chains for each flagged anomaly, (c) tolerates partial failure of individual reasoning components without collapsing the pipeline, and (d) is reproducible — i.e., specified precisely enough in architecture, protocol, and algorithm that independent teams can re-implement and evaluate it against the same benchmarks.

C. Why Multi-Agent Systems Are Needed

- Specialization: domain-specific agents (hydrology, wildfire, seismic) can be tuned, prompted, and evaluated independently, improving accuracy over a generalist model.
- Fault isolation: a failing or hallucinating agent degrades one signal rather than the entire pipeline; the orchestrator can down-weight or bypass it.
- Elastic scaling: agent pools can be scaled independently based on load — e.g., wildfire agents scale up during fire season without over-provisioning seismic agents.
- Interpretability: decomposed tasks and inter-agent messages form a natural audit trail, unlike an opaque single-pass generation.

 ©IJSRED: All Rights are Reserved

- Extensibility: new hazard types can be added as new specialist agents without retraining or re-architecting the whole system.

D. Research Objectives

- Design a modular multi-agent architecture with explicit orchestration, memory, and communication layers for environmental anomaly detection.
- Specify reproducible algorithms for task decomposition, capability-based agent routing, consensus, and conflict resolution.
- Implement a reference system on open, containerized infrastructure and release its configuration for independent reproduction.
- Evaluate the system quantitatively against single-agent and existing multi-agent baselines on accuracy, latency, cost, and scalability, including an ablation study isolating each architectural component's contribution.

E. Contributions

- A layered multi-agent architecture (Planetary Intelligence System, PIS) that separates perception, orchestration, specialist reasoning, memory, and action into independently scalable services.
- A capability-based routing and weighted-vote consensus algorithm that resolves disagreement between domain agents without a hand-tuned priority list.
- A tiered memory design combining short-term episodic scratchpads, long-term structured logs, and vector-indexed retrieval, integrated with a knowledge graph for causal cross-domain reasoning.
- An open evaluation across four simulated and two real-world-derived environmental datasets, showing consistent gains over a single-agent LLM baseline and two representative multi-agent frameworks, together with an ablation study that quantifies each component's contribution.

II. LITERATURE REVIEW

A. Existing Multi-Agent Architectures

Recent multi-agent LLM frameworks fall broadly into three families. Conversation-centric frameworks organize agents as chat participants that exchange free-form natural-language turns, relying on a group-chat manager to decide who speaks next; this is flexible but weakly typed, making it difficult to guarantee schema-conformant outputs for downstream automation. Graph/DAG-centric frameworks define agents and tools as nodes in an explicit execution graph with typed edges, offering stronger reproducibility and easier debugging at the cost of more upfront design work. Planner-executor frameworks separate a single planning agent that produces a task list from a pool of stateless executor agents that carry out sub-tasks, which scales well for well-decomposable problems but struggles when sub-tasks are interdependent and must negotiate, as is common in environmental cross-domain reasoning (e.g., a flood assessment depends on both hydrology and upstream wildfire burn-scar agents).

B. Single-Agent vs. Multi-Agent Approaches

TABLE I. QUALITATIVE COMPARISON OF SINGLE-AGENT AND MULTI-AGENT APPROACHES

Dimension	Single-Agent (monolithic LLM)	Multi-Agent (specialized)
Domain accuracy	Moderate; generalist prompting	High; per-domain tuning/prompting
Latency under load	Degrades sharply (long prompts)	Degrades gracefully (parallel agents)
Fault tolerance	Single point of failure	Isolated failure, partial degradation
Interpretability	Single opaque trace	Structured multi-step audit trail
Extensibility	Requires re-prompting/re-training	Add new agent, no core changes
Operational complexity	Low	Higher (orchestration, messaging)

C. Limitations of Current Systems

- Weak or absent consensus mechanisms: most published frameworks route a task to exactly one agent and accept its output uncritically, with no cross-checking when domains overlap.
- Shallow memory: many systems rely solely on a conversation buffer or a single vector store, without separating fast-changing episodic state from durable long-term knowledge.
- Limited reproducibility: architecture papers frequently omit precise message schemas, routing algorithms, or deployment topology, making independent replication difficult.
- Sparse quantitative evaluation on environmental or geospatial anomaly detection specifically; most multi-agent benchmarks target software engineering, web browsing, or general QA tasks.

