OpenClaw Tool Call Failures: Why Models Break and How to Fix Them

February 14, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Most OpenClaw tool call failures come from three sources: the model doesn't support tool calling (DeepSeek Reasoner, Phi4, Llama 3 non-3.1), the chat template doesn't match the model's expected format, or session history got corrupted by a terminated tool call. Fix the first by switching to Qwen 2.5, Llama 3.1, or Claude. Fix the second by verifying API format in openclaw.json. Fix the third by clearing the session. Run `openclaw doctor --fix` as a first step. It catches most config issues automatically.

📚 **More on this topic:** Function Calling with Local LLMs · Best Models for OpenClaw · Structured Output from Local LLMs · OpenClaw Setup Guide · OpenClaw Model Routing

Your OpenClaw agent was working fine. Then it stopped doing things. It claims it completed a task but nothing happened. Or it loops endlessly, calling the same tool over and over. Or it just goes silent. No response, no error, nothing.

Tool call failures are the most common reason OpenClaw agents break, and the error messages (when they exist at all) rarely point at the actual problem. The agent might display a cryptic JSON error, a “tool result not found” message, or nothing whatsoever.

This guide covers every failure mode, what causes it, and how to fix it.

What Tool Call Failure Looks Like

Before debugging, identify which failure type you’re dealing with:

Symptom	Likely Cause	Section
Agent says it did something but didn’t	Model generated text instead of a tool call	Model doesn’t support tools
No response at all (silent failure)	Model lacks tool support, Ollama returns empty	Model doesn’t support tools
Agent loops calling the same tool		

Symptom	Likely Cause	Section
	Empty input, bad state, or confused model	Malformed JSON / Loop bugs
`tool result's tool id not found (2013)`	Corrupted session from terminated tool call	Session corruption
`tool_use ids were found without tool_result blocks`	Previous tool call never completed	Session corruption
Raw JSON error dumped to screen	Malformed tool call output from model	Malformed JSON
Agent works with Claude but fails with Ollama	Format mismatch between providers	Schema mismatches
Skills load but never execute	Tool definitions not sent to model	Config issue
`JSON parsing failed`	Model output preamble text before JSON	Model too small or wrong template

---

Root Cause 1: The Model Doesn't Support Tool Calling

This is the most common cause of silent failures. You swap to a local model, everything looks configured correctly, and the agent just stops doing things.

Not every model can make tool calls. The model needs to be specifically trained on tool-use data with special tokens and a chat template that handles the tool calling format. Without that training, the model either ignores the tool definitions entirely or tries to "fake" a tool call by writing JSON in plain text, which OpenClaw can't parse.

Models That Support Tool Calling

Model	Tool Support	Notes
Claude (Opus/Sonnet/Haiku)	Excellent	Native Anthropic tool format. Most reliable option.
GPT-4o / GPT-4o mini	Excellent	Native OpenAI function calling. Well-tested.
Qwen 2.5 (7B-72B)	Strong	0.933-0.971 F1 accuracy. Best local option for tools.
Qwen 3 (8B-32B)	Strong	Supports tool calling, good with Ollama.

Model	Tool Support	Notes
Llama 3.1 / 3.3 (8B-70B)	Strong	Native tool support, special tokens baked in.
Mistral 7B v0.3	Good	Fast, handles single tool calls well.
Mistral Small 24B	Good	Best agentic capabilities at mid-range size.
Granite 3.2	Good	Supports tool calling in Ollama.

Models That Don't (Or Barely Do)

Model	Tool Support	What Happens
DeepSeek Reasoner (R1)	No	Strong reasoning but not trained for function calling. Silent failure or plain-text JSON.
DeepSeek Chat	Limited	Requires OpenRouter workaround. No native Ollama tool support.
Phi-4	No	Does not support Ollama tool calling.
Llama 3 (non-3.1)	No	Returns `llama3 does not support tools` error.
Most community fine-tunes	Varies	Unless specifically trained for tools, assume no support.

The DeepSeek Problem

DeepSeek Reasoner is one of the best reasoning models you can run locally. It's also one of the worst for tool calling. Users regularly switch to DeepSeek for its thinking capabilities and then wonder why their agent stopped executing skills.

