MCP Tool Evaluation Benchmark Report

Why Your MCP Tools Fail (And How to Fix Them)

Arcade.dev

About This Report

This is the full technical analysis behind two companion blog posts. Everything is a Test introduces the Arcade Evals framework and how it works. Building High-Quality MCP Tools with Arcade Evals covers a practical checklist for writing schemas that models actually understand. This report goes deeper: full methodology, benchmark data across 14 models, and a real-world evaluation of the Figma MCP Server.

It's written for MCP server builders, agent developers evaluating third-party servers, and engineering teams deciding which models to pair with their tools.

Summary

We ran two experiments to measure how much MCP tool definitions affect LLM performance.

In Experiment 1, we built the same API twice: once with vague tool schemas (generic names, no descriptions, all strings) and once with descriptive schemas (clear names, format hints, documented enums). We tested both versions against 14 models across 3 runs each. Every single model failed the vague schemas on the Sprint Demo case. Every single model passed with the descriptive schemas. Even gpt-3.5-turbo and gpt-4.1-nano hit 100% with good definitions, after scoring 62% and 49% respectively with bad ones.

In Experiment 2, we evaluated Figma's official desktop MCP server, a real third-party MCP server we didn't write. The best-performing models (gpt-4.1, gpt-4o, claude-opus-4-5) passed every tool, but other high-end models like gpt-5 and claude-sonnet-4-5 showed inconsistencies on harder tools.

The core finding: tool definitions are multipliers on model capability. A vague schema handicaps even the best models. A good schema lets even the smallest models succeed. You can measure this before anything hits production.

Background

At Arcade, we maintain over 8,000 tools across 100+ integrations. At that scale, you can't manually QA every schema change. Poorly defined tools don't just cause test failures. They cause retries, wasted tokens, unpredictable agent behavior, and eroded user trust in production.

We built Arcade Evals to solve this: an automated framework that tests whether LLMs can select the right tool and fill arguments correctly, using only tool definitions and prompts. No tool execution required. This lets us catch schema problems before they reach users.

This report documents what we found when we turned that framework on both our own controlled experiment and a real-world, third-party MCP server.

The Problem

You built an MCP server. The tools work when you test them manually. But when you connect them to an LLM, things get weird. The model picks the wrong tool. It formats arguments incorrectly. It calls three tools when it should call one.

This isn't the model's fault. It's the tool definitions.

LLMs don't see your code. They see JSON schemas and descriptions. If those schemas are vague, the model guesses. Sometimes it guesses right. Often it doesn't.

We wanted to quantify this. How much do tool definitions actually matter? Can you predict which models will struggle with which tools? What makes a "good" schema?

To answer these questions, we ran two experiments using Arcade Evals.

What We Tested

Experiment 1: We built the same scheduling/messaging API twice. Once with vague tool definitions (generic names, no descriptions, all strings). Once with descriptive definitions (clear names, format hints, enum documentation). We ran identical test cases against both and compared results across 14 models.

Experiment 2: We evaluated Figma's official desktop MCP server. This let us test a real-world, third-party MCP server we didn't write, to see how well different models handle production-grade tool definitions.

All tests ran 3 times per configuration to measure stability. Pass threshold: 90%.

Setup

```
pip install 'arcade-mcp[evals]'
```

Models tested:

- GPT:
  gpt-3.5-turbo, gpt-4o, gpt-4o-mini, gpt-4.1, gpt-4.1-mini, gpt-4.1-nano, gpt-5, gpt-5-mini, gpt-5-nano
- Claude:
  claude-3-5-haiku-20241022, claude-haiku-4-5-20251001, claude-sonnet-4-20250514, claude-sonnet-4-5-20250929, claude-o

Experiment 1: Vague vs Descriptive Schemas

The Setup

We wrapped a simple API with two endpoints:

- /schedule — create a meeting
- /notify — send a message

Then we wrote two versions of the tool definitions.

