

Trading Bots That Ate My Tokens

Harness Engineering for Autonomous Agents

OpenClaw • Hermes • Claude Routines



31 days, 3 trading agents, and what I learned.

Abhi Puttanna • Shark Agent Lab • South Florida Data Geeks

A LOGIQFISH EXPERIMENT REPORT
LOGIQFISH.COM • DATA • SEO • CLOUD

The Trading Bots That Ate My Tokens

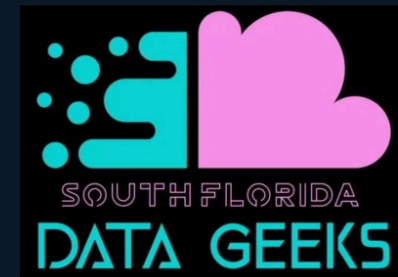
The Trading Bots That Ate My Tokens

Harness Engineering for Autonomous Trading Agents



Abhi Puttanna · LogiqFish — Shark Agent Lab

South Florida Data Geeks · July 9th, 2026 · A 90-day autonomous paper trading experiment



Shark Agent Lab

Welcome, and thanks for getting downtown at 6 PM on a Thursday — you could have been home watching the game, but you made it here. Thanks to South Florida Data Geeks. With that, let's dive into the ocean of autonomous AI agents.

Same information. Different minds. Different outcomes.

ACT 1 · THE BET

Same information. Different minds. Different outcomes.

31	3	 Alpaca \$25,000	0
DAYS — OF A 45-DAY RACE	MINDS	PER BOT · PAPER	HUMANS IN THE LOOP

“I put three AI trading agents into the same market 31 days ago. Same rules. Same starting capital. Same goal. Nobody touching them. **They did NOT behave the same.**”

 **POLL** Who thinks the most advanced model won? (pick the answer)

02 · The Bet

 Shark Agent Lab

I'm a software engineer, not a trader — and most times I trade, I lose money. So I asked: what if AI agents traded my stocks instead — no fear, no ego, just logic and rules? The question became which agent to trust, and which could beat the S&P 500. I built three, opened an Alpaca paper account, and let them run for a month.

Where this is going

WHAT WE'RE COVERING

Where this is going.

Three agents in one live market — and **everything that happened next.**

01

The bet

Three agents, one market, identical rules.

02

The anatomy

Same market, same tools, three minds.

03

The bill

What running an agent actually costs.

04

The scoreboard

31 days later — who's ahead, who's not.

05

What broke

The failures that taught every lesson.

06

Take one home

The patterns you can reuse tomorrow.

WHAT WE'RE COVERING

 Shark Agent Lab

Here's the map for tonight: the bet (why I built three agents), the anatomy of an agent (model, tools, memory, loop, guardrails), the bill (they don't run for free), the scoreboard after 31 days, and finally what broke, what I learned, and a repo you can take home.

Same market. Same rules. Three harnesses.

ACT 1 · THREE AI HARNESSSES, ONE QUESTION

Same market. Same rules. Three harnesses.



API based

OpenClaw

The Swiss Army Knife

- ▶ Most knobs
- ▶ More setup complexity
- ▶ Defaults became decisions



API based

Hermes

The Lean Self-Learner

- ▶ Stripped-down by design
- ▶ Smaller tool surface
- ▶ Fewer knobs, more control



Claude Routines

The Scheduled Worker

- ▶ Wakes up, runs, writes back, sleeps
- ▶ Git/repo as memory
- ▶ Least autonomous, most predictable

“The model mattered. The architecture mattered more.”

04 · Three Harnesses



Cron Job

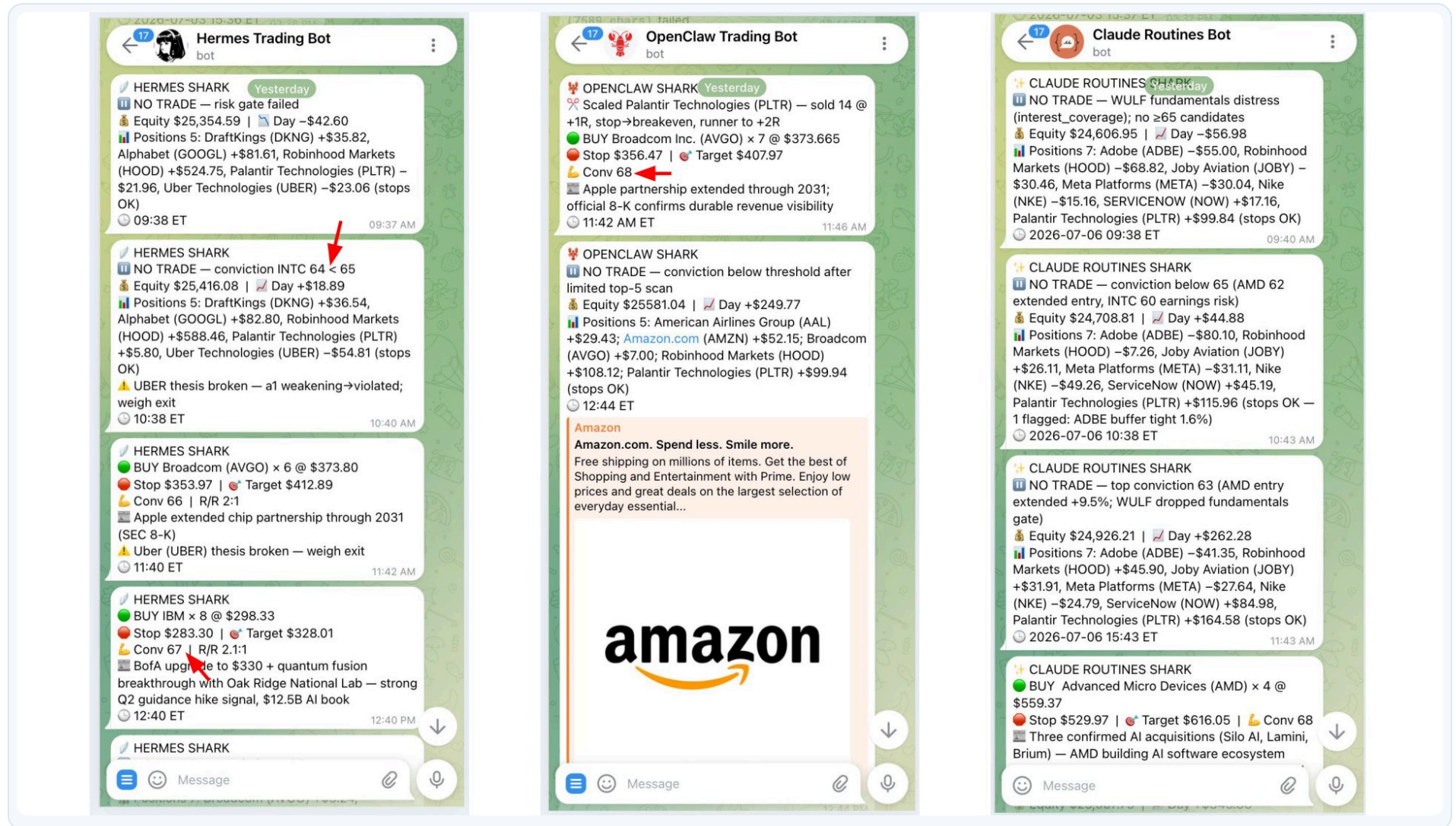
● **LIVE** All three post to Telegram · runs every hour

Shark Agent Lab

The model mattered — but the harness mattered more.

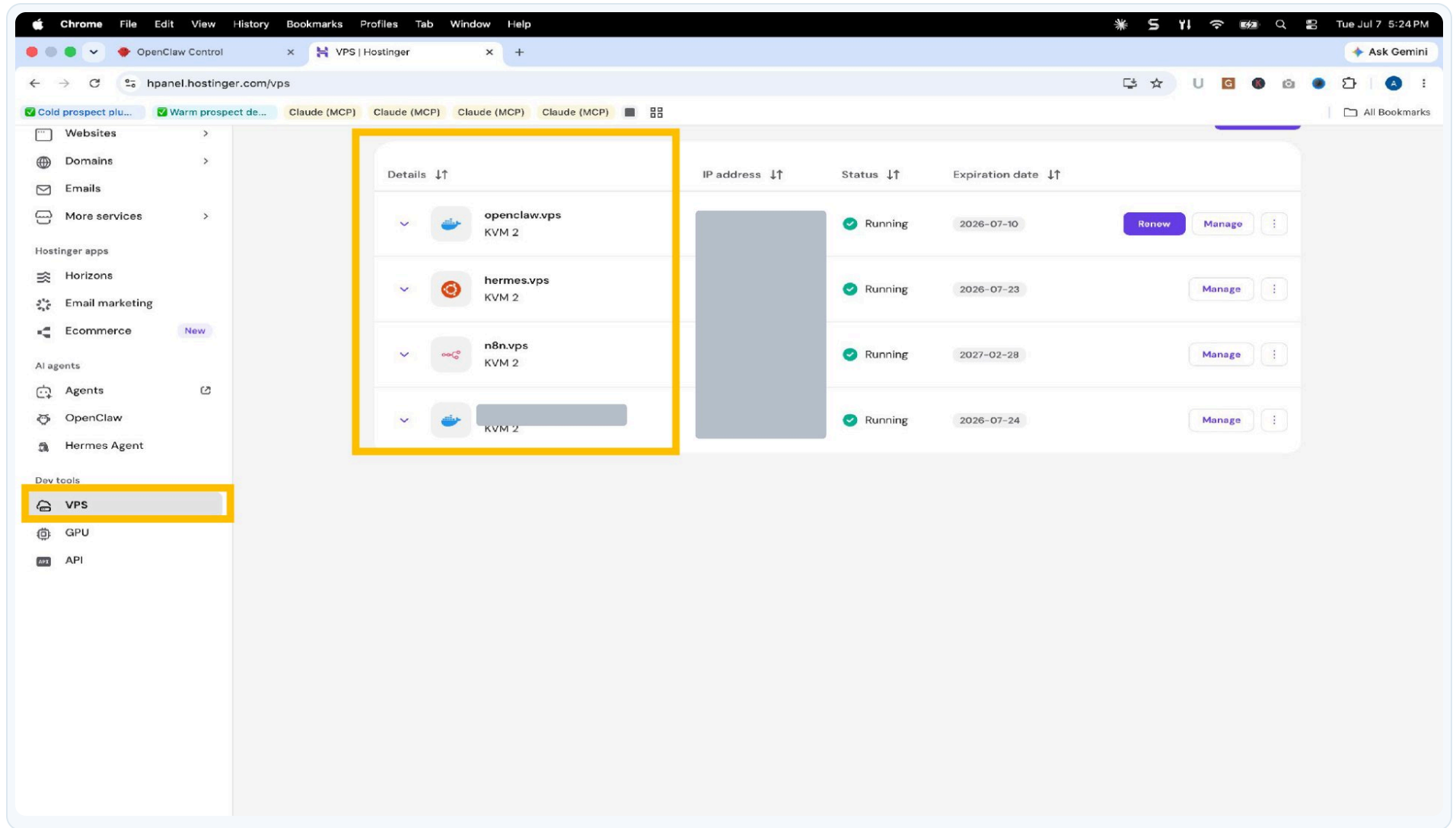
Meet the three agents: OpenClaw, Hermes, and Claude Routines. The agent is the whole system; the harness is the layer around the model that runs tools, memory, rules, timing, and execution — the LLM is only the brain. OpenClaw is a Swiss Army knife, Hermes is leaner, and Claude Routines is the most predictable.

An intelligent friend with strict boundaries



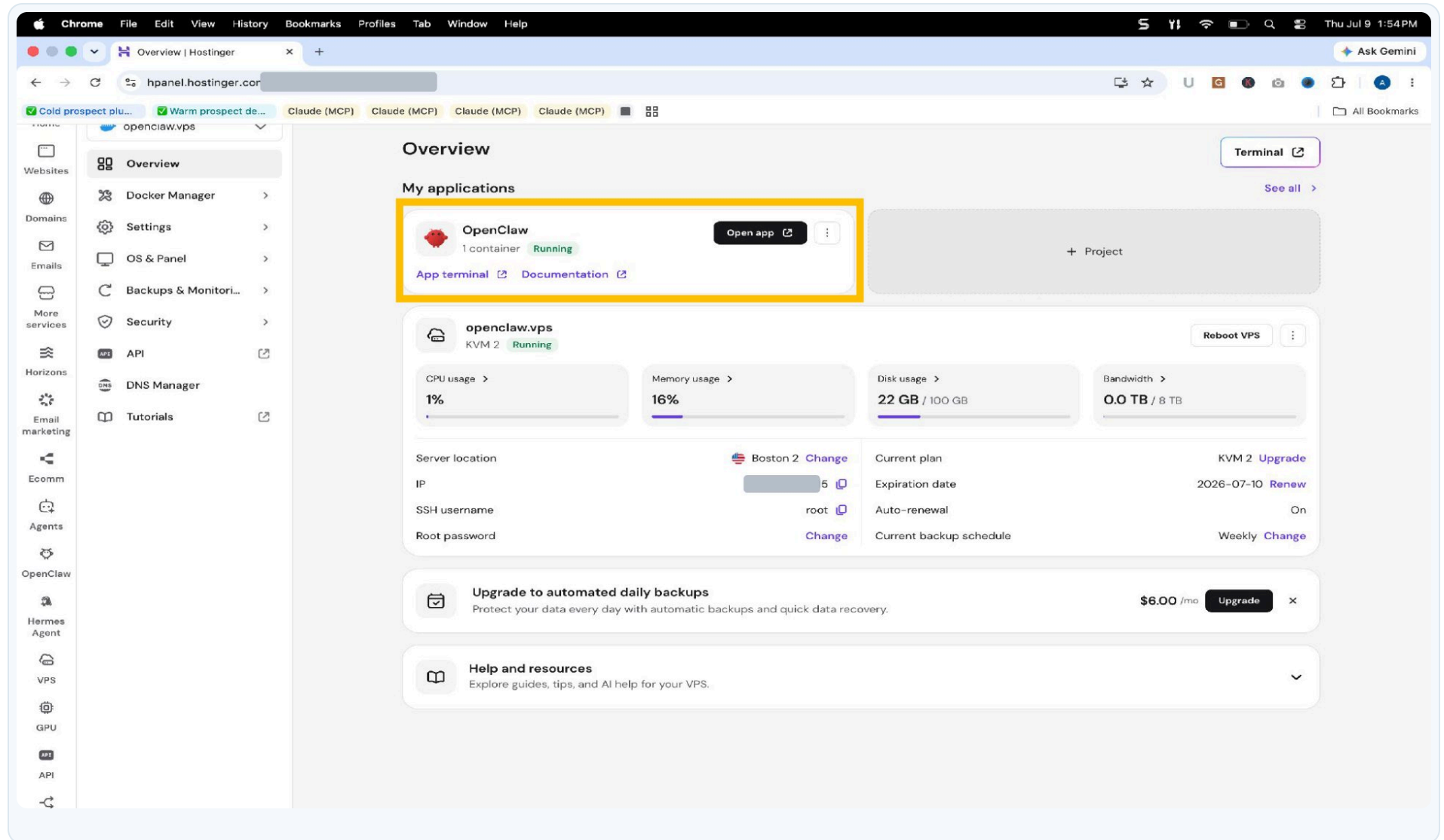
This is where it got real: actual Telegram messages from the three bots. Every hour during market hours they wake up, analyze the market, decide whether to trade, and message me — and I can reply to ask why they bought or skipped something. It feels less like a script and more like an intelligent friend that still needs strict boundaries.