DeepSeek wasn't trained with tool-use tokens. When OpenClaw sends it a list of tools, DeepSeek either ignores them or outputs something that looks like a tool call in plain text but doesn't match the structured format OpenClaw expects. The agent receives no parseable tool call, produces no response, and you see nothing.

The fix: Use DeepSeek for reasoning-heavy tasks that don't require tools. For anything that needs to execute skills, route to Qwen 2.5 or Llama 3.1. Our model routing guide covers how to configure this in `openclaw.json`.

Root Cause 2: Malformed JSON Output

The model supports tools but produces broken output. This is where smaller models fail most often.

Common JSON Failures

Failure	What the Model Outputs	Why It Breaks
Preamble text	`Sure! Here's the tool call: {"name": ...}`	Parser expects raw JSON, chokes on English
Trailing comma	`{"args": {"city": "Tokyo",}}`	Valid JavaScript, invalid JSON
Missing brackets	`{"name": "search", "args": {"q": "test"`	Model hit token limit mid-output
Wrong types	`{"count": "5"}` instead of `{"count": 5}`	String instead of integer
Hallucinated function	`{"name": "super_search_v2", ...}`	Function doesn't exist in tool list
Empty input	`{"name": "search", "args": {}}`	Required parameters missing

Why Smaller Models Fail More

Tool calling accuracy scales with model size. From the function calling benchmarks:

Model Size	Tool Selection Accuracy	Failure Rate on Complex Calls
7B	0.933 F1 (Qwen 2.5)	~10-15% on multi-step
14B	0.971 F1 (Qwen 2.5)	~5-8% on multi-step
70B	94%+ (Llama 3.3)	~2-3% on multi-step
Cloud API	0.974 F1 (GPT-4)	<2% on multi-step

A 7B model picks the right tool 93% of the time for single calls. That sounds good until your agent makes 20 tool calls per task. At 93% per call, you're looking at roughly a 25% chance that at least one call fails in a 20-step workflow. At 97% (14B), that drops to about 45% chance of completing all 20 steps cleanly.

The Empty Input Loop

One specific failure pattern deserves attention: the model sends a tool call with empty arguments (`"args": {}`). OpenClaw executes the tool, which fails or returns an error. The model sees the error, tries again with empty arguments, and loops. This can burn through your API budget in minutes.

Signs: Your agent keeps calling the same tool. Logs show identical requests. Token usage spikes.

Fix: OpenClaw has loop detection, but it doesn't catch every case. If your agent is stuck, kill the task and clear the session. For local models, increase `repeat_penalty` to 1.2-1.3 in your modelfile to make the model less likely to repeat the same output.

Root Cause 3: Schema Mismatches (The "Different Dialects" Problem)

Every model family speaks a slightly different tool-calling language. OpenClaw has to translate between them. When the translation breaks, tools fail even though the model and the config both look correct.

The Three Main Formats

OpenAI format (used by GPT-4o, Ollama's API layer):

```
{
  "tool_calls": [{
    "function": {
      "name": "get_weather",
      "arguments": "{\"city\": \"Tokyo\"}"
    }
  }]
}
```

Anthropic format (used by Claude, OpenClaw internally):

```
{
  "type": "tool_use",
```

```
  "id": "toolu_abc123",
  "name": "get_weather",
  "input": {"city": "Tokyo"}
}
```

Hermes/XML format (used by some local models):

```
<tool_call>
{"name": "get_weather", "arguments": {"city": "Tokyo"}}
</tool_call>
```

The critical difference: Anthropic's format uses unique IDs to match every `tool_use` block with a corresponding `tool_result` block. If an ID is missing or mismatched, the API rejects the entire conversation. OpenAI's format doesn't have this requirement.

What Goes Wrong

Switching providers mid-session. If you start a conversation with Claude (Anthropic format) and then switch to GPT-4o (OpenAI format), orphaned `tool_use` blocks from the Claude session can corrupt the conversation for the OpenAI model. OpenClaw tries to translate, but the ID-matching requirement means some state doesn't convert cleanly.

Ollama chat template mismatches. Ollama converts the OpenAI-format API request into the model's native chat template. If the template is wrong (common with community-uploaded models), the model receives garbled tool definitions. It might output JSON that's correct for its training format but wrong for what Ollama expects back.

