Behavioral Identity for Tool Ecosystems: Why Name and Schema Are Not Enough

Thomas Dionysopoulos, CFA

June 2026 — Draft

Abstract

Every major tool-calling protocol—MCP, OpenAI function calling, LangChain—describes tools with a name and a JSON schema. When multiple tools share the same schema (e.g., `read_query` and `write_query` both accept `{query: string}`), the schema cannot distinguish them. We propose adding three behavioral metadata fields to tool declarations—mutability, action type, and output domain—hashed into a *behavioral identity*. In a controlled experiment (45 tools, 3 language models, 90 agent decisions), behavioral identity catches 88% of tool selection errors compared to 38% for schema validation. Behavioral identity catches every error that schema catches, plus four additional errors that schema is structurally blind to. Schema catches zero errors that behavioral identity misses. The behavioral fields are independently recoverable: three frontier models, given only a tool's schema with the name removed, agree on the behavioral classification 87% of the time. The cost is three fields per tool. The benefit is that same-schema confusion—the most dangerous class of agent tool error—becomes catchable before execution.

1 Introduction

When an AI agent selects a tool, the runtime checks two things: does the tool exist (name resolution), and do the parameters match (schema validation). If both pass, the tool is dispatched.

This is insufficient. Consider a database MCP server that exposes two tools: `read_query` and `write_query`. Both accept `{query: string}`. If an agent selects `read_query` but passes a `DELETE FROM users` statement, schema validation sees nothing wrong: the parameter is a string, and the schema is satisfied. The tool name says "read," but the computation says "delete."

This is not a hypothetical failure. In our experiment, 5 of 8 agent errors are same-schema confusions where the agent selected a tool with identical parameters to the correct one. Schema validation catches zero of these.

We propose a minimal extension: three behavioral metadata fields per tool, hashed into a behavioral identity. The runtime compares the behavioral identity of the selected tool against the identity expected for the task. When they disagree, the call is blocked before dispatch.

The contribution is empirical. We do not propose a new theory. We measure a gap in current tool protocols, demonstrate a fix, and show that the fix is independently verifiable.

2 The Gap

2.1 What Tool Protocols Expose Today

Every major tool-calling protocol describes a tool with:
1. A **name** (string identifier, e.g., `read_query`).

2. A **schema** (JSON Schema for input parameters).
3. Optionally, a **description** (free-text, for model consumption).

The name is the dispatch key. The schema is the validation gate. The description influences model selection but is not machine-verifiable.

2.2 Where This Fails

Same-schema tools are indistinguishable to the runtime. Table 1 shows representative examples from real MCP servers.

Table 1: Same-schema tool groups in MCP servers.

Schema	Tools sharing it	Behavioral difference
`{query: string}`	read_query, write_query	PURE vs MUTATES
`{owner, repo, pull_number}`	get_pull_request, merge_pull_request	READ vs MERGE
`{q: string}`	search_code, search_issues	Different result types
`{path: string}`	read_file, write_file, delete_file	READ vs OVERWRITE vs DELETE

If an agent confuses tools within any of these groups, schema validation cannot detect the error. The name is wrong, but the parameters are valid.

2.3 The Cost of the Gap

Same-schema confusion is the most dangerous class of tool error because:

- The error is *silent*: no validation failure, no exception.
- The error is *semantic*: the wrong computation executes correctly on the right data.
- The error is *high-severity*: read/write confusion, merge/update confusion, create/delete confusion.

3 The Fix: Behavioral Identity

3.1 Three Fields

We propose adding three fields to every tool declaration:

Table 2: Behavioral metadata fields.

Field	Type	Values
`mutability`	enum	`PURE`, `MUTATES`
`action`	enum	`READ`, `SEARCH`, `CREATE`, `UPDATE`, `DELETE`, `MERGE`, `OVERWRITE`, `APPEND`
`output_domain`	enum	`DATA`, `CONTENT`, `STRUCTURE`, `DIFF`, `PR`, `ISSUE`, `REF`, `REPO`, `USER`, `ACK`

These fields are hashed into a **behavioral identity**:

$$\text{BI} = \text{SHA-256}(\texttt{mutability} \mid \texttt{action} \mid \texttt{output\_domain})$$

truncated to 16 hex characters.

