From Black Box to Data Contract: Engineering Accountable AI Agents with UpFormat

Karthik Gomadam Rajagopal, Pablo Mendes and Andrew Rabinovich

Upwork Inc., Palo Alto, CA, USA.

Abstract

Modern multi-agent systems are built on improvisation. Large language model (LLM) agents reason flexibly but communicate opaquely: they make decisions without explaining intent, escalate ambiguously when stuck, and lose context across interactions. Current orchestration frameworks attempt to resolve this by using LLMs to interpret other LLMs' outputs, creating uncertainty cascades where each interpretive layer compounds drift and error.

We argue that reliable hybrid-intelligence systems require a protocol layer that separates coordination from content. We present UpFormat, a communication protocol that treats agent interactions as structured data contracts rather than unstructured text exchanges. UpFormat introduces explicit coordination signals that make communication transparent, traceable, and auditable, while preserving the generative flexibility of LLMs for semantic reasoning. Through lightweight semantic metadata, UpFormat enables interoperability with knowledge graphs and symbolic reasoning systems without mandatory ontology commitment.

Through structured reasoning, targeted feedback, and semantic hooks, UpFormat turns agent collaboration into an engineering problem rather than an interpretation challenge. We show how UpFormat brings deterministic structure to agent collaboration, enabling explainable and accountable AI systems.

1. Introduction

Modern multi-agent systems hold significant promise for automating complex enterprise workflows, yet most are built on improvisation. Frameworks such as LangChain [1], CrewAI [2], and AutoGen [3] treat communication as unstructured text, leaving coordination logic as an implementation detail for developers. Each framework allows developers to design their own state representations, error signals, and coordination patterns—an approach that works within single deployments but fragments the ecosystem. Agents built for one framework cannot easily interoperate with another, and orchestration logic becomes tightly coupled to specific message formats.

Large language model (LLM) agents reason flexibly and adapt to new situations, but they communicate opaquely: their operational state is embedded in conversational responses, forcing orchestrators to infer whether agents can proceed from natural language. This lack of structure creates a fundamental reliability problem. When an agent expresses uncertainty or asks a question, the system cannot distinguish between different operational states. A model might say "I think this might need escalation" (blocked), "I'd suggest considering escalation" (optional follow-up), or "I've completed this, should I escalate?" (done, awaiting next steps). Without explicit state signals, orchestration systems must interpret nuance instead of reading state directly.

Existing frameworks attempt to fix this by using one LLM to interpret another's output—employing language models to decide whether peers are "stuck," "uncertain," or "complete." But LLMs are inherently non-deterministic [4], creating *uncertainty cascades* where each interpretive layer compounds drift and error [5]. As workflows scale, these cascades amplify [6], producing unstable control flow that heuristics cannot prevent [7]. Enterprise deployments reflect this fragility: studies of AI initiative failures frequently cite coordination reliability as a barrier to production adoption [8].

karthikgomadam@upwork.com (K. G. Rajagopal); pablomendes@upwork.com (P. Mendes); andrewrabinovich@upwork.com (A. Rabinovich)

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2025-11-13

We argue that reliable hybrid-intelligence systems require communication protocols that separate coordination from content. We present **UpFormat**, a protocol that makes agent operational state explicit through structured fields rather than conversational text. UpFormat does not remove LLM non-determinism—the same prompt may yield different confidence scores or reasoning—but it ensures that operational state is declared in parseable form, enabling both deterministic orchestration and explainable decision-making.

By enforcing explicit state representation, UpFormat transforms opaque message passing into transparent, debuggable data exchange. It turns agent collaboration from an interpretive exercise into an engineering discipline, enabling accountable orchestration across heterogeneous agents.

The remainder of this paper presents the UpFormat protocol (Section 2), demonstrates its application (Section 3), and discusses implications for accountable AI systems (Section 4).

2. UpFormat Protocol

UpFormat separates *coordination semantics* from *data semantics* in agent interactions. Rather than embedding operational state inside payloads—which may be natural language, JSON, or other ad hoc formats—agents communicate through structured messages that declare state fields explicitly. This abstraction decouples the *coordination layer*, used for orchestration, from the *payload layer*, used for content exchange, allowing systems to reason about coordination independently of payload representation.

Any participant in an agentic workflow—LLM-based or tool—can produce and consume UpFormat messages. An orchestration agent may signal `state: needsHumanDecision` when requirements are ambiguous, while a tool may use the same state after authentication failure. UpFormat thus provides a uniform, interpretable coordination layer across heterogeneous components while allowing payload formats to vary freely.