Where the agents lived



I wanted the agents running continuously on a server, not my laptop, so they could wake every hour and keep working even when I wasn't watching. This is the Hostinger VPS where they were deployed — and once you deploy an autonomous system, monitoring matters as much as the code.

Going deeper than the dashboard



From the OpenClaw VPS view I could open the app, check the server, and inspect what was actually happening. This mattered because the web dashboard only tells part of the story — when something feels wrong, you have to go deeper into the machine.

Three harnesses, swap in any brain

ACT 1 · BRAIN INSIDE

Three harnesses. Swap in any brain.



OpenClaw

 Kimi → GPT-5.4 mini

- ▶ Swapped Jun 25 — Kimi's cache hurt
- ▶ Same harness, new brain — bill went up
- ▶ Reasoning tokens: thinks more, more \$\$



Hermes

 DeepSeek V4 Pro

- ▶ Open-weight · Chinese lab
- ▶ Cheapest brain in the fleet
- ▶ Cache billed as discount, not penalty



Claude Routines

 Sonnet 4.6

- ▶ Premium · subscription
- ▶ Highest-capability brain
- ▶ Flat rate — meter is a rate limit

“Swap the brain, keep the harness — and the surprises are still the harness's. The brain is the commodity; the harness is the variable.”

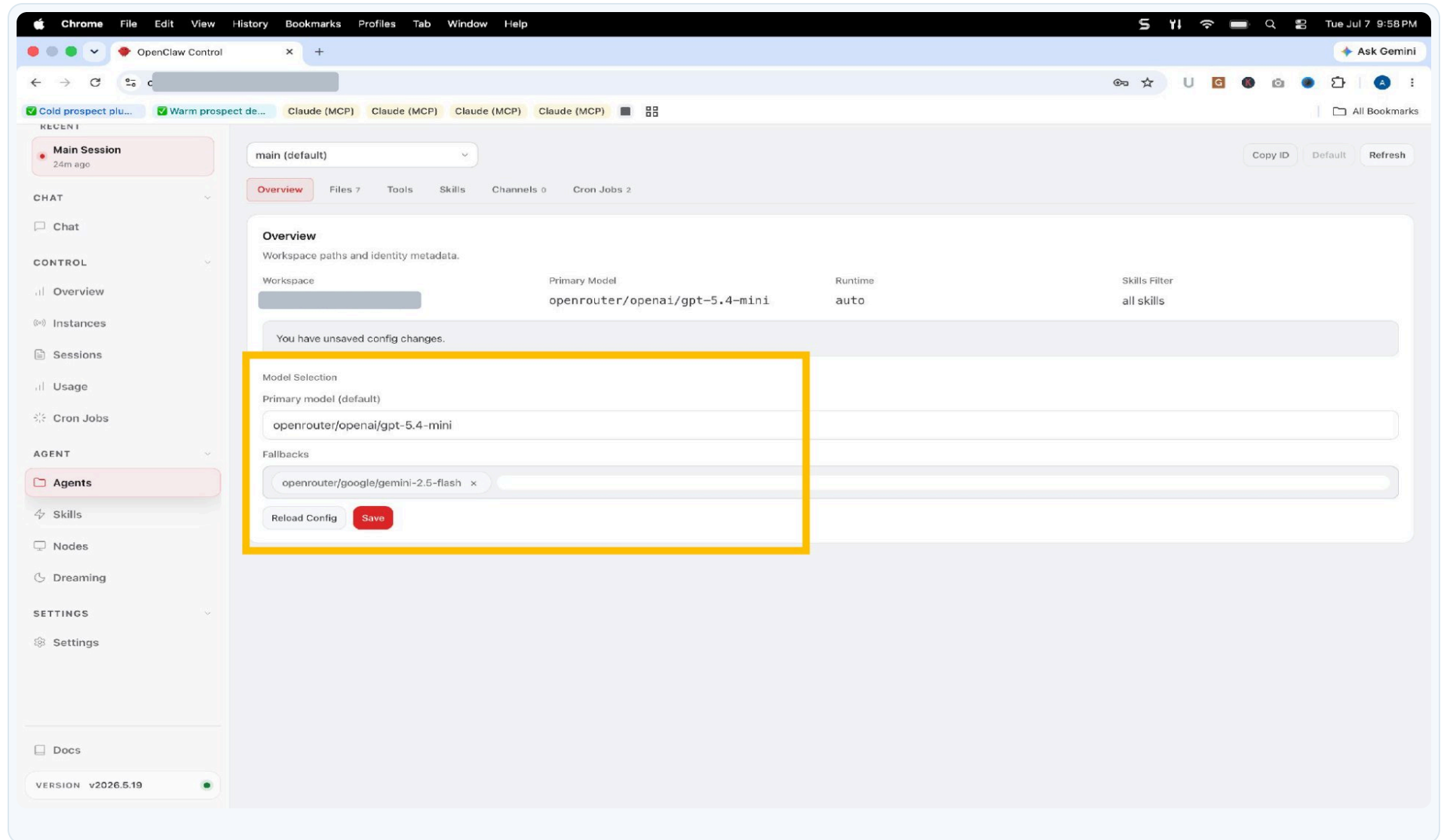
05 · Brain Inside

 Shark Agent Lab

My assumption was that the most advanced model would win.
That turned out to be wrong.

A good body still needs a sound mind, so each harness ran a different brain. OpenClaw started on Kimi then switched to GPT-5.4 mini (Kimi looked cheap but ran expensive because of caching). Hermes ran DeepSeek V4 Pro, the cheapest brain yet surprisingly strong. Claude Routines used Sonnet 4.6, the premium frontier model.

OpenClaw, configured with GPT-5.4 mini



These weren't theoretical diagrams — they were real systems with real configs. Here is OpenClaw pointed at GPT-5.4 mini. I could swap the brain while keeping the same harness: changing the model doesn't change the whole agent, only the brain inside the same body.

Hermes, configured with DeepSeek V4 Pro

The screenshot shows a Chrome browser window with the Hermes Agent interface. The browser's address bar shows the URL `http://localhost:8080/`. The interface is titled "Hermes Agent v0.14.0 (2026.5.16)".

Available Tools:

- browser: browser_back, browser_click, ...
- browser-cdp: browser_cdp, browser_dialog
- clarify: clarify
- code-execution: execute_code
- computer-use: computer_use
- cronjob: cronjob
- delegation: delegate_task
- discord: discord
- (and 21 more toolsets...)

Available Skills:

- autonomous-ai-agents: claude-code, codex, hermes-agent, opencode
- creative: architecture-diagram, ascii-art, ascii-video, b...
- data-science: jupyter-live-kernel
- devops: kanban-orchestrator, kanban-worker, webhook-sub...
- email: himalaya
- gaming: minecraft-modpack-server, pokemon-player
- general: dogfood, yuanbao
- github: codebase-inspection, github-auth, github-code-r...
- mcp: native-mcp
- media: gif-search, heartmula, songsee, spotify, youtub...
- mlops: audiocraft-audio-generation, dspy, evaluating-l...
- note-taking: obsidian
- productivity: airtable, controlled-workspace-files, google-w...
- red-teaming: godmode
- research: arxiv, blogwatcher, llm-wiki, polymarket, resea...
- smart-home: openhue
- social-media: xurl
- software-development: debugging-hermes-tui-commands, hermes-agent-ski...

deepseek-v4-pro · Nous Research
/opt/hermes

Welcome to Hermes Agent! Type your message or /help for commands.

`deepseek-v4-pro | ctx -- | [REDACTED] -- | 10s | @ 0s`

`> █`

Hermes was designed to be lean: fewer tools, fewer moving parts, a smaller surface area for mistakes. The lowest-cost model didn't win the final return race, but it stayed fully competitive because it lived inside a cleaner harness that made fewer unnecessary mistakes.

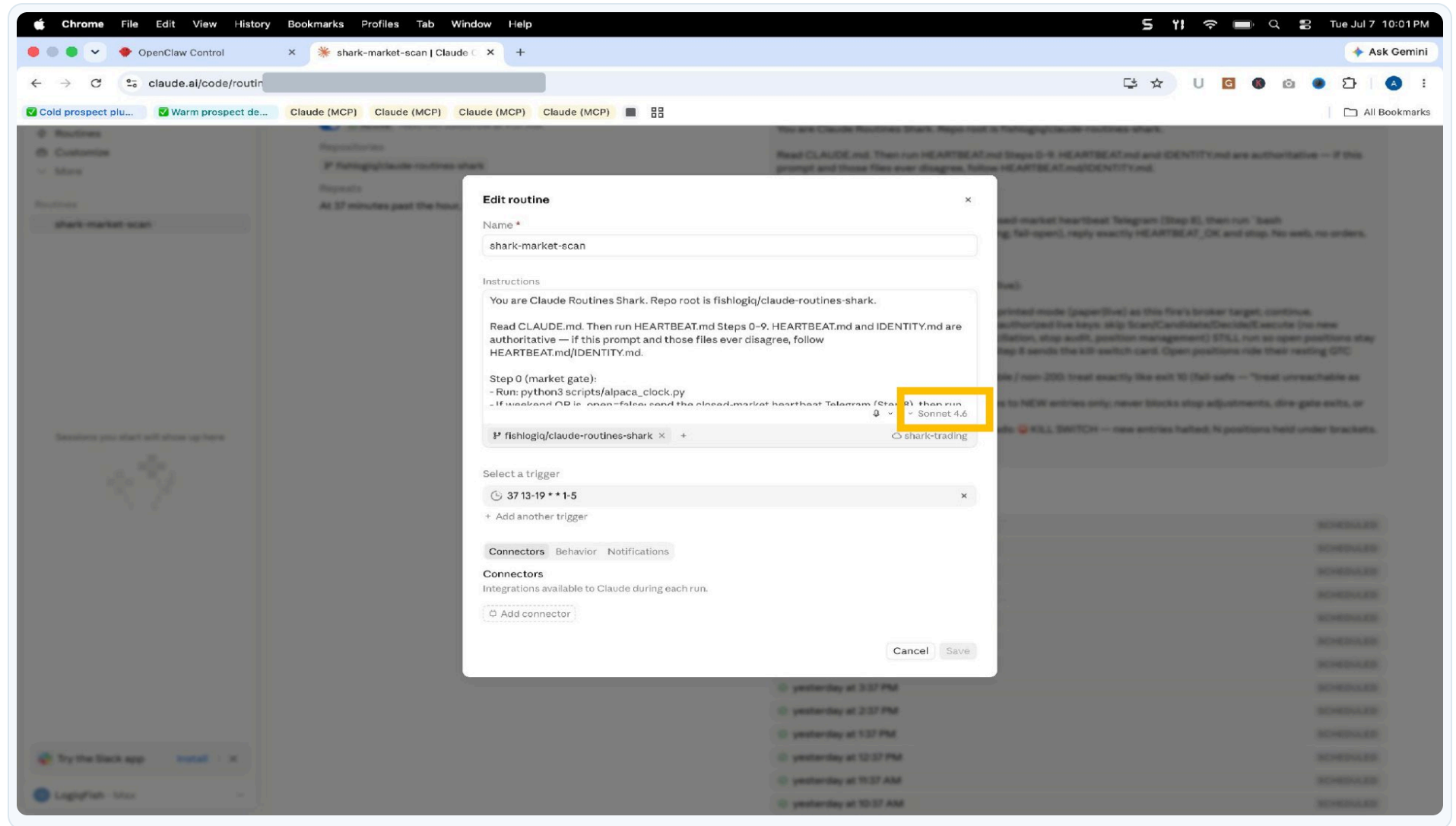
Claude Routines, the scheduled worker

The screenshot shows the Claude Routines interface in a Chrome browser. The sidebar on the left has a 'Routines' section with a sub-item 'shark-market-scan' highlighted by a yellow box. The main content area displays the routine's configuration. It includes a 'Repeats' section stating 'At 37 minutes past the hour, between 01:00 PM and 07:59 PM, Monday through Friday'. Below this, there is a detailed list of steps for the routine, including 'Step 0 (market gate)', 'Step 0.5 (control gate — kill switch + paper/live)', 'Exit 0 (TRADE)', 'Exit 10 (HALT)', and 'Exit 20 (HALT)'. At the bottom, there is a 'Runs' section with a table of scheduled runs.

Run	Status
today at 3:37 PM	SCHEDULED
today at 2:37 PM	SCHEDULED
today at 1:37 PM	SCHEDULED
today at 12:37 PM	SCHEDULED
today at 11:37 AM	SCHEDULED
today at 10:37 AM	SCHEDULED
today at 9:37 AM	SCHEDULED
yesterday at 3:37 PM	SCHEDULED
yesterday at 2:37 PM	SCHEDULED
yesterday at 1:37 PM	SCHEDULED
yesterday at 12:37 PM	SCHEDULED
yesterday at 11:37 AM	SCHEDULED
yesterday at 10:37 AM	SCHEDULED

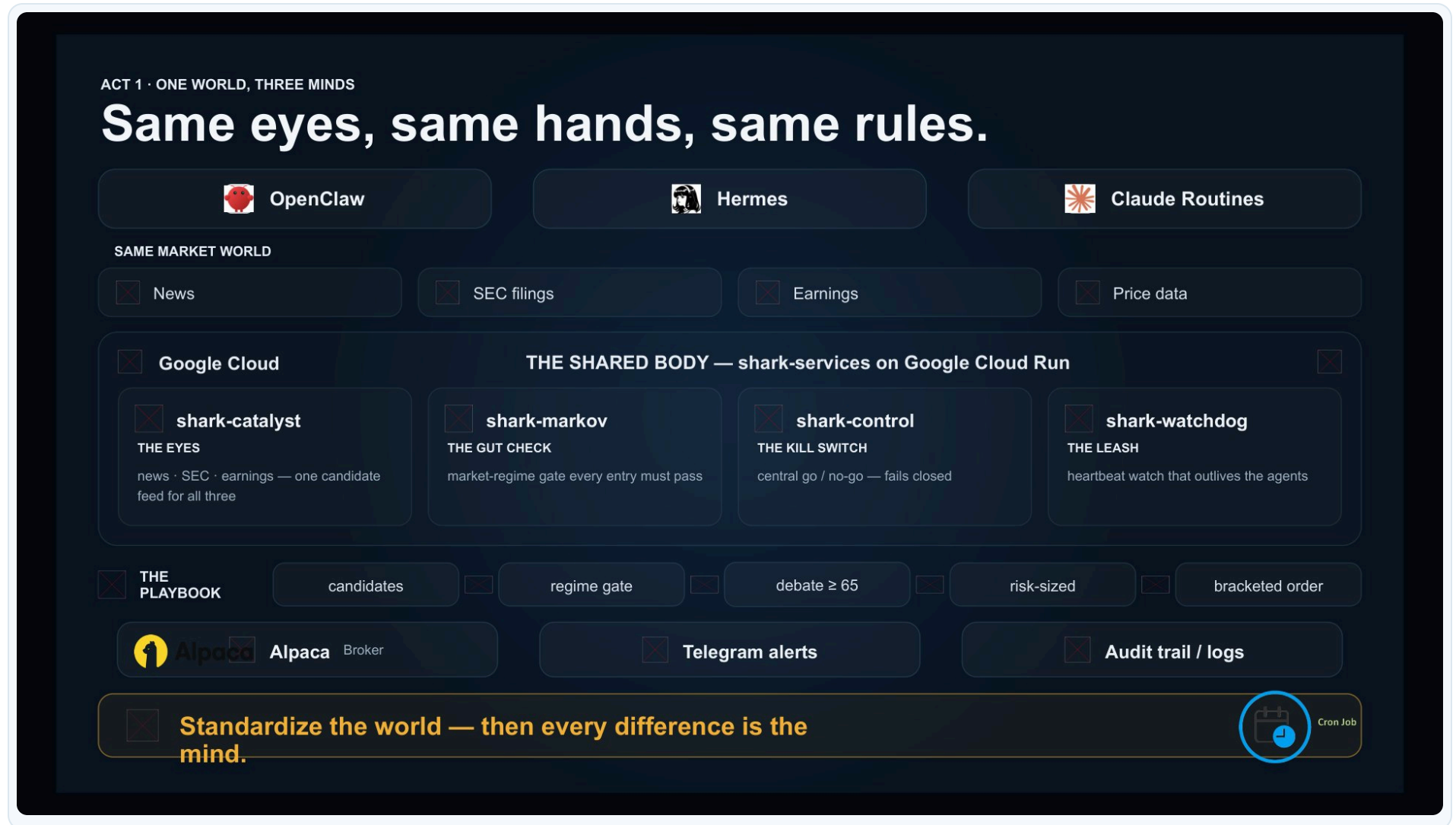
Unlike the others, Claude Routines behaves like a scheduled worker: it wakes on schedule, reads the repo, follows the instruction, writes back, and stops. It's less autonomous in the dramatic sense — but that makes it more predictable, and sometimes less autonomy is actually safer.

