

Agent Backbone – Weekly Reliability Briefing (Issue #2): 7 Ways MCP and Retry Plumbing Quietly Break Production Agents

Published: October 31, 2026

Publisher: Lali Publications

Web Edition & Free Weekly Briefing Signup: <https://tools.lalipublications.com/agent-backbone/>

Issue #2 Permalink: <https://tools.lalipublications.com/agent-backbone/issue-2/>

Previous Issue: <https://tools.lalipublications.com/agent-backbone/issue-1/>

Editorial Contact: larrys@lalipublications.com

The hook: Your agent retried a dropped tool call and refunded a customer twice. Your overnight run is still "retrying" a model that no longer exists. Your code-review bot approved a commit that isn't the one that got merged. None of these are model problems. They are plumbing problems, and all seven show up in public release notes and changelogs from AI coding agents and an app builder dated October 21–28, 2026.

Executive Summary

Last week's issue covered how agents quietly break inside a single conversation. This week the failures live one layer down: in the **transport, retry, and orchestration plumbing** that connects an agent to its tools, its workers, and its model provider. None of these bugs make a demo look worse. All of them make week-three production worse: a refund issued twice, an unattended run that never finishes, a subagent that "can't find" a tool that exists, a code review that approved the wrong commit.

Six of the failure modes below were fixed in public releases of widely used AI coding agents between October 21 and October 28, 2026; the seventh is a documented provider model retirement on an AI app builder. We re-checked each primary source on **October 29, 2026** and cite the exact version and date. The point is not that any one vendor got it wrong. The point is that teams with large engineering budgets shipped these bugs, which means your homegrown agent stack almost certainly has some of them too.

1. **Duplicate Side Effects on Mid-Call Disconnects** (a non-idempotent tool call re-sent after the connection dropped).
2. **Retrying What the Server Said Not to Retry** (retry storms and unattended runs that hang indefinitely).
3. **Subagents Blind to Late-Connecting MCP Servers** (tool registries snapshotted at spawn time).
4. **Cancellation That Never Reaches the Remote Server** (the client stops waiting, the server keeps working).
5. **Failed Worker Requests That Leave the Caller Hanging** (a swallowed error instead of an explicit failure).
6. **Reviewing a Stale Commit After a Late Push** (event-driven automation keyed to "latest" instead of a SHA).
7. **Silent Model Rerouting by the Provider** (a retired model ID quietly served by a different model with different cost and behavior).

Inside this issue:

- **Section 1:** The 7 Failure Modes Checklist (Symptom, Root Cause, Deterministic Fix)
- **Section 2:** Agent Backbone Briefing Issue #2, each failure mode in depth with Preferred vs. Rejected snippets
- **Section 3:** The Fixed 8-Question Reliability Regression Test Block
- **Section 4:** How to Score These Failures With the Locked 4-Criteria Rubric
- **Section 5:** Primary Sources (Verified October 29, 2026)

1. The 7 Failure Modes Checklist

#	Failure Mode	Production Symptom	Root Cause	Deterministic Fix
1	Duplicate Side Effects on Disconnect	Customer charged twice, two tickets opened, an email sent twice after a network blip.	The transport layer retries any call whose response was lost, including calls that write.	Retry automatically only tools a trusted server annotates as read-only or idempotent; for writes, send one idempotency key per logical action where the tool accepts one, and reconcile instead of resending when the outcome is unknown.
2	Retrying the Non-Retryable	A turn stalls for minutes; an overnight unattended run never finishes; error budget burned on a 404.	Retry policy keyed on "any error" instead of the server's retryable signal and error class.	Classify errors first: an explicit do-not-retry signal beats the status code; treat not-found, permission, and validation errors as terminal; refresh auth once on 401; honor <code>Retry-After</code> up to your time budget; add jitter, retry at one layer only, and cap attempts and total wall time.
3	Late MCP Connect Blindness	A subagent says "I don't have a tool for that" when the tool exists; behavior differs between fast and slow starts.	Tool list snapshotted when the subagent spawns, before every MCP server finished connecting.	Re-run <code>tools/list</code> when a server becomes ready, when the cached list's <code>ttlMs</code> expires, and on <code>notifications/tools/list_changed</code> (opt in via <code>subscriptions/listen</code>); gate spawn on required servers being ready, or fail loudly.
4	Cancellation Not Propagated	User hits stop, but the remote job keeps running, writing, and billing.	Local cancellation drops the waiting future without telling the server.	On cancel, timeout, or a dropped future, cancel the request on the wire: over MCP Streamable HTTP, close that request's response stream (the server must treat it as cancellation); over stdio, send <code>notifications/cancelled</code> with the request ID. Make servers stop work when they see it.
5	Hung Caller on Worker Failure	A UI spinner or orchestrator step waits forever; no error in logs.	The worker failed but never sent a response, so the caller's await has nothing to resolve.	Every request path returns exactly one terminal result: success or an explicit typed error; give every await a deadline that also aborts the request.
6	Stale-Commit Review	Automated review passes, but the merged code was never reviewed; or no review runs at all.	Review job resolves "latest commit" when it runs, and a late or out-of-order push event points it at the wrong head.	Pin every review to an explicit commit SHA; re-resolve the PR head before posting and don't post if it moved (the new head's run reviews it).