3.2 What Changes

A tool declaration goes from:

```
{
  "name": "read_query",
  "schema": {"query": "string"}
}
```

to:

```
{
  "name": "read_query",
  "schema": {"query": "string"},
  "mutability": "PURE",
  "action": "READ",
  "output_domain": "DATA"
}
```

`write_query` with the same schema gets:

```
{
  "name": "write_query",
  "schema": {"query": "string"},
  "mutability": "MUTATES",
  "action": "OVERWRITE",
  "output_domain": "DATA"
}
```

Same schema, different behavioral identity. A runtime gate that compares behavioral identity against expected behavior now catches the confusion.

3.3 The Gate

When an agent selects a tool for a task, the runtime:
1. Resolves the tool by name (existing).
2. Validates parameters against schema (existing).
3. Compares the tool's behavioral identity against the expected identity for the task class (**new**).

If the identities do not match, the call is blocked before dispatch. The expected identity can come from: a task specification, a policy rule ("never call a MUTATES tool for a read-only task"), or a prior approved execution.

3.4 Why Three Dimensions

No single dimension separates all confusable pairs:

- **Mutability alone** separates read_query from write_query (PURE vs MUTATES) but not merge_pr from update_pr (both MUTATES).
- **Action alone** separates merge_pr from update_pr (MERGE vs UPDATE) but not search_code from search_issues (both SEARCH).
- **Output domain alone** separates search_code from search_issues (CONTENT vs ISSUE) but not read_query from write_query (both DATA).

The composition of all three is necessary and sufficient to separate every CRITICAL and HIGH severity confusable pair in our benchmark.

4 Experiment

4.1 Tools

45 tools from 4 MCP servers:

- **SQLite** (5 tools): read_query, write_query, create_table, list_tables, describe_table.
- **Filesystem** (15 tools): read, write, edit, search, move, delete, directory operations, streaming.
- **Git** (10 tools): status, diff, commit, log, show, branch operations.
- **GitHub** (15 tools): issues, PRs, search, repositories, branches, users.

4.2 Models

Three frontier language models: Claude Sonnet 4.6, GPT-4o, Gemini 2.5 Flash. Each model received the tool's JSON schema with the tool name removed. No examples, no descriptions beyond parameter names and types.

4.3 Phase A: Can Models Recover Behavioral Identity?

Each model classified each tool into the three behavioral dimensions. This tests whether behavioral identity is an objective property of the tool (independently recoverable) or a subjective label (models disagree).

Table 3: Cross-model agreement on behavioral classification.

Strict agreement (3/3 models)	26/45 (58%)
Majority agreement (2/3 models)	39/45 (87%)
Unknown/unclassifiable	0/135 (0%)

87% majority agreement means that for 39 of 45 tools, at least two of three independent models—with different training data, different architectures, different prompting behavior—converge on the same behavioral fingerprint from schema alone.

The 6 tools without majority agreement are genuinely ambiguous: `append_insight` (is appending a CREATE or an APPEND?), `create_directory` (does it mutate or merely declare?), `move_file` (is moving a CREATE, UPDATE, or OVERWRITE?), `create_or_update_file` (the name itself is ambiguous), `get_file_contents` (is the result CONTENT, STRUCTURE, or REPO-level?), `get_pull_request_files` (files of a PR: DIFF, DATA, or PR object?).

Implication. Behavioral identity is not arbitrary. If a tool author declares `mutability: PURE, action: READ, output_domain: DATA`, an independent consumer can verify this claim by asking a frontier model to classify the tool from its schema. If the consumer's model agrees, the declaration is corroborated. If it disagrees, the tool is flagged for manual review.

4.4 Phase A: Does Behavioral Identity Separate Confusable Tools?

15 confusable pairs—tools with identical or near-identical schemas—tested for separation by behavioral identity.

Every CRITICAL and HIGH severity pair is separated. The 5 unseparated pairs are genuinely similar tools: `get_pull_request` vs `get_pull_request_files` (both pure reads returning PR-related data), `list_directory` vs `directory_tree` (both list filesystem structure), `read_text_file`

Table 4: Separation of confusable pairs by severity.

Severity	**Pairs**	**Separated by BI**
CRITICAL (read/write, merge/update, create/delete)	3	3/3 (100%)
HIGH (write/edit, write/streaming, comment/update)	3	3/3 (100%)
MEDIUM	7	3/7 (43%)
LOW	2	1/2 (50%)
Total	15	10/15 (67%)

vs `get_file_contents` (equivalent functionality on different servers), and two more. These are tools that a human would also struggle to distinguish without reading the documentation.