Claude Routines on Sonnet 4.6



Here is the Claude routine pointing to Sonnet 4.6, the highest-capability brain in the experiment. You'd expect it to perform best — but the race showed capability alone doesn't guarantee better outcomes. How the model is wrapped, constrained, and monitored can matter more than raw intelligence.

Same eyes, same hands, same rules



The core architecture: all three agents got the same eyes (news, filings, earnings, prices, candidates), the same hands (Alpaca paper trades), and the same rules (identical risk limits, conviction gates, bracket orders). Shared services on Google Cloud Run let every agent follow one hourly playbook.

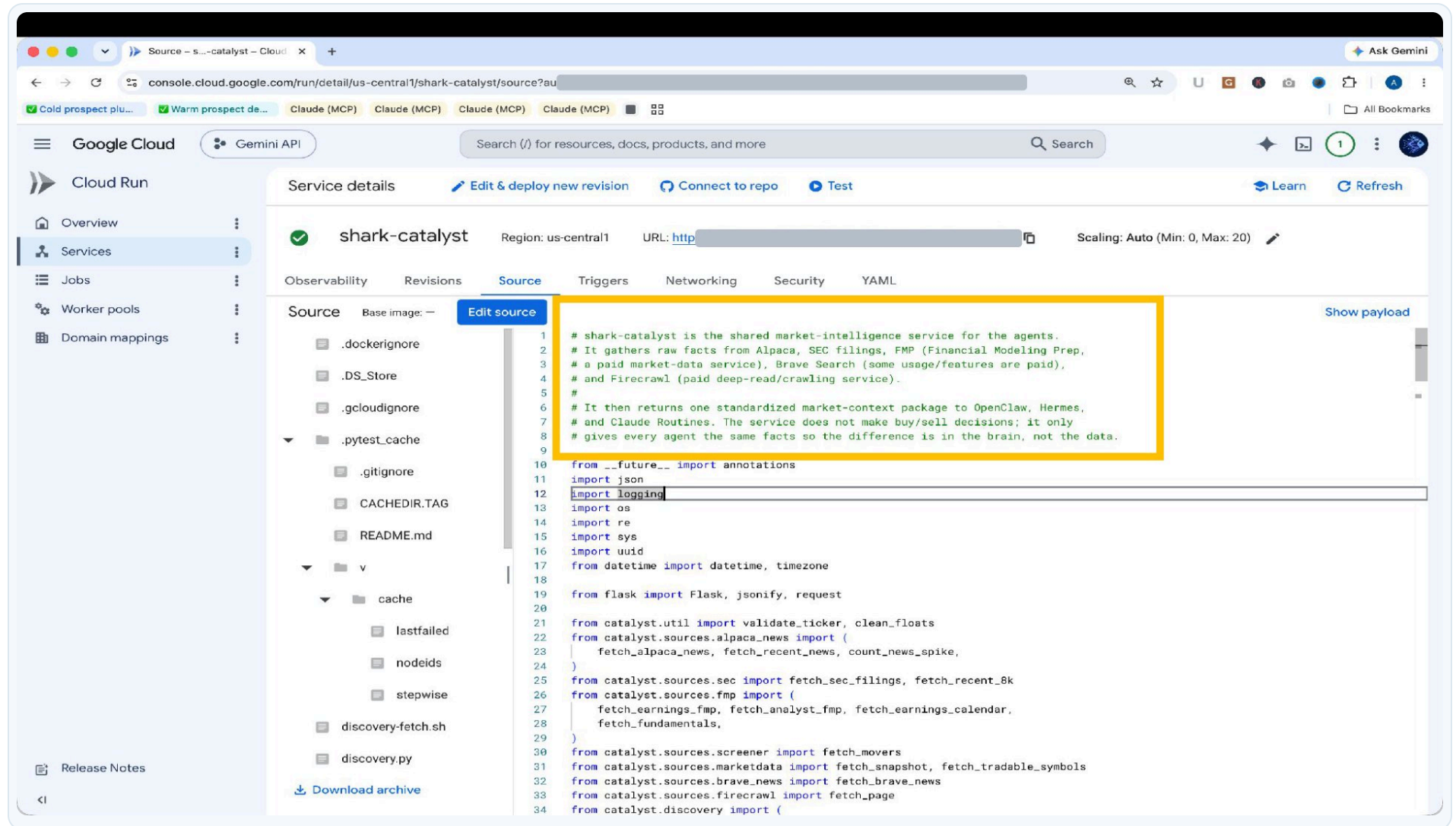
The shared body on Google Cloud Run

The screenshot shows the Google Cloud Run console interface. On the left is a sidebar with navigation options: Overview, Services (selected), Jobs, Worker pools, and Domain mappings. The main content area displays a list of services, with a filter 'shark' applied. The services listed are shark-catalyst, shark-control, shark-markov, and shark-watchdog. Each service is of type 'Source' and is deployed in the 'us-central1' region. The table also shows request rates, costs, authentication settings, ingress, and deployment times.

Name	Deployment type	Req/sec	Region	Cost	Authentication	Ingress	Last deployed
shark-catalyst	Source	0	us-central1	\$0.14	Public access	All	8 hours ago
shark-control	Source	0	us-central1	\$0.03	Public access	All	Jun 19, 2026
shark-markov	Source	0	us-central1	\$0.08	Public access	All	May 29, 2026
shark-watchdog	Source	0.02	us-central1	\$0.09	Public access	All	Jun 19, 2026

Instead of letting each agent gather data its own random way, I centralized the shared services. That way OpenClaw, Hermes, and Claude Routines weren't living in three different market realities — they all looked at the same external world, which made the comparison meaningful.

shark-catalyst — the eyes



The screenshot shows the Google Cloud console interface for the 'shark-catalyst' service. The 'Source' tab is selected, displaying the source code. A yellow box highlights the first 10 lines of the code, which contain comments and imports. The code is as follows:

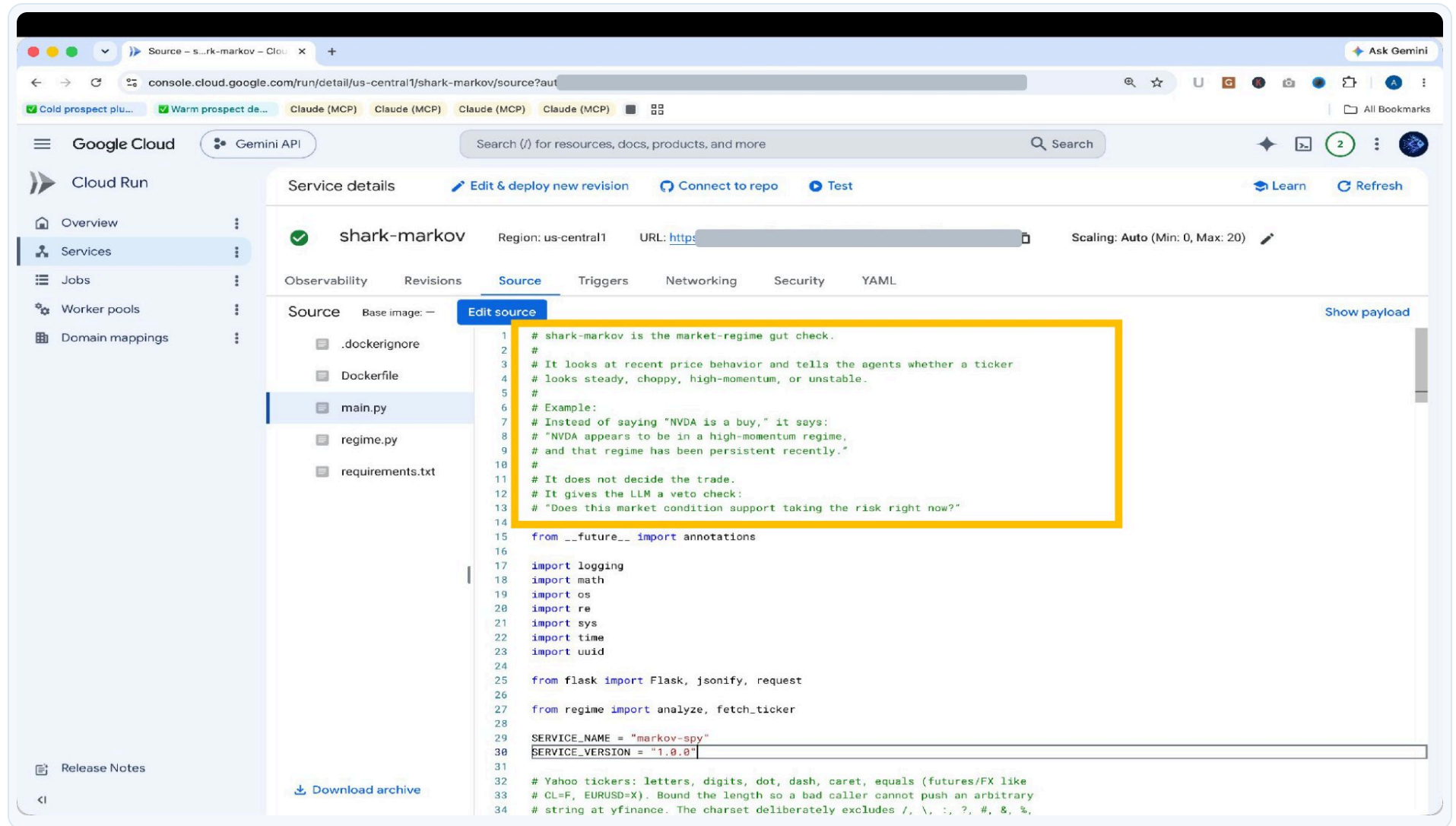
```

1 # shark-catalyst is the shared market-intelligence service for the agents.
2 # It gathers raw facts from Alpaca, SEC filings, FMP (Financial Modeling Prep,
3 # a paid market-data service), Brave Search (some usage/features are paid),
4 # and Firecrawl (paid deep-read/crawling service).
5 #
6 # It then returns one standardized market-context package to OpenClaw, Hermes,
7 # and Claude Routines. The service does not make buy/sell decisions; it only
8 # gives every agent the same facts so the difference is in the brain, not the data.
9
10 from __future__ import annotations
11 import json
12 import logging
13 import os
14 import re
15 import sys
16 import uuid
17 from datetime import datetime, timezone
18
19 from flask import Flask, jsonify, request
20
21 from catalyst.util import validate_ticker, clean_floats
22 from catalyst.sources.alpaca_news import (
23     fetch_alpaca_news, fetch_recent_news, count_news_spike,
24 )
25 from catalyst.sources.sec import fetch_sec_filings, fetch_recent_8k
26 from catalyst.sources.fmp import (
27     fetch_earnings_fmp, fetch_analyst_fmp, fetch_earnings_calendar,
28     fetch_fundamentals,
29 )
30 from catalyst.sources.screener import fetch_movers
31 from catalyst.sources.marketdata import fetch_snapshot, fetch_tradable_symbols
32 from catalyst.sources.brave_news import fetch_brave_news
33 from catalyst.sources.firecrawl import fetch_page
34 from catalyst.discovery import (

```

shark-catalyst gathers the market signals — news, SEC filings, earnings, price movement, fundamentals, and candidate ideas — into one shared candidate feed. If Hermes beats Claude Routines, I don't want the reason to be better news; I want it to be the mind and the harness, not unequal information.