The vague version is what you'd write if you were in a hurry. Generic names, no descriptions, string types for everything:

```
{
    "name": "schedule",
    "inputSchema": {
        "properties": {
            "name": {"type": "string"},
            "when": {"type": "string"},       # What format? ISO? "2pm"? "tomorrow"?
            "length": {"type": "string"},     # Minutes? Hours? "30m"? "PT30M"?
            "invitee": {"type": "string"},
            "location": {"type": "string"},   # Zoom URL? Address? Room name?
            "type": {"type": "string"},       # What are the valid options?
            "items": {"type": "array", "items": {"type": "string"}},
            "privacy": {"type": "string"},
        }
    }
}
```

The descriptive version is what you'd write if you knew the model would read it. Clear naming, explicit formats, documented enums:

```
{
    "name": "create_meeting",
    "description": "Schedule a video meeting with ISO formats and one attendee.",
    "inputSchema": {
        "properties": {
            "title": {"type": "string", "description": "Title Case meeting title"},
            "datetime": {"type": "string", "description": "ISO 8601 datetime (YYYY-MM-DDTHH:MM)"},
            "duration": {"type": "string", "description": "ISO 8601 duration (PT30M)"},
            "attendee": {"type": "string", "description": "Single attendee email"},
            "platform": {"type": "string", "description": "Video platform: Zoom | Google Meet | Teams"},
            "meeting_type": {"type": "string", "description": "Meeting type: demo | review | planning | 1:1"},
            "agenda_items": {"type": "array", "description": "Ordered list of agenda topics"},
            "privacy": {"type": "string", "description": "Visibility: internal | external"},
        }
    }
}
```

Both MCP servers also included decoy MCP tools (similar but wrong options) to test selection accuracy.

The Test Cases

Case 1 — Sprint Demo:
"Schedule a 30-minute Sprint demo with Alex on February 4th, 2026 at 2pm. Use Zoom. This is an internal demo. Agenda: demo, Q&A."

This tests:
Can the model format the datetime as ISO? Can it express duration as PT30M?
Does it pick the right enum value for meeting type?

Case 2 — Send to me (no identity):
"Send me an urgent project status update with subject 'Project Update'. CC Alex then Priya. No action required."

Context includes team roster but no user identity. The model should recognize it doesn't know who "me" is and call the identity tool first. Expected result: FAIL (intentionally).

Case 3 — Send to me (with identity):
Same message, but context includes a prior who_am_i call result. Now the model has everything it needs.

Results: Vague Track

Here's what happened when we ran the vague toolkit. The Sprint Demo case was a bloodbath.

Run 1

Model	Sprint Demo	Send+ID
gpt-4.1-nano	73% FAIL	90% FAIL
gpt-4.1-mini	73% FAIL	85% FAIL
gpt-4.1	73% FAIL	95% PASS
gpt-4o	68% FAIL	95% PASS
gpt-4o-mini	73% FAIL	95% PASS
gpt-3.5-turbo	62% FAIL	85% FAIL
gpt-5	70% FAIL	0% FAIL
gpt-5-mini	73% FAIL	90% FAIL
gpt-5-nano	62% FAIL	0% FAIL
claude-3-5-haiku	73% FAIL	85% FAIL
claude-haiku-4-5	73% FAIL	85% FAIL
claude-sonnet-4	73% FAIL	85% FAIL
claude-sonnet-4-5	73% FAIL	95% PASS
claude-opus-4-5	73% FAIL	90% PASS

Run 2

Model	Sprint Demo	Send+ID
gpt-4.1-nano	0% FAIL	90% FAIL
gpt-4.1-mini	73% FAIL	85% FAIL
gpt-4.1	70% FAIL	95% PASS
gpt-4o	68% FAIL	95% PASS
gpt-4o-mini	73% FAIL	95% PASS
gpt-3.5-turbo	62% FAIL	90% FAIL
gpt-5	70% FAIL	0% FAIL
gpt-5-mini	73% FAIL	90% FAIL
gpt-5-nano	62% FAIL	0% FAIL
claude-3-5-haiku	73% FAIL	85% FAIL
claude-haiku-4-5	73% FAIL	85% FAIL
claude-sonnet-4	73% FAIL	85% FAIL
claude-sonnet-4-5	73% FAIL	95% PASS
claude-opus-4-5	73% FAIL	90% FAIL