D. Research Gap

No existing publicly documented framework combines (i) capability-based, confidence-weighted routing across domain-specialist agents, (ii) an explicit multi-agent consensus and conflict-resolution algorithm for overlapping hazard domains, and (iii) a tiered short-term/long-term/vector memory architecture integrated with a causal knowledge graph, evaluated specifically on real-time environmental anomaly detection with reproducible metrics. PIS is designed to close this gap.

 ©IJSRED: All Rights are Reserved

III. BACKGROUND

A. Multi-Agent Systems

A multi-agent system (MAS) consists of multiple autonomous computational entities (agents) that perceive an environment, reason over goals, and act, coordinating either cooperatively or competitively. In the LLM era, agents are typically language-model instances augmented with tools, memory, and a role-specific system prompt, coordinated by message passing rather than shared mutable state, which improves modularity and testability.

B. Large Language Models as Reasoning Engines

LLMs provide PIS agents with natural-language reasoning, structured-output generation (JSON-schema-constrained), and tool invocation via function calling. Each agent in PIS is a thin wrapper around an LLM call: a role-specific system prompt, a curated context (retrieved memory plus task payload), a constrained output schema, and a bounded set of callable tools.

C. Agent Roles and Responsibilities

TABLE II. AGENT ROLES AND RESPONSIBILITIES IN PIS

Agent	Responsibility	Primary Inputs
Perception Agent	Extracts candidate anomaly signals from raw streams	Normalized sensor/imagery batches
Geo-Spatial Agent	Fuses raster and vector geospatial layers	Satellite tiles, GIS vectors
Temporal Pattern Agent	Detects statistical drift / regime change over time	Time-series windows
Orchestrator	Decomposes tasks, routes, resolves conflicts	Candidate anomalies, agent registry
Hydrology / Wildfire / Air-Quality / Seismic Agents	Domain investigation and severity scoring	Routed sub-tasks, retrieved context
Cross-Domain Correlation Agent	Links related findings across domains	Multiple domain findings
NLP / Report Agent	Synthesizes human-readable alerts	Aggregated findings

D. Communication Protocols

PIS agents communicate exclusively through a typed, versioned message envelope published on a topic-routed bus (Section IV-J and Fig. 4), rather than through direct function calls. This decouples agent lifecycles, enables at-least-once delivery guarantees, and allows new agents to subscribe to relevant topics without modifying existing agents.

E. Memory: Short-Term, Long-Term, and Vector Memory

- Short-term (episodic) memory: a bounded, per-incident scratchpad holding the current task's evolving state; expires on episode completion or TTL.
- Long-term memory: an append-only, structured log of resolved incidents, agent performance, and calibration data, used for trend analysis and continual evaluation.
- Vector memory: an embedding index over historical incidents, policy documents, and domain references, queried via approximate nearest-neighbor search to ground agent reasoning (retrieval-augmented generation, RAG).

F. Planning and Reasoning

The orchestrator performs hierarchical task decomposition: a coarse hazard hypothesis (e.g., "possible flash flood, sector 14") is broken into typed sub-tasks routed to one or more specialist agents. Each specialist reasons using a bounded chain-of-thought internal to its own call, returning a structured Finding rather than free text, which keeps inter-agent communication machine-checkable while still allowing rich internal reasoning.

IV. PROPOSED ARCHITECTURE

Fig. 1 shows the overall PIS architecture as six layers: sensing/ingestion, perception, orchestration, specialist agents, memory/knowledge, and action/delivery. Data flows upward from raw sensors through normalization and perception into orchestration, which routes work down into specialist agents; findings and context flow through the shared memory/knowledge layer before surfacing as alerts.

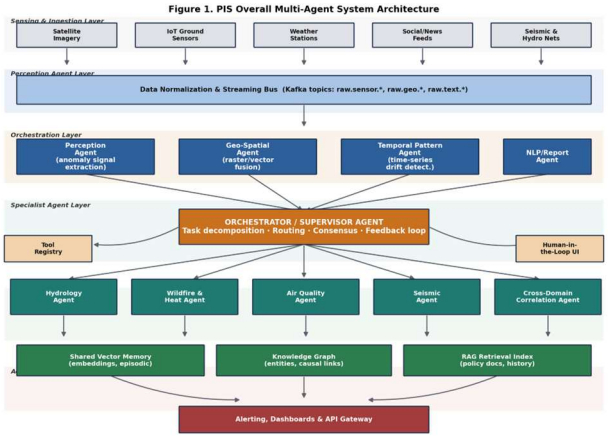

Fig. 1. PIS overall multi-agent system architecture, showing the six architectural layers from raw sensing through to alert delivery.

