The Seam Protocol

Compositional Verification for Agent Pipelines

John Komkov

February 2026

Abstract

Bilateral verification of agent tool pipelines has computable blind spots: internal assumptions that shape every output but appear in no schema. We define the *coherence fee*—the number of such invisible dimensions—show it is the rank deficiency of the observable restriction map, and prove that *bridge annotations* reduce it to zero. Applied to the source code of deployed MCP servers, the diagnostic identifies three undeclared convention dimensions in the Filesystem–Git composition—encoding, line-ending format, and path separator—corresponding to failure modes that have plagued cross-platform development for decades. For trustless deployment, tools commit to a semantic manifest at composition time, enabling fraud proofs that certify compositional inconsistency from selective openings alone.

1 Introduction

A financial firm deploys three AI agents in a pipeline. Agent A retrieves market data using calendar days. Agent B computes risk-adjusted returns—silently annualizing with 252 trading days. Agent C verifies the result against portfolio thresholds and recommends a trade. Every schema check passes. Every type matches. The system ships a wrong trade, because the day-count convention drifted between A and B, and no interface ever carried it.

This is not a bug in any single tool. It is a structural property of *bilateral verification*—the regime in which correctness is checked one edge at a time, using only the information each tool publishes in its output schema. The day-count convention shaped every output but appeared in no schema. It was *projected away.* The *coherence fee* counts these independent, structurally invisible degrees of freedom.

On a DAG, a hidden inconsistency is **attributable**: you can trace it upstream to a specific tool's implicit choice and fix it with provenance or a bridge. On a cycle, it can be **non-localizable**: the inconsistency is a property of the loop, distributed around it, with no single edge to blame. **Cycles need witnesses; DAGs need blame.**

Observation 1.1 (Bilateral blindness)**.** *If two adjacent tools neither emit nor commit to a variable, no edge-local verifier can check equality of that variable across the edge.*

The intuition is ancient. A Florentine merchant and a Venetian merchant each keep internally consistent ledgers. A bill of exchange crosses the border between them. If neither ledger records the exchange rate used, no bilateral audit of the two ledgers can detect that they assumed different rates. The notary's art—the bridge annotation—is a small object that carries the conditions under which a claim can cross the border. Without it, the seam between two locally valid systems is a blind spot.

The Seam Protocol is a minimal addition to tool-calling protocols[1] that makes this blind spot computable and eliminable:

[1]MCP (Anthropic, 2024), A2A (Google, 2025), OpenAPI (2021). All enforce bilateral schema compatibility; none address compositional consistency across cycles.

1. **Diagnose**: compute the coherence fee from the tool schemas and bilateral interface specifications.
2. **Bridge**: for each blind-spot dimension, add a named field to both adjacent tools' output schemas.
3. **Verify**: confirm the coherence fee drops to zero.
4. **Enforce**: in trusted mode, check bridge fields at runtime; in trustless mode, commit to a semantic manifest and allow fraud proofs.

Empirical evidence. The failure mode is not hypothetical. In a companion experiment [5], three production LLMs (GPT-4o-mini, Claude 3.5 Haiku, Gemini 2.0 Flash) operating as financial agents on independent databases produce exactly two bilateral blind spots ($\dim H^1 = 2$) in every cyclic composition—matching the coboundary rank deficiency. All $3! = 6$ model-role permutations exhibit identical blind-spot patterns on ambiguous events (30/30), confirming the failure is structural (schema-driven) rather than behavioral (model-personality-driven). Three independent LLMs tasked with diagnosing the failure converge on the topologically prescribed bridge types (15/15). The present paper abstracts the mathematical core—the coherence fee—and the protocol layer (manifests, fraud proofs, enforcement) from that empirical foundation.

Contributions.

- A two-layer model (internal state vs. observable projection) that makes the bilateral blind spot computable (Section 2).
- The coherence fee: $\dim H^1(\mathcal{F}_{\text{obs}}) - \dim H^1(\mathcal{F}_{\text{full}})$, the precise count of consistency-relevant dimensions the projection kills (Section 3).
- A protocol with commitment-based fraud proofs—portable receipts for compositional inconsistency (Section 4).
- A diagnostic tool (`seam-lint`) and results across nine compositions, including three derived from the source code of deployed MCP servers in the official `modelcontextprotocol/servers` repository (Sections 6 and 6.1).

2 The Model

2.1 Internal State and Observable Projection

Definition 2.1 (Tool specification)**.** A *tool specification* is a triple $(v, S(v), F(v))$ where:

- v is the tool's identifier.
- $S(v) = \mathbb{R}^{s_v}$ is the *internal semantic state*: every assumption the tool carries, whether or not it appears in the output. Dimensions include output fields, internal parameters, conventions, and configuration.
- $F(v) = \mathbb{R}^{f_v}$ is the *observable schema*: the subset of $S(v)$ declared in the output. $F(v) \subseteq S(v)$.
- The *projection* $\pi_v : S(v) \to F(v)$ drops the dimensions of $S(v)$ not in $F(v)$.

Example 2.2 (Financial pipeline)**.** Three tools form a directed cycle:

$$\begin{tikzcd}
\texttt{data\_provider} \arrow[rr] & & \texttt{financial\_analysis} \arrow[dl] \\
& \texttt{portfolio\_verif.} \arrow[ul] &
\end{tikzcd}$$

Tool	$S(v)$ (internal)	$F(v)$ (observable)	π kills
`data_provider`	prices, dividends, period_unit, data_start, data_end, **day_convention**	prices, dividends, period_unit, data_start, data_end	day_convention
`financial_analysis`	risk_adj_return, volatility, max_drawdown, ann_return, **day_convention**, **risk_metric**	risk_adj_return, volatility, max_drawdown, ann_return	day_convention, risk_metric
`portfolio_verif.`	pass, score, recommendation, **risk_metric**	pass, score, recommendation	risk_metric

Two dimensions—`day_convention` and `risk_metric`—live in $S(v)$ but not in $F(v)$. The projection π kills them.

