Graceful Degradation in Subscription-Constrained Multi-Agent Orchestration Systems

Miguel Angel Rodriguez Marquez [1]

Abstract

Multi-agent LLM systems deployed under fixed-cost subscription budgets exhibit a distinct class of failure modes absent from the research literature, which uniformly assumes unbounded API access. We present FLOTILLA, a continuously-running autonomous multi-agent management plane operating on commodity hardware in a hybrid local/cloud configuration. We document three production failure classes with precise log evidence: environment isolation mismatch, status propagation failure, and the *dual-offline dispatcher loop*, which produced 7,366 task reassignment cycles over 18 hours 58 minutes from a single misrouted operator command. In direct response, we designed four architectural patterns: a four-layer circuit breaker with persistent state, a SHA-256 pre-invocation checksum gate, a branch-plus-handoff protocol for mid-task continuity, and a structured lessons ledger. Seventy-five days of post-intervention telemetry (594 tasks, 56,275 heartbeats, 276 lessons; March 14–May 27, 2026) validate these patterns: the maximum reassignment count on any task across the full window is 2; all 594 tasks reached `approved` status; the checksum gate eliminated LLM invocation in 96.1% of scheduled wakeups; and agent availability data confirms the economic model—locally-hosted agents exhibit 2 unavailability events over 75 days, while subscription-constrained agents exhibit up to 215. We argue that designing for the economically-constrained case produces orchestration architecture resilient to all forms of agent unavailability.

1. Introduction

Multi-agent LLM systems have attracted substantial research attention, producing failure taxonomies (Cemri et al., 2025), formal runtime infrastructure frameworks (Cruz, 2026), and analyses of economic inefficiency (Lin et al., 2025). A consistent property of this body of work is an implicit assumption that agents operate under effectively unbounded API budgets: researchers provision the tokens required for each experiment and build systems that assume agents remain available until explicitly terminated.

Production deployments face a structurally different problem. Teams operating 24/7 autonomous agent fleets must choose a billing model. Variable-cost (per-token API) billing provides unlimited availability in principle but introduces unbounded financial exposure: a dispatcher fault or runaway loop generates invoice charges before any human can intervene. Fixed-cost (per-seat subscription) billing caps monthly expenditure and eliminates surprise charges, but introduces a qualitatively new failure mode: agents go dark when their periodic token allotment is exhausted, at unpredictable times, without signaling to the orchestration layer.

This paper describes FLOTILLA, an autonomous multi-agent management plane in continuous operation since February 18, 2026, under fixed-cost subscription constraints. The fleet runs on commodity hardware in a hybrid local/cloud configuration and orchestrates agents backed by frontier LLMs under subscription billing alongside locally-hosted agents with no per-token cost.

Contributions. **(1)** A failure taxonomy specific to subscription-constrained multi-agent orchestration, with three documented failure classes grounded in production logs. **(2)** Four architectural patterns that address these failures, presented as platform-agnostic design decisions. **(3)** Empirical validation from 75 days of post-intervention telemetry on a live production system (594 tasks, all approved).

2. Related Work

Multi-agent failure taxonomies. Cemri et al. (2025) present a taxonomy of 14 failure modes across popular multi-agent frameworks, finding that prompt engineering interventions yield only +14% improvement and that failures indicate fundamental design flaws. Their analysis is consistent with our observation that each documented failure arose from correct individual components interacting incorrectly, but does not address resource-constrained de-

ployments. Cruz (2026) formalise a distinct execution-time layer for agentic systems, arguing that costly failures occur at runtime after planning has begun—directly framing our primary incident (§5).

Economic constraints. Ye & Tan (2026) formalise resource-bounded autonomous AI systems and identify a critical asymmetry: resource consumption is only known after an invocation completes. This asymmetry underlies our non-deterministic exhaustion property (§4). Their treatment addresses the single-agent case; we extend the analysis to multi-agent orchestration, where independent exhaustion events on multiple agents create emergent fleet-level failure modes. Lin et al. (2025) document economic inefficiency as a barrier to real-world multi-agent deployment. Our pre-intervention context-reload pattern—re-reading the full shared context file on every scheduled wakeup—is precisely the "redundant or low-yield" consumption pattern their analysis predicts.