2.1. Core Message Fields

Each UpFormat message includes fields for identity, lineage, and operational state:

- `messageId`, `threadId`, `parentMessageId`: traceable message lineage;
- `state`: operational state (`submitted`, `waiting`, `completed`, `failed`, `needsHumanDecision`, `followup`);
- `timestamp`: message creation time;
- `payload`: structured data or content being communicated;
- `explanation`: natural-language context for the state.

These fields allow orchestrators to interpret progress and escalation consistently across agents. When an agent signals `needsHumanDecision`, the orchestrator can route to human review or intervene without secondary model interpretation. The `explanation` field provides rationale, while lineage fields support transparent, auditable workflows.

2.2. Extensions

UpFormat defines optional extensions that add structure to reasoning, feedback, and semantics.

upThought – Structured Reasoning. Captures an agent's reasoning process, questions, confidence (0–1), and alternative approaches. It distinguishes between genuine blocking states and exploratory clarifications, reducing ambiguity at its source.

upFeedback – Targeted Responses. Structures human or agent feedback through explicit `type` (`edit`, `approve`, `rank`, `comment`, `instruction`), `target` references, and metadata (severity, priority). Feedback may include text edits, annotations, or diffs—enabling precise, machine-readable revision cycles rather than unstructured comments.

upContext – Semantic Metadata. Provides lightweight, self-describing metadata within the payload layer. Rather than enforcing a shared ontology, `upContext` attaches semantic hints automatically derived from structured model definitions (e.g., Pydantic or JSON Schema). Each context may include:

- `entity`: brief description of represented content;
- `fields`: field-level semantic descriptions;
- `concepts`: tags for categorization or routing;
- `constraints`: validation hints (ranges, requirements);
- `schema`: optional JSON Schema reference.

This design promotes interoperability among agents exchanging heterogeneous payloads. For example, one agent may send analysis results as a JSON object while another expects tabular input; the attached `upContext` describes field meanings and constraints, allowing receivers to map data correctly without brittle name-based inference. Orchestrators rely on the coordination layer to manage workflow logic, while agents use the payload layer—augmented with context—for semantic alignment.

By separating coordination from content and embedding minimal semantics, UpFormat turns agent communication into structured, interpretable data exchange. The coordination layer governs interaction flow, while the payload layer carries self-described content that remains adaptable to domain and agent variability. This division reduces ambiguity cascades while preserving the generative flexibility that makes LLM-based agents valuable.

We next illustrate how UpFormat's dual-layer structure enables transparent collaboration and explainable decision-making in a hybrid agent workflow.

2.3. Implementation Approach

In practice, agents implement UpFormat through structured output validation and deterministic state mapping. **Coordination states are set by agent code, not by LLMs.** The LLM generates structured outputs; the agent deterministically maps these outputs to states.

When an LLM generates a response, agent code validates it against a schema (e.g., Pydantic). Consider a freelancer search agent: the LLM outputs `{canProceed: true, confidence: 0.45, candidates: [...]}`. The agent's deterministic logic examines these fields: if `confidence < 0.5`, it sets `state: needsHumanDecision` regardless of the LLM's `canProceed` claim. If `canProceed=false`, the state becomes `needsHumanDecision` with the LLM's explanation attached. Only when `canProceed=true` and confidence exceeds the threshold does the agent set `state: completed`.

If validation fails—the LLM omits required fields or produces malformed JSON—the agent prompts: "Either fix the validation errors, or explain why human intervention is needed." Using structured output formats (e.g., OpenAI structured output), the LLM responds with an enumerated choice: `fixed`, `needHuman`, or `beyondCapability`. The agent deterministically maps these: `fixed` triggers re-validation; `needHuman` becomes `state: needsHumanDecision`; `beyondCapability` becomes `state: failed`. In both validation paths, the LLM provides explanations and reasoning, but state assignment is purely deterministic agent logic operating on structured LLM outputs.

This hybrid architecture channels LLM flexibility through explicit checkpoints. The same prompt may yield `confidence: 0.45` in one run and `0.52` in another, but the agent's threshold-based logic ensures consistent routing: low-confidence outputs always escalate to humans. By making the

LLM declare `canProceed` and `confidence` explicitly rather than embedding uncertainty in conversational hedging, UpFormat transforms coordination from an interpretation problem into a deterministic mapping problem.