2.2 Bilateral Interfaces and the Coboundary

Definition 2.3 (Bilateral interface)**.** A *bilateral interface* at directed edge $e = (u, v)$ specifies a set of *semantic dimensions*—consistency requirements that must hold across the edge. Each dimension d names:

- A dimension `from_field` in $S(u)$ (or $\emptyset$).
- A dimension `to_field` in $S(v)$ (or $\emptyset$).

The dimension is *observable* if both fields lie in $F(u)$ and $F(v)$ respectively. It is a *blind spot* if either field lies in $S \setminus F$—i.e., π has killed the information the bilateral checker would need.

Definition 2.4 (Observable and full coboundary)**.** Let $G = (V, E)$ be the directed composition graph. The *cochain spaces* are:

$$C^0_{\text{obs}} = \bigoplus_{v \in V} F(v), \qquad C^0_{\text{full}} = \bigoplus_{v \in V} S(v), \qquad C^1 = \bigoplus_{e \in E} C(e)$$

where $C(e) = \mathbb{R}^{d_e}$ collects the semantic dimensions at edge e.

The *observable coboundary* $\delta^0_{\text{obs}} : C^0_{\text{obs}} \to C^1$ is defined by: for edge $e = (u, v)$ and semantic dimension d,

$$(\delta^0_{\text{obs}}\, s)_{e,d} = \rho^{\text{obs}}_{v \to e,d}(s_v) - \rho^{\text{obs}}_{u \to e,d}(s_u)$$

where $\rho^{\text{obs}}_{u \to e,d}$ projects $F(u)$ onto dimension d of $C(e)$—and is *zero* if d's `from_field` is not in $F(u)$ (i.e., π_u killed it).

The *full coboundary* $\delta^0_{\text{full}} : C^0_{\text{full}} \to C^1$ is defined identically, but using $S(v)$ instead of $F(v)$. Since every `from_field` and `to_field` exists in S by definition, no restriction map is zero.

2.3 The Coboundary Matrices

For the financial pipeline (Example 2.2):

$$\dim C^0_{\text{obs}} = 5 + 4 + 3 = 12, \quad \dim C^0_{\text{full}} = 6 + 6 + 4 = 16, \quad \dim C^1 = 3 + 3 + 2 = 8.$$

The observable coboundary δ^0_{obs} (8×12) has six nonzero rows—each with a single -1 at the source tool's observable field—and **two zero rows**:

Row	Semantic dimension	Entry
0	`data_match`	$-1 \cdot$ `dp.prices`
1	`time_match`	$-1 \cdot$ `dp.period_unit`
2	`day_conv_match`	**all zeros — blind spot**
3	`rar_match`	$-1 \cdot$ `fa.risk_adj_return`
4	`vol_match`	$-1 \cdot$ `fa.volatility`
5	`metric_type_match`	**all zeros — blind spot**
6	`feedback_match`	$-1 \cdot$ `pv.score`
7	`granularity_match`	$-1 \cdot$ `pv.recommendation`

The full coboundary δ^0_{full} (8×16) has **no zero rows**. Rows 2 and 5 become:

$$\text{row 2: } -1 \cdot \texttt{dp.day\_convention} + 1 \cdot \texttt{fa.day\_convention}$$

$$\text{row 5: } -1 \cdot \texttt{fa.risk\_metric} + 1 \cdot \texttt{pv.risk\_metric}$$

Proposition 2.5. $\operatorname{rank}(\delta^0_{\text{obs}}) = 6$, $\dim H^1(\mathcal{F}_{\text{obs}}) = 8 - 6 = 2$.
$\operatorname{rank}(\delta^0_{\text{full}}) = 8$, $\dim H^1(\mathcal{F}_{\text{full}}) = 8 - 8 = 0$.

Proof. In δ^0_{obs}, the six nonzero rows each place a single -1 in a distinct column, so they are linearly independent. Rows 2 and 5 are identically zero (both fields are projected away), contributing nothing. In δ^0_{full}, every row has at least one nonzero entry in a column unique to that row, giving full rank. □

Remark. The coherence fee is not "count of missing fields." It is the *rank deficiency of the observable restriction map*—the number of consistency requirements that touch dimensions the projection π has killed. $H^1(\mathcal{F}_{\text{full}}) = 0$ confirms that full internal state resolves everything; the gap $\dim H^1(\mathcal{F}_{\text{obs}}) - \dim H^1(\mathcal{F}_{\text{full}}) = 2$ is exactly the structural blind spot. In the simple case where each projected-away dimension appears in exactly one edge constraint, the fee equals the count of projected fields. The framework's value is that it generalizes: when projected fields participate in multiple constraints, or when restriction maps have nontrivial kernel, the rank deficiency can differ from the naive count.

3 The Coherence Fee

Definition 3.1 (Coherence fee)**.** The *coherence fee* of a tool composition is $\dim H^1(\mathcal{F}_{\text{obs}}) - \dim H^1(\mathcal{F}_{\text{full}})$: the number of independent semantic dimensions that bilateral verification cannot reach because π projects them away, net of any purely topological obstruction. Under the construction of Section 2, $H^1(\mathcal{F}_{\text{full}}) = 0$ (every dimension in $S(v)$ is reachable), so the fee reduces to $\dim H^1(\mathcal{F}_{\text{obs}})$. The gap formulation survives generalization to settings where $S(v)$ is incomplete.

The coherence fee is a structural property of the schema graph. It is zero if and only if every consistency-relevant dimension at every edge is covered by at least one adjacent tool's observable schema. It is the *irreducible cost* of making claims compose across tool boundaries without a trusted intermediary.

A bridge annotation eliminates a chokepoint: it makes an implicit dimension bilaterally checkable without requiring anyone to trust the orchestrator's interpretation. The coherence fee is what you pay when you choose not to bridge. The *rent* is what a semantic chokepoint extracts when it controls the only interpretation of an unbridged dimension.

In a marketplace of tools, a tool author who refuses to add bridge fields forces every downstream orchestrator to absorb the coherence fee—accepting silent failure modes that no bilateral check can detect. Competing tools that do support bridges offer a lower fee and are preferred by risk-sensitive orchestrators. This creates pressure toward $\dim H^1 = 0$ at equilibrium: the same competitive dynamic that drives API providers to publish comprehensive schemas today will drive them to publish bridge fields tomorrow.