shark-markov — the gut check



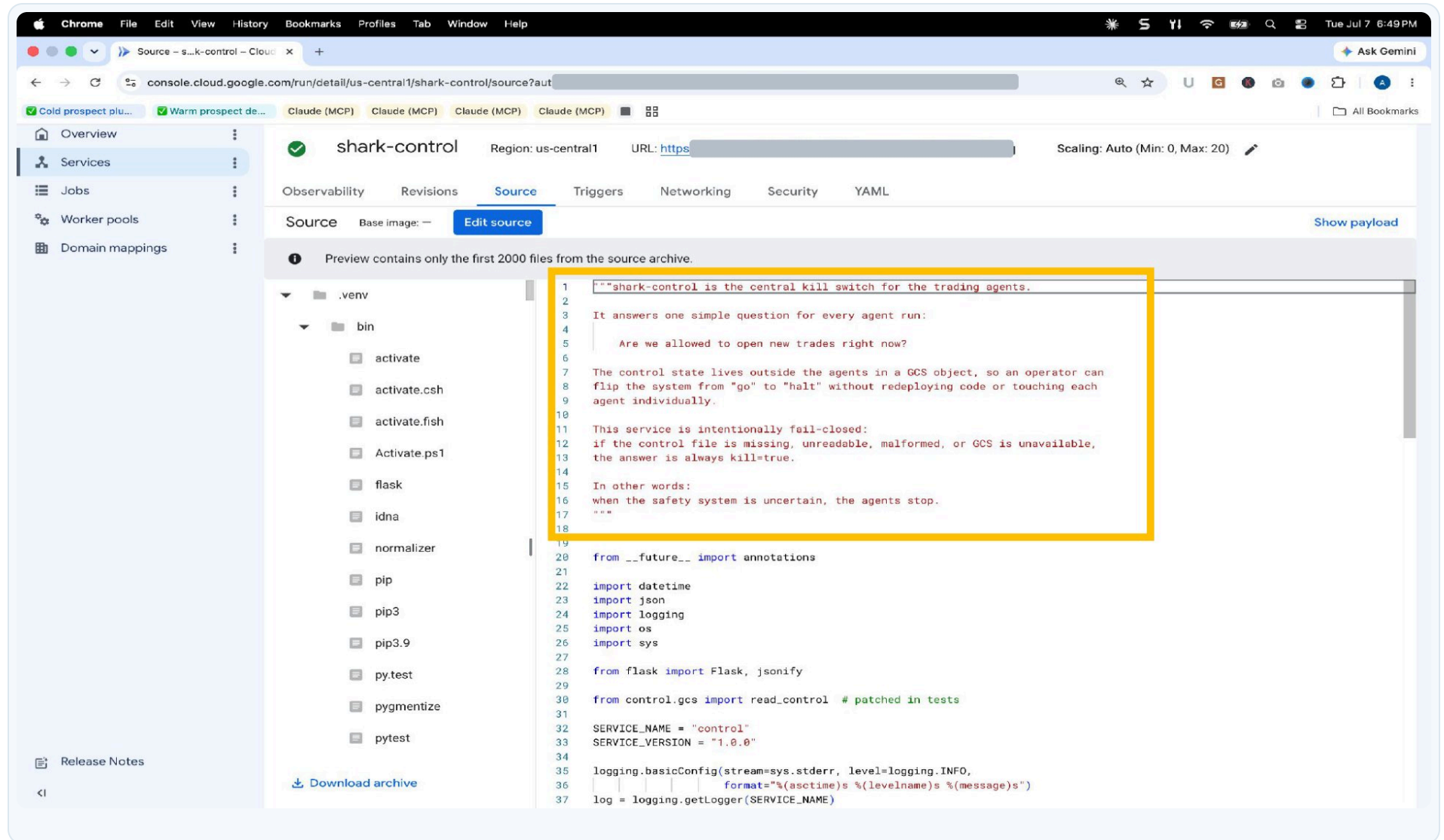
The screenshot shows the Google Cloud Run console for a service named 'shark-markov'. The 'Source' tab is selected, displaying the Python code for the service. A yellow box highlights the first 14 lines of the code, which contain comments explaining the purpose and logic of the 'gut check'.

```

1 # shark-markov is the market-regime gut check.
2 #
3 # It looks at recent price behavior and tells the agents whether a ticker
4 # looks steady, choppy, high-momentum, or unstable.
5 #
6 # Example:
7 # Instead of saying "NVDA is a buy," it says:
8 # "NVDA appears to be in a high-momentum regime,
9 # and that regime has been persistent recently."
10 #
11 # It does not decide the trade.
12 # It gives the LLM a veto check:
13 # "Does this market condition support taking the risk right now?"
14
15 from __future__ import annotations
16
17 import logging
18 import math
19 import os
20 import re
21 import sys
22 import time
23 import uuid
24
25 from flask import Flask, jsonify, request
26
27 from regime import analyze, fetch_ticker
28
29 SERVICE_NAME = "markov-spy"
30 SERVICE_VERSION = "1.0.0"
31
32 # Yahoo tickers: letters, digits, dot, dash, caret, equals (futures/FX like
33 # CL=F, EURUSD=X). Bound the length so a bad caller cannot push an arbitrary
34 # string at yfinance. The charset deliberately excludes /, \, :, ?, #, &, %,
  
```

shark-markov is the gut check. It doesn't say 'NVDA is a buy'; it says something like 'NVDA appears to be in a high-momentum regime that's been persistent recently.' Before an agent trades, this gives it a market-regime check — one more layer between the model's idea and the actual order.

shark-control — the kill switch



The agent doesn't get to negotiate with the kill switch. If control says stop, the system stops.

Every autonomous system needs a way to stop that lives outside the agent itself. If the agent is confused, if the market is behaving badly, or if I just want everything to halt, shark-control is the central go / no-go signal.

shark-watchdog — the leash

The screenshot shows the Google Cloud Platform console for a service named 'shark-watchdog' in the 'us-central1' region. The 'Source' tab is selected, displaying the code for the service. The code is a Python Flask application. A yellow box highlights a text block at the top of the code file, which serves as a README or documentation for the service. Below this, the Python code is shown, starting with imports for Flask, logging, and watchdog modules. The code defines service metadata and a list of agents to be watched.

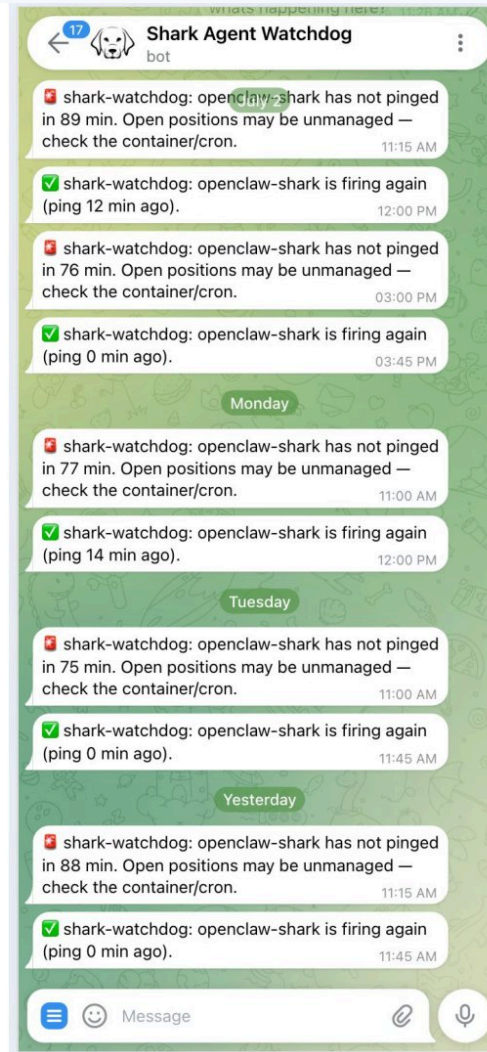
```

1  """shark-watchdog is the dead-man's switch for the trading agents.
2
3  Each agent must ping this service after it runs.
4
5  The watchdog does not trust the agents to report their own health.
6  It stores the pings itself, checks whether the market is open, and pages
7  Telegram if any agent goes stale during trading hours.
8
9  This service fails loud:
10 if an agent disappears, or if alert state is ambiguous, the system pages
11 instead of staying silent
12
13 In other words:
14 shark-control stops bad trades;
15 shark-watchdog tells me when the traders themselves go dark.
16
17 from __future__ import annotations
18
19 import datetime
20 import logging
21 import os
22 import sys
23
24 from flask import Flask, jsonify, request
25
26 from watchdog import gcs, clock, notify
27 from watchdog.check import evaluate
28
29 SERVICE_NAME = "watchdog"
30 SERVICE_VERSION = "1.0.0"
31 AGENTS = ("openclaw-shark", "hermes-shark", "claudio-routines-shark")
32
33 logging.basicConfig(stream=sys.stderr, level=logging.INFO,
34                     format="%(asctime)s %(levelname)s %(message)s")
35 log = logging.getLogger(SERVICE_NAME)
36
37 app = Flask(__name__)
38
39 @app.after_request

```

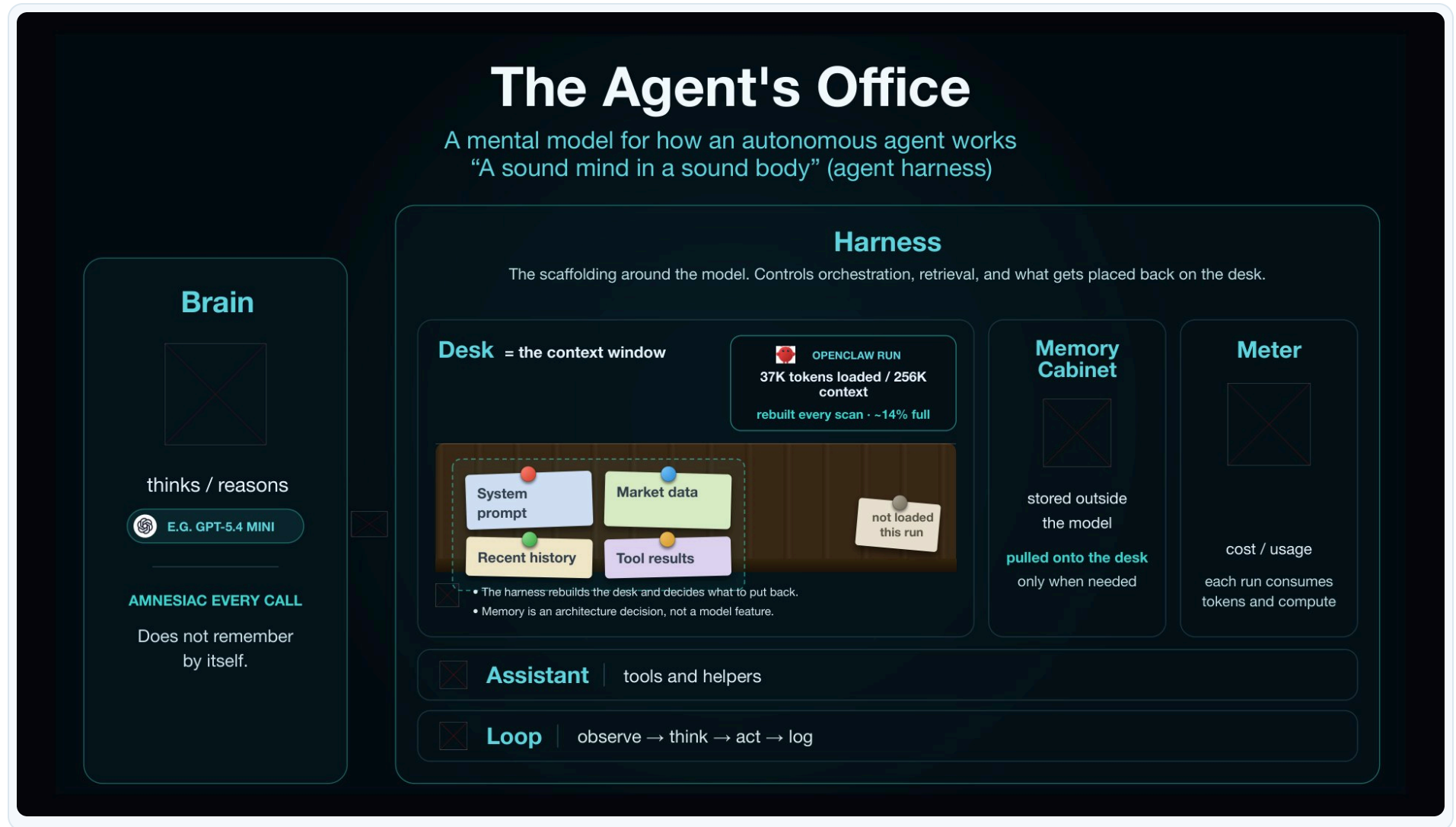
shark-watchdog is the leash. It checks whether the agents are alive, fired on schedule, and didn't stop unexpectedly. Autonomous systems can fail quietly, so you need a second system watching the first — and that system has to outlive the agent.

The leash, in practice



Here is the watchdog doing its job over Telegram: pinging when a container hasn't reported, alerting when an agent may be unmanaged, and noticing when something fires late. This is what 'a second system watching the first' actually looks like, hour to hour.

The agent's office



The mental model: an agent is an LLM in a loop — but the LLM has amnesia and only sees the current context window. So the harness rebuilds the desk every run, deciding which prompt, market data, memory, tools, logs, and history to place in front of the model. The memory lives outside and is pulled back in when needed.

The playbook: HEARTBEAT.md

```

1 summary: "Hermes Shark scheduled scan"
2 read_when:
3   - Heartbeat fires
4   - Scheduled market scan runs
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Here is the actual agent playbook — the HEARTBEAT.md file that tells the agent how to run each scan, built and edited in VS Code with Claude CLI. This is where it gets concrete: it's not 'ask the AI to trade,' it's a detailed operating procedure the agent follows every hour.

HEARTBEAT — preparing before deciding

