

TECHNICAL REPORT

CVC Gateway Forensic Audit

and Hermes Brain Fork Plan

CVC Gateway Forensic Audit & Hermes Fork Plan

Principal: Jai Kumar Meena **Prepared by:** Sofia (Master Orchestrator) with Samantha (AI Research) and Tina (Code Audit) **Date:** 2026-05-20 **Subject:** Root-cause analysis of CVC dashboard gateway failures + plan to fork Hermes Agent's backend into CVC

Executive Summary

The CVC dashboard's WebSocket chat path silently dies after 4–5 tool calls. The CLI does not exhibit this bug. Root cause: `cvc/gateway.py` contains a **third, divergent copy** of the agentic loop that is missing 7+ guardrails present in the SSE path and CLI. Three structural defects account for nearly all observed failures.

Recommendation: Execute **Path A** (incremental ~90-minute patch fixing 5 of 6 candidate causes) immediately, then begin **Path B** (full Hermes-brain fork into CVC) as a 2–3 day refactor behind a feature flag.

Part 1 — Tina's Code Audit (CVC Gateway)

1.1 Dashboard Chat Path Map

Stage	File / Lines
WS endpoint	<code>gateway.py:5459</code>
Socket accept + first frame	<code>gateway.py:5460–5520</code>
Resume hand-off branch	<code>gateway.py:5519–5559</code>
New turn init	<code>gateway.py:5560–5598</code>
Receiver loop (control frames)	<code>gateway.py:5567–5580</code> — swallows non-disconnect exceptions
LLM availability gate	<code>gateway.py:5602–5637</code>
Agentic loop header	<code>gateway.py:5672</code>
Per-event LLM stream watchdog	<code>gateway.py:5712–5756</code>
	<code>gateway.py:5807–5871</code>

Stage	File / Lines
No-tool-call branch (nudge + synthesis)	
Loop detector (3× same sig)	<code>gateway.py:5873–5894</code>
<code>no_progress</code> counter	<code>gateway.py:5896–5909</code>
Assistant message reconstruction	<code>gateway.py:5911–5931</code>
Tool execution + confirm	<code>gateway.py:5933–6118</code>
Synthesis fallback (v2.71.x)	<code>gateway.py:6132–6239</code>

1.2 WS vs SSE vs CLI Feature Matrix

Capability	WS	SSE	CLI
Per-event stream watchdog	YES (60s)	YES (60s)	NO
Total-turn wall-clock	NO	YES	NO
Budget-aware iter cap	NO	YES	YES
Expensive-model max_tokens cap	NO	YES	YES
Synthesis fallback in finally	YES (v2.90.7)	YES	YES
Loop detector	YES	YES	YES
<code>no_progress</code> abort	YES (≥ 5)	YES (≥ 5)	n/a
Tool truncation per provider	NO	YES	YES
Sub-agent timeout exemption	NO	YES	YES
Dead-client check in loop	NO	YES	n/a
Receiver exception handling	silent swallow	n/a	n/a

1.3 Top Three Root Causes (Ranked)

#1 — `no_progress` Counter Kills Silent Tool Sequences (HIGH CONFIDENCE)

`gateway.py:5896–5909` aborts the loop when 5 iterations pass without assistant text. Claude/Copilot's natural style is to announce a plan, then execute 5 silent tool calls, then summarize. **Exactly 5 silent tool calls trigger the abort.** Symptom matches user report verbatim.

Fix: Raise threshold to ≥ 8 AND require repeating tool signatures across all 5 iterations before counting as no-progress.

#2 — Receiver Loop Swallows Errors + Socket Silently Detaches (HIGH CONFIDENCE)

`gateway.py:5576–5578` catches all non- `WebSocketDisconnect` exceptions and does nothing — no disconnect event queued. Meanwhile `_send_safe` (line 5408) sets `ts.websocket = None` on backpressure timeout. The agentic loop has no `_client_alive()` check (SSE has one at 4609). Loop continues spinning while browser sees nothing.

Fix: Add `_client_alive(ts)` check at top of every iteration and after every tool. In `receive_loop`, enqueue a `disconnect` event on any exception, not only `WebSocketDisconnect`.

#3 — Sub-Agent Timeout + Uncapped Tool Output (MEDIUM-HIGH CONFIDENCE)

`gateway.py:6058` wraps `agent / parallel_agents` tools in 300s `asyncio.wait_for` (SSE exempts them at line 5030). `gateway.py:6111` uses flat `_MAX_TOOL_OUTPUT_LLM = 15000` regardless of provider (SSE uses `_MAX_TOOL_OUTPUT_EXPENSIVE = 5000` for Claude/Opus). After 4 tool calls × 15K chars, Claude's working context is destroyed.

Fix: Port `_effective_timeout = None if _is_subagent_tool` from SSE 5030. Use `_MAX_TOOL_OUTPUT_EXPENSIVE` when `_is_expensive_model(_agent_llm.model)`.