Run 3

Model	Sprint Demo	Send+ID
gpt-4.1-nano	73% FAIL	90% FAIL
gpt-4.1-mini	73% FAIL	85% FAIL
gpt-4.1	70% FAIL	90% PASS
gpt-4o	68% FAIL	95% PASS
gpt-4o-mini	73% FAIL	95% PASS
gpt-3.5-turbo	62% FAIL	85% FAIL
gpt-5	70% FAIL	95% PASS
gpt-5-mini	68% FAIL	90% FAIL
gpt-5-nano	62% FAIL	0% FAIL
claude-3-5-haiku	73% FAIL	85% FAIL
claude-haiku-4-5	73% FAIL	85% FAIL
claude-sonnet-4	73% FAIL	85% FAIL
claude-sonnet-4-5	73% FAIL	95% PASS
claude-opus-4-5	73% FAIL	90% FAIL

Aggregated (mean +/- std, pass rate)

Model	Sprint Demo	Send+ID
gpt-4.1-nano	48.7%+/-34.4%(0/3)	90%+/-0%(0/3)
gpt-4.1-mini	73%+/-0%(0/3)	86.7%+/-2.4%(0/3)
gpt-4.1	71%+/-1.4%(0/3)	93.3%+/-2.4%(2/3)
gpt-4o	68%+/-0%(0/3)	95%+/-0%(3/3)
gpt-4o-mini	73%+/-0%(0/3)	95%+/-0%(3/3)
gpt-3.5-turbo	62%+/-0%(0/3)	86.7%+/-2.4%(0/3)
gpt-5	70%+/-0%(0/3)	31.7%+/-44.8%(1/3)
gpt-5-mini	71.3%+/-2.4%(0/3)	90%+/-0%(0/3)
gpt-5-nano	62%+/-0%(0/3)	0%+/-0%(0/3)
claude-3-5-haiku	73%+/-0%(0/3)	85%+/-0%(0/3)
claude-haiku-4-5	73%+/-0%(0/3)	85%+/-0%(0/3)
claude-sonnet-4	73%+/-0%(0/3)	85%+/-0%(0/3)
claude-sonnet-4-5	73%+/-0%(0/3)	95%+/-0%(3/3)
claude-opus-4-5	72%+/-1.4%(0/3)	90%+/-0%(0/3)

What went wrong: Every single model failed the Sprint Demo case across all runs. The scores cluster around 68-73%, well below the 90% threshold.

The models got partial credit for picking the right tool and filling some arguments, but they couldn't nail the formatting. Without explicit hints, models passed "2pm" instead of "2026-02-04T14:00", "30 minutes" instead of "PT30M", and made up meeting types instead of using the expected enum values.

Notice gpt-4.1-nano's wild variance (48.7% +/- 34.4%). In Run 2, it scored 0% because it selected the wrong tool entirely. The vague schema gave it no signal to differentiate between "schedule" and the decoy "event" tool.

Results: Descriptive Track

Same test cases, same models. Only difference: better tool definitions.

Run 1

Model	Sprint Demo	Send+ID
gpt-4.1-nano	100% PASS	95% PASS
gpt-4.1-mini	100% PASS	95% PASS
gpt-4.1	100% PASS	100% PASS
gpt-4o	100% PASS	100% PASS
gpt-4o-mini	100% PASS	100% PASS
gpt-3.5-turbo	100% PASS	95% PASS
gpt-5	100% PASS	100% PASS
gpt-5-mini	100% PASS	95% PASS
gpt-5-nano	100% PASS	90% PASS
claude-3-5-haiku	100% PASS	95% PASS
claude-haiku-4-5	100% PASS	95% PASS
claude-sonnet-4	100% PASS	95% PASS
claude-sonnet-4-5	100% PASS	100% PASS
claude-opus-4-5	100% PASS	100% PASS

Run 2

Model	Sprint Demo	Send+ID
gpt-4.1-nano	97% PASS	90% PASS
gpt-4.1-mini	100% PASS	95% PASS
gpt-4.1	100% PASS	100% PASS
gpt-4o	100% PASS	100% PASS
gpt-4o-mini	100% PASS	100% PASS
gpt-3.5-turbo	100% PASS	90% PASS
gpt-5	100% PASS	100% PASS
gpt-5-mini	100% PASS	90% PASS
gpt-5-nano	100% PASS	90% PASS
claude-3-5-haiku	100% PASS	95% PASS
claude-haiku-4-5	100% PASS	95% PASS
claude-sonnet-4	100% PASS	95% PASS
claude-sonnet-4-5	100% PASS	100% PASS
claude-opus-4-5	100% PASS	100% PASS