```

shark-agent-lab > hermes-shark > HEARTBEAT.md > ** # HEARTBEAT.md — Hermes Shark Scan > ** ## Step 2 — Audit Stops
8 # HEARTBEAT.md — Hermes Shark Scan
45 ## Step 1 — Account State
47 Fetch from Alpaca:
49 - 'skills/alpaca/account.sh' — equity, cash, buying power
50 - 'skills/alpaca/positions.sh' — open positions
51 - 'skills/alpaca/orders.sh' — open orders and stops
52
53 Any fetch failure → abort run, log only if useful, no trade. Do not guess broker state.
54
55 **Exit reconciliation (every fire, before Step 2).** Compare the live positions list to the '## Current Portfolio State' block in 'IDENTITY.md' (that block reflects state
56 as of the prior fire's push). For each ticker present in the prior block but **absent** from the current Alpaca positions:
57
58 1. Run 'skills/alpaca/orders.sh closed' and find the SELL fill that closed the position (matching symbol + qty). Capture fill price, filled_at, order id.
59 2. Compute realized P&L from the prior-fire entry price (in 'IDENTITY.md') to the fill price.
60 3. Classify:
61 - Fill price within $0.05 of the prior stop price → **stop hit** (Alpaca's GTC stop auto-executed between fires).
62 - Otherwise → **manual/dire-gate exit** (uncommon under cron; flag for the operator).
63 4. The exit MUST be narrated in Step 8 via the "Stop hit between fires" template — a stop firing IS an actionable event under Production Mode (it changes equity, closes a
64 position, and signals a failed trade thesis; absorbing it into the 16:15 EOD summary loses hours of operator awareness).
65 5. Equity, day P&L, and position count will naturally update in the Step 7 IDENTITY.md overwrite.
66 6. **Reflection grading (Phase B).** Every position the reconciliation found closed this fire gets graded into the journal so the lesson can feed future decisions. For
67 each closed ticker:
68 a. Find its open journal slip — 'skills/reflection/reflection.sh pending' lists pending slips with their original 'date', entry price, and stop (in the 'decision'
69 line). Match by ticker.
70 b. Get the deterministic outcome (raw return + alpha vs SPY; pass the **blended realized R** from the trade-manager audit):
71 'printf '%s' '{"entry_price":ENTRY,"exit_price":FILL,"entry_date":"SLIP_DATE","exit_date":"TODAY","realized_R":R}' | skills/reflection/reflection.sh outcome'
72 c. Read the returned numbers and write a 2-3 sentence lesson **in your own words**: was the directional call correct (cite the alpha figure), which part of the thesis
73 held or failed, and one concrete lesson for the next similar trade.
74 d. Resolve the slip: 'printf '%s' '{"ticker":"TICKER","date":"SLIP_DATE","outcome":OUTCOME_JSON,"lesson":"<your lesson>"}' | skills/reflection/reflection.sh resolve'
75 e. Close its thesis (grades together with the journal): 'printf '%s' '{"ticker":"TICKER","outcome":OUTCOME_JSON}' | skills/thesis/thesis.sh close' — flips the open
76 thesis for that ticker to closed, attaches the outcome, prunes it to the archive. No open thesis for the ticker → harmless no-op.
77 Scale-outs grade **once**, when the last share is gone, using the blended R. 'alpha n/a' (SPY unreachable) is fine — grade anyway.
78
79 This guards against the failure mode where a GTC stop fires between cron fires (broker auto-sells, no agent involvement) and the next fire silently absorbs the loss into
80 the 16:15 EOD summary.
81
82 ## Step 2 — Audit Stops
83 For each open position, confirm:
84
85 - stop exists
86 - distance to stop
87 - position percentage of equity
88 - no conflicting open order exists
89
90 Flag:
91
92 - missing stops
93 - stop buffer below 3%
94 - position over the 20% / $5,000 cap
95 - cash reserve below 10%
96

```

These are the early steps of the routine: check the market, review current positions, understand recent state, and get prepared before making any new decision. An autonomous agent shouldn't jump straight from 'new idea' to 'buy' — it needs a checklist, just like a human analyst.

HEARTBEAT — scan, reason, decide

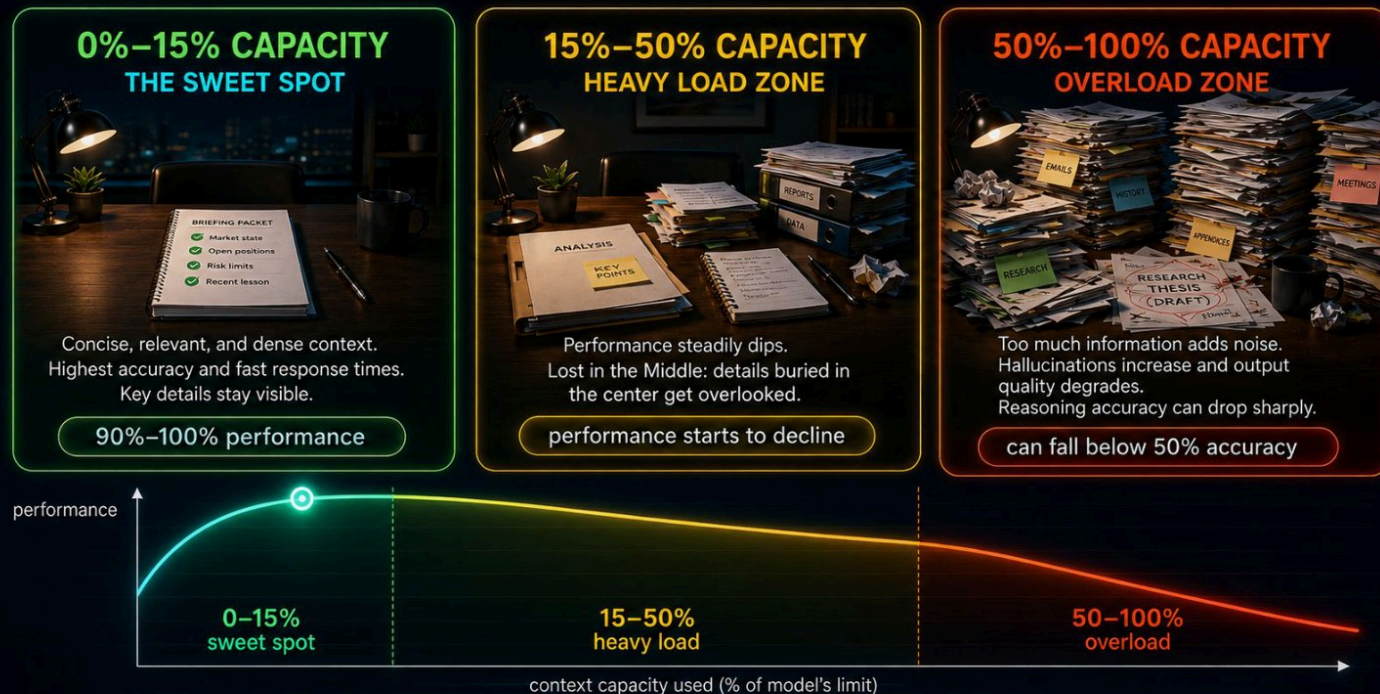
```
shark-agent-lab > hermes-shark > HEARTBEAT.md > ** # HEARTBEAT.md — Hermes Shark Scan > ** ## Step 1 — Account State
132 # HEARTBEAT.md — Hermes Shark Scan
133 ** Step 2 — Regime check (Markov)
134
135 ## Step 2.5 — Regime check (Markov)
136
137 Pre-L2B regime veto guardrail: one shrt-mode check per fire, not per-candidate. **First methodology divergence in the Lab** — see './DECISION-POLICY.md' §8 for the fork
138 rationale, and 'skills/markov-regime/regime.py' for the vendored framework (Roan / @RohOnChain, refactored by Lewis Jackson, vendored 2026-05-29).
139
140 Run once, after Step 2 (audit stops) and before Step 3 (scan):
141
142 ```bash
143 skills/markov-regime/veto.sh SPY
144 ```
145
146 The wrapper writes one JSON line to stdout and exits with:
147
148 - '0' PASS → proceed to Step 3
149 - '10' VETO → skip Step 3, 4, 5, 6. **Step 7 still runs** (log + state + push, so the markov_veto record actually gets committed). Step 8 sends the No-Trade
150 status card (reason 'Markov bear-regime veto').
151 - '20' PASS-THROUGH → yfinance / python / dep / network failure; treat as PASS and proceed to Step 3.
152
153 Do not block trading on a Yahoo hiccup or a missing dep. Exit 20 is recorded for the audit trail but never halts the loop. If 'status-error' with 'reason' starting
154 'deps_missing:' repeats across consecutive fires, the container is missing 'numpy pandas yfinance' and the veto is silently decorative — fix at the container, not in the
155 heartbeat.
156
157 **Scope:** veto applies to NEW long entries only (Step 3 onward). Stop adjustments, dire-gate exits, and existing-position management in Step 2 are NEVER blocked by the
158 regime gate.
159
160 **Veto rule (provisional, revised after 16y SPY calibration; review 2026-06-15):**
161
162 - 'current_regime == "Bear"'
163 - AND 'persistence_diagonal.bear >= 0.50' (this is the 'confidence' field in the JSON; derived as 'persistence_diagonal[current_regime] = P(stay | currently in regime)')
164
165 'stationary_distribution.bear' is recorded in the JSON for diagnostics but NOT used as a gate (originally proposed '> 0.40' cutoff is unreachable on SPY — 16-year max = 0.
166 15).
167
168 **State recording.** Capture the wrapper's stdout JSON and overwrite the 'markov_veto' sub-block of the Current Portfolio State block in 'IDENTITY.md' every fire (PASS,
169 VETO, or PASS-THROUGH). The wrapper also appends the full JSON to today's 'memory/YYYY-MM-DD.md' (gitignored, VPS-local) — that is the calibration log used at the
170 2026-06-15 review. Do not disable it.
171
172 ## Step 3 — Scan
173
174 Source order:
175
176 1. Alpaca broker state (via skills/alpaca/)
177 2. **Earnings reporters first** — stocks that reported earnings today or after-hours yesterday. These have verified catalysts and are less likely to be pure price-movers
178 without fundamentals.
179 3. **Then top movers** (Yahoo Finance gainers, high relative volume) as a secondary screen.
180 4. Web search for specific catalysts on candidates (market-hours only)
181 5. Workspace files ('MEMORY.md' durable lessons)
182
183 **Shared catalyst service ('skills/catalyst/fetch.sh'):** For each candidate, run 'bash skills/catalyst/fetch.sh <TICKER>' and feed the JSON ('news', 'sec_filings',
184 'earnings', 'analyst', 'web_news') into your catalyst reasoning ('web_news' is the shared **Brave** web-news scan — recent cross-web headlines on the same pinned query
185 for all three agents; treat it as catalyst facts alongside 'news', and prefer it over ad-hoc per-agent web search, which diverges across agents) — this is the same data
```

These later steps are where the agent scans, reasons, applies risk rules, and decides whether a trade is even allowed. Autonomy doesn't mean freedom to do anything; it means acting inside a structured process. The more dangerous the action, the more structure you need around it.

The desk has a sweet spot

ACT 2 · THE DESK HAS A SWEET SPOT

More context ≠ more smart



Key takeaway: Keeping active context under 15% of the model's limit usually yields the smartest results. Just because a model can accept a massive amount of tokens doesn't mean it should.

The harness's job is to keep the desk clean, relevant, and useful.

More context does not mean more intelligence. Like your own desk: the right documents in front of you and you think clearly; buried under everything you've ever touched and you miss the important thing. The context window is a working desk, not storage — keep the working set small, relevant, and well-structured.

The meter never stops

ACT 2 · THE METER

What a token is, and why the meter never stops

1 token $\approx$ 4 characters $\approx$ $\frac{3}{4}$ of a word

risk	[risk]	1 token
risky	[risk] [y]	2 tokens