The pressure can be made concrete through a three-layer market mechanism:

1. *Fee-footprint publication.* Each tool publishes the number of blind-spot dimensions it contributes to any composition (its "fee footprint"). Orchestrators publish a "max coherence fee tolerated" for their pipelines. Tools compete on fee footprint alongside latency, cost, and accuracy.
2. *Procurement thresholds.* A rule ("only compose tools with coherence fee $\leq k$") turns the fee into a qualification threshold. A registry that scores tools by fee footprint gives orchestrators a quantitative basis for tool selection.
3. *Priced risk.* Insurance priced to the coherence fee makes the cost of non-bridging visible in the budget, not just in the risk register. An escrow pool funded by the coherence fee itself—tool authors deposit proportional to the fee they impose; slashed when a valid fraud proof is submitted—creates a direct incentive to bridge.

3.1 The Failing Scenario

We construct concrete tool outputs demonstrating the blind spot. All three bilateral checks pass:

Tool	Observable output	Internal assumption
`data_provider`	prices, period_unit= "calendar"	day_convention= "calendar"
`financial_analysis`	risk_adj_return= 1.67, vol= 0.19	day_convention= "trading", risk_metric= "sortino"
`portfolio_verif.`	pass= true, score= 0.91	risk_metric= "sharpe"

The bilateral checker at each edge sees only observable fields and reports success. But the internal states disagree on two dimensions: `day_convention` ("calendar" $\neq$ "trading") and `risk_metric` ("sortino" $\neq$ "sharpe").

These are the two generators of $H^1(\mathcal{F}_{\text{obs}})$, made concrete.

Negative control: existing validation passes. We submitted the three tool outputs from the failing scenario to standard MCP JSON Schema validation. The validator checks: every required field present, every type correct, every value within its declared range. All three tools pass. The validator *cannot* flag the `day_convention` drift (calendar vs. trading) or the `risk_metric` mismatch (sortino vs. sharpe), because neither field appears in any schema. This is not a limitation of the validator's implementation—it is a structural impossibility. The information needed to detect the inconsistency has been projected away by π. No improvement to bilateral schema validation can close this gap; only bridge annotations can.

3.2 Eliminating the Fee

Definition 3.2 (Bridge annotation)**.** A *bridge annotation* for blind-spot dimension d at edge $e = (u, v)$ extends the observable schemas: add field f_d to both $F(u)$ and $F(v)$, so that π_u and π_v no longer kill f_d. The previously-zero row of δ^0_{obs} becomes:

$$-1 \cdot (u, f_d) \; + \; +1 \cdot (v, f_d)$$

Concretely, the `financial_analysis` tool's output schema before and after bridging:

Listing 1: Before: risk_metric is projected away

```
{
  "output": {
    "risk_adjusted_return": {"type": "number"},
    "volatility":           {"type": "number"},
    "max_drawdown":         {"type": "number"},
    "annualized_return":    {"type": "number"}
  }
}
```

Listing 2: After: bridge annotation makes risk_metric observable

```
{
  "output": {
    "risk_adjusted_return": {"type": "number"},
    "volatility":           {"type": "number"},
    "max_drawdown":         {"type": "number"},
    "annualized_return":    {"type": "number"},
    "risk_metric": {"type": "string",
                    "enum": ["sharpe","sortino","alpha"]}
  }
}
```

Theorem 3.3 (Bridge elimination)**.** *If* $\dim H^1(\mathcal{F}_{\text{obs}}) = k > 0$*, applying a bridge annotation for each of the* k *generators produces* $\mathcal{F}'_{\text{obs}}$ *with* $\dim H^1(\mathcal{F}'_{\text{obs}}) = 0$*.*

Proof. Each generator corresponds to a zero row in δ^0_{obs}. The bridge adds two columns (one per tool) and sets the row's entries to -1 and $+1$ in those columns. The row becomes linearly independent from all others (it is the only row with nonzero entries in those columns). Repeating for all k generators raises the rank by k, so $\dim H^1(\mathcal{F}'_{\text{obs}}) = (k + r) - (r + k) = 0$ where r was the original rank. □

For the financial pipeline, two bridges suffice:

Bridge field	Added to	Blind spot eliminated
`day_convention`	`dp`, `fa`	row 2 (`day_conv_match`)
`risk_metric`	`fa`, `pv`	row 5 (`metric_type_match`)

After bridging: $\delta^0_{\text{obs}'}$ is 8×16, rank $= 8$, $\dim H^1 = 0$. The same failing-scenario data now triggers two bilateral failures: `day_convention` = “calendar” $\neq$ “trading” at `dp`→`fa`, and `risk_metric` = “sortino” $\neq$ “sharpe” at `fa`→`pv`.

The blind spots become type errors.

4 The Protocol

4.1 Phases

The protocol enforces a single invariant:

No edge may rely on semantics that neither endpoint emits nor commits to.

Every phase below exists to make this invariant computable, achievable, and enforceable.

Registration. Each tool registers its specification $(v, S(v), F(v))$ with a bridge registry or directly with the orchestrator. The composition graph is built, the coboundary is computed, and the coherence fee is reported. If $\dim H^1(\mathcal{F}_{\text{obs}}) > 0$, the generators are identified and bridge annotations are recommended. The registry is one implementation of this computation; it is not the protocol's essence. The essence is: **compute fee → bridge → enforce**.

Composition. An orchestrator queries the registry before composing a pipeline. If the fee is zero, bilateral checks suffice. If not, the orchestrator either applies the recommended bridges (reducing the fee to zero) or accepts the residual risk.

Enforcement. Two modes, corresponding to different trust assumptions.

Trusted mode: the orchestrator has access to all outputs and checks all bridge fields at every edge. Cost: $O(|E| \cdot d_{\max})$.

Trustless mode: tools are operated by independent parties. No single party is trusted to check correctly. This mode requires one additional primitive: the *semantic manifest commitment*.