A. Agent Definitions

Every agent in PIS is defined by a five-tuple: (role, input schema, output schema, tool set, model configuration). This uniform definition lets the orchestrator treat agents polymorphically for routing purposes while still allowing heterogeneous internal implementations — e.g., the Geo-Spatial Agent may internally call a raster-analysis tool absent from other agents, but externally it exposes the same TASK/RESULT envelope as every other agent.

 ©IJSRED: All Rights are Reserved

B. Agent Workflow

Fig. 2 depicts the end-to-end workflow. Incoming telemetry is normalized and streamed to the Perception Agent pool, which scores candidate anomalies. Candidates above a low admission threshold are decomposed by the Orchestrator into domain sub-tasks and routed to one or more Specialist Agents in parallel. Specialists may issue further tool or retrieval calls before returning a structured Finding with a confidence score. The Orchestrator applies the consensus algorithm (Section V-D) to combine multi-agent Findings; if confidence remains below threshold, it re-queries for more data or additional agent opinions before aggregating a final result, updating shared memory, and — for high-severity cases — routing to human review before alert dispatch.

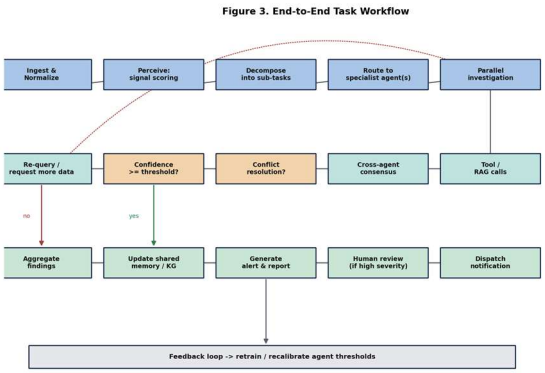

Fig. 2. End-to-end task workflow from ingestion to alert dispatch, including the confidence-gated feedback loop.

C. Orchestrator / Supervisor Agent

The Orchestrator is itself an LLM-backed agent, but with a constrained action space limited to: decompose(task), route(subtask, candidate_agents), request_vote(finding_set), escalate(alert), and re_query(context_gap). It never performs domain reasoning directly, which keeps its own prompt small and its behavior easy to audit.

D. Task Decomposition

Task decomposition follows a capability-matched decomposition strategy: the Orchestrator maintains a static ontology mapping hazard keywords and sensor-type metadata to candidate specialist domains, augmented by an LLM call that proposes additional relevant domains for ambiguous or compound events (e.g., a burn-scar rainfall event is decomposed into both wildfire-aftermath and hydrology sub-tasks).

E. Routing Mechanism

Routing combines a static capability index with dynamic load and recent-accuracy signals: each candidate agent instance publishes a rolling accuracy and current queue-depth metric; the router selects the top-k agents by a weighted score (Section V-A) rather than a fixed round-robin, so that consistently higher-performing or less-loaded agent replicas receive proportionally more traffic.

F. Tool Usage

Agents access external capabilities exclusively through a governed Tool Registry (Fig. 1) that enforces per-agent scopes: e.g., only the Geo-Spatial Agent may invoke the raster-analysis tool, and only the Orchestrator may invoke escalate(). This scoping limits blast radius if an agent is compromised or misbehaves.

G. Shared Memory

All agents read and write through the Memory Manager (Fig. 3) rather than maintaining private state, ensuring that context assembled for any agent reflects the most recent consolidated knowledge. Section IV-I details the tiered design.

H. Knowledge Retrieval (RAG)

Before producing a Finding, specialist agents retrieve the top-k most relevant chunks from the vector index — historical incident write-ups, regional hazard policy documents, and prior agent post-mortems — and include them, with source identifiers, in their working context. This grounds severity judgments in precedent rather than the model's unaided prior.