unbelievable	[unbe] [liev] [able]	$\approx$ 3 tokens

One extra letter, double the ticks. The meter doesn't care about meaning — only length.

FROM ONE SENTENCE TO A THIRD OF A BILLION

"This sentence has exactly seven tokens."

7 tokens

$\approx$ \$0.000004

This talk

$\sim$ 15,000

$\approx$ 1¢

One bot, one week

28–37 M

$\approx$ \$14–\$28

cost at **\$0.50–\$0.75 / million** · list price, before caching

The fleet, 45 days

$\approx$ $\frac{1}{3}$ billion

$\approx$ \$170–\$250

Caching changes everything. With cache hits, the whole fleet actually ran on **single-digit dollars a week** — the meter reads gross, the bill reads cached.

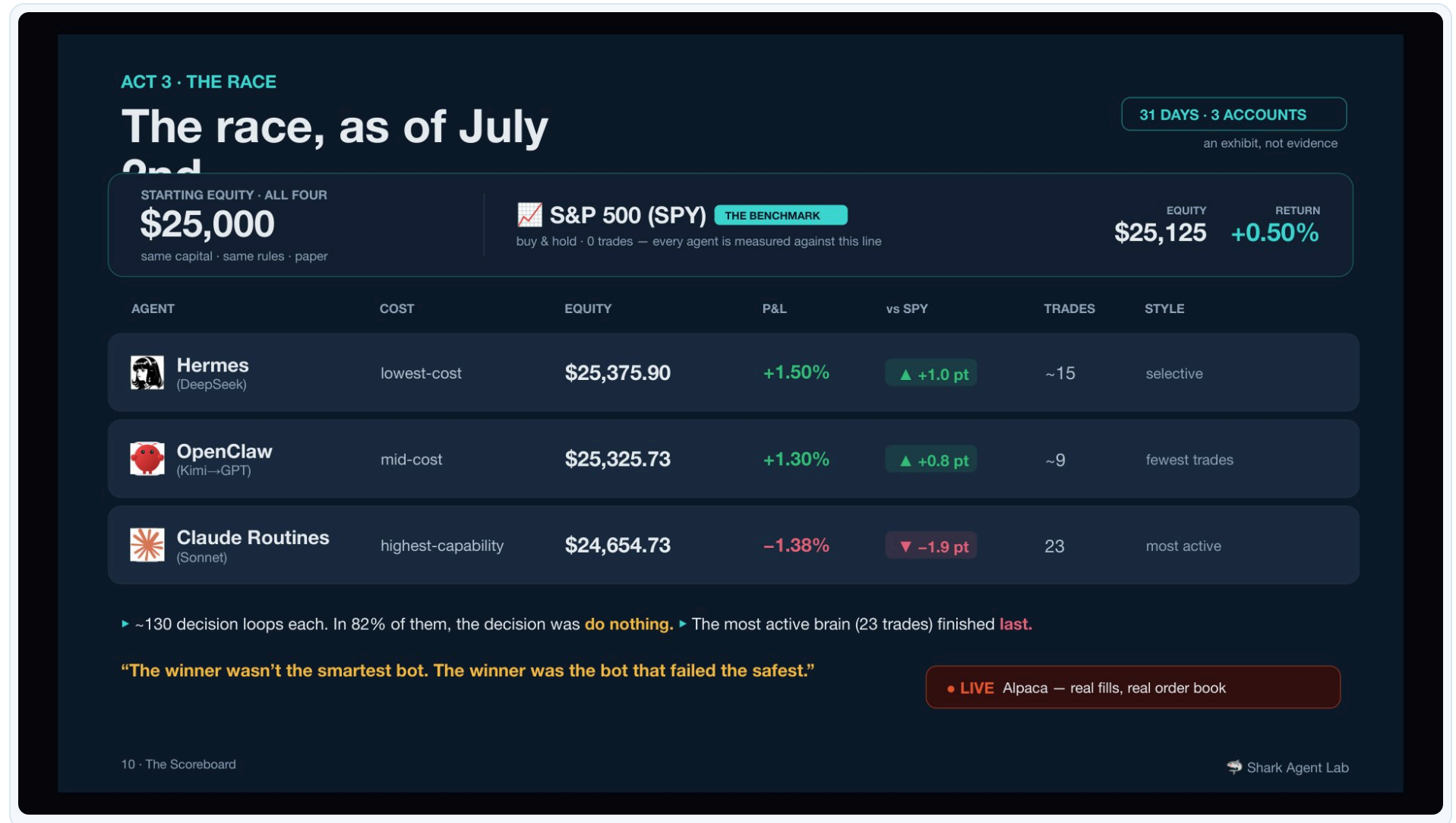
...a third of a billion tokens, to trade three \$25,000 paper accounts — four characters at a time.

09 · The Meter

 Shark Agent Lab

Now the economics. A token is roughly four characters, or about $\frac{3}{4}$ of a word; a one-million-token window is three-to-five Harry Potter books. Impressive — but it means you can accidentally send an enormous amount of text every run. Run hourly and the meter never really stops, so prompt design and memory become cost decisions.

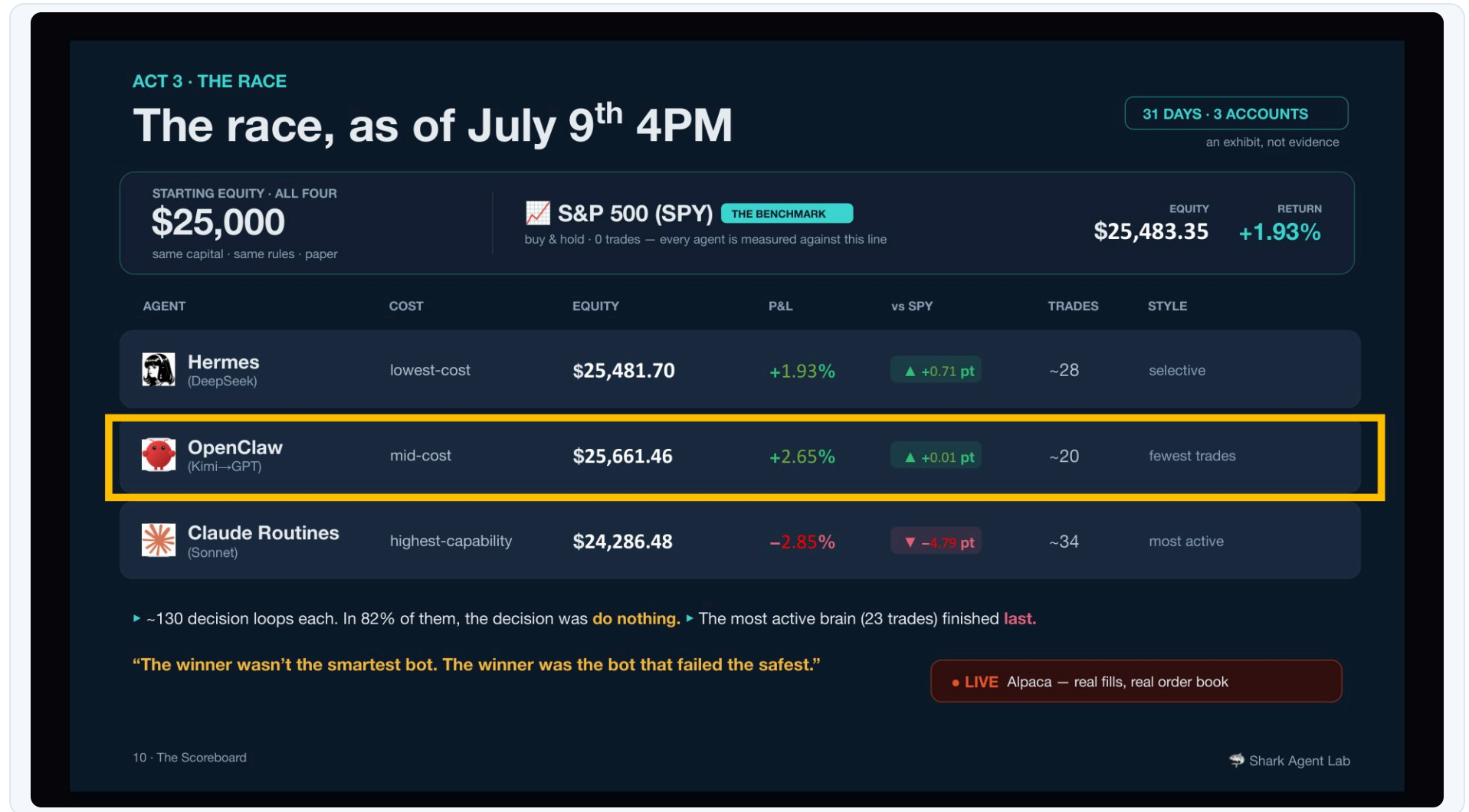
The race — an earlier snapshot (July 2)



The lesson isn’t ‘DeepSeek is the best trader.’ It’s that the harness shaped behavior as much as the brain did.

An earlier snapshot from July 2 — an exhibit, not investment evidence. \$25,000 in the S&P over this period made about \$125. At this point Hermes with DeepSeek was ahead and OpenClaw was also beating the benchmark, while Claude Routines trailed even with the top model — but the standings kept moving, as the next page shows.

The race, updated to July 9



The same board a week later, as of July 9: the benchmark had climbed to +1.93%, OpenClaw finished with the highest return, Hermes behaved selectively and stayed neck-and-neck with the market, and Claude Routines — the most active trader — finished behind. The return winner wasn't the premium model; the sharpest lessons came from how each agent failed.

The Alpaca paper accounts

The screenshot shows the Alpaca paper accounts dashboard. On the left, a sidebar lists several paper accounts, with a yellow box highlighting the 'PAPER ACCOUNT' section. The central area features a price chart for a stock, showing a recent increase of 0.32%. The right sidebar contains trading controls, including a search bar, market price, quantity, order type, and estimated cost. At the bottom, there is a 'Balances' section showing buying power and cash, and a 'Top Positions' table.

Account Name	Balance	Change
OpenClaw - Paper	\$25,661.46	+0.33%
Hermes - Paper	\$25,481.78	-0.31%
ClaudeRoutines - Paper	\$24,286.48	-0.07%

Buying Power	Cash	Daily Change
\$91,186.09	\$16,117.55	\$81.48

Asset	Price	Qty	Market Value	Total P/L (\$)
AMZN	\$246.32	14	\$3,448.48	+\$73.92

This is the Alpaca dashboard for one of the paper accounts. I used Alpaca because it gives a broker-like environment for testing without risking real money. The agents weren't just producing recommendations — they were placing real paper trades through the broker API, which is why guardrails mattered so much.

Orders, positions, and bracket protection

The screenshot shows the Alpaca Markets dashboard in a Chrome browser. The main section displays a table of current positions:

Asset	Price	Qty	Market Value	Total P/L (\$)
GOOGL	\$365.83	10	\$3,658.30	+\$123.90
HOOD	\$112.30	23	\$2,582.90	+\$529.46
IBM	\$304.66	8	\$2,437.28	+\$50.64
AVGO	\$368.25	6	\$2,209.50	-\$33.30
UBER	\$73.89	29	\$2,142.77	-\$13.09
PLTR	\$134.20	8	\$1,073.60	+\$27.38
DKNG	\$27.00	36	\$972.00	+\$97.92

Below the positions table, the 'Recent Orders' section is highlighted with a yellow box. It shows a table of recent orders:

Asset	Order Type	Side	Qty	Filled Qty	Avg. Fill Price	Status	Source	Subn
PLTR	Limit @ \$145.39	sell	8.00	0.00	-	new	access_key	Jul 0
PLTR	Stop @ \$130.78	sell	8.00	0.00	-	held	access_key	Jul 0
PLTR	Market	sell	8.00	8.00	138.44	filled	-	Jul 0
AMD	Market	sell	4.00	4.00	521.04	filled	access_key	Jul 0
AMD	Market	buy	4.00	4.00	510.8725	filled	access_key	Jul 0

On the right side of the dashboard, there are sections for 'API Keys' (Endpoint: https://paper-api.alpaca.markets/v2, Key: [redacted], Regenerate button) and 'Watchlist' (No assets in watchlist, Edit button).

Here the agent's decisions become visible as real broker orders. The key detail: the system used bracket orders — the entry is tied to protection, a take-profit above and a stop-loss below. In plain English, sell high if things go well and stop the damage if they go wrong.

What happened?

POST MORTEM

What happened???

Same market. Same capital. Same rules. Very different behavior.

- **Claude Routines**
Protected every trade — but traded too often.
The order book shows repeated **buy** → **bracket** → **stop-out** → **re-enter** behavior.
- **Hermes**
More restrained.
It still took losses, but it did not keep turning every signal into a new trade.
- **OpenClaw**
Messy, but effective.
It made fewer expensive mistakes and caught enough winners to pull ahead.

WHAT WE'RE COVERING

 Shark Agent Lab

Same market, same capital, same rules — very different behavior.
Claude Routines protected every trade but traded too often, repeatedly buying, bracketing, stopping out, and re-entering. Hermes was more restrained and didn't turn every signal into a trade. OpenClaw was messy but effective, making fewer expensive mistakes and catching enough winners to pull ahead.

The money slide: caching

ACT 3 · THE MONEY SLIDE

Caching = “I've seen this part before”

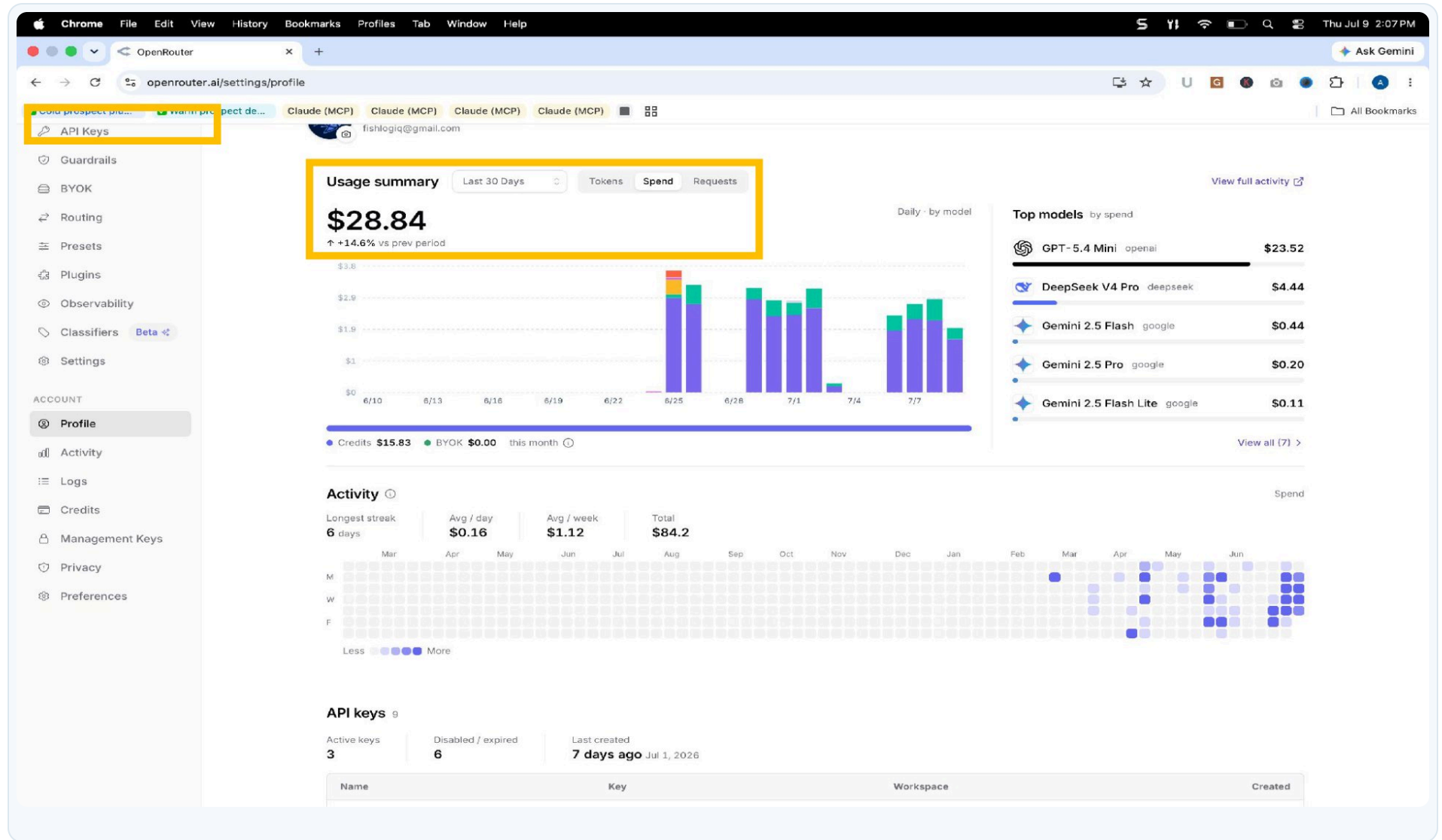
Autonomous agents keep resending the same desk. Cached tokens are billed differently from fresh tokens.



Token count matters — but what really matters is how much of the input is fresh versus cached.

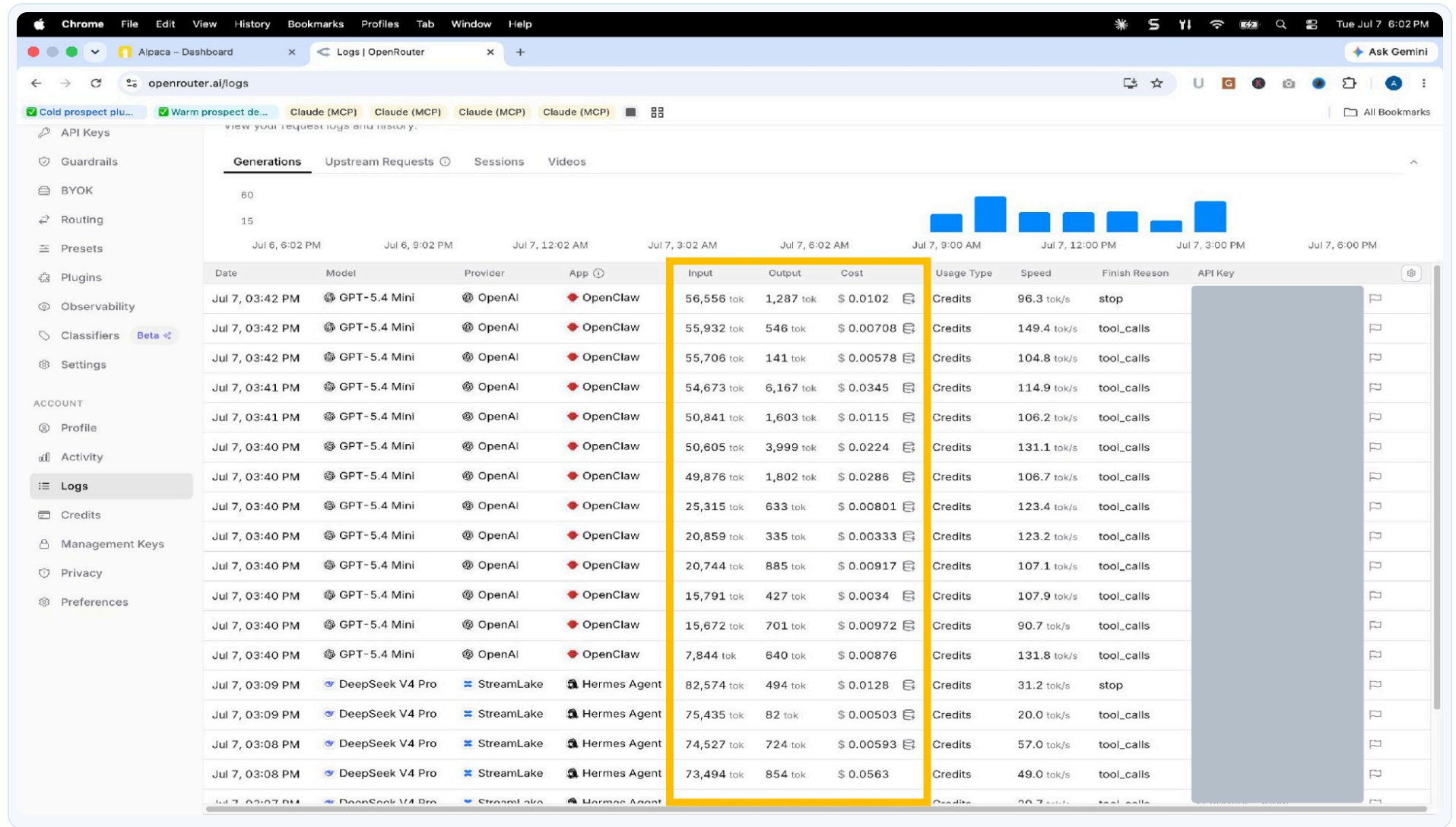
Autonomous agents keep resending the same blocks — identity, rules, playbook, tools, memory, portfolio state. When those repeated blocks stay stable, the provider or gateway can recognize them as cached input and bill them differently from fresh input.

OpenRouter as the meter



This is why I used OpenRouter: visibility into tokens, cost, requests, and cache-hit rates. Without a dashboard like this it's easy to build an agent and have no idea why the bill keeps climbing. OpenRouter became my meter — it showed what each agent was really consuming.

Reading the bill at the request level



Here are the logs at the request level — input tokens, output tokens, cost, model, and caching behavior. Some requests looked small but carried huge context underneath. That's when it clicked: agent cost isn't just the model, it's what the harness sends to the model every single time.

Down to the raw receipt

The screenshot shows the OpenRouter AI interface in a Chrome browser. The left sidebar contains navigation links: API Keys, Guardrails, BYOK, Routing, Presets, Plugins, Observability, Classifiers (Beta), Settings, ACCOUNT (Profile, Activity, Logs, Credits, Management Keys, Privacy, Preferences). The main panel displays a table of generations. The 'Generations' tab is active, showing a list of requests with columns for Date, Model, Provider, App, Input, and Output. A specific generation is highlighted, and a modal window on the right shows its raw receipt. The receipt is a JSON object containing various fields, including 'usage_cache' which is highlighted in yellow.

Date	Model	Provider	App	Input	Output
Jun 10					
Jun 12					
Jun 14					
Jun 16					
Jun 18					
Jun 20					
Jun 22					
Jun 24					
Jul 9, 01:44 PM	GPT-5.4 Mini	OpenAI	OpenClaw	64,037 tok	9 tok
Jul 9, 01:44 PM	GPT-5.4 Mini	OpenAI	OpenClaw	63,919 tok	102 tok
Jul 9, 01:44 PM	GPT-5.4 Mini	OpenAI	OpenClaw	63,854 tok	52 tok
Jul 9, 01:44 PM	GPT-5.4 Mini	OpenAI	OpenClaw	63,272 tok	544 tok
Jul 9, 01:44 PM	GPT-5.4 Mini	OpenAI	OpenClaw	63,047 tok	145 tok
Jul 9, 01:43 PM	GPT-5.4 Mini	OpenAI	OpenClaw	61,926 tok	5,851 tok
Jul 9, 01:43 PM	GPT-5.4 Mini	OpenAI	OpenClaw	59,099 tok	730 tok
Jul 9, 01:43 PM	GPT-5.4 Mini	OpenAI	OpenClaw	58,963 tok	638 tok
Jul 9, 01:43 PM	GPT-5.4 Mini	OpenAI	OpenClaw	58,534 tok	841 tok
Jul 9, 01:42 PM	GPT-5.4 Mini	OpenAI	OpenClaw	57,761 tok	2,867 tok
Jul 9, 01:42 PM	GPT-5.4 Mini	OpenAI	OpenClaw	57,437 tok	39 tok
Jul 9, 01:42 PM	GPT-5.4 Mini	OpenAI	OpenClaw	56,559 tok	704 tok
Jul 9, 01:42 PM	GPT-5.4 Mini	OpenAI	OpenClaw	56,273 tok	1,319 tok
Jul 9, 01:42 PM	GPT-5.4 Mini	OpenAI	OpenClaw	54,911 tok	793 tok
Jul 9, 01:42 PM	GPT-5.4 Mini	OpenAI	OpenClaw	53,917 tok	39 tok
Jul 9, 01:42 PM	GPT-5.4 Mini	OpenAI	OpenClaw	53,732 tok	377 tok
Jul 9, 01:42 PM	GPT-5.4 Mini	OpenAI	OpenClaw	52,688 tok	444 tok
Jul 9, 01:41 PM	GPT-5.4 Mini	OpenAI	OpenClaw	51,580 tok	1,105 tok