4.2 Semantic Manifests

Definition 4.1 (Semantic manifest)**.** At composition time, each tool v publishes:

1. Its observable output $F(v)$ (in the clear).
2. A commitment $r_v = H(\texttt{manifest}_v)$ where $\texttt{manifest}_v$ contains values for *all* dimensions in $S(v)$, including those not in $F(v)$.

In normal operation, only $F(v)$ and r_v are visible. The internal dimensions remain private.

The commitment is a hash of the concatenated per-dimension hashes: $r_v = H(h_1 \| h_2 \| \cdots \| h_{s_v})$ where $h_i = H(\texttt{key}_i : \texttt{value}_i)$. A *selective opening* for dimension d reveals $(\texttt{key}_d, \texttt{value}_d, h_d)$ and lets any verifier check h_d against r_v.

4.3 Composition Fraud Proofs

Definition 4.2 (Composition fraud proof)**.** A *composition fraud proof* (a receipt for compositional inconsistency) is a tuple $\pi = (\{h_i\}, \{r_i\}, C, B)$:

- $\{h_i\}$: hashes of the tool schemas (commits to the specification).
- $\{r_i\}$: manifest root hashes (commits to the internal state at composition time).
- C: the cycle $v_1 \to v_2 \to \cdots \to v_1$.
- B: the *blind-spot certificate*: for each failing dimension, selective openings from both adjacent tools showing inconsistent values for a dimension that π projected away.

Verification ($O(n)$).

1. **Confirm all observable bilateral checks pass.** This is the step that makes the fraud proof non-trivial. If bilateral checks *also* fail, that is a bilateral error—a bug, not a blind spot. The receipt is interesting precisely when the composition looks correct to every local verifier and is globally inconsistent.
2. Verify schema hashes against the registry.
3. Verify manifest roots match the commitments published at composition time.
4. For each opening, verify the dimension hash matches $H(\texttt{key} : \texttt{value})$.
5. Confirm each blind spot exhibits a nonzero residual (the two tools' values disagree on a projected-away dimension).

If all five steps succeed, the receipt is valid: bilateral verification certified a composition that is globally inconsistent.

Theorem 4.3 (Soundness)**.** *If all bridges are applied (*$\dim H^1(\mathcal{F}_{\text{obs}}) = 0$*) and all bilateral checks pass, no valid fraud proof exists.*

Proof. When $\dim H^1(\mathcal{F}_{\text{obs}}) = 0$, every consistency-relevant dimension at every edge is observable. A bilateral check that passes on all observable dimensions therefore passes on *all* dimensions. Step 4 of verification requires a nonzero residual on a projected-away dimension, but with all bridges applied, no dimensions are projected away at any edge. Step 4 always fails; no valid receipt can be constructed. □

In plain terms: a fully bridged pipeline cannot be receipted for compositional failure. The receipt mechanism proves nothing because there is nothing to prove.

Remark. A fraud proof is not an accusation against a specific tool. On a cycle, the inconsistency is a property of the composition—it does not localize to any single edge. The receipt names the cycle, the openings, and the residual. It is a *global witness* for a non-localizable failure.

4.4 Why Cycles Require Receipts

On a DAG, a hidden inconsistency is attributable: follow the directed edges backward from the point of failure and you find the tool whose implicit choice caused it. The fix is local: bridge that tool's output or document its assumptions.

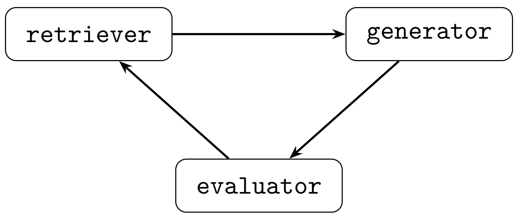
On a cycle, attribution fails. The inconsistency circulates: `dp` assumes calendar days; `fa` switches to trading days; `pv` interprets the result as if it were Sharpe when `fa` computed Sortino; `pv`'s feedback reaches `dp`, closing the loop. No single edge is "wrong." The composition is wrong.

This is why a receipt must exhibit the *entire cycle*: the schema commitments, the manifest commitments, and the openings around the loop. Cycles need witnesses; DAGs need blame.

Security model. The trustless mode assumes an honest-observer (1-of-N) model: correctness holds unless every observer is compromised. Manifest roots must be posted to a data-available channel. A dispute game specifies consequences when a valid fraud proof is submitted (reputational, financial, or operational). The full security model—roles, data availability modes, dispute mechanics, and collusion boundary—is in Appendix B.

5 A Second Example: RAG with Feedback

To confirm the model generalizes, we apply it to a retrieval-augmented generation (RAG) pipeline with a feedback loop:



Tool	π kills	Semantic role
`retriever`	chunk_size, relevance_threshold	How documents were chunked; what threshold was used
`generator`	chunk_size, citation_mode	Whether chunks were reassembled; strict vs. loose citation
`evaluator`	citation_mode, relevance_threshold	What citation fidelity means; what relevance means

Three bilateral interfaces carry 9 semantic dimensions (3 per edge). Three of those dimensions are blind spots: `chunk_size_match` (retriever→generator), `citation_mode_match` (generator→evaluator), `threshold_match` (evaluator→retriever).

$$\delta^0_{\text{obs}} : 9 \times 8, \quad \text{rank} = 6, \quad \dim H^1(\mathcal{F}_{\text{obs}}) = 3.$$
$$\delta^0_{\text{full}} : 9 \times 14, \quad \text{rank} = 9, \quad \dim H^1(\mathcal{F}_{\text{full}}) = 0.$$

The coherence fee is 3.