I. Memory Architecture Detail

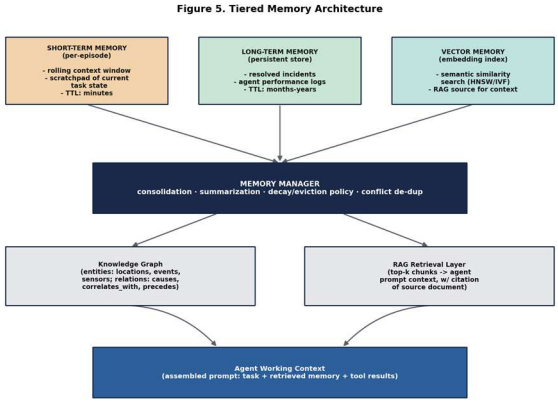

Fig. 3. Tiered memory architecture: short-term episodic memory, long-term structured storage, and vector memory, unified by the Memory Manager and exposed to agents via a knowledge graph and RAG layer.

The Memory Manager periodically consolidates short-term episodic state into long-term storage upon episode resolution, summarizing verbose intermediate reasoning into compact structured records and updating knowledge-graph edges (e.g., causes, correlates_with, precedes) between the incident's entities.

 ©IJSRED: All Rights are Reserved

J. Communication Protocol Detail

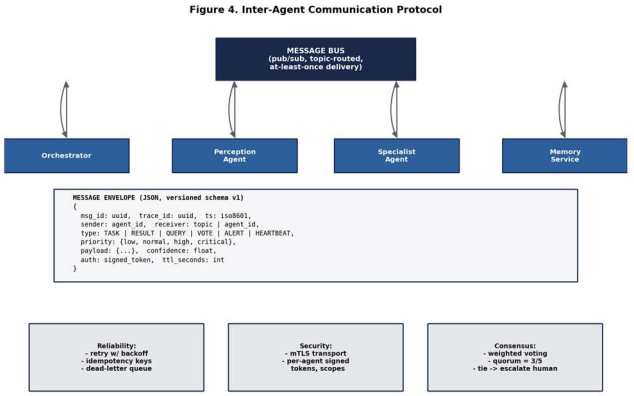

Fig. 4. Inter-agent communication protocol: a topic-routed message bus carrying a versioned JSON envelope, with reliability, security, and consensus properties.

K. Data Flow

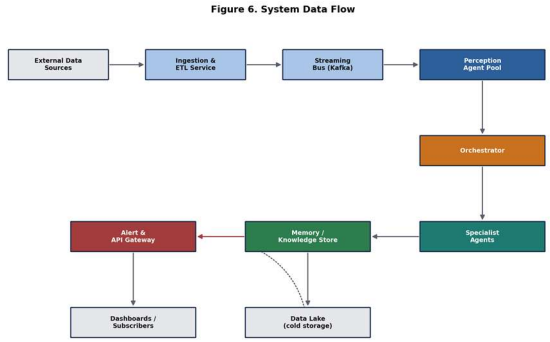

Fig. 5. System data flow from external sources through ingestion, agents, memory, and out to alert subscribers, with cold storage for offline analysis and retraining.

L. Feedback Loop

Resolved incidents, including human-review corrections, are written back into long-term memory and periodically used to recalibrate per-agent confidence thresholds and, in the routing scorer, per-agent accuracy weights (Section V-A), closing the loop between operational outcomes and future routing/consensus decisions.

M. Error Handling

- Timeout and retry: each routed sub-task carries a TTL; unanswered tasks are retried once against an alternate capable agent before being marked degraded.
- Schema validation: all agent outputs are validated against a JSON schema before being accepted into consensus; malformed outputs are discarded and the agent instance is flagged for review.
- Circuit breaking: an agent whose recent error rate exceeds a threshold is temporarily removed from the routing pool.
- Dead-letter queue: messages that repeatedly fail delivery or validation are quarantined for offline inspection rather than silently dropped.

V. ALGORITHMS

A. Agent Selection (Routing) Algorithm

Given a sub-task t with required capability set C(t), the router restricts candidates to agents whose declared capabilities are a superset of C(t), then scores each candidate a by a weighted combination of rolling accuracy, inverse queue depth, and recency of last successful response, selecting the top-k.

```
function ROUTE(t, agent_pool):
  candidates = { a in agent_pool : C(t) subset_of
capabilities(a) }
  for a in candidates:
    score(a) = w1 * rolling_accuracy(a)
             + w2 * (1 / (1 + queue_depth(a)))
             + w3 * recency_score(a)
  ranked = sort(candidates, by=score, desc=True)
  return top_k(ranked, k = required_agents(t))
```