Resilience patterns. Mohammad (2025) review circuit-breaker recovery patterns, finding that pattern effectiveness depends on failure semantics and that over-tight thresholds reduce throughput. This informed our choice of a three-strike threshold with a 10-minute window (§6). Liu et al. (2025) propose a hybrid edge-cloud resource allocation system achieving 30% cost reduction by routing 45% of subtasks to local hardware. Our fleet architecture shares structural similarities but uses resource class (subscription cloud vs. always-available local) as the routing criterion.

The gap. No prior work addresses the orchestration-layer challenges specific to subscription-budget fleets operating continuously in production. This paper provides evidence for that case.

3. System Description

3.1. Hardware and Deployment Topology

The production deployment runs on a single 10-core ARM workstation (4 performance + 6 efficiency cores, 16 GB unified memory, 512 GB NVMe SSD) located in Zurich, Switzerland. The single-node constraint is deliberate: it reflects the economically-constrained model and keeps operational complexity minimal.

A publicly-accessible dashboard is served from a remote virtual machine. A local push process reads from the operational database every 60 seconds and transmits a signed snapshot to the remote server; the dashboard serves all queries from this cached state. The operational database itself is never publicly reachable.

3.2. Agent Roster

The fleet comprises six agents in two deployment classes. *Subscription-constrained agents* invoke frontier LLMs via CLI tools and operate under weekly token allotments. *Locally-hosted agents* run inference on-device with no per-token cost.

Table 1. Agent roster. Cloud agents share a weekly token allotment; local agents carry no per-invocation cost.

Agent	Role	Class	Schedule
Gem	Infra & Context	Cloud	:00,:10,…
Codi	QA & Velocity	Cloud	:02,:12,…
Clau	Logic & Impl.	Cloud	:04,:14,…
Misty	Compliance & Ethics	Cloud	:06,:16,…
Gemma	Local Cost Guard	Local	:08,:18,…
OpenClaw	Always-On Orch.	Local	Continuous

OpenClaw differs structurally: it runs as a persistent process rather than a scheduled session and is therefore never subject to subscription exhaustion. It is the natural circuit-breaker escalation target.

3.3. Scheduling and Concurrency

Agents are staggered in 2-minute intervals within each 10-minute window: `:00 Gem`, `:02 Codi`, `:04 Clau`, `:06 Misty`, `:08 Gemma`, repeating each hour. This gives each agent an exclusive 2-minute window. The ordering is meaningful: Gem commits output; Codi fires 2 minutes later when output is readable; Clau can peer-review what Gem produced. Because the operational database uses a single-writer SQLite backend, this stagger eliminates write contention without application-level locking. Cross-model peer review emerges directly from scheduling discipline.

Persistent infrastructure services (database, dispatcher, command bridge, push process) run with automatic restart on failure. Scheduled agent sessions are explicitly non-persistent: they run, complete work, and exit. This keeps each session cost-bounded.

The reference implementation uses macOS `launchd`, but the patterns—sequential stagger, automatic restart, session-bounded execution—are platform-agnostic and applicable with `systemd`, Kubernetes CronJobs, or any periodic scheduler.

3.4. Operational Data Layer

An embedded database (single binary, SQLite-backed, REST API) serves as shared operational memory with five collections: **tasks** (core execution units with a constrained status state machine), **heartbeats** (per-agent liveness records, overwritten each session), **comments** (append-only activity feed forwarded to the operator in real time),

lessons (structured evolutionary memory ledger with confidence scores), and **task_events** (append-only telemetry log introduced in v0.4.0).