#	Failure Mode	Production Symptom	Root Cause	Deterministic Fix
7	Silent Model Rerouting	Same prompts, different answers; per-request cost shifts with no deploy on your side.	Provider retires a model ID and routes calls to a replacement automatically.	Pin full model IDs, log the model the response says served it, alert on any mismatch (or on cost-per-request jumps and canary drift where it isn't exposed), and re-run your regression set before accepting a swap.

2. Agent Backbone Briefing — Issue #2: 7 Ways MCP and Retry Plumbing Quietly Break Production Agents

1. Duplicate Side Effects When a Connection Drops Mid-Call

What goes wrong in production: An agent calls a tool over HTTP: `create_refund`, `send_invoice`, `open_ticket`. The server does the work, but the connection closes before the response arrives. The client sees a failure, retries, and the server does the work a second time. From the agent's point of view, one tool call happened. From your customer's bank statement, two did.

Verified example: Claude Code **2.1.307** (published to npm October 21, 2026) lists: *"Fixed HTTP MCP tool calls being sent twice when the connection closed mid-call; only read-only or idempotent tools are now retried"*.

Root cause: A lost response is ambiguous. The request may have failed before execution, or it may have succeeded with only the reply lost. Blind transport-level retries treat both cases as "safe to resend." In MCP revision 2026-07-28, a broken Streamable HTTP response stream can't be resumed. The in-flight request is lost, the server treats the disconnect as cancellation, and the changelog says clients **MUST** re-issue it as a new request with a new request ID. For a write, that re-issue is only safe with an idempotency key; without one, re-issue a status lookup instead of the raw write, because the server may have committed the write before it noticed the disconnect.

The deterministic fix:

- **Use tool annotations as the first gate.** The MCP specification (current revision 2026-07-28) defines `readOnlyHint`, `destructiveHint`, and `idempotentHint` on `ToolAnnotations`. Retry automatically only when a tool is read-only or idempotent. The spec also says annotations are **hints** and clients **MUST** treat them as untrusted unless they come from trusted servers, so pair them with your own server allowlist. Tool names are only unique within one server, so key that allowlist by (server, tool), never by tool name alone.
- **Make writes idempotent with a key.** MCP has no standard idempotency field, so this works only when the tool's `inputSchema` declares one (for example `idempotency_key`) and the server dedupes on it. Generate one key per logical action (not per attempt) and reuse it on every retry. Each retry is still a new JSON-RPC request with a new request ID. If the tool has no key, never resend: return an explicit unknown outcome and reconcile with a status lookup.

Rejected (retries every tool the same way):

```
def call(name, args):
    for attempt in range(3):
        try:
            return mcp.call_tool(name, args)
        except ConnectionError:
            continue # resends create_refund too
```