2.4. Semantic Interoperability

UpFormat's upContext extension bridges pragmatic self-description and formal semantics. While avoiding mandatory ontology commitment, upContext enables optional semantic enrichment that supports integration with knowledge graphs, symbolic reasoning systems, and existing Semantic Web infrastructure.

Concept Linking: Tags in upContext.concepts may reference formal ontologies (Schema.org, Wikidata, domain ontologies) when semantic precision or cross-system integration is required. An agent describing freelancer candidates might use concepts: ["schema:Person", "wikidata:Q5", "software_engineer"], mixing formal terms with domain vocabulary. Fields can reference RDF predicates, constraints can encode validation rules, and the JSON structure maps to JSON-LD contexts for systems requiring RDF serialization.

Hybrid Reasoning: The separation between LLM-based agents and symbolic reasoners is deliberate. UpFormat messages serve as structured interfaces: LLMs produce natural language explanations, confidence scores, and content within the payload; symbolic systems consume the coordination layer (state, upThought) and upContext annotations to apply formal logic. A compliance system might validate upContext.constraints against SHACL shapes, or a policy engine might evaluate upThought.confidence thresholds using rule-based decision trees. The LLM does not perform symbolic reasoning—it declares what it knows and how confident it is, leaving deductive inference to specialized systems designed for that purpose.

This design philosophy—"semantic hooks without semantic mandates"—allows UpFormat to span the neural-symbolic divide without forcing either paradigm into the other's computational model. LLMs excel at content generation and pattern recognition; symbolic systems excel at formal validation and logical inference. UpFormat provides the structured communication layer that allows each to operate in its domain of strength while coordinating effectively. The protocol supports a spectrum of integration: from purely neural systems using free-text tags to tightly coupled hybrid systems where every upContext field references a formal ontology and external reasoners validate semantic consistency.

3. Illustrative Use Case: Human-in-the-Loop Shortlisting

We demonstrate UpFormat in a hybrid hiring workflow involving an **Orchestrator**, a **Freelancer Recommender Agent**, an **Email Tool**, and a **Human** reviewer. The scenario highlights explicit state coordination and payload mediation through `upContext`.

3.1. Scenario

The Orchestrator requests freelancer recommendations. The Recommender detects a missing location and signals `state: needsHumanDecision`, with `upContext` describing the required field. The Orchestrator routes the request to the Human, who provides the location via `upFeedback`. The Orchestrator forwards this feedback to the Recommender.

The Recommender returns a ranked candidate list with semantic metadata. The Orchestrator then signals `state: followup` to the Human, offering the option to select specific candidates; if no response is received, it defaults to inviting the top two. Finally, the Orchestrator invokes the Email Tool to send invitations. Because the Email Tool expects a different schema, the Orchestrator uses `upContext` from both agents to map fields (e.g., `candidates[].email` → `recipients[]`), avoiding brittle, hardcoded conversions.