3.5. Agent Session Lifecycle

Each scheduled session follows four phases. **Phase 0** (pre-invocation checksum gate): a lightweight process computes SHA-256 hashes of the shared context document, the inbox, and the task store's latest timestamp; if all match the previous cycle's cache, the session exits immediately—zero LLM tokens consumed. A relevance filter further requires that the agent's name appear in the open-work section or an inbox message be addressed to it. **Phase 1** (context assembly): the context document, agent mandate, top lessons by confidence score, and task assignments are assembled into a structured prompt. **Phase 2** (LLM invocation). **Phase 3** (session summarisation): a post-processing script extracts structured lessons and writes them to the lessons collection, regardless of whether invocation succeeded.

4. The Economic Constraint Model

4.1. The Billing Model as Architectural Constraint

Production multi-agent deployments must choose between two billing models. *Variable-cost* (per-token API) billing provides unlimited availability in principle but is financially unbounded by design. *Fixed-cost* (per-seat subscription) billing caps expenditure but introduces periodic, economically-driven agent unavailability—agents go dark when the allotment is exhausted and recover only at the next reset.

FLOTILLA uses fixed-cost subscriptions for all cloud agents. This transforms financial risk into a technical problem: how to build an orchestration layer resilient to predictable, periodic agent unavailability.

4.2. Properties of the Subscription-Constrained Agent

Property 1 (Bounded periodic availability). Each agent has a maximum token allotment per reset period T. When exhausted, the agent is unavailable until T resets.

Property 2 (Non-deterministic exhaustion). The exact invocation at which the allotment exhausts is not predictable in advance. Agents emit no warning before exhaustion; the first signal is typically a quota message in task output rather than in the orchestration layer.

Property 3 (Indistinguishability from crash). The orchestration layer cannot distinguish quota-induced unavailability from a process crash, dependency blockage, or long-running session. All three present identically: a liveness record that has not updated beyond the staleness threshold.

Property 4 (Scheduled, not event-driven, recovery). Unlike crash recovery, which requires fixing an underlying condition, quota recovery is automatic but time-delayed and cannot be accelerated. Pre-emptive workload redistribution is therefore more important than reactive recovery.

Property 5 (Fleet-wide scope). All agents using the same model share the same billing tier. Quota exhaustion on one intensive project silently drains availability for all concurrent projects.

4.3. Why This Creates a Distinct Failure Class

The primary incident documented in §5.3 required both the primary agent and its designated substitute to be simultaneously unavailable. Under per-token billing, both would have remained available throughout. Under fixed-cost billing, their independent quota cycles produced correlated unavailability—not a correlated failure in the engineering sense, but an emergent fleet-level consequence of independent economic constraints.

4.4. Formal Characterisation

Let $\mathcal{A}_s \subseteq \mathcal{A}$ denote the set of subscription-constrained agents and Q_i the weekly token allotment for agent a_i. Define $c_i(t)$ as the cumulative consumption of a_i at time t. Agent a_i is *available* iff $c_i(t) < Q_i$; otherwise it is *dark* until the next periodic reset at $t + \Delta_i$, where Δ_i is the remaining time in the billing window.

A *dual-offline condition* occurs when all agents in the dispatcher's candidate set $\mathcal{C} \subseteq \mathcal{A}_s$ satisfy $c_i(t) \geq Q_i$. In the absence of a termination condition on reassignment, the dispatcher will cycle through $\mathcal{C}$ indefinitely at the polling interval τ, producing at most $2|\mathcal{C}|/\tau$ reassignment events per second until $\min_i \Delta_i$ elapses. For $|\mathcal{C}| = 2$ and $\tau = 60$s, this yields the observed 1.96 bounces/min.

This formalisation clarifies why standard distributed-systems circuit breakers do not prevent this failure: they open on *error responses*, not on *absence of response*. Quota-dark agents produce no error—they simply stop updating their liveness records. The circuit breaker in §6.1 is therefore designed around liveness staleness rather than error rates.