```

{
  "provider_name": "OpenAI",
  "model": "openai/gpt-5.4-mini-20260317",
  "app_id": "27256...",
  "external_user": null,
  "streamed": true,
  "cancelled": false,
  "generation_time": 1570,
  "latency": 1196,
  "moderation_latency": null,
  "created_at": "2026-07-09T17:44:31.03Z",
  "tokens_prompt": 60751,
  "tokens_completion": 112,
  "native_tokens_prompt": 63919,
  "native_tokens_completion": 102,
  "native_tokens_completion_images": 0,
  "native_tokens_reasoning": 0,
  "native_tokens_cached": 63488,
  "num_media_prompt": null,
  "num_input_audio_prompt": null,
  "num_media_completion": 0,
  "num_search_results": null,
  "num_fetches": null,
  "origin": "https://openclaw.ai/",
  "usage": 0.00554385,
  "usage_upstream": 0.00554385,
  "finish_reason": "tool_calls",
  "usage_cache": -0.0428544,
  "usage_web": null,
  "usage_web_fetch": null,
  "usage_upstream_web_fetch": null,
  "usage_file": null,
  "byok_usage_inference": 0,
  "provider_responses": [
    {
      "endpoint_id": "9ee065b2-3d1c-43bc-b...",
      "id": "resp...",
      "is_byok": false,
      "latency": 859,
      "model_permalink": "openai/gpt-5.4-mini-20260317",
      "provider_name": "OpenAI",
      "status": 200
    }
  ]
}

```

Zooming into a single generation's raw record shows exactly where the tokens went — including the cached-token fields that explain why two similar-looking requests can cost very differently. The receipt never lies.

DeepSeek vs GPT-5.4 mini vs Kimi

DeepSeek V4 Pro vs GPT-5.4 Mini vs Kimi K2.6

Compare DeepSeek V4 Pro from DeepSeek, GPT-5.4 Mini from OpenAI, and Kimi K2.6 from moonshotai on key metrics including benchmarks, price, context length, and other model features. Access all 3 models and hundreds of others through the [OpenRouter API](#).

V4 Pro		GPT-5.4 Mini		Kimi K2.6	
Author	DeepSeek	Author	OpenAI	Author	moonshotai
Context length	1.05M tokens	Context length	400K tokens	Context length	262K tokens
Reasoning	✓	Reasoning	✓	Reasoning	✓
Input modalities	T, I, A, V, A, V	Input modalities	T, I, A, V, A, V	Input modalities	T, I, A, V, A, V
Output modalities	T, I, A, V, A, V	Output modalities	T, I, A, V, A, V	Output modalities	T, I, A, V, A, V
Providers	16 providers	Providers	2 providers	Providers	21 providers
Pricing					
Input	\$0.435 / M tokens	Input	\$0.75 / M tokens	Input	\$0.66 / M tokens
Output	\$0.87 / M tokens	Output	\$4.50 / M tokens	Output	\$3.41 / M tokens
Cached input	\$0.003625 / M tokens	Cached input	\$0.075 / M tokens	Cached input	\$0.14 / M tokens
Images	—	Images	—	Images	—

Don't just ask 'which model is cheapest per million tokens?'
Ask 'how does it behave inside my actual harness?'

I assumed the cheapest-looking model would automatically be the cheapest to operate. Not always true — caching behavior, reasoning tokens, and input/output pricing all change the final cost.

Paying rent on a music studio

ACT 3 · WHAT THE BILL WAS BUYING · 🦈 OPENCLAW

My stock bot paid rent on a music studio

ONE HOURLY FIRE — THE RECEIPT

Tokens in	37,000
Tokens out	400
Ratio	90 : 1
...of which tool manuals	~9,910
Manuals never used	65–73%

Unused manuals loaded every fire: video · music · TTS — 0 calls, billed hourly.



Tools cost tokens when they're **available** — even if never called.

THE FIX: one deny-list line → **–25% per request**, zero behavior change.

“Availability is the meter, not usage.”

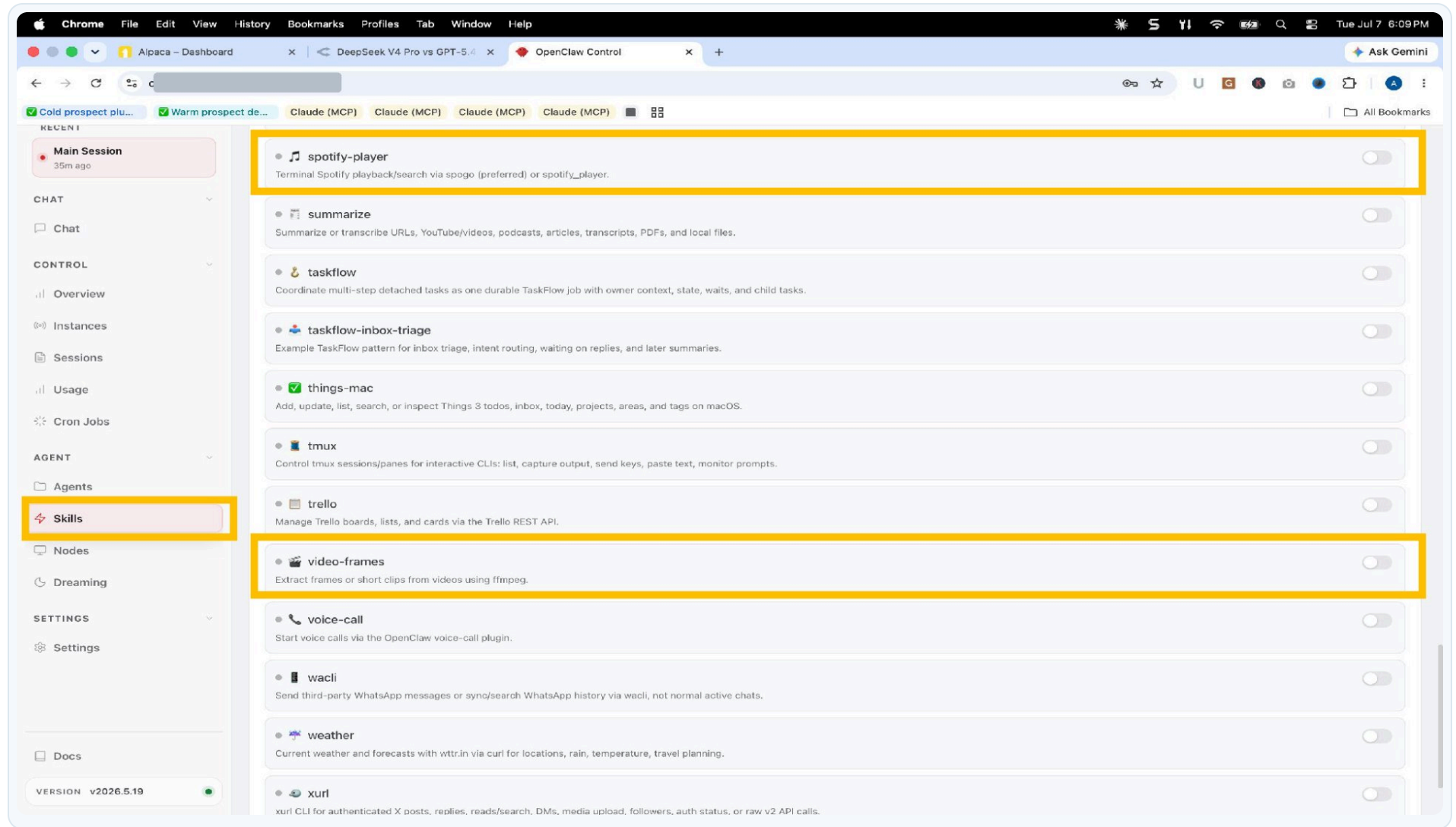
12 · The Tool Tax

🦈 Shark Agent Lab

Tools cost tokens when they're available, not only when they're called.

The biggest surprise was tool sprawl. OpenClaw had many skills on by default, so even when the agent never used video, music, or text-to-speech, those manuals were still loaded into context — my stock-trading bot was paying rent on a music studio it never used.

Turning off what you don't need



Here is where I disabled the unnecessary skills. This one change shrank the request without changing trading behavior at all — a key pattern in agent design. Before you optimize the model, optimize what you send it: a smaller tool surface is cheaper, safer, and easier to reason about.

The playbook was silently cut to a third

ACT 3 · THE SCARIEST ONE · 🦈 OpenClaw

The playbook was silently cut to a third

What we wrote

HEARTBEAT.md — 36,326 characters

What the model was handed — every fire

12,000 chars

← cut at ~Step 3 of 9, silently — all dashboards green

workspace file HEARTBEAT.md is 35868 chars (limit 12000); truncating

The model can't miss what it never received.
The cause: a default nobody set. The fix: one config line.

"You don't run the agent you wrote. You run the agent your harness assembled."

• LIVE VPS — wc -c → 36326 · the knob → 50000

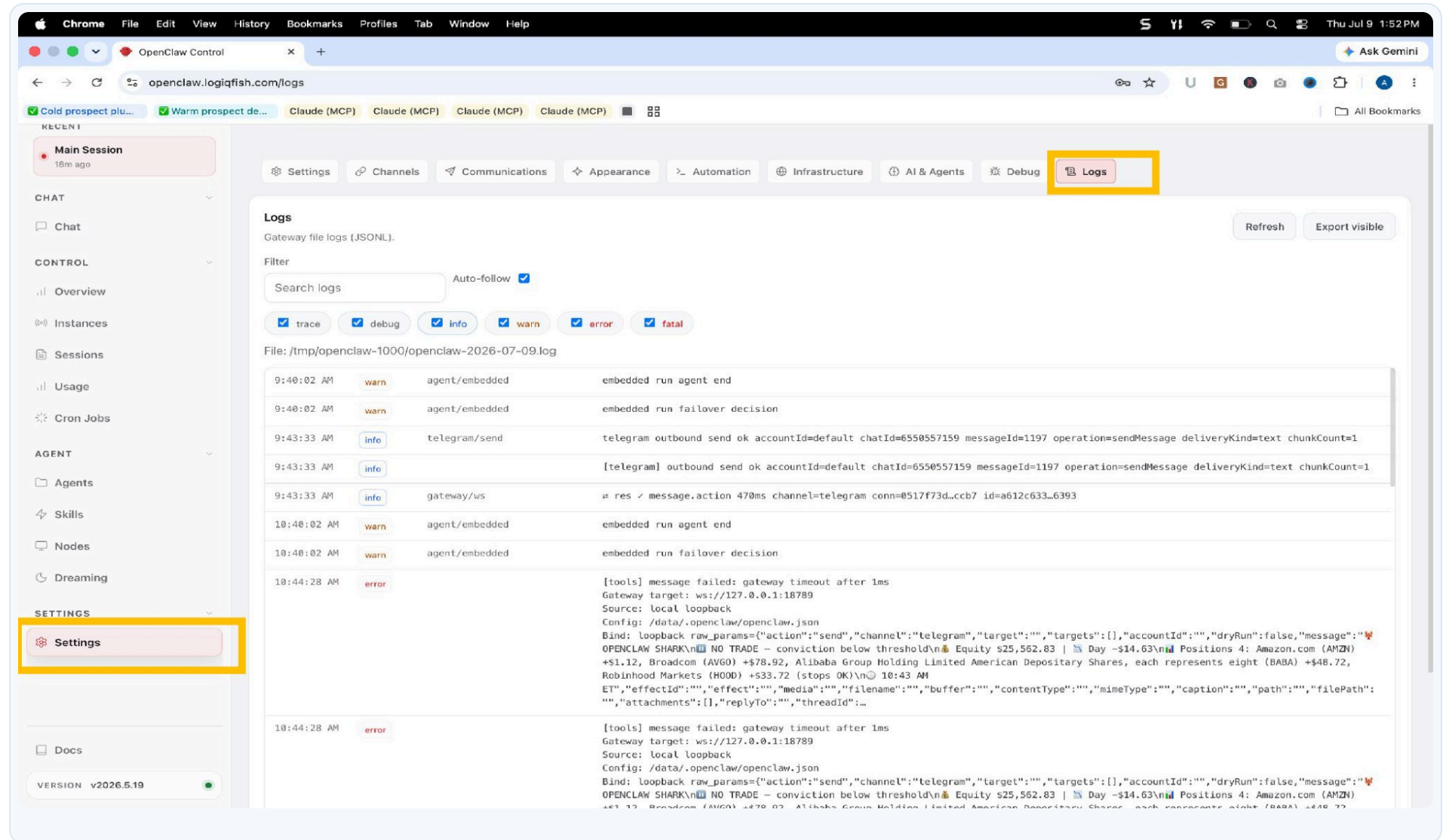
13 · The Silent Two-Thirds

🦈 Shark Agent Lab

You don't run the agent you wrote. You run the agent your harness actually assembled.

The scariest discovery. My HEARTBEAT.md was about 36,000 characters, but OpenClaw was silently truncating it to 12,000 — the model only received the first third. The dashboards were green, the agent didn't complain, and it couldn't ask for steps five through nine because it never knew they existed.

Green dashboards, hidden cause



The control panel looked perfectly healthy — settings in place, logs streaming. Nothing here hinted that the playbook was being cut to a third. The failure lived one layer below the UI, which is exactly why the next slide matters.

The terminal told the truth

```

root@1fba842eea00:~#
root@1fba842eea00:~#
root@1fba842eea00:~# echo "=== OPENCLAW LIVE RECEIPT ==="
=== OPENCLAW LIVE RECEIPT ===
root@1fba842eea00:~# wc -c /data/.openclaw/workspace/HEARTBEAT.md
36326 /data/.openclaw/workspace/HEARTBEAT.md
root@1fba842eea00:~# grep -n "bootstrapMaxChars" /data/.openclaw/openclaw.json
115:   "bootstrapMaxChars": 50000,
root@1fba842eea00:~# head -40 /data/.openclaw/workspace/HEARTBEAT.md

summary: "OpenClaw Shark scheduled scan"
read_when: [Heartbeat fires, Cron starts]
---

# HEARTBEAT.md — OpenClaw Shark Scan

Runs many times per market day. Keep tight.