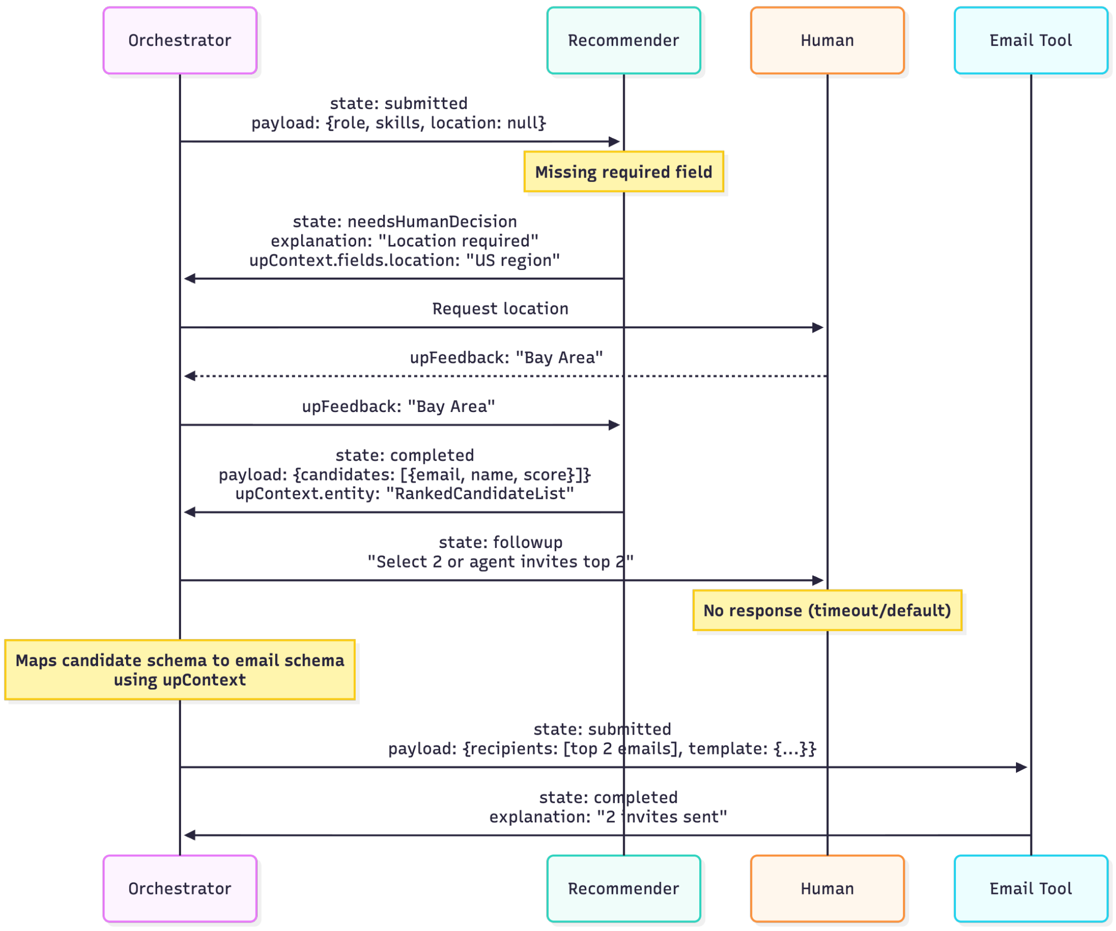
Orchestrator
Recommender
Human
Email Tool
state: submitted
payload: {role, skills, location: null}
Missing required field
state: needsHumanDecision
explanation: "Location required"
upContext.fields.location: "US region"
Request location
upFeedback: "Bay Area"
upFeedback: "Bay Area"
state: completed
payload: {candidates: [{email, name, score}]}
upContext.entity: "RankedCandidateList"
state: followup
"Select 2 or agent invites top 2"
No response (timeout/default)
Maps candidate schema to email schema using upContext
state: submitted
payload: {recipients: [top 2 emails], template: {...}}
state: completed
explanation: "2 invites sent"
Orchestrator
Recommender
Human
Email Tool

Figure 1: UpFormat coordination in a human-in-the-loop hiring workflow. The Orchestrator uses explicit states for routing and `upContext` for schema mediation between the Recommender and Email Tool. The `followup` state enables optional human input with a default fallback.

3.2. Key Observations

Coordination remains explicit through operational state signals. When the Recommender signals `needsHumanDecision`, the Orchestrator routes to the Human for required input; when the Orchestrator signals `followup`, it indicates an optional refinement step with a default fallback.

The `upContext` extension enables semantic interoperability. The Recommender's output may include `fields.email: "contact address"`, while the Email Tool expects `fields.recipients: "email addresses"`. The Orchestrator uses these semantic descriptions to align payloads without name-based inference. Together, explicit state transitions and self-describing payloads make coordination transparent, extensible, and resilient to schema variation across agents.

4. Related Work

UpFormat builds on decades of agent communication research, adapting classical approaches for the LLM era. The challenge of coordinating autonomous agents through structured communication has deep roots in distributed AI, and UpFormat inherits key insights while addressing new requirements introduced by neural language models.

Early Agent Communication Languages: KQML (Knowledge Query and Manipulation Language), developed in the early 1990s as part of the DARPA Knowledge Sharing Effort, pioneered the use of performatives—speech acts defining permissible agent operations [9]. Building on KQML, FIPA-ACL

(Agent Communication Language) emerged as a standardized approach with formal speech-act semantics, defining message parameters including performatives, content, and ontology references [10]. Both protocols established the principle that agent coordination requires explicit communication primitives rather than implicit inference from unstructured messages. UpFormat inherits this architectural insight: operational state should be declared, not inferred. However, KQML and FIPA-ACL rely on shared ontologies for semantic interoperability. This can be a barrier when agents do not agree on or have access to the shared vocabulary. UpFormat relaxes this requirement through upContext's lightweight semantic hooks that leverage LLMs for functional natural language annotations, while allowing opportunistic ontology/URI referencing.