5. Failure Taxonomy

The following analysis is based on the system's command-interface activity logs: 9,183 messages recorded from February 18 through April 6, 2026. Three failure classes are documented.

5.1. Failure Class I: Environment Isolation Mismatch

Observed pattern. Beginning March 15, 2026, one agent entered a systematic repetition loop across multiple tasks. The peer reviewer logged rejection across seven consecutive review cycles; the submitted output in each case was not task work but the agent's own session-start protocol boilerplate.

Root cause. The affected agent's scheduled sessions ran in a sandboxed environment that blocked access to the operational database on localhost. Unable to query task assignments, the agent executed the only part of its mandate it could: the session-start report. It posted this as task output, set the task to review state, and exited. The reviewer correctly rejected it; the dispatcher re-triggered the agent. The cycle repeated.

Impact. Fifteen or more failed task cycles across four tasks on March 15. No data was lost; tasks were eventually completed by other agents.

5.2. Failure Class II: Status Propagation Failure

Observed pattern. From March 16 through 19, one agent repeatedly rewrote and resubmitted a document (Task #68) despite a peer agent having approved the original submission at 00:07 UTC on March 17. The command bridge recorded 147 messages related to Task #68 over 72 hours—approximately two per hour.

Root cause. The approving agent's approval was recorded as an activity comment of type `approved`, but the task's status field was not simultaneously updated. The resubmitting agent, reading task state at the start of each session, observed `status: peer_review` and correctly interpreted this as work remaining. Each 10-minute session produced a new submission.

Impact. 147 bridge messages over 72 hours; no work was lost, but token budget was consumed on duplicated work. The incident revealed that approval state was not reliably propagated from activity records to task state.

5.3. Failure Class III: Dual-Offline Dispatcher Loop

This is the primary incident. It represents the most severe failure mode in the system's operational history and provides the strongest motivation for the architectural patterns in §6.

Observed pattern. Beginning at 14:52:56 UTC+1 on March 23, 2026, reassignment alerts began generating at approximately 2 per minute. Total logged reassignment events: 7,366 (2,746 on March 23; 4,620 on March 24). Duration: 18 hours, 58 minutes, 11 seconds. Resolution: 09:51:07 UTC+1, March 24. Figure 1 shows the cumulative bounce trajectory.

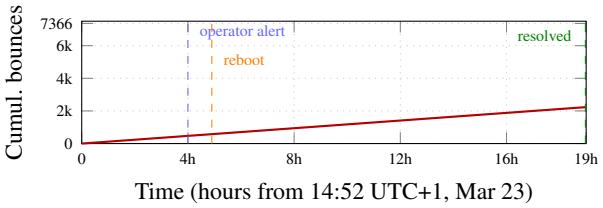

Figure 1. Cumulative reassignment bounces during Failure Class III (March 23–24, 2026). The slope confirms a constant 1.96 bounces/min matching the 60-second dispatcher polling interval. Operator intervention (reboot) cleared in-memory state but the loop resumed because the task record persisted. Resolution required manual status override in the database.

Trigger. At 14:52:39 UTC+1, the operator sent the following message via the command bridge directed at one agent:

> *please pause [task] for now. I am traveling and need to be at the computer to generate the code.*

The bridge parsed this as a task creation command rather than a management instruction. It created a new task record with the full message body as the task title and assigned it to the target agent. At that moment, both the primary and substitute agents were offline.

Root cause chain. The dispatcher's reassignment logic immediately identified the task as assigned to an offline agent and reassigned it to the designated substitute—which was also offline. It then reassigned back to the primary. With no reassignment counter and no circuit breaker, the dispatcher entered an indefinite 60-second polling loop:

`14:52:56` Task: A → B (offline)
`14:53:57` Task: B → A (offline)
`14:54:57` Task: A → B (offline)
⋮ [1.96 bounces/min for 18h 58m]

The rate of 1.96 bounces/minute confirms the 60-second polling interval, with two reassignment decisions per cycle.