Preferred (retry only safe tools; keyed writes):

```
# Pseudocode: adapt names to your MCP client SDK.
import random
from time import sleep

# Key by (server, tool): tool names are only unique per server.
SAFE_TO_RETRY = {(srv.id, t.name)
                  for srv in servers if srv.id in TRUSTED_SERVERS
                  for t in srv.tools
                  if t.annotations.readOnlyHint
                     or t.annotations.idempotentHint)}
AMBIGUOUS = (ConnectionError, TimeoutError) # response lost: outcome unknown

def call(server, name, args, action_id, reconcile_step):
    tool = registry.get(server, name)
    keyed = "idempotency_key" in tool.inputSchema.get("properties", {})
    if keyed: # only where the schema allows it
        args = {**args, "idempotency_key": action_id} # one key per logical action
    safe = (server, name) in SAFE_TO_RETRY or (keyed and server in TRUSTED_SERVERS)
    read_only = (server in TRUSTED_SERVERS and tool.annotations is not None
                 and bool(tool.annotations.readOnlyHint))
    attempts = 3 if safe else 1
    for attempt in range(1, attempts + 1):
        try:
            return mcp.call_tool(server, name, args) # new request ID each attempt
        except AMBIGUOUS:
            if attempt < attempts:
                sleep(random.uniform(0, min(8, 0.5 * 2 ** attempt))) # full jitter
    if read_only:
        return {"status": "error", "code": "TOOL_UNREACHABLE", "tool": name}
    # Anything that writes may have committed before the response was lost.
    return {"status": "unknown_outcome", "action_id": action_id,
            "next_step": reconcile_step}
```

The key detail in the Preferred version: when a write's outcome is unknown, the agent gets an explicit `unknown_outcome` result and a reconcile step, not a silent second attempt.

2. Retrying Requests the Server Marked Non-Retryable

What goes wrong in production: The model API or a downstream service returns an error and explicitly says it should not be retried. The client retries anyway, with backoff, again and again. In an interactive session, the user waits minutes. In an unattended overnight run, the job never ends and never reports failure.

Verified examples:

- Claude Code **2.1.311** (published to npm October 26, 2026): *"Fixed Claude Code retrying requests the API marked not to retry, which could hold a turn for minutes or, in unattended retry mode (CLAUDE_CODE_RETRY_WATCHDOG), indefinitely".*

- OpenAI Codex CLI **0.165.0** (changelog dated October 27, 2026): *"Stopped retrying when a provider reports that a model was not found, preserving its error message."*

Root cause: Retry logic written as "catch error, back off, try again" with no error classification. A model-not-found or explicit do-not-retry response will never succeed on attempt 50.

The deterministic fix:

- **Classify before you retry.** An explicit server signal beats the status code (Anthropic's SDKs, for example, obey an `x-should-retry` header). Terminal: 400/422 validation, 403 permission, 404 not found (including unknown model IDs), 501, quota-exhausted 429s, and any explicit do-not-retry signal. On 401, or an MCP 403 `insufficient_scope`, refresh or re-authorize once, then treat a second failure as terminal. Retryable: connection resets, 408, 429, 500, 502, 503, 504, and provider overload codes such as 529. For writes, retry these only when the request carries an idempotency key. An MCP tool result with `isError: true` is not a transport failure; return it to the model instead of retrying.
- **Honor Retry-After** (seconds or an HTTP date) on 429/503 as the minimum wait, but cap it: if the server asks for longer than your remaining time budget, stop now and surface the error instead of sleeping.
- **Cap both attempts and total wall time**, and surface the original error message to the agent and the logs. Unattended mode should fail fast and loudly, never wait forever.
- **Add jitter and retry at one layer only.** Use exponential backoff with full jitter so clients don't retry in lockstep. Turn off retries in either the inner layers (SDK, HTTP client) or the outer ones (agent loop). Three layers that each make up to 4 attempts can send $4 \times 4 \times 4 = 64$ requests for one call. Stop calling a dependency whose recent failure rate crosses a threshold (a circuit breaker).

Rejected:

```
retry:
  on: any_error
  max_attempts: unlimited # "unattended mode should be resilient"
  backoff: exponential
```

Preferred:

```
# Illustrative config, not a specific library's schema
retry:
  layer: agent_client # SDK/HTTP-client retries off (max_retries: 0)
  precedence: explicit_signal_over_status # e.g. x-should-retry: false wins
  terminal: [400, 403, 404, 422, 501, model_not_found, quota_exhausted, do_not_retry_signal]
  refresh_auth_once_then_terminal: [401, insufficient_scope_403]
  retryable: [408, 429, 500, 502, 503, 504, 529, connection_reset]
  retryable_writes: idempotency_key_required # otherwise return unknown_outcome
  tool_result_is_error: return_to_model # not a transport retry
  honor_retry_after: { enabled: true, fail_fast_if_over_budget: true }
  backoff: { type: exponential, base_seconds: 1, max_seconds: 30, jitter: full }
  max_attempts: 4
  max_total_seconds: 120
  on_exhausted: return_error_with_original_message # never hang
```

3. Subagents Blind to Tools From Late-Connecting MCP Servers

What goes wrong in production: Your orchestrator spawns a subagent while one MCP server (say, a slow remote CRM connector) is still connecting. The subagent's tool list is built at spawn time, so the CRM tools never appear.