Semantic Web Services: SAWSDL (Semantic Annotations for WSDL and XML Schema) defines extension attributes for adding semantic annotations to web service descriptions, allowing concepts from external semantic models to be referenced within WSDL components [11]. OWL-S provides an ontology within the OWL framework for describing semantic web services, enabling automatic discovery, invocation, and composition through service profiles, process models, and groundings [12]. These approaches demonstrated how semantic metadata could augment service descriptions without requiring complete ontological commitment at every layer.

UpFormat's upContext draws directly from this lineage: like SAWSDL, it attaches semantic annotations to structured definitions (here, agent message schemas rather than WSDL); like OWL-S, it enables discovery and composition through self-describing metadata. The key adaptation for LLM agents is automation: upContext derives semantics from model definitions (Pydantic, JSON Schema) rather than requiring manual SAWSDL annotation, reducing the engineering overhead that limited Semantic Web service adoption.

Contemporary Agent Protocols: Recently, the Agent2Agent (A2A) Protocol by Google and supported by industry partners, addresses agent-to-agent communication through standardized HTTP-based interactions, agent cards for capability discovery, and task-oriented collaboration [13]. A2A focuses on cross-platform agent interoperability and treats agents as opaque services that can delegate tasks without revealing internal implementations. Similarly, Anthropic's Model Context Protocol (MCP) provides standardized interfaces for LLM-tool integration, enabling LLMs to interact with external data sources and services [14].

UpFormat complements these protocols by addressing a different architectural layer. While A2A and MCP define how agents discover and delegate to each other or access external resources (the "outer loop" of task management), UpFormat specifies how agents structure their operational state and payload semantics within messages (the "inner loop" of coordination). A2A's task delegation or MCP's tool invocations could carry UpFormat messages: requests might contain UpFormat's state and upThought fields to make agent reasoning explicit during task execution. Where A2A and MCP optimize for cross-platform interoperability, UpFormat optimizes for transparency and accountability within orchestrated workflows.

UpFormat's Distinctive Position: To our knowledge, no prior protocol explicitly separates agent coordination semantics from payload semantics while remaining both ontology-agnostic and LLM-compatible. KQML and FIPA-ACL require ontology commitment; SAWSDL and OWL-S target web services rather than conversational agents; A2A and MCP focus on task delegation and tool access rather than explicit operational state. UpFormat fills this gap by making coordination state (needsHumanDecision, followup, completed) a first-class protocol concern while keeping semantic annotation lightweight and optional, enabling deterministic orchestration of non-deterministic LLM agents.

5. Conclusion and Future Work

We presented UpFormat, a structured communication protocol that separates coordination semantics from content generation in multi-agent systems. By requiring agents to declare operational state explicitly through structured fields, UpFormat enables deterministic orchestration even when LLM-generated content varies. The protocol's separation of coordination layer (explicit states like `needsHumanDecision`, `followup`, `completed`) from payload layer (self-describing data with `upContext`) eliminates the need to use LLMs to interpret other LLMs' outputs, addressing the uncertainty cascade problem in current multi-agent frameworks.

5.1. Future Work

Several directions merit exploration: (1) establishing UpFormat as an open standard with reference implementations and framework integrations (LangChain, AutoGen, CrewAI); (2) extending to federated settings where agents from different organizations collaborate; (3) integration with formal ontologies and knowledge graphs for richer semantic reasoning; and (4) empirical evaluation comparing UpFormat-based systems against traditional approaches on metrics like escalation precision, coordination failures, and system debuggability. As multi-agent systems move from prototypes to production, explicit communication protocols become essential infrastructure for building accountable, debuggable, and reliable AI collaboration.

Declaration on Generative AI

In preparing this manuscript. the authors used Claude (Anthropic) for assisting in drafting portions of the paper text and for providing editorial suggestions. The authors take full responsibility for the content including content informed or generated by Claude and have thoroughly reviewed, verified, and edited the content of the manuscript.