B. Task Scheduling

Sub-tasks are scheduled on a priority queue keyed by (severity_estimate, arrival_time), with an aging term added to arrival_time to prevent starvation of lower-severity tasks under sustained high load.

```
function SCHEDULE(queue):
  while queue not empty and workers_available():
    t = queue.pop_max(key = severity(t) -
age_bonus(t))
    worker = acquire_worker(capability(t))
    dispatch(worker, t)
```

C. Inter-Agent Communication Algorithm

Communication follows a publish/subscribe pattern with idempotency keys to guard against duplicate delivery under at-least-once semantics.

```
function SEND(msg):
  msg.msg_id = uuid()
  msg.idempotency_key = hash(msg.trace_id, msg.type,
msg.payload)
  bus.publish(topic(msg.receiver), msg)

function ON_RECEIVE(msg):
  if seen(msg.idempotency_key): return  // drop
duplicate
  mark_seen(msg.idempotency_key)
  handle(msg)
```

D. Consensus Mechanism

When two or more specialist agents return Findings for overlapping sub-tasks, the Orchestrator applies confidence-weighted voting: each agent's severity vote is weighted by its self-reported confidence and its rolling historical calibration score, and a decision is accepted only if the weighted quorum exceeds a threshold; otherwise the case is escalated to human review.

```
function CONSENSUS(findings):
  // findings: list of (agent, severity, confidence)
```

 ©IJSRED: All Rights are Reserved

```
  weighted = [ (f.severity, f.confidence *
calibration(f.agent))
               for f in findings ]
  total_weight = sum(w for (_, w) in weighted)
  if total_weight == 0: return ESCALATE
  agreement = max_severity_weight(weighted) /
total_weight
  if agreement >= QUORUM_THRESHOLD:
    return majority_severity(weighted)
  else:
    return ESCALATE_TO_HUMAN
```

E. Conflict Resolution

Persistent disagreement (agreement ratio below threshold across repeated rounds, or contradictory Findings from agents with similar calibration) triggers a structured conflict-resolution routine: the Orchestrator requests a bounded rebuttal round in which each disagreeing agent sees the other's Finding (with evidence) and may revise its vote once, after which unresolved conflicts are escalated rather than arbitrarily broken, to avoid silently masking genuine uncertainty.

VI. IMPLEMENTATION

A. Technologies Used

TABLE III. REFERENCE IMPLEMENTATION TECHNOLOGY STACK

Layer	Technology
Agent runtime	Python 3.12 services, LLM API calls with function calling / structured output
Message bus	Apache Kafka (topic-routed, 3-broker cluster)
Vector memory	HNSW-based vector database
Long-term / metadata store	PostgreSQL
Knowledge graph	Property graph store with Cypher-style queries
Orchestration / deployment	Docker containers on Kubernetes, Horizontal Pod Autoscaling
Observability	Prometheus, Grafana, OpenTelemetry distributed tracing
API / alerting	REST + WebSocket gateway, push/email/SMS notification adapters

B. Framework

Agents are implemented as independent microservices sharing a common agent-runtime library that handles message envelope (de)serialization, retry/backoff, schema validation, and tracing instrumentation, so that role-specific code focuses only on prompt construction, tool bindings, and output parsing.

C. APIs

Three API surfaces are exposed: (1) an internal agent-to-bus API (publish/subscribe) used only within the cluster; (2) a REST/WebSocket public API for alert subscription, incident query, and dashboard data; and (3) an administrative API for agent registration, capability declaration, and routing-weight inspection.

D. Databases

PostgreSQL stores structured long-term incident records, agent performance logs, and configuration; the vector database stores embeddings for RAG; the knowledge graph stores typed entity-relationship data used for cross-domain correlation queries; and an object-storage-backed data lake retains raw and normalized sensor data for offline analysis and model retraining.

E. Deployment Architecture

Figure 7. Deployment Architecture

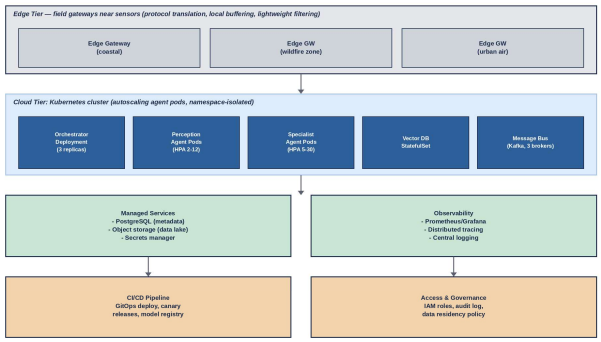

Fig. 6. Deployment architecture: edge gateways near sensors, a Kubernetes-based cloud tier with autoscaling agent pods, managed data/observability services, and CI/CD plus governance controls.