Composition	Tools	Edges	β_1	Fee	Blind-spot dimensions
Auth-Data-Audit	3	3	1	0	—
Financial Analysis	3	3	1	2	`day_convention`, `risk_metric`
RAG with Feedback	3	3	1	3	`chunk_size`, `citation_mode`, `relevance_threshold`
Code Review	4	4	1	3	`diff_format`, `severity_threshold`, `style_convention`
Web Research	4	4	1	3	`search_language`, `citation_style`
Multi-Source ETL	4	4	1	4	`timezone`, `null_convention`, `decimal_precision`
From deployed MCP server source code (Section 6.1)					
Filesystem $\leftrightarrow$ Git	2	2	1	3	`encoding`, `line_ending`, `path_separator`
Fetch $\leftrightarrow$ Memory	2	2	1	2	`encoding`, `content_format`
Fetch $\rightarrow$ Filesystem $\rightarrow$ Git	3	3	1	4	`encoding`$\times 2$, `line_ending`, `path_separator`

Table 1: Coherence fee diagnostic across nine compositions. The first six use author-constructed schemas; the last three are extracted from the source code of deployed MCP servers (the `modelcontextprotocol/servers` repository). After bridging, every composition drops to fee $= 0$.

A concrete RAG failure. The retriever chunks documents at 512 tokens with `relevance_threshold` $= 0.7$. The generator receives the chunks (observable), reassembles them into a response, and internally uses `chunk_size` $= 256$ (it was fine-tuned on smaller chunks) and `citation_mode` = "strict" (verbatim spans only). The evaluator checks citation fidelity using `citation_mode` = "loose" (paraphrase acceptable) and `relevance_threshold` $= 0.5$.

Every bilateral check passes: the chunks arrive, the response cites them, the quality score is high. But the generator is silently re-chunking at 256 tokens, breaking the retriever's span boundaries. The evaluator accepts paraphrased citations that the generator flagged as strict verbatim matches. And the evaluator's feedback loop sends a relaxed relevance threshold back to a retriever that expects 0.7—gradually degrading retrieval quality over iterations.

These are three generators of $H^1(\mathcal{F}_{\text{obs}})$, made concrete. Three bridge annotations (`chunk_size`, `citation_mode`, `relevance_threshold`) reduce the fee to zero and convert each silent drift into a type error at the offending edge.

Remark. The recurring implicit dimensions across both examples—conventions, thresholds, metric definitions, strategy choices—are exactly the assumptions that tool authors treat as "obvious" and omit from schemas. The coherence fee makes the cost of that omission computable.

6 Diagnostic Results

To demonstrate that the coherence fee is both computable and practically informative across domains, we implemented `seam-lint`: a diagnostic tool that takes a YAML composition specification (tool schemas, bilateral interfaces) and reports the coherence fee, identifies blind-spot dimensions, and recommends bridge annotations. We ran it on nine compositions spanning finance, retrieval-augmented generation, DevOps, data engineering, security, web research, and three compositions derived directly from the source code of deployed MCP servers (Section 6.1).

Five findings stand out.

Real MCP tool types produce nonzero fees. The web research pipeline is composed of four tools modeled on commonly-deployed MCP servers: Brave Search (web search API), a content extraction server (Firecrawl, Jina Reader), an LLM-based summarizer, and a fact-checking tool, with a feedback loop from fact-checker to search for iterative verification. The coherence fee is 3, corresponding to two implicit dimensions: `search_language` (assumed by the search engine, propagated through scraping and summarization, but declared in no schema—three blind-spot edges from a single hidden convention) and `citation_style` (the summarizer produces inline citations; the fact-checker expects numbered references). Two bridge annotations—adding `search_language` and `citation_style` to the relevant schemas—reduce the fee to 0. No existing MCP schema validation would flag either dimension. The `search_language` case demonstrates a structural point: a single omitted dimension can contribute multiple blind-spot edges—the fee measures *topological footprint*, not merely the count of hidden assumptions.

Deployed MCP servers confirm the prediction. Section 6.1 analyzes three compositions built entirely from the source code of servers in the official MCP repository. The Filesystem $\leftrightarrow$ Git composition—the single most common MCP multi-tool workflow—has fee 3. The three blind-spot dimensions (`encoding`, `line_ending_convention`, `path_separator`) correspond to failure modes that practitioners already encounter: CRLF/LF inconsistencies on Windows, encoding mismatches with non-ASCII filenames, and OS-dependent path separators in diff output. None appears in any MCP tool schema. A 3-tool composition (Fetch $\rightarrow$ Filesystem $\rightarrow$ Git) raises the fee to 4: the `encoding` convention, undeclared at all three tools, creates blind spots at every boundary it crosses.

The fee varies by domain. Finance and security compositions, where schemas tend to be well-specified, have lower fees (0–2). RAG and data-engineering compositions, where conventions (chunk boundaries, null handling, decimal precision) are often left implicit, have higher fees (3–4). The tool quantifies a difference that practitioners sense but cannot currently measure.

Zero fee is achievable. The auth-data-audit pipeline has $S(v) = F(v)$ for all tools—every consistency-relevant dimension is already in the output schema. Its $H^1(\mathcal{F}_{\text{obs}}) = 1$ is purely topological (one cycle creates one redundant constraint among the `user_id` matching dimensions). The coherence fee, correctly defined as $\dim H^1(\mathcal{F}_{\text{obs}}) - \dim H^1(\mathcal{F}_{\text{full}}) = 0$, separates topological from observability contributions.

Bridge recommendations are minimal. The ETL pipeline has four blind-spot dimensions but only three unique bridge fields (`timezone`, `null_convention`, `decimal_precision`), because `null_convention` appears on two edges. The tool deduplicates and reports three bridge annotations that jointly reduce the fee from 4 to 0.

The fee scales with composition size. An undeclared convention contributes a blind-spot dimension at *every* composition boundary it crosses. The coherence fee is therefore $O(|E| \cdot c)$, where c is the number of undeclared conventions and $|E|$ is the number of edges each convention spans—not merely $O(c)$. The 3-tool deployed composition (Fetch $\rightarrow$ Filesystem $\rightarrow$ Git) provides direct evidence: `encoding`, undeclared at all three tools, crosses two boundaries and contributes 2 to the fee, raising it from 3 (2-tool) to 4 (3-tool). In a 10-tool pipeline with 3 undeclared conventions, the fee could reach 15, not 3. Bridge annotations have *increasing* marginal returns: declaring a convention once eliminates its contribution at every boundary simultaneously.