The `arguments` type mismatch. OpenAI format sends `arguments` as a JSON string (`"{\"city\": \"Tokyo\"}"`). Some templates expect it as an object (`{"city": "Tokyo"}`). When the types don't match, the parser fails silently.

Signs of a Template Mismatch

- Model outputs valid JSON but OpenClaw doesn't recognize it as a tool call
- Model responds with plain text describing what it would do instead of calling the tool
- Tools work with one model but not another, same config
- Logs show `finish_reason: "stop"` instead of `finish_reason: "tool_calls"`

Fixes

1. **Stick to one provider per session.** Don't switch between Claude and GPT-4o mid-conversation. Start a fresh session when changing providers.
2. **Use official Ollama models.** Community models often have incorrect templates. The official `qwen2.5`, `llama3.1`, and `mistral` models have tested templates.
3. **Check the chat template.** Run `ollama show <model> --template` and verify it includes tool handling sections. If it doesn't mention `tools` or `tool_calls`, the model won't handle them.

Root Cause 4: Context Overflow

Tool definitions eat tokens. Every tool registered with your agent gets injected into the prompt before your actual message. With enough tools, you can overflow the context window before the conversation even starts.

How Many Tokens Tools Cost

Setup	Token Overhead	Impact
1-3 simple tools	400-1,500 tokens	Negligible
5-10 tools with detailed schemas	3,000-5,000 tokens	Noticeable on 8K context models
MCP server (single, e.g. Jira)	~17,000 tokens	Significant
5 MCP servers	~55,000 tokens	Extreme. Exceeds most local model context.
58 tools from mixed sources	55,000+ tokens	Reported by Anthropic. Required optimization.

Each individual function definition costs roughly 400-550 tokens depending on how detailed the description and parameter schemas are. That adds up fast.

What Happens When Context Overflows

1. **Hard failure:** The request exceeds the model's context limit. You see errors like `input length and max_tokens exceed context limit`. The agent stops.
2. **Soft degradation:** The request fits but leaves almost no room for the actual conversation. The model "forgets" tool schemas partway through, producing malformed calls or ignoring tools entirely.

3. **Large tool results:** Your agent calls a tool that returns 20,000 tokens of JSON data. That result stays in context. The next tool call has 20,000 fewer tokens to work with. After a few large results, the context is full and the agent stalls.

Signs of Context Overflow

- Agent works for the first few messages, then starts failing
- Tool calls get less accurate as conversations get longer
- Agent ignores tools it was using successfully earlier in the conversation
- Error messages about exceeding context or token limits

Fixes

1. **Reduce tool count.** Keep active tools under 5-10 for local models. Only load tools relevant to the current task.
2. **Trim tool descriptions.** "Get weather for a city" works as well as "Retrieves comprehensive current weather information including temperature, humidity, wind speed, and precipitation for any specified city worldwide."
3. **Increase context window.** Set `num_ctx` in your Ollama modelfile. But watch VRAM: 16384 context on a 32B model at Q4 uses ~22GB.
4. **Purge session history.** Run "new session" before starting tasks with many tools. Old conversation history competes with tool definitions for context space.
5. **Summarize large tool results.** If a tool returns thousands of tokens, have the agent summarize the relevant parts before proceeding rather than keeping the raw output in context.

Root Cause 5: Session Corruption

This is the most frustrating failure because it's invisible. Your agent was working, something interrupted a tool call (process killed, timeout, crash), and now every subsequent request fails. The agent appears broken but the config is fine.

What Happens

OpenClaw tracks tool calls in session history files (`.jsonl`). Every `tool_use` block must have a matching `tool_result` block. If a tool call is interrupted before the result is recorded, the session file contains an orphaned `tool_use` without its `tool_result`.

From that point forward, every API request sends this corrupted history. The API rejects it:

```
invalid params, tool result's tool id not found (2013)
```

Or:

```
tool_use ids were found without tool_result blocks
```

Or:

```
Each 'tool_use' block must have a corresponding 'tool_result' block in the next message
```

OpenClaw tries to repair this automatically by inserting synthetic error results:

```
[openclaw] missing tool result in session history; inserted synthetic error result for transcript
```

But the repair can create its own problems. The synthetic result sometimes triggers a follow-up error:

```
unexpected 'tool_use_id' found in 'tool_result' blocks
```

Now you're stuck. The session is corrupted, the auto-repair made it worse, and every message fails.