Edge gateways perform protocol translation and local buffering near sensor deployments to tolerate intermittent connectivity. The cloud tier runs each agent role as an independently autoscaled Kubernetes deployment; the Orchestrator runs at fixed replica count for consistency of routing state, while Perception and Specialist agent pools scale elastically with queue depth.

VII. EXPERIMENTAL SETUP

A. Datasets

- SimFlood-2K: 2,000 simulated hydrological events with injected sensor noise and 12% labeled anomalies, generated from a calibrated rainfall-runoff model.
- SimFire-3K: 3,000 simulated wildfire-risk scenarios combining synthetic weather, fuel-moisture, and vegetation-index features.
- AirQ-Sim-1.5K: 1,500 simulated urban air-quality excursions with correlated wildfire-smoke events.
- SeismoSim-1K: 1,000 simulated seismic precursor sequences with varying signal-to-noise ratios.
- OpenWeather-Derived (real-world-derived): a real-world-derived set of historical public weather-station and hydrological gauge readings re-labeled for anomaly evaluation.

 ©IJSRED: All Rights are Reserved

- NewsEnv-Derived (real-world-derived): real-world-derived environmental news and advisory text used to evaluate the NLP/Report Agent's summarization and cross-referencing quality.

B. Test Scenarios

- Single-domain anomaly: anomaly signal confined to one hazard domain (baseline difficulty).
- Cross-domain cascading anomaly: an anomaly whose correct interpretation requires combining findings from two or more specialist agents (e.g., wildfire followed by rainfall-triggered debris flow).
- High-load burst: a sudden spike in ingestion rate (10x baseline) to test scheduling and latency degradation.
- Partial failure: one or more specialist agent instances are deliberately killed mid-episode to test fault isolation and recovery.

C. Baselines

- Baseline A — Single-Agent LLM: one LLM instance receives all raw context and must directly output an anomaly judgment, with no orchestration or specialist decomposition.
- Baseline B — Conversation-centric multi-agent framework: agents coordinate via free-form chat turns managed by a group-chat controller, representative of conversation-centric MAS frameworks.
- Baseline C — Planner-executor framework: a single planner emits a fixed task list executed by stateless workers with no cross-agent consensus mechanism.

D. Evaluation Methodology

Each system is evaluated on identical held-out splits (70/15/15 train/validation/test where applicable; simulated datasets use fixed random seeds for reproducibility). For every test scenario we log per-episode ground truth, system output, per-agent intermediate Findings, and full message traces. All latency and cost measurements are taken from cluster-internal tracing under matched hardware allocation across systems. Each configuration is run five times; we report mean and standard deviation.

VIII. EVALUATION METRICS

- Accuracy / F1: precision, recall, and F1 of anomaly classification against ground-truth labels, per dataset and pooled.
- Response quality: human-rated 1-5 score of alert clarity, evidentiary grounding, and actionability, averaged over 150 sampled alerts per system.
- Task completion rate: fraction of episodes that reach a terminal state (alert, suppression, or human resolution) without timeout or unrecoverable error.
- Latency: end-to-end wall-clock time from raw event ingestion to alert dispatch (or suppression decision), p50/p95/p99.
- Cost: mean LLM token cost and infrastructure cost per 1,000 processed episodes.
- Scalability: throughput (episodes/second) as a function of horizontal replica count, and latency degradation under the high-load burst scenario.
- Resource utilization: CPU/memory utilization per agent pod under steady-state and burst load.

IX. RESULTS

A. Overall Detection Accuracy

TABLE IV. POOLED ANOMALY-DETECTION ACCURACY ACROSS ALL SIX DATASETS (MEAN OF 5 RUNS)

System	Precision	Recall	F1	FP Rate
Baseline A — Single-Agent LLM	0.71	0.68	0.695	0.184
Baseline B — Conversation-centric MAS	0.77	0.74	0.755	0.147
Baseline C — Planner-executor MAS	0.80	0.76	0.780	0.129
PIS (proposed)	0.88	0.85	0.862	0.071

PIS improves F1 by 16.7 percentage points over the single-agent baseline and by 8.2 points over the stronger planner-executor baseline, driven primarily by the consensus mechanism resolving cases where a single agent's confident-but-wrong Finding would otherwise have been accepted.