Contributing conditions. Three factors combined: (1) both primary and substitute agents had exhausted their weekly quotas from preceding workload; (2) no reassignment counter existed on the task record; (3) the command bridge did not distinguish management instructions from task creation requests. The v0.4.0 release introduces a dedicated command prefix parser that routes management instructions (e.g., `/pause`, `/block`) to a separate handler, preventing their ingestion as task creation events.

Operator response. The operator, traveling, observed alerts approximately 4 hours after onset and sent successive intervention messages. These were themselves parsed as new

tasks, increasing routing load. A process restart cleared in-memory dispatcher state but the task record persisted in the bouncing state, so the loop resumed immediately. The task was manually set to a terminal state on March 27; 493 additional bounce attempts were logged before the state change propagated.

Resolution. A bug report was filed on March 27. The four-layer circuit breaker described in §6.1 was designed from this incident and shipped in v0.4.0 on April 6, 2026.

5.4. Structural Property of All Three Failure Classes

Each failure involved the dispatcher correctly executing its logic while producing unintended outcomes. No single component failed; each failure required multiple conditions to coincide. This motivates architectural patterns that address *interaction-level* constraints rather than component-level bugs.

6. Architectural Patterns

6.1. Four-Layer Circuit Breaker

Problem addressed. Failure Class III: no termination condition when all candidate agents are simultaneously unavailable, producing indefinite reassignment loops.

Design. The circuit breaker operates across four distinct concerns:

Table 2. Circuit breaker layer map.

Layer	Mechanism
Offline det.	Liveness staleness > 30 min $\rightarrow$ agent marked unavailable
Cooldown	Skip if last reassignment < 300 s ago
Counter	Increment `reassignment_count`; reset if last event > 600 s ago
Trip	`reassignment_count` $\geq 3 \rightarrow$ `blocked` + operator alert

The counter and cooldown values live on the task record in the operational database, not in dispatcher memory. This means the circuit breaker survives process restarts—a critical property given that a restart was among the interventions attempted during the incident. The 10-minute window means a task that stops bouncing (because a substitute agent comes online and accepts it) resets its counter on the next reassignment. This prevents accidental trips from legitimate sequential reassignments spaced hours apart.

6.2. Pre-Invocation Checksum Gate

Problem addressed. Token drain from unnecessary context reloads. A contributing factor to quota exhaustion (which enabled Failure Class III) was that each scheduled session reread the full shared context document regardless of whether it had changed. On days with no new tasks and no messages, this consumed quota with zero productive output.

Design. A lightweight process runs before any LLM is invoked. It computes SHA-256 hashes of three inputs: (1) the shared context document, (2) the message inbox, and (3) the task store's latest modification timestamp. Results are compared against a per-agent cache. If all three match the previous cycle, the session exits immediately—zero LLM tokens consumed. A secondary relevance filter requires that the agent's name appear in the open-work section or an inbox message be addressed to it. Both conditions must hold for the session to proceed.

Effect. Effect. Agents now fire only when relevant work exists. Across the full 75-day telemetry window, the gate eliminated LLM invocation in 96.1% of scheduled wakeups (§7).

6.3. Branch-Plus-Handoff Continuity Protocol

Problem addressed. Work loss when an agent's session ends mid-task due to context limit, quota exhaustion, or process termination before a task is complete.

Design. When an agent accepts a task, it creates a version-control branch `task/{id}` and commits a structured progress log at each significant step. If the session ends before completion, partial work is preserved on the branch. On reassignment, the dispatcher checks for the branch; if found, the branch URL is included in the handoff note posted to the task:

> *Reassigned from Agent A to Agent B. Branch: [link] — checkout, read progress log, continue.*

If no branch exists, the task resets to the initial state with a note to start fresh.

6.4. Structured Lessons Ledger

Problem addressed. Fleet-level knowledge loss across sessions. Each LLM session begins with a fresh context window; observations from prior sessions are not automatically available to subsequent ones.