The Fix

Clear the corrupted session:

1. Stop the agent
2. Delete the affected session's `.jsonl` file from `~/.openclaw/sessions/`
3. Restart

You lose the conversation history for that session, but the agent works again. If you need to preserve context, copy the important parts to the agent's memory file before deleting the session.

Prevention

- Don't kill the agent process while a tool call is in progress. Wait for the current operation to complete.
- Set reasonable timeouts so tools don't hang indefinitely.

- Use `maxConcurrent` limits on sub-agents to avoid overwhelming the system.

Root Cause 6: Ollama Streaming Bug

This one is specific and fixable. Ollama's streaming implementation doesn't properly emit `tool_calls` delta chunks. When OpenClaw sends `stream: true` (which it does by default), the model decides to call a tool, but the streaming response returns empty content with `finish_reason: "stop"`. The tool call is lost entirely.

The agent receives an empty response. No tool call, no text, nothing. From your perspective, the agent goes silent.

The Fix

If you're running Ollama as your provider, check whether your OpenClaw version handles this. Recent versions work around the streaming issue for tool calls, but if you're on an older version:

1. Update OpenClaw to the latest version
2. Update Ollama to the latest version
3. If still failing, check if your config allows disabling streaming for tool-enabled requests

How to Tell This Is Your Problem

- Agent works fine for chat (no tools) but fails silently when tools are involved
- Switching from Ollama to an API provider (Claude, GPT-4o) fixes the problem immediately
- Ollama logs show the model generating a tool call response but OpenClaw receiving nothing

---

Model Reliability Scorecard

Based on community reports and testing, here's how each model handles tool calling in OpenClaw:

Model	Tool Reliability	Common Failure Mode	Recommended for OpenClaw?
Claude Opus	Excellent	Rarely fails	Yes — best overall
Claude Sonnet	Excellent	Rarely fails	

Model	Tool Reliability	Common Failure Mode	Recommended for OpenClaw?
			Yes — best cost/quality ratio
Claude Haiku	Very Good	Occasionally oversimplifies complex tool chains	Yes — good for simple tasks
GPT-4o	Excellent	Rarely fails	Yes
GPT-4o mini	Good	Struggles with 5+ step chains	Yes for simple routing
Qwen 2.5 14B+	Strong	Occasional wrong argument types	Yes — best local option
Qwen 2.5 7B	Good	Eager invocation, multi-step degradation	Yes for single-step tools
Qwen 3 32B	Strong	Similar to Qwen 2.5 but better reasoning	Yes on 24GB
Llama 3.1 8B	Good	Bad-state loops after errors	Yes with guardrails
Llama 3.3 70B	Strong	Rarely fails	Yes if you have 48GB
Mistral 7B	Good	Parallel tool calls fail	Yes for single tools
Mistral Small 24B	Good	Template issues with some versions	Yes with verified template
DeepSeek R1	Poor	Not trained for tools. Silent failure.	No — use for reasoning only
DeepSeek Chat	Poor	Requires workarounds	No for tool-heavy work
Phi-4	None	Does not support tool calling	No

Debugging Step by Step

When tools fail, work through this sequence:

Step 1: Run the Doctor

```
openclaw doctor --fix
```

This catches and auto-repairs most configuration issues: missing providers, invalid JSON, broken permissions.

Step 2: Check Your Model

Does your model support tool calling? Verify:

```
# For Ollama models
ollama show <model-name> --template
```

Look for `tools`, `tool_calls`, or `function` in the template output. If none of these appear, the model doesn't handle tools.

Step 3: Test with a Simple Tool Call

Bypass OpenClaw and send a tool call directly to your model through the API:

```
curl http://localhost:11434/api/chat -s -d '{
  "model": "qwen2.5:7b",
  "messages": [
    {"role": "user", "content": "What is 5 + 3?"}
  ],
  "stream": false,
  "tools": [{
    "type": "function",
    "function": {
      "name": "calculator",
      "description": "Calculate a math expression",
      "parameters": {
        "type": "object",
        "properties": {
          "expression": {"type": "string"}
        },
        "required": ["expression"]
      }
    }
  }]
}'
```

If this works but OpenClaw doesn't, the problem is in OpenClaw's config, not the model. If this also fails, the problem is the model or Ollama.