1.4 Additional Confirmed Gaps

- **C3:** No `_budget_max_iterations` — WS uses raw `_MAX_AGENT_ITERATIONS`, ignoring expensive-model halving
- **C4:** No orphan `tool_call/tool_result` backfill — message corruption on denial paths produces 400 errors
- **C7:** Loop-detector false-positives on legitimate re-reads (3-repeat threshold too aggressive)
- **C10:** No pre-flight context estimator — Claude can be fed 90K input asking for 16K output, silently truncated

Part 2 — Samantha's Hermes Architecture Research

2.1 The One-Brain Principle

Hermes Agent uses a single agentic loop for every surface (CLI, Telegram, MCP). Both CLI and Telegram converge at `AIAgent.run_conversation` (forwarder in `run_agent.py:3865`) which calls `agent.conversation_loop.run_conversation` (lines 187–3900). There is no second copy, no third copy.

2.2 Why Hermes Never Breaks

1. **Hard iteration cap with grace** — `IterationBudget` (`agent/iteration_budget.py:17`) allows one final summary attempt after exhaustion

2. **Synthesis fallback** — `chat_completion_helpers.handle_max_iterations` (line 910) always strips tools, injects "summarize, no more tools", forces a final reply
3. **Empty-response nudge** — `conversation_loop.py:3503–3540` injects synthetic assistant message on empty turns
4. **Per-iteration message sanitization** — `repair_message_sequence`, `_sanitize_api_messages`, `_drop_thinking_only_and_merge_users` make orphan tool_calls impossible
5. **Per-tool error isolation** — `tool_executor._run_tool` (line 197) wraps every call; failures become messages, never break loop
6. **Provider/credential fallback** — `try_activate_fallback` (`chat_completion_helpers.py:673`) rotates models/keys on 429/5xx
7. **Stateful streaming scrubbers** — `<think>` and memory-context tags split across deltas never leak
8. **Inactivity watchdog** — `gateway/run.py:16320–16440` polls `agent.get_activity_summary()` every 5s; converts hung agent to clean timeout
9. **Per-iteration persistence** — `SessionDB.append_message` called 15+ times per turn; crash-mid-loop leaves DB consistent
10. **Backup interrupt detection** — `gateway/run.py:16348, 16408` checks for interrupts even if dedicated monitor task died
11. **CLI safety net** — `cli.py:~11010` wraps `run_conversation` in try/except, always returns a result dict

2.3 Tool Registry and Provider Adapters

- **Discovery:** `tools/registry.py` (lines 234–334), `model_tools.py` (lines 731–890)
- **Toolsets:** `toolsets.py` — grouped, toggleable via `enabled_toolsets` / `disabled_toolsets`
- **Execution:** `agent/tool_executor.py` — concurrent (line 64) + sequential (line 474) dispatch
- **Schema converters:**
 - OpenAI/Copilot: `agent/transport/chat_completions.py`
 - Anthropic: `agent/anthropic_adapter.py` (lines 1422, 1609)
 - Gemini: `agent/gemini_native_adapter.py`, `agent/gemini_schema.py`
 - Bedrock: `agent/bedrock_adapter.py`
 - Codex Responses: `agent/codex_responses_adapter.py`

2.4 Session Persistence (the CVC-Merkle equivalent)

`hermes_state.py:SessionDB` (line 309) — SQLite WAL + FTS5 + trigram index.

- `append_message` (line 1433) — per-message write after every turn
- `get_messages_as_conversation` (line 1884) — the recall API (explicit only)
- `search_messages` (line 2078) — on-demand FTS query

Critical: Next turn does NOT auto-load history. Recall is on-demand via tools. This is exactly the CVC philosophy — Merkle DAG should be queryable storage, not auto-injected context.

2.5 Telegram Bridge

`gateway/platforms/telegram.py` (~4837 lines) + `gateway/platforms/base.py` (~3778 lines, abstract `MessageEvent` contract at line 919). Same brain as CLI — both call `AIAgent.run_conversation`. Differences are only callbacks (stream sink, approval, sudo prompt) and timeout policy (gateway has inactivity monitor; CLI is unbounded).

Part 3 — Fork Manifest

3.1 Copy Verbatim (The Brain)

```
run_agent.py
agent/conversation_loop.py
agent/chat_completion_helpers.py
agent/tool_executor.py
agent/agent_runtime_helpers.py
agent/agent_init.py
agent/iteration_budget.py
agent/stream_diag.py
agent/auxiliary_client.py
agent/transport/__init__.py
agent/transport/base.py
agent/transport/types.py
agent/transport/chat_completions.py
agent/transport/anthropic.py
agent/transport/bedrock.py
agent/transport/codex.py
agent/transport/codex_app_server.py
agent/transport/codex_app_server_session.py
agent/transport/hermes_tools_mcp_server.py
agent/anthropic_adapter.py
agent/bedrock_adapter.py
agent/codex_responses_adapter.py
agent/gemini_native_adapter.py
agent/gemini_schema.py
agent/moonshot_schema.py
tools/registry.py
tools/*
model_tools.py
toolsets.py
hermes_state.py
```