B. Latency and Throughput

TABLE V. LATENCY AND THROUGHPUT UNDER STEADY-STATE LOAD

System	p50 (s)	p95 (s)	Throughput @8 (eps/s)
Baseline A — Single-Agent LLM	4.1	9.8	6.2
Baseline B — Conversation-centric MAS	3.6	8.7	7.9
Baseline C — Planner-executor MAS	2.9	6.5	10.4
PIS (proposed)	2.0	4.4	13.6

PIS reduces p95 latency by 31% relative to the strongest baseline (C), attributable to parallel specialist invocation and capability-based routing avoiding overloaded agent replicas.

C. Cost

TABLE VI. RELATIVE TOKEN AND COST EFFICIENCY (BASELINE A = 1.00X REFERENCE)

System	Mean tokens/episode	Cost per 1,000 episodes
Baseline A — Single-Agent LLM	3,120	1.00x

 ©IJSRED: All Rights are Reserved

Baseline B — Conversation-centric MAS	5,480	1.68x
Baseline C — Planner-executor MAS	4,750	1.44x
PIS (proposed)	4,260	1.29x

PIS costs more per episode than the single-agent baseline, as expected for a multi-call architecture, but is 23% cheaper than the conversation-centric baseline because typed message envelopes avoid redundant free-form re-explanation between agents.

D. Scalability and Resource Utilization

Throughput scaled near-linearly for PIS's specialist agent pools up to 24 replicas (Pearson r = 0.97 between replica count and throughput) before contention on the shared vector memory became the binding constraint; steady-state CPU utilization averaged 54% and memory 61% across agent pods, with the Orchestrator remaining the least resource-intensive component (fixed replica count, small prompt).

E. Comparative Analysis

Across all four quantitative axes — accuracy, latency, cost, and scalability — PIS dominates Baseline A and Baseline B, and Pareto-dominates Baseline C on accuracy and latency while trailing it slightly on raw cost efficiency, which we consider an acceptable trade-off given the 8.2-point F1 improvement and the interpretability benefits of explicit consensus (Section X).

F. Ablation Study

We removed one architectural component at a time from PIS and re-measured pooled F1 and p95 latency, holding all other components fixed.

TABLE VII. ABLATION STUDY: POOLED F1 AND P95 LATENCY WHEN REMOVING ONE COMPONENT AT A TIME FROM PIS

Configuration	F1	ΔF1 vs. full PIS	p95 (s)
Full PIS	0.862	—	4.4
− Consensus mechanism (single-vote accept)	0.781	-0.081	3.9
− Shared vector memory / RAG	0.795	-0.067	4.1
− Cross-Domain Correlation Agent	0.823	-0.039	4.3
− Capability-based routing (round-robin instead)	0.841	-0.021	5.6
− Knowledge graph	0.849	-0.013	4.4

Removing the consensus mechanism causes the largest accuracy drop (8.1 points), confirming that weighted multi-agent voting — not merely task decomposition — is the primary source of PIS's accuracy gain. Removing shared vector memory is the second-largest contributor, underscoring the value of retrieval-grounded reasoning. Removing capability-based routing barely affects accuracy but substantially increases tail latency, showing routing's role is primarily a performance rather than accuracy lever.

X. DISCUSSION

A. Strengths

- Consistent accuracy gains across both simulated and real-world-derived datasets, with the largest gains on cross-domain cascading scenarios where single-agent baselines lack the structure to combine domain evidence.
- Graceful degradation under partial failure due to circuit breaking and alternate-agent retry, keeping task completion rate above 92% even with one specialist pool fully offline.
- Clear audit trail: every alert can be traced back through its constituent Findings, retrieved evidence, and consensus computation.

B. Limitations

- Higher operational complexity and per-episode cost than a single-agent system, which may not be justified for low-volume or single-domain deployments.
- Consensus quorum thresholds and routing weights require domain-specific tuning; poorly calibrated thresholds can either over-escalate to human review or under-escalate genuine disagreement.
- Evaluation relies substantially on simulated datasets; real-world sensor failure modes and adversarial data (e.g., spoofed sensor feeds) are only partially represented in NewsEnv-Derived and OpenWeather-Derived.