The subagent tells the user it has no way to look up the account, or worse, guesses. The bug is timing-dependent, so it passes in staging and fails on cold starts.

Verified example: Claude Code **2.1.311** (published to npm October 26, 2026): *"Fixed subagents being unable to use MCP tools from servers that finish connecting after the subagent starts"*.

Root cause: The tool registry is treated as immutable for the life of a subagent, while MCP servers connect asynchronously and their tool lists can change.

The deterministic fix:

- **Refresh the registry on connect and on change.** In MCP 2026-07-28, servers that declare `"tools": {"listChanged": true}` send `notifications/tools/list_changed` only to clients that opened a `subscriptions/listen` stream with `toolsListChanged: true`, so open one for every server that supports it. Re-run `tools/list` and update every live subagent's toolset when the notification arrives, when a server becomes ready, or when the cached list's `ttlMs` expires.
- **Declare required servers per task.** If a task needs the CRM server, either wait for it (with a deadline) before spawning or return an explicit "required tool server unavailable" error. Never let the model improvise around a missing tool.

Rejected:

```
def start(task):
    return spawn_subagent(task, tools=registry.snapshot()) # frozen at t=0
```

Preferred:

```
# Pseudocode: adapt to your orchestrator. Register handlers before spawning.
# once, at startup
registry.on("server_ready", registry.refresh)           # re-run tools/list
registry.on("notifications/tools/list_changed", registry.refresh) # needs subscriptions/listen

async def start(task):
    await registry.wait_for(task.required_servers, timeout_s=15) # raises -> explicit error
    return spawn_subagent(task, tools=registry.live_view())      # sees every refresh
```

4. Remote MCP Cancellation That Never Reaches the Server

What goes wrong in production: A user hits stop, or your orchestrator times out a slow tool call. Locally, the request is abandoned. Remotely, the server has no idea and keeps running a long query, a batch write, or a paid API loop. You pay for work nobody will read, and side effects land after the user thinks the action was stopped.

Verified example: OpenAI Codex CLI **0.165.0** (changelog dated October 27, 2026): *"Cancelled remote MCP tool requests when their local response futures are dropped."*

Root cause: Cancellation is implemented as "stop awaiting" rather than "tell the other side to stop."

The deterministic fix:

- **Propagate cancellation on every exit path** (user stop, timeout, parent task cancelled, future dropped). In MCP 2026-07-28 the mechanism depends on transport. Over Streamable HTTP, closing the request's response stream *is* the cancellation; no `notifications/cancelled` is required or expected. Over stdio, the client sends

notifications/cancelled with the request ID and an optional reason. Either way, ignore late responses, and treat a write's outcome as unknown until you check, because the server may have finished before the cancel arrived.

- **Set timeouts per tool.** If a tool sends progress notifications, you may extend the wait on each one, but keep a hard maximum; the cancellation spec says implementations SHOULD always enforce one.
- **Make servers honor it:** check for cancellation between steps of long-running tools, and make partial writes either transactional or reported.
- **Log it:** record `cancel_sent` with the request ID and time, then whether any response or side effect appeared after it. MCP has no cancel acknowledgment, so a late side effect is the only evidence that the cancel lost the race.

