OmnisensAI

TCP/AP

Category: Standards Track (Draft)

Version 0.1

Status: Draft · Request for Comments

2026

Trusted Cognition Protocol / Agentic Protocol (TCP/AP)

A specification for Deterministic AI: deterministic interpretation of probabilistic model output.

> **Abstract.** TCP/AP is an open protocol that places a deterministic checkpoint between machine cognition and execution. A probabilistic model produces an **Artifact**; the protocol evaluates that Artifact against a declared **Agentic Protocol** and resolves it to exactly one admissible decision, or to an explicit failure, before any action is taken. Resolution is deterministic and independent of the model, sampling parameters, runtime, and environment. This document defines the objects, the resolution semantics, the wire protocol, the verification model, the runtime, and the requirements an implementation must meet to be considered conformant.

Status of This Document

This is a working draft (v0.1) published for comment. It is not final; identifiers, fields, and failure codes may change. Send feedback via contact or the project repository.

Contents

---

1. Introduction

Large language models exhibit output variance even under deterministic decoding (temperature 0). This variance arises not primarily from sampling but from **interpretation drift**: when a single input admits multiple valid interpretations, identical inputs yield different outputs across runs and models. When such outputs drive actions, non-determinism becomes operational risk.

TCP/AP does not attempt to make models deterministic. It makes **decisions** deterministic. Just as TCP/IP standardized the transport of data over unreliable networks, TCP/AP standardizes the interpretation of meaning over probabilistic systems, so that identical evidence resolves to identical decisions regardless of which model produced it.

---

2. Terminology

The key words **MUST**, **MUST NOT**, **SHOULD**, **SHOULD NOT**, and **MAY** in this document are to be interpreted as described in RFC 2119.

- **Artifact**: a structured output submitted for evaluation (e.g. an LLM tool call, an API response, a function result). Only the Artifact is evaluated; the producing model is not part of the decision contract.
- **Agentic Protocol**: the content-hashed decision contract: the constraints an Artifact must satisfy, the set of admissible labels, and the rules that resolve an Artifact to a label. Held as data, versioned, model-independent.
- **Kernel**: the deterministic, non-LLM component that evaluates an Artifact against an Agentic Protocol and returns a verdict. Stateless with respect to the decision; the sole locus of resolution.
- **Conformance Event**: the signed, self-contained record of a resolution, verifiable offline against a pinned root.
- **Objective**: a unit of work (Python or LLM) that produces an Artifact and emits an outcome. It decides nothing.
- **Switchboard**: the routing table, expressed as data, mapping `(objective, status, label)` to the next objective.
- **Operator**: the control loop that runs objectives and connects them per the Switchboard. It holds no policy.
- **admissible**: an Artifact is admissible under an Agentic Protocol when it satisfies the constraints and resolves to a single decision.

---

3. Architecture

TCP/AP separates **Cognition** from **Action**.

- **Cognition (probabilistic).** Models read, reason, and interpret unstructured input without constraint, and emit an Artifact.
- **Action (deterministic).** Before an Artifact drives an action it is evaluated by the Kernel against a declared Agentic Protocol. Admissible Artifacts resolve to a single authorized decision; non-admissible Artifacts are rejected before any side effect.

An application is a loop of objectives connected by an Operator over a Switchboard:

```
# the governed loop
Objective → Artifact → Kernel → (conformant + label) → Switchboard → Operator → next Objective
                              → (non-conformant)      → explicit failure → blocked
```

---

4. Objects

4.1 Artifact

An Artifact is any structured value. The Kernel evaluates the Artifact only; it **MUST NOT** consider the identity of the producing model. Two models that emit equivalent Artifacts **MUST** resolve to the same decision under the same Agentic Protocol.

4.2 Agentic Protocol

An Agentic Protocol is a JSON object with a `slug`, a `version`, a set of `constraints` (JSON Schema), a list of `labels` (the result vocabulary), and the `rules` that declare which label applies. The bundle is content-hashed; its identity is independent of any model. Constraints **SHOULD** set `additionalProperties: false` so that undeclared fields render an Artifact non-conformant.

4.3 Conformance Event

A Conformance Event binds the Agentic Protocol hash, the Artifact hash, the generation metadata hash, the resolved decision, and the signer. It is self-contained: any relying party **MUST** be able to verify it offline against a pinned root, without access to the originating model or runtime.

---

5. Decision Resolution

The Kernel validates an Artifact `A` against an Agentic Protocol `P` and returns a verdict. Validation **MUST** be deterministic: the same `(A, P)` always yields the same verdict, independent of model, sampling parameters, runtime, or environment.

A verdict is exactly one of three outcomes:

- **Conformant** (HTTP 200). `A` is **admissible** under `P`; the Kernel returns a single decision label drawn from `P.labels`.
- `INVALID_REQUEST` (HTTP 400). `A` did not satisfy `P.constraints`.
- `INTERPRETATION_DRIFT` (HTTP 422). `A` is well-formed but did not resolve to a single admissible decision under `P`.

A conforming Kernel **MUST NOT** emit a label that is not declared in `P.labels`, and **MUST NOT** produce content of its own.

How the Kernel reaches a verdict (the evaluation of `P`, any precedence among candidate decisions, and the handling of ambiguous or conflicting matches) is **implementation-defined and out of scope for this document**. The Kernel is a black box: to a client it returns only the verdict (a status, and a label when conformant), never the internal reasoning that produced it. A conforming Kernel **MUST NOT** expose that reasoning in its response.