Run 3

Model	Sprint Demo	Send+ID
gpt-4.1-nano	100% PASS	90% PASS
gpt-4.1-mini	100% PASS	95% PASS
gpt-4.1	100% PASS	100% PASS
gpt-4o	100% PASS	100% PASS
gpt-4o-mini	100% PASS	100% PASS
gpt-3.5-turbo	100% PASS	90% PASS
gpt-5	100% PASS	100% PASS
gpt-5-mini	100% PASS	95% PASS
gpt-5-nano	100% PASS	90% PASS
claude-3-5-haiku	100% PASS	95% PASS
claude-haiku-4-5	100% PASS	95% PASS
claude-sonnet-4	100% PASS	95% PASS
claude-sonnet-4-5	100% PASS	100% PASS
claude-opus-4-5	100% PASS	100% PASS

Aggregated (mean +/- std, pass rate)

Model	Sprint Demo	Send+ID
gpt-4.1-nano	99%+/-1.4%(3/3)	91.7%+/-2.4%(3/3)
gpt-4.1-mini	100%+/-0%(3/3)	95%+/-0%(3/3)
gpt-4.1	100%+/-0%(3/3)	100%+/-0%(3/3)
gpt-4o	100%+/-0%(3/3)	100%+/-0%(3/3)
gpt-4o-mini	100%+/-0%(3/3)	100%+/-0%(3/3)
gpt-3.5-turbo	100%+/-0%(3/3)	90%+/-0%(3/3)
gpt-5	100%+/-0%(3/3)	100%+/-0%(3/3)
gpt-5-mini	100%+/-0%(3/3)	93.3%+/-2.4%(3/3)
gpt-5-nano	100%+/-0%(3/3)	90%+/-0%(3/3)
claude-3-5-haiku	100%+/-0%(3/3)	95%+/-0%(3/3)
claude-haiku-4-5	100%+/-0%(3/3)	95%+/-0%(3/3)
claude-sonnet-4	100%+/-0%(3/3)	95%+/-0%(3/3)
claude-sonnet-4-5	100%+/-0%(3/3)	100%+/-0%(3/3)
claude-opus-4-5	100%+/-0%(3/3)	100%+/-0%(3/3)

The difference is stark. Every model passes every run. Even gpt-4.1-nano, which had 34% variance with vague schemas, now scores 99% +/- 1.4%. The model didn't get smarter. The schema got clearer.

This is the core finding: tool definitions are multipliers on model capability. A vague schema handicaps even the best models. A good schema lets even smaller models succeed.

Experiment 2:
Figma Desktop MCP Server

Why Test Someone Else's Tools?

You can't always control the tool definitions. When you're building an MCP gateway or integrating third-party servers, you need to know:

- Which models work best with these specific tools?
- Where are the weak spots?
- Can I ship this to production?

Figma's desktop MCP server is a good test case. It's a real, production tool with 6 endpoints covering design-to-code workflows. We didn't write the schemas, so this tests how well models handle "the real world."

Loading Tools from an MCP Server

```python
@tool_eval()
async def eval_figma_desktop() -> EvalSuite:
    suite = EvalSuite(
        name="Figma Desktop MCP Tools",
        system_message="You are a coding assistant with Figma access.",
        rubric=EvalRubric(fail_threshold=0.8, warn_threshold=0.9),
    )
    await suite.add_mcp_server(url="http://127.0.0.1:3845/mcp", timeout=10)
    return suite
```

The Test Cases

Each case targets a different tool with a specific challenge:

Case	Expected Tool	What Makes It Hard
Generate React component	get_design_context	Must parse node-id=42-789 to 42:789, infer forceCode=True
Screenshot for docs	get_screenshot	Must choose image over code generation, handle branch URL format
Layer structure overview	get_metadata	Must choose metadata over code context when user says "overview"
Extract design tokens	get_variable_defs	Must recognize "tokens" / "color palette" as variable extraction
FigJam whiteboard	get_figjam	Must distinguish /board/ URL (FigJam) from /design/ (Figma)
Design system rules	create_design_system_rules	No URL provided (uses current selection), must not confuse with code gen