Rejected:

```
import asyncio

async def call(client, name, args):
    try:
        return await asyncio.wait_for(client.call_tool(name, args), timeout=30)
    except asyncio.TimeoutError:
        return "Tool timed out"    # server may still be running; no cancel on the wire
```

Preferred:

```
# Pseudocode: assumes your SDK cancels the request on the wire (closes the HTTP
# stream, or sends notifications/cancelled on stdio) when the awaiting task is
# cancelled. Verify that with test Q5 below.
import asyncio

async def call(client, name, args):
    try:
        async with asyncio.timeout(tool_timeout_s(name)):    # Python 3.11+; per tool
            return await client.call_tool(name, args)
    except TimeoutError:
        log.info("cancel_sent", tool=name, reason="timeout")
        return {"status": "cancel_requested", "outcome": "unknown_until_checked", "tool": name}
    except asyncio.CancelledError:
        log.info("cancel_sent", tool=name, reason="user_stop_or_parent_cancel")
        raise    # never swallow cancellation
```

5. Failed Worker Requests That Leave the Caller Hanging

What goes wrong in production: A UI process or orchestrator sends a request to a worker. The worker throws, crashes, or hits an unhandled branch, and never answers. The caller awaits a response that will never come. There is no error in the logs, just a spinner or a stuck pipeline step.

Verified example: OpenCode **v1.19.5** (GitHub release published October 28, 2026 UTC; late October 27 Pacific):
"Failed TUI worker requests now return an error instead of leaving the caller hanging." (PR #54936)

Root cause: The request/response contract only defines the success path. Failure is "no message," which is indistinguishable from "still working."

The deterministic fix:

- **Exactly one terminal response per request:** build the success or typed-error result inside `try / catch`, then send it with a single `reply` after the block, so a failing `reply` can't trigger a second one.
- **A deadline on every await** on the caller side that also aborts the request (so the worker stops too), converted into an explicit error the agent and user can act on.

Rejected:

```
worker.on("request", async (req) => {
  const out = await handle(req);          // throws -> no reply ever sent
  worker.reply(req.id, { ok: true, out });
});
```

Preferred:

```
worker.on("request", async (req) => {
  let res;
  try {
    res = { ok: true, out: await handle(req) };
  } catch (err) {
    res = { ok: false, error: { code: "WORKER_FAILED", message: String(err) } };
  }
  worker.reply(req.id, res); // exactly one reply on every path
});

// caller side: the deadline also aborts the request, so the worker stops too
const res = await call(req, { signal: AbortSignal.timeout(20_000) }).catch((err) => ({
  ok: false,
  error: { code: err?.name === "TimeoutError" ? "WORKER_TIMEOUT" : "WORKER_FAILED", message:
String(err) },
})));
```

6. Code Review Running on a Stale Commit After a Late Push

What goes wrong in production: An automated reviewer (human-in-the-loop bot, AI review, or CI gate) is triggered by a push event. Events arrive late or out of order, and the job reviews whatever commit it resolves at run time, or the wrong one. The check goes green on code that is not what gets merged.

Verified example: Claude Code **2.1.312** (published to npm October 27, 2026): *"[Code Review] Fixed Code Review reviewing an outdated commit, or none at all, when GitHub delivered a push event late; it now reviews the pull request's current commit"*.

Root cause: The review is keyed to an event or to "latest," not to an immutable commit identity.

The deterministic fix:

- **Pin every review run to a commit SHA** and print it in the review output.
- **Re-check the PR head before posting.** If `head.sha` changed while the review ran, don't post. The new push's own run reviews the new head, and a `concurrency` group with `cancel-in-progress` stops the stale run early.
- **Make merge gates require a review result whose SHA equals the merge candidate's SHA.**

Rejected (fragment):

```
# fragment: steps excerpt, not a complete workflow
on: push
steps:
  - run: git checkout origin/${BRANCH}      # whatever is latest when the job starts
  - run: ai-review --post-result
```

Preferred (complete workflow; `ai-review` stands in for your review CLI):

```
name: ai-review
on:
  pull_request:
    types: [opened, synchronize, reopened]
concurrency:
  group: ai-review-${{ github.event.pull_request.number }}
  cancel-in-progress: true      # a new push cancels the stale run
permissions:
  contents: read
  pull-requests: write
jobs:
  review:
    runs-on: ubuntu-latest
    env:
      SHA: "${{ github.event.pull_request.head.sha }}"
      PR: "${{ github.event.pull_request.number }}"
      GH_TOKEN: "${{ github.token }}"
    steps:
      - uses: actions/checkout@v7
        with:
          ref: "${{ github.event.pull_request.head.sha }}"
      - run: ai-review --sha "$SHA" --out review.json
      - name: Post only if the PR head has not moved
        run: |
          CURRENT=$(gh pr view "$PR" --repo "$GITHUB_REPOSITORY" --json headRefOid -q .headRefOid)
          if [ "$CURRENT" != "$SHA" ]; then
            echo "Head moved to $CURRENT; not posting review for $SHA"
            exit 0
          fi
          ai-review --post-result review.json --sha "$SHA"
```

7. Silent Model Rerouting by the Provider

What goes wrong in production: You never changed a line of code, but answers drift and the bill moves. The provider retired the model ID you call and now serves those requests with a different model.