---

6. Wire Protocol

A hosted Kernel exposes a single evaluation endpoint. A conforming implementation **MUST** accept a request bearing the Artifact, the Agentic Protocol (or its hash), and a generation-metadata hash, and **MUST** return the unified envelope below.

```
POST /v1/validate
{
  "artifact": { "name": "issue_refund", "arguments": { "amount": 20 } },
  "agentic_protocol": { /* declared { constraints, labels, rules } or its hash */ },
  "generation_metadata_hash": "sha256:4b81…"
}
```

Responses share one envelope; the HTTP status encodes the class of result.

Status	failure_code	Meaning
200	(none)	Conformant. `data.conformant = true`, `data.label` is the resolved decision.
400	INVALID_REQUEST	The Artifact failed the constraints (malformed or undeclared fields).
422	INTERPRETATION_DRIFT	The Artifact is valid but did not resolve to a single admissible label.

```
< 200 OK
{ "status": "EVALUATED", "data": { "conformant": true, "label": "AUTO_APPROVE" }, "error": null, "attestati

< 422 Unprocessable Entity
{ "status": "EVALUATED", "data": { "conformant": false, "label": null },
  "error": { "failure_code": "INTERPRETATION_DRIFT", "message": "the artifact did not resolve to a single a
```

The response is deliberately coarse: it carries only the verdict class and a generic reason (for example, `"artifact is not an object"`). It **MUST NOT** expose the Kernel's internal reasoning. Any finer detail **MAY** be retained in a private, server-side audit record, but **MUST NOT** be returned to the client.

---

7. Verification

The Kernel is the sole holder of the signing key and signs every conformant resolution as a Conformance Event. A client **MUST NOT** hold a key capable of minting events that chain to the trust root: a client that could sign could forge verdicts.

A relying party **MUST** be able to verify an event offline: confirm that the event is signed by an issuing key, that the issuing key is authorized by a certificate signed by the pinned root, and that the bound hashes match the Artifact and Agentic Protocol under evaluation. Verification **MUST NOT** require a call back to the Kernel.

---

8. Runtime

A runtime composes objectives into an application. The Operator **MUST NOT** encode decisions; it runs the current objective, obtains an outcome, and connects to the next objective named by the Switchboard for `(objective, status, label)`.

Routing **SHOULD** be total and deny-by-default: any `(objective, status, label)` not explicitly mapped **MUST** route to a safe terminal (e.g. human review) rather than proceed. Only decisions verified as admissible **MAY** drive state transitions; non-conformant, ambiguous, or invalid outcomes **MUST** fail explicitly before any side effect.

---

9. Conformance

An implementation is a **conforming Kernel** if and only if it satisfies all of the following:

- C-1. `resolve(A, P)` is deterministic and independent of model, sampling, runtime, and environment.
- C-2. Validation is total: an Artifact that satisfies the constraints resolves either to a single admissible label (conformant) or to `INTERPRETATION_DRIFT`. The resolution procedure itself is implementation-defined and need not be disclosed.
- C-3. It rejects Artifacts that fail the declared constraints with `INVALID_REQUEST`.
- C-4. It emits only labels declared in `P.labels`, and generates no content of its own.
- C-5. It signs conformant resolutions as offline-verifiable Conformance Events, and never requires a relying party to call back to verify.
- C-6. It exposes the wire contract of Section 6 and returns the unified envelope.

Because the Artifact and Agentic Protocol are content-hashed and validation is deterministic, a verdict is reproducible: the same `(Artifact, Agentic Protocol)` yields the same verdict and the same signed Conformance Event. The wire contract, verification, and outcomes are open; the resolution engine that produces the verdict is an implementation concern.

---

10. Security Considerations

Interpretation drift is an attack surface: an adversary can craft input that looks benign to a human while steering a model toward an unintended interpretation. TCP/AP contains this by bounding every model output to a declared, admissible decision or an explicit failure before execution. The gate is deterministic; a hijacked model can only propose, never decide.

Signing authority is the primary asset. Clients verify but never sign. Deploying a Kernel with a development or shared root **MUST NOT** be treated as production-grade attestation.

Appendix A. Example

A refund-approval Agentic Protocol, as declared by the caller:

```
{
  "slug": "issue_refund.guard",
```

```
  "version": "1.0",
  "constraints": { "type": "object", "required": ["amount"], "additionalProperties": false,
                   "properties": { "amount": { "type": "number" } } },
  "labels": ["AUTO_APPROVE", "MANAGER_REVIEW", "BLOCK"],
  "rules": [
    { "label": "AUTO_APPROVE",   "condition": { "<=": [{ "var": "amount" }, 50] } },
    { "label": "MANAGER_REVIEW", "condition": { "and": [ /* 50 < amount <= 1000 */ ] } },
    { "label": "BLOCK",          "condition": true }
  ]
}
```

Submitting the Artifact `{ "amount": 20 }` against this protocol returns a conformant verdict with label `AUTO_APPROVE`. An Artifact carrying an undeclared field returns `INVALID_REQUEST`. An Artifact that the protocol does not resolve to a single admissible decision returns `INTERPRETATION_DRIFT`. In every case the caller sees only the verdict, not how the Kernel reached it.