3.2 Copy and Adapt (Transport Layer — including Gateway)

```
gateway/run.py           # HTTP/queue server + inactivity monitor + run_sync
gateway/delivery.py      # DeliveryRouter chunking + fan-out + persistence sin
gateway/session.py       # session/task state
gateway/platforms/base.py # MessageEvent contract
gateway/platforms/telegram.py # for CVC's future Telegram bridge
```

The gateway IS included in the fork. Hermes's `gateway/run.py` carries the inactivity watchdog (lines 16320–16440), the `_handle_message_with_agent` router, and the `run_sync` worker pattern that keeps the agent thread alive while the HTTP layer polls `get_activity_summary()`. That is the exact "never stops, never zombies" property required. Renamed "Copy and adapt" only because CVC's dashboard exposes different endpoints (`/api/ws/chat`, `/api/chat`) — we rewire URL surface, keep brain + plumbing.

3.3 Do NOT Touch (CVC's UI Stays Identical)

```
cvc/web/*                # CVC dashboard React/web UI
cvc/agent/chat.py (TUI portion) # CVC terminal UI
```

3.4 Integration Contract for CVC's UI

```
from cvc.hermes_brain import AIAgent, SessionDB

agent = AIAgent(
    model=...,
    enabled_toolsets=...,
    session_db=SessionDB(path),
    session_id=session_id,
    max_iterations=90,
)
agent.set_stream_callback(your_ws_sender)
agent.set_approval_callback(your_approval_handler)
result = agent.run_conversation(user_message=msg, task_id=session_id)
final_reply = result["final_response"]
```

One brain. Three skins (TUI, Dashboard, future Telegram).

Part 4 — Execution Plan

Path A — Incremental Patch (TONIGHT, ~90 min, LOW RISK)

All edits in `cvc/gateway.py`:

1. Raise `no_progress` threshold from 5 to 8, require repeating tool signatures
2. Add `_client_alive(_ts)` check at iteration top and after every tool
3. Fix `receive_loop` to enqueue `disconnect` on any exception
4. Exempt sub-agents from `_TOOL_EXEC_TIMEOUT`
5. Use `_MAX_TOOL_OUTPUT_EXPENSIVE` for Claude/Opus
6. Use `_budget_max_iterations()` instead of raw cap
7. Add total-turn wall-clock + orphan `tool_call` backfill
8. Optional: borrow pre-flight context estimator from `chat.py:1433–1454`

Total: ~90 minutes coding + 1 hour smoke testing. Risk: low — every change has SSE or CLI precedent.

Path B — Full Hermes-Brain Fork (THIS WEEK, 2–3 days, MEDIUM RISK)

1. Create branch `cvc-hermes-brain`
2. Lift Fork Manifest modules into `cvc/hermes_brain/` namespace
3. Adapt `gateway/run.py` to expose CVC's `/api/ws/chat` + `/api/chat` endpoints
4. Wire CVC's TUI and Dashboard to `AIAgent.run_conversation`
5. Replace `SessionDB` with Merkle-DAG-backed implementation satisfying the same call surface
6. Ship behind `CVC_USE_HERMES_LOOP=1` feature flag for one week of daily-driver use
7. Flip default, delete ~1500 LOC of divergent gateway code

Payoff: kills 5 of 6 candidate root causes structurally; one brain across all surfaces; matches Hermes's never-stops guarantee.

Appendix A — Key File Reference

Concern	Hermes file	CVC equivalent
Agentic loop	<code>agent/conversation_loop.py:187–3900</code>	<code>cvc/gateway.py:5672–6130</code> (WS) + <code>cvc/agent/chat.py</code> (CLI)
API call + synthesis	<code>agent/chat_completion_helpers.py:910</code>	<code>cvc/gateway.py:6132–6239</code>
Tool dispatch	<code>agent/tool_executor.py:64, 474</code>	inline in <code>cvc/gateway.py:5933–6118</code>
Iteration budget	<code>agent/iteration_budget.py:17</code>	scattered constants in <code>cvc/gateway.py</code>
Persistence	<code>hermes_state.py:SessionDB</code> (line 309)	CVC Merkle DAG layer
Anthropic adapter	<code>agent/anthropic_adapter.py:1422, 1609</code>	<code>cvc/agent/llm.py:_fix_github_claude_messages</code>
Inactivity monitor	<code>gateway/run.py:16320–16440</code>	absent in CVC
Telegram bridge	<code>gateway/platforms/telegram.py</code>	not yet built

Appendix B — Audit Provenance

- **Samantha's run:** 50 tool calls, 445 seconds, 2.63M input tokens. Read-only inspection of `/Users/jkm/.hermes/hermes-agent`.
 - **Tina's run:** 12 tool calls, 235 seconds, 757K input tokens. Read-only inspection of `/Users/jkm/Projects/cvc`.
 - **No source files were modified during this audit.**
-

End of Report