Step 4: Check Session Health

If tools were working and suddenly stopped, session corruption is likely:

- Look at the latest `.jsonl` file in `~/.openclaw/sessions/`
- Search for orphaned `tool_use` blocks without matching `tool_result` blocks
- If found, clear the session

Step 5: Compare Providers

If you have both a local model and an API key configured, try the same task on each:

- Works on Claude but not Ollama → Ollama streaming bug or model doesn't support tools
- Works on GPT-4o but not Claude → Anthropic format issue, check for orphaned tool IDs
- Fails on both → The tool definition itself is broken, or context is overflowing

---

Config Fixes Reference

Verify API Format

Make sure your `openclaw.json` uses the correct API format for your provider:

```
{
  "models": {
    "providers": {
      "ollama": {
        "baseUrl": "http://127.0.0.1:11434/v1",
        "apiKey": "ollama-local",
        "api": "openai-completions"
      }
    }
  }
}
```

The `api` field matters. `"openai-completions"` tells OpenClaw to use the OpenAI-compatible format that Ollama expects. Wrong format = tools silently not sent.

Fix the Beta Flag Error (Bedrock/Vertex AI)

If you're routing through AWS Bedrock or Google Vertex AI and seeing:

```
ValidationException: invalid beta flag
```

Add this to your config:

```
{
  "beta_features": []
}
```

OpenClaw's SDK sends beta headers that Bedrock/Vertex don't support. Disabling them fixes the error.

Reduce Tool Count for Local Models

If you have MCP servers or many skills loaded, the total tool count might exceed what your model can handle. Check with:

```
openclaw skills list
```

Disable non-essential skills for sessions where reliability matters. Five well-defined tools work better than fifty poorly-defined ones on a 7B model.

Temperature for Tool Calls

Lower temperature produces more consistent structured output:

```
PARAMETER temperature 0.6
```

For tool-heavy workflows, 0.6-0.7 beats the default 0.8. The model needs to be precise with JSON syntax and argument types, not creative.

When to Switch Models vs Fix Config

Situation	Action
Model doesn't support tools at all (DeepSeek R1, Phi-4)	Switch models. No config fix will help.
Tools work sometimes but fail 10-20%	Fix config: lower temperature, trim tool count, increase context
Tools work for simple calls but fail on chains	Upgrade model size (7B → 14B → 32B)
Same model works via API but fails locally	Fix Ollama config: check template, update version, test streaming
Everything failed after a crash	Clear corrupted session, restart
Works with one provider, fails with another	Format mismatch. Start fresh session, don't mix providers.
Agent loops on same tool call	Increase repeat_penalty, add max iteration limits, clear session
`error 2013` or orphaned tool IDs	Session corruption. Delete the `.jsonl` file and restart.

Bottom Line

Most OpenClaw tool call failures aren't bugs in OpenClaw. They're model incompatibility, format mismatches, or corrupted sessions. The debugging process:

1. **Run `openclaw doctor --fix`** — catches config issues automatically.
2. **Verify your model supports tools** — DeepSeek R1, Phi-4, and base Llama 3 don't. Switch to Qwen 2.5, Llama 3.1, or Claude.
3. **Test the model directly** — Send a curl request to Ollama with a simple tool. If that fails, the problem is the model, not OpenClaw.
4. **Check for session corruption** — If tools suddenly stopped working after a crash, clear the session file.
5. **Match the format** — Don't switch providers mid-session. Use official Ollama models with tested templates.

For the most reliable tool calling setup: Claude Sonnet as primary with Qwen 2.5 14B as local fallback. For budget-only: Qwen 2.5 14B on 12GB VRAM with `temperature 0.6` and tools

limited to 5-10 per session. For the full model selection guide, see Best Local Models for OpenClaw.

Related Guides

- Function Calling with Local LLMs — how tool calling works, model benchmarks, working code
- Structured Output from Local LLMs — getting reliable JSON from models
- Best Local Models for OpenClaw — which models handle agent tasks
- OpenClaw Model Routing — route different tasks to different models
- OpenClaw Token Optimization — cut costs while keeping reliability
- OpenClaw Setup Guide — installation from scratch

Get notified when we publish new guides.

Subscribe — free, no spam