Verified example: Lovable's changelog (entry dated October 22, 2026): Gemini 2.5 Pro, 2.5 Flash, and 2.5 Flash Lite chat models for app AI features *"retired on October 20, 2026."* Requests for 2.5 Pro and Flash now go to Gemini 3.8 Flash, and Flash Lite requests go to Gemini 3.1 Flash Lite; *"the replacement can cost a different amount per request and respond differently."* To its credit, Lovable documented the change publicly, and rerouting keeps apps running. The architectural lesson is that **"keeps working" is not the same as "behaves the same,"** and your agent needs to notice.

Root cause: The application treats a model name as a stable contract. In practice it is a lease with an end date.

The deterministic fix:

- **Pin full model IDs** in config, never "latest" aliases, and keep a dated retirement calendar for every model you call.
- **Log the served model** from each response's metadata where the API exposes it, and alert when it differs from the requested one. Where the served model isn't exposed (for example, a platform's built-in AI gateway), alert on a jump in cost per request and run a small fixed canary set daily to catch behavior changes.
- **Treat any model change as a deploy:** re-run the 8-question regression set and compare weighted scores and cost per request before accepting it.

Rejected:

```
{ "model": "fast-default", "alerts": [] }
```

Preferred:

```
{
  "model": "provider/model-name-2026-xx-xx",
  "on_served_model_mismatch": "alert_and_flag_responses",
  "if_served_model_hidden": ["alert_on_cost_per_request_jump", "daily_canary_set"],
  "retirement_dates": { "provider/model-name-2026-xx-xx": "2027-xx-xx" },
  "retirement_check": "weekly",
  "on_model_change": ["run_regression_8q", "compare_cost_per_request", "require_signoff"]
}
```

3. The Fixed 8-Question Reliability Regression Test Block

Issue #1's template tested conversational behavior. This block tests plumbing. Run it in staging with fault injection (drop connections, delay servers, return synthetic errors) before and after any change to your tool layer, retry policy, orchestration, or model config.

Each slot targets exactly one failure mode, so a failure can be attributed. Every criterion is binary pass/fail with a numeric threshold; score the agent's user-facing reply separately with the rubric in Section 4.

Slot	Mode	Fault-Injection Test	Deterministic Pass Criterion (pass/fail; score the user-facing reply separately with the rubric)
Q1	1	<i>"Refund order #1042, then tell me its status."</i> Drop the connection after the server executes <code>create_refund</code> but before the response arrives. Then drop the connection once on the read-only <code>get_order</code> call.	Exactly 1 refund record exists for the action ID. <code>create_refund</code> was sent exactly 1 time, or at most 3 times all carrying the same idempotency key. The agent reports an unknown outcome or the reconciled result, never an unverified "done". <code>get_order</code> makes at most 3 attempts in total and returns the correct status, with 0 errors shown to the user.
Q2	2	During an unattended run, return a model-not-found error from the model API. In a second run, return a 503 carrying an explicit do-not-retry signal. Optional third run: a tool returns a result with <code>isError: true</code> .	Runs 1 and 2: each sends exactly 1 request to the model API and ends with a failed status within 10 s. The original error message appears verbatim in the logs and in the user-facing reply. Run 3: the tool result reaches the model exactly once, with 0 transport retries.