Results: Figma Eval

Model	React	Screenshot	Metadata	Variables	FigJam	Design Rules
gpt-4.1	PASS	PASS	PASS	PASS	PASS	PASS
gpt-4o	PASS	PASS	PASS	PASS	PASS	PASS
claude-opus-4-5	PASS	PASS	PASS	PASS	PASS	PASS
claude-sonnet-4-5	PASS	PASS	PASS	PASS	PASS	2/3
claude-sonnet-4	PASS	PASS	PASS	2/3	PASS	2/3
gpt-4o-mini	PASS	2/3	PASS	PASS	PASS	1/3
gpt-4.1-mini	PASS	2/3	PASS	PASS	PASS	1/3
claude-haiku-4-5	PASS	2/3	PASS	2/3	PASS	1/3
claude-3-5-haiku	2/3	2/3	PASS	2/3	PASS	1/3
gpt-5-mini	PASS	1/3	PASS	2/3	PASS	1/3
gpt-4.1-nano	2/3	1/3	PASS	2/3	PASS	FAIL
gpt-5-nano	1/3	FAIL	2/3	1/3	PASS	FAIL
gpt-3.5-turbo	1/3	FAIL	PASS	1/3	PASS	FAIL
gpt-5	2/3	PASS	PASS	2/3	PASS	2/3

Reading this table: The top-left is green (big models, easy tools). The bottom-right is red (small models, hard tools). The pattern tells you where to focus.

get_figjam and get_metadata are easy. Every model passes these consistently. The tool definitions are clear, the use cases are unambiguous.

get_screenshot and create_design_system_rules are hard. Screenshot requires URL parsing (models pass the full URL instead of extracting the nodeId). Design rules is selection-dependent (no URL in the request), so models often call get_design_context first to gather information.

Smaller models struggle more. gpt-5-nano and gpt-4.1-nano fail multiple tools. But even they pass get_figjam and get_metadata, which shows this is about schema clarity, not raw model capability.

Failure Analysis

We categorized failures into three types:

Tool Selection Errors (picked the wrong tool)

React code generation:
- gpt-4.1-nano and gpt-5-nano called get_metadata instead of get_design_context
- They didn't recognize that "generate a React component" meant code generation

Metadata retrieval:
- gpt-5-nano called get_design_context instead of get_metadata when asked for "layer structure overview"
- The word "overview" wasn't a strong enough signal

Design rules:
- Several models called get_design_context instead of create_design_system_rules
- This case has no URL (it uses the current Figma selection), so models tried to fetch context first

Multiple Tool Call Errors (called extra tools)

Extract design tokens:
- gpt-4.1-mini, claude-haiku-4-5, claude-sonnet-4-5, claude-opus-4-5, and claude-sonnet-4 called 2 tools: get_variable_defs + get_design_context
- They wanted more context before the main action
- Our eval expects a single tool call, so this scores 0% even though the intent was reasonable

React code:
- claude-haiku-4-5 and gpt-5-nano called multiple tools instead of just get_design_context

Parameter Extraction Errors (right tool, wrong arguments)

nodeId parsing:
- gpt-5-mini and gpt-4.1-mini passed the full URL instead of extracting 100:200 from node-id=100-200
- The schema says "nodeId" but doesn't explicitly say "not the full URL"

artifactType inference:
- gpt-4o and gpt-5 chose REUSABLE_COMPONENT instead of COMPONENT_WITHIN_A_WEB_PAGE_OR_APP_SCREEN for a login form
- Subtle semantic difference that changes the code output

clientLanguages/clientFrameworks:
- gpt-5-nano and gpt-4.1-nano returned "unknown" or None instead of inferring from context

What This Tells Us

Figma's schemas are good, but not bulletproof. A few models (gpt-4.1, gpt-4o, claude-opus-4-5) passed everything, but some high-end models showed cracks on harder tools. Mid-tier and smaller models fail more consistently.

URL parsing is a common failure mode. If your tool expects a parsed value from a URL (like a nodeId), smaller models will often pass the whole URL. Be explicit: "Format: 123:456 (not the full URL)".