4.5 Phase B: Does Behavioral Identity Catch Agent Errors?

30 adversarial prompts, each with a correct tool and a designated confusable wrong tool. All 45 tools presented simultaneously to each model. Prompts targeted same-schema groups with ambiguous phrasing:

- "Check how many sessions are expired and clean them out of the sessions table" (read or write?)
- "Verify the data in users—check if any ages are negative and correct them" (read or write?)
- "Mark issue #301 as a duplicate of #250 and close it" (update_issue or add_comment?)

Each prompt answered by all 3 models: 90 total decisions.

Table 5: Error detection: behavioral identity vs schema validation ($n = 90$).

Agent correct	82 (91%)
Agent wrong	8 (9%)
When the agent is wrong, who catches it?	
Both BI and schema catch	3
BI catches, schema misses	**4**
Schema catches, BI misses	**0**
Neither catches	1
BI catch rate	7/8 (88%)
Schema catch rate	3/8 (38%)

The headline. In our evaluation, behavioral identity strictly dominated schema validation:

- Behavioral identity catches every error that schema catches (3/3).
- Behavioral identity catches 4 additional errors that schema is blind to.
- Schema catches zero errors that behavioral identity misses.

The 4 identity-only catches. All are same-schema confusions. Prompts G02 and G23 use ambiguous phrasing that 2/3 models interpret as read-only. Both Claude and GPT-4o chose `read_query` instead of `write_query`. Schema sees nothing wrong: both accept `{query: string}`. Behavioral identity catches it: `read_query` is PURE|READ|DATA, `write_query` is MUTATES|OVERWRITE|DATA.

Prompt G15: Gemini chose `add_issue_comment` instead of `update_issue`. Both accept `{owner, repo, issue_number, ... }`. Schema sees nothing wrong. Behavioral identity catches it: UPDATE|ISSUE vs CREATE|ISSUE.

The 1 uncaught error. Claude chose `read_media_file` instead of `read_text_file`. Both are PURE|READ with similar output domains. Neither schema nor behavioral identity distinguishes them because the difference is content type (text vs binary), which is not captured in either the schema or the three behavioral fields. This is a genuine limitation: a fourth field (`content_type`) would catch it.

4.6 Out-of-Sample Evaluation

30 additional tools not seen during taxonomy development.

Table 6: Out-of-sample generalization (30 held-out tools).

Tools evaluated	30
Strict agreement (3/3)	17/30 (57%)
Majority agreement (2/3)	consistent with main
Unknown output domain	16/90 (18%)

57% strict agreement on held-out tools matches 58% on the development set. The taxonomy generalizes. The 18% unknown rate on output domain indicates room for additional categories, not a failure of the approach.

5 Cost-Benefit Analysis

Table 7: What the extension costs and what it buys.

Cost	**Benefit**
3 enum fields per tool declaration	Same-schema confusion becomes catchable
1 hash comparison per dispatch	88% error catch rate (vs 38%)
Author declares, consumer verifies	4 additional errors caught per 8
No changes to tool implementation	Zero schema-only catches (BI $\supseteq$ schema)

The cost is three lines in a JSON declaration. The benefit is that the most dangerous class of agent error—silent same-schema confusion—becomes visible to the runtime.

Backward compatibility. The three fields are additive. Existing tools without behavioral metadata continue to work. The gate degrades gracefully: if behavioral identity is absent, only schema validation runs (current behavior). Adoption is incremental.

6 Relation to Computational Identity

The behavioral identity proposed here is related to but distinct from the Computational Identity (CI) defined in Paper 12 [1].

CI operates on computation graphs: it canonicalizes a full expression plan and hashes the result. Two computations have the same CI if and only if they compute the same function. This requires access to the computation's internal structure—a parsed expression, a query plan, a DAG of operations.

Behavioral identity operates on opaque tools: it hashes three declared metadata fields. Two tools have different behavioral identities if their declared behaviors differ. This requires no access to the tool's implementation.

The tradeoff is precision for generality:

- CI is fine-grained: it can distinguish two SQL queries that read different tables. Behavioral identity cannot.
- Behavioral identity is universal: it applies to any tool that can be described by mutability, action, and output domain. CI requires a canonicalizable computation graph.

For tools that *do* expose computation graphs (SQL tools with query plans, DSL tools with expression plans), CI provides strictly finer identity. Behavioral identity is what remains when the computation graph is unavailable.