Slot	Mode	Fault-Injection Test	Deterministic Pass Criterion (pass/fail; score the user-facing reply separately with the rubric)
Q3	2	Return 429 with <code>Retry-After: 20</code> three times, then succeed. In a second run, return 429 with <code>Retry-After: 600</code> .	Run A makes exactly 4 attempts, waits ≥ 20 s before each retry, finishes in ≤ 120 s, succeeds, and logs all 4 attempts. Run B makes 0 retries and fails within 5 s with the original error (the requested wait exceeds the 120 s budget).
Q4	3	Delay the CRM MCP server's readiness by 10 s, then ask a subagent "What plan is account ACME-77 on?" Repeat with a 30 s delay.	Run A: the subagent calls the CRM tool at least once and states the plan the tool returned. Run B: within 16 s of the question it returns "required tool server unavailable". In both runs: 0 replies say the tool doesn't exist, and 0 replies state a plan the tool didn't return.
Q5	4	Start a 60 s remote export, then press stop after 5 s.	Within 2 s of stop, the server log shows that request ID cancelled (HTTP: client closed the response stream; stdio: notifications/cancelled received). 0 export files are written after the stop timestamp. The agent says the export was stopped or must be checked; 0 replies say "completed".
Q6	5	Make the worker throw on <code>summarize_ticket</code> . In a second run, kill the worker process mid-request.	Run A: the caller gets exactly 1 reply, <code>WORKER_FAILED</code> , within 2 s of the throw. Run B: the caller gets <code>WORKER_TIMEOUT</code> between 20 s and 21 s after the request. In both runs, the agent says the step failed and gives exactly 1 next step.
Q7	6	Push a new commit to a PR while its automated review is running.	0 reviews are posted for the old SHA after the push. Exactly 1 review naming the new head SHA is posted. The merge gate blocks 100% of merge attempts until a result whose SHA equals the merge candidate exists.
Q8	7	In staging, have a mock provider answer requests for the pinned model ID with a different served model.	An alert fires within 60 s of the first mismatched response. 100% of mismatched responses are flagged in the logs. The 8-test regression run starts automatically within 1 hour. The model config is not accepted without 1 recorded sign-off.

The 4-Step Reliability Measurement Loop

1. **Baseline Run:** Execute all 8 fault-injection tests and record pass/fail, duplicate side-effect count, time-to-terminal-error, and weighted rubric score of the agent's user-facing reply.
2. **Install the Fixes:** Retry classification, idempotency keys where tools accept them, annotation-gated retries keyed by (server, tool), registry refresh, cancellation propagation, worker deadlines, SHA pinning, and model pinning with alerts.
3. **Production Soak:** Run live traffic for several days with the same logging.
4. **Post-Fix Re-Run:** Re-run the identical 8 tests and compare: all 8 tests should meet their numeric thresholds, duplicate writes should be zero, hangs should be zero, and every failure should end in an explicit, actionable message.

4. Scoring Note: Applying the Locked 4-Criteria Rubric to Plumbing Failures

Infrastructure bugs still show up as bad replies, so score the agent's user-facing turn on each test with the same locked weights from Issue #1 (1–5 per criterion):

Weighted Score = (0.35 * Accuracy & Policy Adherence) + (0.25 * Conciseness & Clarity) + (0.20 * Tone & Brand Alignment) + (0.20 * Actionability)

- **Accuracy & Policy Adherence (35%):** Does the reply state what actually happened? "Your refund was issued" after an unknown-outcome write scores **1**. "The connection dropped mid-refund, so I checked with the payment system: the refund was issued once. No further action needed." scores **5**.
- **Conciseness & Clarity (25%):** A hung run or retry loop that floods the user with repeated "trying again" messages scores low. One clear status line scores high.
- **Tone & Brand Alignment (20%):** Calm, direct failure reporting without blame-shifting or over-apology.
- **Actionability (20%):** Every failure path ends with one concrete next step (retry later, check status, reconnect a server) or an explicit limit. A spinner that never resolves scores **1** by definition.

Group failures into Primary / Secondary / Minor weaknesses by frequency exactly as in the full rubric: [download the rubric PDF](#).