Selection-dependent tools are harder. When there's no URL to anchor the request, models make assumptions. If your tool depends on implicit context (like the current Figma selection), consider making that context explicit in the schema or in the system message.

Multi-tool calling is a design decision. Some models naturally want to gather context before acting. If your evals penalize this, you'll see false negatives. Decide upfront whether you're testing "single tool call" or "correct end state."

Putting It Into Practice

What Makes a Good Tool Schema

Based on our data, here's what consistently helps:

- Names are verb + object: create_meeting beats schedule
- Descriptions state constraints: "Does X. Does NOT do Y."
- Parameters match user language: datetime not when
- Formats are explicit: "ISO 8601 datetime (YYYY-MM-DDTHH:MM)"
- Enums are listed: "Meeting type: demo | review | planning | 1:1"
- No hidden requirements: If context is needed, make it a parameter

How to Apply This to the Figma Failures

The failure analysis above isn't just diagnostic. It's a roadmap for schema improvements. Here's what specific changes to Figma's schemas would likely fix the failures we observed:

nodeId parsing failures (get_screenshot, get_design_context):
Models passed the full URL instead of just the node ID. The current schema says "nodeId" but doesn't constrain the format. Adding a description like "Node ID in format 123:456. Extract from the URL's node-id parameter (replace the hyphen with a colon). Do NOT pass the full URL." would likely resolve this for mid-tier models.

Tool selection ambiguity (create_design_system_rules):
Models confused this with get_design_context because both tools relate to design information. The fix: add a negative constraint to the description. Something like "Creates design system rules from the current Figma selection. Does NOT generate code or retrieve design context. No URL required." Negative constraints ("does NOT do Y") are one of the most effective disambiguation tools we've found.

Implicit context (create_design_system_rules):
This tool relies on the current Figma selection, but that isn't explicit in the schema. When models don't see a required URL or file parameter, they try to gather context first (hence the multi-tool calls). Adding "Uses the active Figma selection. No file URL or node ID needed." would give the model the confidence to call this tool directly.

artifactType inference (get_design_context):
The enum values REUSABLE_COMPONENT and COMPONENT_WITHIN_A_WEB_PAGE_OR_APP_SCREEN are semantically close. Adding brief descriptions to each enum value ("REUSABLE_COMPONENT: standalone design element used across multiple screens" vs "COMPONENT_WITHIN_A_WEB_PAGE_OR_APP_SCREEN: a UI element that exists within a specific page layout, like a login form") would help models make the right call.

What Makes a Good Eval

To get reliable signal:

- Provide necessary context (unless you're testing "missing context" behavior)
- Decide upfront: single tool call or multi-step chain?
- Run 3+ times to measure stability
- Separate failure types: tool selection vs argument format vs extra calls
- Match critic strictness to intent: exact match for enums, similarity for free text

Limitations

Arcade Evals tests tool selection without execution. It won't catch:
- Output format issues (tool returns bad data)
- Context load problems (tool returns too much data)
- Error handling quality
- Runtime failures

Also: failing an eval doesn't mean the tool won't work. It means models will have a harder time getting expected behavior on the first try. Real agents can ask clarifying questions, call intermediate tools, and retry. But if you want reliable first-shot accuracy, these numbers matter.

Conclusion

Two findings stand out from this work.

First, tool definitions matter more than model selection. In Experiment 1, the gap between vague and descriptive schemas was larger than the gap between any two models. gpt-3.5-turbo with good schemas outperformed claude-opus-4-5 with bad ones. If you're spending time choosing between models but not investing in your tool definitions, you're optimizing the wrong variable.

Second, you can measure this before production. Every failure we found in both experiments was detectable with schema-level testing alone, no tool execution required. That means you can catch these problems in CI, not in production logs.

If you're building MCP tools, start by running your existing schemas through Arcade Evals. The results will tell you exactly where models struggle with your tools, and the failure categories in this report will tell you how to fix them.

If you're integrating third-party MCP servers, run evals against those tools with the models you plan to use. The Figma experiment shows that even well-built schemas have edge cases, and the model you choose determines whether those edges matter.

Reproduce These Results

```
pip install 'arcade-mcp[evals]'
arcade evals .
```

Docs: https://docs.arcade.dev/guides/create-tools/evaluate-tools/run-evaluations

Full source code:

Check out repo