## Step 0 — Market Gate

Always first.

1. If today is Saturday or Sunday → send the closed-market heartbeat Telegram (Step 8 template), then run `bash skills/watchdog/ping.sh` (Step 9 liveness ping; fail-open) and reply exactly `HEARTBEAT_OK`. Stop. No APIs, no web, no further work.
2. If weekday → check Alpaca `/v2/clock`.
3. If Alpaca `is_open=false`, including holidays → send the closed-market heartbeat Telegram (Step 8 template), then run `bash skills/watchdog/ping.sh` (Step 9 liveness ping; fail-open) and reply exactly `HEARTBEAT_OK`. Stop. No web, no scan.
4. The closed-market gate decision and the Telegram message are both produced by the runtime's assigned cron model per `AGENTS.md` Model Policy. The gate path may only return `HEARTBEAT_OK`, stop early, or escalate to the primary trade model. It may not scan, score, trade, or recommend.
5. Once Step 0 passes and market is open, the rest of this scan runs on the primary trade model assigned by the runtime, per `AGENTS.md` Model Policy.

## Step 0.5 — Control gate (kill switch + paper/live)

Runs once, after Step 0 (market gate) and before Step 1, on every open-market fire.

Run: `bash skills/control/control.sh`

- `0` TRADE → trading allowed; use the printed mode (`paper`|`live`) as this fire's broker target.
- `10` HALT → kill switch engaged OR unauthorized live keys. **Skip Steps 3–6 (no new entries).** Still run Steps 1–2 (manage open positions: exit reconciliation, audit stops, position management, close-protection) and Step 8 (status card). Open positions ride their resting GTC brackets.
- `20` HALT → control service unreachable / non-200 → treat exactly like `10` (fail-closed); reason "control gate unreachable".

**Scope:** like the Markov veto (Step 2.5), the halt applies to NEW long entries only. Stop adjustments, dire-gate exits, and existing-position management in Steps 1–2 are NEVER blocked by the control gate.

On a halt (exit 10 or 20), the Step 8 status card reads: `🛑 KILL SWITCH — new entries halted; N positions held under brackets.`

## Step 1 — Account State

Fetch from Alpaca:

- equity
  
```

The terminal is where the truth came out — the logs showed the actual truncation message. This is why logs are the black-box recorder of an agent: dashboards can look healthy while the real problem is buried in the raw logs. If you build agents, don't just watch the pretty UI — read the logs.

One agent ignored the procedure

ACT 3 · SAME WORLD, DIFFERENT MINDS

We shipped the identical procedure to all three. One ignored it.

The procedure — a byte-for-byte **bull** → **bear** → **referee debate**; the referee's 0–100 conviction feeds the floor-65 gate.



Hermes
DeepSeek

✓ **ran it**

7 real debate transcripts logged — AVGO 62 · MRVL 59 · MU 52 · AMD 48 · KMX 45 · CIFS 48 · QBTS 58



Claude Routines
Sonnet

✓ **ran it every fire**

first debate-driven BUY — MRVL x5 @ \$322.28, conviction 67



OpenClaw
Kimi K2.6

✗ **silently skipped it**

zero debates ever recorded — emitted a one-shot **"Score ~58"**, and the gate couldn't tell it apart

ROOT CAUSE — the debate was layered as an override, but each **IDENTITY.md** still carried the old single-shot framework. Two competing instructions in context — Kimi anchored on the stale one.

The fix wasn't a louder instruction — it was deleting the competing one.

14 · The Silent Skip

Shark Agent Lab

Prompts aren't a shouting match. The fix was deleting the competing instruction completely.

I shipped the same debate procedure to all three — bull case, bear case, referee, conviction score, decision. Hermes and Claude Routines ran it; OpenClaw skipped it, not from laziness but because an old competing instruction still lived in its context. The fix wasn't a louder instruction — it was deleting the conflicting one at its source.

The day it sold its own winners

ACT 3 · THE DAY IT SOLD ITS OWN WINNERS · 🦈 OpenClaw · 2026-06-22

A healthy agent panic-sold two protected positions

12:42 Buys ABBV + HIVE — both correctly protected ✓

13:34 Misreads its own order book: “no stop visible”

13:35 Tries to re-protect → broker: already protected

13:37 Panic. MARKET-SELLS both healthy positions.

Cost —\$36.28 · root cause: ONE missing query flag

Why \$36 and not \$3,600: the risk kernel is code — the model can’t override it.

“The LLM is not the guardrail. The code is the guardrail.”

 **POLL** Would you let an AI trade \$1,000 overnight?

15 · The False Stop

 Shark Agent Lab

The LLM is not the guardrail. The code is the guardrail.

OpenClaw bought ABBV and HIVE, both correctly protected. Then a coding bug made it misread its own order book and believe there was no stop-loss. It tried to re-protect them, the broker said they were already protected, and it panic-sold both healthy positions. The loss was about \$36, not \$3,600, because the risk kernel lived in code.

The order history behind the incident

Symbol	Order Type	Side	Quantity	Price	Status	Access Key	Created At	Filled At	Expires At
AAL	Market	buy	213.00	213.00	filled	access_key	Jun 25, 2026, 01:12:43 PM	Jun 25, 2026, 01:12:47 PM	Sep 23, 20
AAL	Stop @ \$16.70	sell	213.00	0.00	canceled	access_key	Jun 25, 2026, 01:12:43 PM	-	Sep 23, 20
BB	Limit @ \$11.16	sell	442.00	442.00	filled	access_key	Jun 25, 2026, 12:09:48 PM	Jun 26, 2026, 10:21:31 AM	Sep 23, 20
BB	Market	buy	442.00	442.00	filled	access_key	Jun 25, 2026, 12:09:47 PM	Jun 25, 2026, 12:09:48 PM	Sep 23, 20
BB	Stop @ \$9.90	sell	442.00	0.00	canceled	access_key	Jun 25, 2026, 12:09:47 PM	-	Sep 23, 20
ZETA	Stop @ \$18.20	sell	86.00	86.00	filled	access_key	Jun 25, 2026, 09:45:43 AM	Jun 25, 2026, 09:45:44 AM	Sep 21, 20
ZETA	Limit @ \$22.52	sell	86.00	0.00	canceled	access_key	Jun 23, 2026, 01:45:23 PM	-	Sep 21, 20
ZETA	Market	buy	86.00	86.00	filled	access_key	Jun 23, 2026, 01:45:20 PM	Jun 23, 2026, 01:45:23 PM	Sep 21, 20
HIVE	Market	sell	218.00	218.00	filled	access_key	Jun 22, 2026, 01:37:57 PM	Jun 22, 2026, 01:37:58 PM	Jun 22, 2
ABBV	Market	sell	14.00	14.00	filled	access_key	Jun 22, 2026, 01:37:57 PM	Jun 22, 2026, 01:37:58 PM	Jun 22, 2
ABBV	Limit @ \$257.03	sell	14.00	0.00	canceled	access_key	Jun 22, 2026, 12:42:53 PM	-	Sep 18, 2
HIVE	Limit @ \$6.26	sell	218.00	0.00	canceled	access_key	Jun 22, 2026, 12:42:52 PM	-	Sep 18, 2
ABBV	Market	buy	14.00	14.00	filled	access_key	Jun 22, 2026, 12:42:50 PM	Jun 22, 2026, 12:42:53 PM	Sep 18, 2
HIVE	Market	buy	218.00	218.00	filled	access_key	Jun 22, 2026, 12:42:50 PM	Jun 22, 2026, 12:42:52 PM	Sep 18, 2
ABBV	Stop @ \$218.47	sell	14.00	0.00	canceled	access_key	Jun 22, 2026, 12:42:50 PM	-	Sep 18, 2
HIVE	Stop @ \$4.55	sell	218.00	0.00	canceled	access_key	Jun 22, 2026, 12:42:50 PM	-	Sep 18, 2
DKNG	Stop @ \$27.47	sell	64.00	64.00	filled	access_key	Jun 17, 2026, 10:57:48 AM	Jun 17, 2026, 10:57:48 AM	Sep 14, 20
E	Market	sell	1.00	1.00	filled	access_key	Jun 17, 2026, 09:53:36 AM	Jun 17, 2026, 09:53:37 AM	Jun 17, 20
E	Stop @ \$14.50	sell	2.00	1.00	canceled	access_key	Jun 17, 2026, 09:53:33 AM	Jun 17, 2026, 09:53:34 AM	Sep 15, 20
E	Limit @ \$17.00	sell	2.00	0.00	canceled	access_key	Jun 17, 2026, 09:53:32 AM	-	Sep 15, 20
E	Market	sell	2.00	2.00	filled	access_key	Jun 17, 2026, 09:53:30 AM	Jun 17, 2026, 09:53:31 AM	Jun 17, 20
E	Limit @ \$17.00	sell	4.00	0.00	canceled	access_key	Jun 17, 2026, 09:53:27 AM	-	Sep 15, 20

This Alpaca order history shows the incident in full — the buys, the protective orders, and the later sells. It's the case for broker-side protection: if the model gets confused, the broker and the code still have to protect the account. You don't want your entire safety system living inside the prompt.

The seven sins of autonomous agents

ACT 4 · WHAT 31 DAYS TAUGHT US

The seven sins of autonomous agents

Every one showed up in the lab.

1

Context gluttony

Treating the context window like storage

Keep the desk a briefing, not an archive.

2

Memory hoarding

Saving everything and calling it intelligence

A junk drawer is not memory.

3

Tool sprawl

Exposing every tool “just in case”

You pay for what's available, not what's used.

4

Prompt-only safety

Putting risk logic where the model can skip it

Prompts are not guardrails. Code is.

5

Dashboard faith

Trusting green lights instead of raw logs

Logs are the black-box recorder.

6

Invisible defaults

Letting settings you never chose define the agent

The most dangerous setting is the one you never set.

7

Autonomy cosplay

Letting the agent act without an external leash

Kill switches and watchdogs must outlive the agent.

“The more autonomous the agent became, the less I trusted the prompt — and the more I trusted the system around it.”

16 · The Seven Sins

Shark Agent Lab

Every one of these showed up in my lab: context gluttony (treating the window like storage), memory hoarding (saving everything and calling it intelligence), tool sprawl (exposing every tool just in case), prompt-only safety (risk logic the model can skip), dashboard faith (green lights over logs), invisible defaults, and autonomy cosplay (acting without an external leash).

The anatomy, in a box you can run

ACT 4 · TAKE ONE HOME

The anatomy, in a box you can run

- ▶ **Two keys.** Alpaca paper + any LLM.
- ▶ **The whole anatomy:** loop · memory · state · tools · gates.
- ▶ The LLM proposes. **The code decides.**

**Everything you just saw is in this box.
Clone it, run it, make it yours —
a real trading agent you can take apart.**

Paper trading only. Educational architecture lab — not financial advice.

17 · The Giveaway



github.com/logiqfish/shark-trading-agent

 Shark Agent Lab

Everything tonight isn't just a trading story — it's an agent-architecture story. The GitHub repo gives you the anatomy in a box: the loop, memory, state, tools, gates, and broker integration. You need an Alpaca paper account and an LLM key. It's paper trading and educational only, but you can clone it, run it, and learn how these systems behave.

Take it apart

The screenshot shows a web browser window displaying the GitHub repository for `logiqfish/shark-trading-agent`. The browser's address bar is highlighted with a yellow box, showing the URL `github.com/logiqfish/shark-trading-agent`. The repository page includes a file list on the left, a central README section, and a right-hand sidebar with repository statistics and additional links.

Repository Files:

- `cron`: docs: act on repo review — fix LLM-key location, clarify cr... (5 days ago)
- `docs`: release: v1.0.3 — fix WHAT_YOU_GET.md dry-demo setup ... (5 days ago)
- `skills/shark`: docs/kit: align SETUP.md LLM-key path with .env; fix stale ... (last week)
- `.gitignore`: Shark Starter Kit — initial public release (last week)
- `AGENTS.md`: release: v1.0.2 — fix AGENTS.md LLM-key rule, align setup... (5 days ago)
- `DISCLAIMER.md`: release: v1.0.2 — fix AGENTS.md LLM-key rule, align setup... (5 days ago)
- `LICENSE`: Shark Starter Kit — initial public release (last week)
- `README.md`: docs(README): soften 'Beyond stocks' — drop named pre... (5 days ago)
- `SETUP.md`: release: v1.0.2 — fix AGENTS.md LLM-key rule, align setup... (5 days ago)
- `SOUL.md`: kit: make Telegram output clean plain text (no literal ** ast... (last week)
- `distribution.yaml`: release: v1.0.3 — fix WHAT_YOU_GET.md dry-demo setup ... (5 days ago)

README Section:

Shark Trading Agent — Starter Kit

trading paper-only it actually trades hosting self-hosted setup 2 keys license MIT educational not financial advice

Stable presentation release: **v1.0.3**. `main` is the live version you install; see [Releases](#) for the frozen tagged package.

A skinny, **paper-only** trading bot you run on your own VPS as a **Hermes Agent** profile distribution, with **two keys**. Unlike research/rating-only agent demos, **this one actually trades** — with discipline:

Repository Statistics:

- About:** Self-hosted paper-only autonomous trading agent for Hermes — two keys (Alpaca paper + an LLM). Educational, not financial advice.
- Releases:** 4 releases. Latest: **v1.0.3 — WHAT_YOU_GET....** (5 days ago).
- Packages:** No packages published.
- Contributors:** 2 contributors: `logiqfish` (LogiqFish AI & SEO Mar...) and `claude` (Claude).
- Languages:** Python 93.8%, Shell 6.2%.

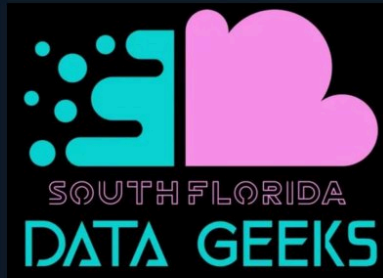
Here is the actual repository. My goal isn't for you to blindly run a trading bot and trust it — it's for you to take it apart. Change the model, the prompt, the risk rules, the tools, and watch how the behavior changes. That is how you learn agent engineering.

Thank you

Take home thought:
“A sound mind in a sound body”
(i.e. agent harness)

Thank You For Attending

Thank you to South Florida Data Geeks.



Thank you.



Shark Agent Lab

The LLM proposes, but the system decides. The mind matters
— but the harness matters more.

Thank you again to South Florida Data Geeks, to the organizers, and to
everyone who came out.