5. Primary Sources (All Re-Checked October 29, 2026)

Failure Mode	Source	Version & Date
1. Duplicate MCP tool calls on disconnect	Claude Code CHANGELOG: https://github.com/anthropics/claude-code/blob/main/CHANGELOG.md	2.1.307, npm publish Oct 21, 2026
2. Retrying non-retryable requests	Claude Code CHANGELOG (above)	2.1.311, npm publish Oct 26, 2026
2. Model-not-found treated as terminal	OpenAI Codex changelog: https://developers.openai.com/codex/changelog	Codex CLI 0.165.0, Oct 27, 2026
3. Subagents and late-connecting MCP servers	Claude Code CHANGELOG (above)	2.1.311, npm publish Oct 26, 2026
4. Remote MCP cancellation	OpenAI Codex changelog (above)	Codex CLI 0.165.0, Oct 27, 2026
5. Failed worker request hangs caller	OpenCode release: https://github.com/anomalyco/opencode/releases/tag/v1.19.5	v1.19.5, Oct 28, 2026 (UTC)
6. Code Review on stale commit	Claude Code CHANGELOG (above)	2.1.312, npm publish Oct 27, 2026
7. Silent model rerouting	Lovable changelog: https://docs.lovable.dev/changelog	Entry dated Oct 22, 2026 (retirement Oct 20, 2026)

Failure Mode	Source	Version & Date
Tool annotations, list_changed via subscriptions/listen, tool errors (isError), tool-name scope	MCP specification: https://modelcontextprotocol.io/specification/2026-07-28/server/tools	Revision 2026-07-28 (current)
Cancellation (HTTP stream close vs. stdio notifications/cancelled), timeouts	https://modelcontextprotocol.io/specification/2026-07-28/basic/utilities/cancellation	Revision 2026-07-28
ToolAnnotations fields and defaults, ttls	https://modelcontextprotocol.io/specification/2026-07-28/schema	Revision 2026-07-28
No SSE stream resumability; re-issue with a new request ID	MCP changelog: https://modelcontextprotocol.io/specification/2026-07-28/changelog	Revision 2026-07-28
401 and 403 insufficient_scope re-authorization	https://modelcontextprotocol.io/specification/2026-07-28/basic/authorization	Revision 2026-07-28
x-should-retry checked before status code	Anthropic Python SDK: https://github.com/anthropics/anthropic-sdk-python/blob/main/src/anthropic/_base_client.py	main branch, checked Oct 29, 2026

Seen in Public Threads This Week

Two public discussions show the same failure shapes outside vendor release notes (both checked October 29, 2026; paraphrased, no names):

- Mode 2 (retry ceilings and circuit breakers):**
https://www.reddit.com/r/AI_Agents/comments/1xedkfx/moving_beyond_context_soup_how_are_you_strictly/ (posted about October 24, 2026). A builder running a self-hosted multi-agent pipeline asks which retry ceilings stop two models from endlessly bouncing rejections before a human steps in. It is a fit for the retry-cap and circuit-breaker advice in Mode 2, with a secondary fit to Mode 5's typed, bounded handoffs.
- Mode 3 (a tool the agent can't see):** <https://x.com/i/status/2114821583412011384> (posted October 26, 2026). A team describes a support agent that started handing simple follow-ups to a human after it gained more tools: *"the tool it needed was still there. the model couldn't see it"*. The root cause there is **lazy tool loading** (the tool sat in a group that wasn't loaded), not a late MCP connect. The symptom and the fix are the same: make tool availability explicit, and load or refresh before declaring a tool missing.

Related Reading on Tool Use Cases

- Agent Backbone Issue #1: 5 Ways Production AI Agents Quietly Break (And the Deterministic Fixes):**
<https://tools.lalipublications.com/agent-backbone/issue-1/>
- Free 4-Criteria Evaluation Rubric (PDF):** <https://tools.lalipublications.com/agent-backbone/agent-backbone-evaluation-rubric.pdf>
- Best AI Coding Agents:** <https://tools.lalipublications.com/use-cases/best-ai-coding-agents/>
- Best AI for Fixing Broken Code:** <https://tools.lalipublications.com/use-cases/best-ai-for-fixing-broken-code/>
- Best AI for Vibe Coding** (for builders on hosted app platforms like Lovable):
<https://tools.lalipublications.com/use-cases/best-ai-for-vibe-coding/>

Subscribe & Share a Production Failure Case

- **Free Weekly Briefing & Rubric Updates:** Subscribe free at <https://tools.lalipublications.com/agent-backbone/> (one email field, no account needed; signup is handled by FormSubmit).
- **Submit a Failure Trace or Question:** Email larrys@lalipublications.com (Lali Publications). Anonymized retry logs and tool-call traces are welcome; we may feature patterns, never your data.