A testable prediction. We predict that *in practice*, every MCP pipeline with a feedback loop ($\beta_1 \geq 1$) contains at least one convention-type dimension (encoding, line-ending format, time basis, null representation) that shapes outputs at multiple tools but appears in no tool's output schema. This is an empirical claim about how tool authors actually design schemas—they treat conventions as "obvious" and omit them—not a mathematical tautology. The claim is falsifiable: find a cyclic MCP composition where the tool authors *did* declare their encoding, line-ending convention, timezone, and other convention-type dimensions in their output schemas. That would show the "omission is universal" premise is wrong. A zero-fee cyclic pipeline where conventions were *never relevant* (like the auth-data-audit pipeline) does not falsify the prediction—it confirms that tools without convention-type requirements can achieve fee 0. In the nine compositions analyzed here—including three from deployed server source code—every cyclic pipeline with at least one convention-type hidden dimension has fee ≥ 1. We further predict that in production pipelines with $n > 5$ tools, the fee will grow superlinearly in the number of tool boundaries each undeclared convention crosses—making the cost of *not* bridging increasingly visible at scale.

The diagnostic tool, all composition specs (YAML), and output are at `papers/seam/seam-lint/`.

6.1 Diagnostic on Deployed MCP Servers

The six compositions above use schemas constructed by the authors to illustrate the framework. A stronger test is to apply the diagnostic to tool compositions that someone else built for a different purpose and ask whether the framework discovers something about them.

Methodology. We selected MCP servers from the official `modelcontextprotocol/servers` repository (commit `a83b145`). For each server, we identified $F(v)$ from the tool's declared output format in the source code—the fields that actually appear in `CallToolResult`—and $S(v)$ by reading the implementation to identify configuration parameters, environment variables, and hardcoded conventions that shape output but do not appear in the schema. Bilateral interfaces were inferred from shared semantic requirements: when two tools in a composition must agree on a convention for their composed output to be consistent, we record that convention as a semantic dimension at the connecting edge. This identification step is necessarily manual—it requires domain understanding of what "consistency" means for each dimension—and we report it transparently: the YAML composition specs are published alongside the paper for independent verification.

Composition 1: Filesystem $\leftrightarrow$ Git. This is the canonical AI-assisted coding workflow: the Filesystem server reads and writes local files; the Git server tracks changes, produces diffs, and commits. A feedback loop (Git status/diff $\to$ Filesystem edit) makes this a cyclic composition ($\beta_1 = 1$). We identified $|S(v)| = 6$ internal dimensions for Filesystem and $|S(v)| = 9$ for Git by reading the source. Three dimensions are in $S(v) \setminus F(v)$ at *both* endpoints:

- **`encoding`.** Filesystem hardcodes UTF-8 (`lib.ts:157`). Git uses locale-dependent encoding for subprocess output and hardcodes UTF-8 only for binary diffs (`server.py:216`). Neither tool declares its encoding assumption in any output field.
- **`line_ending_convention`.** Filesystem returns raw bytes from disk for reads (preserving CRLF on Windows) but normalizes to LF for `edit_file` (`lib.ts:56`). Git assumes LF in string splits (`server.py:154`). The MCP server has no `core.autocrlf` awareness. Neither tool declares the convention.
- **`path_separator`.** Filesystem uses OS-native separators via `path.join` (`path-utils.ts:38`). Git always outputs POSIX `/` in diff headers and status. A developer on Windows sees `src\file.txt` from Filesystem and `src/file.txt` from Git—for the same file.

The coherence fee is 3. All three blind spots are convention-type dimensions—exactly the kind our testable prediction targets. The `path_separator` dimension appears on both edges of the cycle, but the coboundary correctly identifies this redundancy: $H^1(\mathcal{F}_{\text{full}}) = 1$, so the net fee is $H^1(\mathcal{F}_{\text{obs}}) - H^1(\mathcal{F}_{\text{full}}) = 4 - 1 = 3$. Three bridge annotations (adding `encoding`, `line_ending_convention`, and `path_separator` to both servers' output schemas) reduce the fee to 0.

The failing scenario. A developer on Windows edits a source file. The Filesystem server's `read_file` returns the contents with CRLF line endings (raw bytes from disk). An MCP client stores this content and calls `git_diff_unstaged` to obtain the diff—whose context lines use LF only (Git convention). The client attempts to match diff hunks against the file content: line n in the diff reads `function foo() {\n` while the file has `function foo() {\r\n`. A patch application or line-number correlation fails silently. Both bilateral checks pass: `read_file` returns valid UTF-8 text, `git_diff` returns a syntactically correct unified diff. The inconsistency is *between* the two outputs, in a dimension—`line_ending_convention`—that neither output schema declares.

Composition 2: Fetch $\leftrightarrow$ Memory. A knowledge-acquisition pipeline: the Fetch server retrieves web content and converts HTML to markdown; the Memory server stores entities and relations as a JSONL knowledge graph. A feedback loop (Memory search identifies gaps $\rightarrow$ Fetch retrieves more) makes this cyclic. Two blind spots:

- **`encoding`.** Fetch relies on `httpx`'s charset detection (Content-Type header $\rightarrow$ UTF-8 fallback $\rightarrow$ `charset_normalizer`). Memory reads and writes JSONL as UTF-8 (`fs.readFile(..., "utf-8")`). If Fetch encounters a non-UTF-8 page, the decoded Python string may contain lossy substitutions that Memory stores without awareness.
- **`content_format`.** Fetch converts HTML to markdown via `readabilipy` + `markdownify`—producing ATX headings, inline links, and emphasis markers. Memory stores observations as opaque strings. A Memory search for "important concept" will fail to match `**important concept**` from the markdown conversion. Neither tool declares the format convention.

The coherence fee is 2. Both bridges are actionable: Fetch could declare `content_format: "markdown"|"raw"` in its output; Memory could declare `observation_format: "plaintext"` and normalize on ingestion.

Composition 3: Fetch $\rightarrow$ Filesystem $\rightarrow$ Git. A web-content-to-version-control pipeline: Fetch retrieves documentation or data, Filesystem writes it locally, Git commits the changes. A feedback loop (Git log reveals committed URLs $\rightarrow$ Fetch checks for updates) makes this cyclic. The composition chains all three official servers, producing a 3-tool, 3-edge graph with $\beta_1 = 1$.