References

[1] LangChain Inc., Langchain, https://github.com/langchain-ai/langchain, 2022.
[2] crewAI Inc., Crewai: Framework for orchestrating role-playing, autonomous ai agents, https://github.com/crewAIInc/crewAI, 2023.
[3] Q. Wu, G. Bansal, J. Zhang, et al., Autogen: Enabling next-gen llm applications via multi-agent conversation, https://github.com/microsoft/autogen, 2023. ArXiv:2308.08155.
[4] Non-determinism of "deterministic" llm settings, arXiv preprint arXiv:2408.04667v5 (2025). URL: https://arxiv.org/html/2408.04667v5.
[5] C. Greyling, Llm drift, prompt drift, chaining & cascading, Medium (2023). URL: https://cobusgreyling.medium.com/llm-drift-prompt-drift-chaining-cascading-fa8fbf67c0fd.
[6] Position: Towards a responsible llm-empowered multi-agent systems, arXiv preprint arXiv:2502.01714v1 (2025). URL: https://arxiv.org/html/2502.01714v1.
[7] M. Cemri, et al., Why do multi-agent llm systems fail?, arXiv preprint arXiv:2503.13657 (2025). URL: https://arxiv.org/abs/2503.13657.
[8] MIT Media Lab NANDA Initiative, The GenAI Divide: State of AI in Business 2025, Technical Report, Massachusetts Institute of Technology, 2025. Based on 300+ AI initiative analyses, 52 executive interviews, and 153 leader surveys.
[9] T. Finin, R. Fritzson, D. McKay, R. McEntire, KQML as an agent communication language, in: Proceedings of the Third International Conference on Information and Knowledge Management, ACM, 1994, pp. 456–463. doi:10.1145/191246.191322.
[10] Foundation for Intelligent Physical Agents, FIPA ACL Message Structure Specification, Technical Report SC00061G, FIPA, 2002. URL: http://www.fipa.org/specs/fipa00061/.
[11] J. Kopecký, T. Vitvar, C. Bournez, J. Farrell, Semantic Annotations for WSDL and XML Schema, W3C Recommendation, W3C, 2007. URL: https://www.w3.org/TR/sawsdl/.
[12] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, K. Sycara, N. Srinivasan, OWL-S: Semantic markup for web services, 2004. URL: https://www.w3.org/Submission/OWL-S/, w3C Member Submission.

[13] Google Cloud and A2A Partners, Agent2agent (A2A) protocol specification, 2025. URL: https://a2aprotocol.ai, open protocol for agent-to-agent communication.
[14] Anthropic, Model context protocol, 2024. URL: https://modelcontextprotocol.io, standardized protocol for LLM-tool integration.

A. Appendix: UpFormat Specification Reference

This appendix provides a reference for UpFormat's core message structure and operational states. UpFormat is under active development, and the specification continues to evolve based on implementation experience and community feedback.

Field	Description	Type/Values
`messageId`	Unique message identifier	UUID
`threadId`	Conversation thread identifier	UUID
`parentMessageId`	Reference to parent message	UUID (optional)
`timestamp`	Message creation time	ISO 8601 datetime
`state`	Operational state (see below)	Enum
`payload`	Domain-specific data	JSON object
`explanation`	Natural language context	String
`agentId`	Sending agent identifier	String (optional)

Table 1
Core UpFormat message fields

Operational States	
`submitted`	Task submitted for processing
`waiting`	Waiting for external dependency
`completed`	Task successfully completed
`failed`	Task failed (unrecoverable error)
`needsHumanDecision`	Requires human input to proceed (blocking)
`followup`	Task complete, optional refinement available (non-blocking)
`cancelled`	Task cancelled before completion
Optional Extensions	
`upThought`	Structured reasoning: `reasoning` (array), `questions` (array), `confidence` (0-1), `canProceed` (boolean), `alternatives` (array), `assumptions` (array)
`upFeedback`	Targeted feedback: `type` (edit/approve/reject/rank/comment/instruction), `target` (message/part reference), `content` (text/edits/annotations), `metadata` (severity, priority)
`upContext`	Semantic metadata: `entity` (description), `fields` (field descriptions), `concepts` (tags), `constraints` (validation hints), `schema` (JSON Schema reference)

Table 2
UpFormat operational states and optional extensions

From State	**To State**	**Trigger Condition**
`submitted`	`needsHumanDecision`	Missing required information
`submitted`	`completed`	Task finished successfully
`submitted`	`waiting`	External dependency required
`needsHumanDecision`	`submitted`	Human provides input
`completed`	`followup`	Optional refinement offered
`followup`	`submitted`	Human requests changes

Table 3
Common state transitions