7 Limitations

1. **Scale.** 45 tools is a proof of concept. The MCP ecosystem has thousands of tools. Whether 87% agreement and 100% CRITICAL separation hold at scale is unknown.
2. **Taxonomy completeness.** The 7-category output domain is coarse. 18% of held-out tools produce unknown output types. The taxonomy needs extension for domains like ML pipelines, payment processing, and infrastructure management.
3. **The uncaught error.** Text vs binary content type is a real behavioral difference not captured by the three fields. A fourth field (`content_type`) would help.
4. **Adversarial robustness.** A malicious tool author can declare false behavioral metadata. Independent verification (asking a model to classify the tool) mitigates this, but is not foolproof.
5. **No production deployment.** We measure detection rates in a controlled experiment. We have not deployed the gate in a production agent system. Latency, false positive rates under real workloads, and user experience are unmeasured.
6. **Prompt sensitivity.** The adversarial prompts were designed to trigger same-schema confusion. The 9% error rate may not reflect real-world distributions. The detection *rates* (88% vs 38%) are what matter, not the absolute error count.

8 Related Work

Tool-calling protocols. MCP [2], OpenAI function calling, and LangChain tools all describe a tool as name + schema + optional human-readable description. None includes *machine-checkable* behavioral metadata. Anthropic's MCP specification defines tool annotations (`readOnlyHint`, `destructiveHint`) that partially overlap with our mutability field, but these are advisory hints for model consumption, not hashable identity components. They do not cover action type or output domain, and they cannot be compared at dispatch time because they lack a canonical form. The distinction is not whether behavioral information exists in the ecosystem—descriptions already carry some—but whether it is structured enough for a runtime gate to act on.

Capability-based security. Dennis & Van Horn (1966) and Miller (2006) define capabilities by what a component *can do* rather than who it *is*. Our behavioral fields are a lightweight capability

declaration. The difference: we hash them into a comparable identity, enabling runtime verification without a central authority.

Computational Identity. Paper 12 [1] defines CI for computation languages (DSL, SQL) where canonical plans are available. Behavioral identity is a coarser construct: it captures what kind of operation a tool performs, not the specific computation. The IronClaw case study (Paper 12 supplement) demonstrates CI for agent tool governance where computation graphs are available; behavioral identity addresses the complementary case where they are not.

9 Conclusion

Tool-calling protocols describe tools with a name and a schema. This is not enough. Same-schema tools are indistinguishable to the runtime, and same-schema confusion is the most dangerous class of agent tool error: silent, semantic, and high-severity.

Adding three behavioral fields—mutability, action type, and output domain—to tool declarations costs three lines of JSON and catches 88% of agent errors, compared to 38% for schema validation alone. On this benchmark, behavioral identity caught every error that schema caught. Schema caught zero errors that behavioral identity missed.

The behavioral fields are not arbitrary labels. Three independent language models, given only a tool's schema with the name removed, agree on the classification 87% of the time. Behavioral identity is recoverable from the tool's interface alone, without access to its implementation.

When name, schema, and behavioral identity all agree, the runtime has high confidence that the selected tool matches the intended operation. When they disagree, the runtime knows *something* is wrong—but resolving that disagreement, whether by blocking the call, prompting the agent, or exchanging additional execution information, is a question this paper does not address.

The proposal is minimal: three fields per tool, one hash comparison per dispatch, no changes to tool implementations. The experiment is controlled: 45 tools, 3 models, 90 decisions, all cached and reproducible. The result: on this benchmark, behavioral identity strictly dominated schema validation for detecting same-schema tool confusion.

References

[1] T. Dionysopoulos. Computational Identity. BLISP Research Program, Paper 12, 2026. DOI: 10.5281/zenodo.20830084.

[2] Anthropic. Model Context Protocol Specification. https://modelcontextprotocol.io, 2024.

A Artifact Inventory

Reproducibility. The experiment requires API keys for Claude, GPT-4o, and Gemini. Cached results are provided in the JSON files. All behavioral identity computations are deterministic given the cached classifications.

Table 8: Data and code locations.

Artifact	**Path**	**Format**
This draft	`artifacts/draft_13/`	LaTeX + PDF
Experiment code	`artifacts/paper13_protocol/dc_full_experiment.py`	Python
Phase A certificates	`artifacts/paper13_protocol/dc_full_phase_a_raw.json`	JSON
Phase B proposals	`artifacts/paper13_protocol/dc_full_phase_b_raw.json`	JSON
Aggregated results	`artifacts/paper13_protocol/dc_full_results.json`	JSON
Output type mapping	`artifacts/paper13_protocol/dc_full_raw_output_types.json`	JSON
Pilot report	`artifacts/paper13_protocol/dc_pilot_report.md`	Markdown
OOS evaluation	`artifacts/paper13_protocol/dc_pilot_oos_results.json`	JSON