The coherence fee is 4—higher than either 2-tool sub-composition. The increase comes from `encoding`, which now appears as a blind spot on *two* edges: Fetch$\rightarrow$Filesystem (httpx charset detection vs. hardcoded UTF-8) and Filesystem$\rightarrow$Git (UTF-8 vs. locale-dependent subprocess). A single hidden convention contributes two independent blind-spot dimensions because it crosses two composition boundaries. The remaining two blind spots (`line_ending_convention`, `path_separator`) are the same as in the 2-tool Filesystem$\leftrightarrow$Git composition. Three bridge annotations—`encoding` (added to all three tools), `line_ending_convention`, and `path_separator`—reduce the fee to 0.

Unlike the 2-tool Filesystem$\leftrightarrow$Git case, $H^1(\mathcal{F}_{\text{full}}) = 0$ here: the `path_separator` dimension appears on only one edge (Filesystem$\rightarrow$Git, not Git$\rightarrow$Fetch), so there is no topological redundancy. The fee equals the raw $H^1(\mathcal{F}_{\text{obs}})$, and the gap formulation and the standalone formulation agree—confirming that the gap matters only when the same dimension appears on multiple edges of the same cycle.

Summary. All three deployed compositions have nonzero coherence fees. The blind spots correspond to real engineering problems: CRLF/LF inconsistencies, encoding mismatches, path separator mismatches, and markdown-vs-plaintext confusion. The 3-tool composition demonstrates that fees accumulate across composition boundaries: a single undeclared convention (`encoding`) that is harmless in a 2-tool DAG becomes a blind spot at every boundary it crosses. In every case, bilateral MCP schema validation (JSON Schema on inputs, type checking on outputs) would pass—the inconsistency is structurally invisible to pairwise checks, exactly as the theory predicts.

7 Scope and Limitations

Known unknowns, not unknown unknowns. The model requires that someone enumerate the semantic dimensions at each edge, including implicit ones. If a dimension is not recognized and listed, it will not appear in the coboundary and will not be detected. The protocol diagnoses known unknowns. It does not discover unknown unknowns—but once a dimension is discovered (by testing, auditing, or failure), it becomes permanently checkable.

Three mechanisms make discovery systematic rather than artisanal:

1. *Standard manifest vocabulary.* A canonical set of dimension categories—time conventions, unit systems, rounding modes, objective functions, model identifiers, retrieval settings—with a requirement that tools declare `unknown/NA` explicitly for each. The vocabulary grows as domains are onboarded.
2. *Schema linting.* A static pass over tool docstrings, tests, and output schemas that proposes candidate hidden dimensions ("this tool annualizes—what day-count convention?"). The lint does not need to be perfect; it converts unknown unknowns into known unknowns for human review.
3. *Domain invariant classes.* For well-understood domains, a curated set of always-bridged dimensions: `day_convention` and `risk_metric` in finance, `chunk_size` and `citation_mode` in RAG, `timezone` and `currency` in cross-border transactions. These encode hard-won domain knowledge as reusable infrastructure.

The protocol does not solve discovery perfectly—but it prevents the critique "this is only as good as manual enumeration" by providing concrete tooling that monotonically expands the known-unknown set.

Static analysis. The coherence fee is computed from schemas, not runtime values. It identifies structural blind spots. Whether bridge field values are *truthful* at runtime is the enforcement layer's job (bilateral checks in trusted mode; commitments and fraud proofs in trustless mode).

Cycles. The distinction between attributable (DAG) and non-localizable (cycle) failures means the fraud-proof machinery is most valuable for cyclic compositions. As agent pipelines mature—feedback loops, self-improvement cycles, multi-agent negotiation, monitoring agents feeding parameters back to data agents—cycles will become the norm, not the exception.

Bridge adoption. The protocol's value depends on tool authors adding bridge fields. This is a coordination problem, not a technical one. The protocol provides the diagnostic (the fee is nonzero) and the prescription (which fields to add to which tools). It cannot force adoption, but it makes the cost of non-adoption explicit and computable.

8 Related Work

Tool-calling protocols. MCP [2] defines bilateral tool schemas with JSON Schema types. A2A [4] adds agent-to-agent negotiation. OpenAPI [6] provides REST-level schema validation. All three enforce bilateral compatibility; none address compositional consistency across cycles.

Schema mapping composition. Fagin et al. [3] prove that composing schema mappings requires second-order dependencies: first-order constraints specified pairwise between two schemas do not compose transitively to a third. This is the database-theoretic foundation for why pairwise reconciliation does not compose—and the closest prior result to our bilateral completeness claim. Our contribution is the specific quantification: the coherence fee measures *how much* composition fails, not just *that* it fails, and the protocol layer makes the gap enforceable.

Sheaf-theoretic data integration. The use of cellular sheaves for data consistency has roots in applied algebraic topology. Our two-layer model (observable vs. full sheaf) is a direct application: the rank deficiency of the observable coboundary measures the information lost by projection. Appendix A provides the full sheaf-theoretic formulation.

Optimistic rollups and fraud proofs. The composition fraud proof (Definition 4.2) follows the optimistic rollup pattern [1]: assume correctness, allow any party to challenge with a succinct proof, revert if valid. The key difference: our proofs target semantic inconsistency across tool boundaries, not state-transition invalidity. The semantic manifest commitment (Definition 4.1) provides the cryptographic primitive that makes the analogy precise.

A Sheaf-Theoretic Foundation

The linear-algebraic model of Section 2 is an instance of a *cellular sheaf* on the directed composition graph $G = (V, E)$.

Definition A.1 (Schema sheaf)**.** The *observable schema sheaf* $\mathcal{F}_{\text{obs}}$ assigns:

- To each vertex $v \in V$, a stalk $\mathcal{F}_{\text{obs}}(v) = F(v) = \mathbb{R}^{f_v}$.
- To each edge $e \in E$, a stalk $\mathcal{F}_{\text{obs}}(e) = C(e) = \mathbb{R}^{d_e}$.
- Restriction maps $\rho^{\text{obs}}_{v \to e} : F(v) \to C(e)$ defined by coordinate projection (or zero if π_v kills the relevant dimension).