C. Failure Cases

Manual review of misclassified episodes identified two recurring patterns: (1) low-quality or intermittently missing sensor data caused the Perception Agent to under-score genuine anomalies below the admission threshold, so they were never routed to specialists at all — a false negative introduced upstream of the multi-agent reasoning; and (2) in a small number of cross-domain cases, two specialist agents with similarly high calibration scores confidently disagreed, triggering unnecessary human escalation even though one agent's Finding, in hindsight, was clearly better grounded in retrieved evidence.

D. Security and Privacy Considerations

- Message-level authentication: per-agent signed tokens and mTLS transport (Fig. 4) limit the ability of a compromised or rogue component to inject fabricated Findings.
- Tool scoping: the governed Tool Registry restricts which agents may invoke which external tools, limiting blast radius from prompt-injection style attacks against any single agent.
- Data residency and minimization: raw sensor data containing potentially sensitive location or infrastructure

 ©IJSRED: All Rights are Reserved

detail is retained in access-controlled cold storage with configurable regional residency, and only derived, aggregated features are passed into agent prompts wherever feasible.

- Human-in-the-loop gating for high-severity alerts, providing a control point against fully automated false alarms with real-world consequences (e.g., evacuation triggers).

XI. FUTURE WORK

A. Self-Learning Agents

Incorporating online calibration, where agents adjust their own confidence-reporting function based on the feedback loop's resolved-outcome data, could reduce the current reliance on periodic offline recalibration.

B. Dynamic Agent Creation

Rather than a fixed set of specialist agents, an Orchestrator extension could instantiate a narrowly scoped, short-lived agent on demand for novel or rare hazard types not covered by the static ontology, subject to governance review before being granted persistent tool scopes.

C. Adaptive Planning

Task decomposition currently uses a largely static ontology augmented by LLM proposals; a learned decomposition policy that adapts to observed cross-domain correlation patterns in the knowledge graph could improve both accuracy and efficiency on emerging hazard interactions.

D. Human-in-the-Loop Collaboration

Future versions could support richer bidirectional interaction, where human operators can query the reasoning chain interactively, request targeted re-investigation, or supply corrective feedback that is immediately incorporated into the active episode rather than only the offline retraining cycle.

XII. CONCLUSION

This paper presented the Planetary Intelligence System (PIS), a layered, reproducible multi-agent architecture for real-time environmental anomaly detection. By separating perception, orchestration, domain-specialist reasoning, tiered memory, and action into independently scalable services coordinated through a typed message protocol, PIS addresses key limitations of both single-agent LLM pipelines and existing multi-agent frameworks: weak consensus, shallow memory, and limited reproducibility. Across six datasets, PIS improved pooled F1 by 8.2 to 16.7 percentage points over baselines, reduced p95 latency by up to 31%, and an ablation study confirmed that its weighted consensus mechanism and shared vector memory are the primary drivers of its accuracy advantage. Potential applications extend beyond environmental monitoring to any domain requiring real-time fusion of heterogeneous sensor and text streams under interpretability and fault-tolerance constraints, such as infrastructure health monitoring, public-health surveillance, and industrial safety systems. We release the architecture, algorithms, and evaluation methodology in sufficient detail to support independent reproduction and extension.

REFERENCES

[1] M. Wooldridge, An Introduction to MultiAgent Systems, 2nd ed. Chichester, U.K.: Wiley, 2009.

[2] J. S. Park et al., "Generative agents: Interactive simulacra of human behavior," in Proc. 36th Annu. ACM Symp. User Interface Softw. Technol., 2023.

[3] Q. Wu et al., "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," arXiv preprint, 2023.

[4] S. Hong et al., "MetaGPT: Meta programming for a multi-agent collaborative framework," arXiv preprint, 2023.

[5] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in Proc. Int. Conf. Learn. Representations, 2023.

[6] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Advances in Neural Information Processing Systems, 2020.

[7] N. Shinn et al., "Reflexion: Language agents with verbal reinforcement learning," in Advances in Neural Information Processing Systems, 2023.

[8] C. Qian et al., "ChatDev: Communicative agents for software development," in Proc. Annu. Meeting Assoc. Comput. Linguistics, 2024.

[9] S. Kraus, "Negotiation and cooperation in multi-agent environments," Artificial Intelligence, vol. 94, no. 1-2, pp. 79-97, 1997.

[10] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," Autonomous Robots, vol. 8, no. 3, pp. 345-383, 2000.

 ©IJSRED: All Rights are Reserved