The *full schema sheaf* $\mathcal{F}_{\text{full}}$ is defined identically with $S(v)$ replacing $F(v)$, and no restriction map is ever zero.

The cochain complex is:

$$0 \to C^0 \xrightarrow{\delta^0} C^1 \to 0$$

where $C^0 = \bigoplus_v \mathcal{F}_{\text{obs}}(v)$, $C^1 = \bigoplus_e \mathcal{F}_{\text{obs}}(e)$, and δ^0 is as in Definition 2.4. The first cohomology is:

$$H^1(\mathcal{F}_{\text{obs}}) = \operatorname{coker}(\delta^0) = C^1 / \operatorname{im}(\delta^0).$$

Theorem A.2 (Coherence fee as cohomological gap)**.**

$$\textit{coherence fee} = \dim H^1(\mathcal{F}_{\text{obs}}) - \dim H^1(\mathcal{F}_{\text{full}}).$$

When every semantic dimension at every edge has a corresponding internal-state dimension at both adjacent vertices and each such dimension indexes a unique column *of the full coboundary, all rows of δ^0_{full} are linearly independent and $H^1(\mathcal{F}_{\text{full}}) = 0$, so the fee reduces to* $\dim H^1(\mathcal{F}_{\text{obs}})$.

Remark. The condition can fail when the *same* internal-state dimension participates in bilateral requirements on multiple edges of a cycle. In the Filesystem $\leftrightarrow$ Git composition (Section 6.1), `path_separator` appears on both edges; the two corresponding rows of δ^0_{full} are linearly dependent, giving $H^1(\mathcal{F}_{\text{full}}) = 1$. This is a *topological* contribution—it reflects the redundancy inherent in cycles, not missing internal state. The gap formulation correctly subtracts it: fee $= 4 - 1 = 3$.

Proof. When the uniqueness condition holds, in δ^0_{full} every row has at least one nonzero entry in a column unique to that row (the `from_field` or `to_field` in $S(v)$). The rows are therefore linearly independent, giving $\operatorname{rank}(\delta^0_{\text{full}}) = \dim C^1$ and $H^1(\mathcal{F}_{\text{full}}) = 0$. When it does not hold, $H^1(\mathcal{F}_{\text{full}}) \geq 1$, and the gap formulation accounts for the topological contribution.

Each bridge annotation extends $F(v)$ to include a dimension previously in $S(v) \setminus F(v)$, making the observable restriction map nonzero where it was previously zero. Applying bridges for all blind-spot generators recovers $\delta^0_{\text{obs}'} = \delta^0_{\text{full}}$ (restricted to the bridged observable dimensions), yielding $H^1(\mathcal{F}'_{\text{obs}}) = H^1(\mathcal{F}_{\text{full}})$—i.e., the fee drops to zero. □

Availability. The worked examples (Python source, twelve checkpoints) are at `papers/seam/example/seam_ex`
The `seam-lint` diagnostic tool, composition specs, and results (Table 1) are at `papers/seam/seam-lint/`.

B Participants & Security Model

Roles. Four roles participate in the protocol. A single entity may occupy multiple roles.

- *Tool operators*: publish $(v, S(v), F(v))$, produce outputs, commit to manifests.
- *Orchestrators*: compose tools into pipelines, run bilateral checks, apply bridges, optionally post bonds.
- *Verifiers* (challengers): any party that reads commitments and constructs fraud proofs.
- *Registry*: stores schema and manifest commitments; adjudicates fraud proofs.

Honest observer (1-of-N). The fraud-proof mechanism requires that at least one verifier along the cycle can read all committed manifest roots and observable outputs, and is willing to raise a fraud proof if an inconsistency exists. Correctness holds unless *every* observer is compromised or colluding—the same assumption as optimistic rollups.

Data availability. Manifest root hashes and observable outputs must be posted to a location all parties can read—an orchestrator log, a shared ledger, or a broadcast channel. If a tool can withhold its manifest root, verification cannot proceed. Two deployment modes apply:

- *Trusted DA*: the orchestrator stores and serves all commitments. Sufficient when the orchestrator is a first party.
- *Challenge-response DA*: manifest roots are posted publicly; a challenge triggers selective opening within a dispute window Δt. Failure to open within Δt is treated as an admission of inconsistency.

Dispute game. When a verifier submits a fraud proof:

1. The registry verifies the proof ($O(n)$, Section 4).
2. If valid: the accused composition is marked *faulted*. Consequences are application-specific but fall into three tiers: (a) *reputational*—the tools' fee-footprint scores increase, reducing their ranking in future compositions; (b) *financial*—an orchestrator's bond is slashed or an insurance claim is triggered against the unresolved coherence fee; (c) *operational*—the pipeline is halted pending bridge resolution.
3. If invalid (verification fails at any step): the proof is discarded; optionally the challenger's deposit is forfeit to discourage spam.

Collusion boundary. Bilateral collusion (two adjacent tools agreeing to report matching bridge values that differ from their actual computation) is detectable by an end-to-end audit that compares final outputs against claimed bridge values. Full-cycle collusion—where *all* tools agree to misrepresent—is outside the scope of compositional verification. It reduces to the trust assumption on individual tools, not on their composition.

References

[1] John Adler and Mikerah Quintyne-Collins. Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities. 2018. arXiv:1809.09044.

[2] Anthropic. Model context protocol specification. 2024. `https://modelcontextprotocol.io`.

[3] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Transactions on Database Systems*, 30(4):994–1055, 2005.

[4] Google. Agent-to-agent protocol (A2A). 2025. `https://google.github.io/A2A/`.

[5] John Komkov. The coherence fee: Edge-local blindness at the string-table seam and the topological price of cross-system composition. 2026. Companion paper.

[6] OpenAPI Initiative. OpenAPI specification v3.1, 2021. `https://spec.openapis.org/oas/v3.1.0`.