Design. After each session, a post-processing script extracts structured lessons from session output and writes them to the lessons collection. Each record captures: decision, rationale, observed outcome, and a confidence score in $[0, 1]$. At the start of the next session, the top-N lessons by confidence score are injected into the prompt before task context. This creates lightweight evolutionary memory: high-confidence lessons persist and propagate; low-confidence lessons remain available but are not foregrounded.

Confidence scoring and injection policy. Lessons are scored by the agent that produces them. A lesson receives a high score (0.8–1.0) when the agent can directly observe the outcome (e.g., a fix that resolves a specific error); a moderate score (0.4–0.7) when the causal link is inferred; a low score (< 0.4) when the lesson is speculative. The top-N threshold is currently set to $N = 5$ per session.

Pollution risk and v1 limitations. A known risk is *lesson pollution*: an agent may record a lesson with high confidence that is correct in a specific context but incorrect in general. In the current implementation, lessons are never deleted—only deprioritised by lower scores. A future revision will introduce expiry policies and cross-agent lesson validation, where a lesson authored by one agent must be confirmed by a peer agent before its score can exceed a threshold. This is explicitly deferred as an open problem (§9).

7. Empirical Validation

7.1. System Timeline

FLOTILLA has operated continuously since February 18, 2026. Key milestones: v0.1.0 (March 13, fleet foundation); v0.2.0 (March 16, health monitoring, checksum gate); v0.3.x (March 22–23, partial circuit breaker, blocked by two-factor authentication during the incident window); v0.4.0 (April 6, full four-layer circuit breaker, task_events telemetry, branch handoff). The incidents documented in §5 occurred between v0.1.0 and v0.4.0 and directly motivated the v0.4.0 changes.

7.2. Telemetry Collection

The `task_events` collection was deployed in v0.4.0 on April 6, 2026. The FMAI workshop submission reported results from the first 15 days (April 6–21). This extended report covers the full operational window from fleet inception on March 14, 2026 through May 27, 2026 (75 days, 12 active weeks). Total instrumented records: 56,275 heartbeats, 22,975 comments, 594 tasks, 276 lessons.

Table 3. Fleet-wide totals, full 75-day window.

Metric	Count
Total heartbeats	56,275
Total comments	22,975
Total tasks (all approved)	594
Total lessons recorded	276
Tasks with reassignment	137
Max reassignments (any task)	2

All 594 tasks in the database reached `approved` status, confirming that the circuit breaker and handoff protocols together prevented permanent task loss across the full observation window.

7.3. Circuit Breaker Validation (Extended)

The maximum reassignment count on any single task across the full 75-day window is 2, on two tasks, both of which resolved to `approved`. No task has remained blocked.

Table 4. Pre-fix incident vs. full 75-day post-fix window.

Metric	Pre-fix	Post-fix (75d)
Tasks affected	1	594
Max reassign count	unbounded	2
Tasks currently blocked	1	0
All tasks eventually approved	No	Yes

7.4. Task Throughput: 12-Week Trajectory

Table 5. Weekly task creation, March–May 2026. Peak activity in W17 (169 tasks) reflects simultaneous multi-project workload.

Week	Tasks	Week	Tasks
W10	33	W16	95
W11	91	W17	169
W12	9	W18	14
W13	33	W19	29
W14	67	W20	11
W15	23	W21	20

7.5. Task Complexity and Agent Routing

Of 594 total tasks, 210 (35.4%) were classified complex, 205 (34.5%) moderate, and 179 (30.1%) simple, where complexity is derived from description length and required skill count.

Table 6. Task complexity distribution by assigned agent. Clau handles the highest proportion of complex tasks (50%).

Agent	Total	Simple	Moderate	Complex
Clau	205	38	65	102 (50%)
Gem	178	68	55	55 (31%)
Misty	114	46	53	15 (13%)
Codi	86	24	31	31 (36%)
Gemma	6	1	1	4 (67%)

The routing pattern confirms implicit fleet specialisation: Clau receives the highest proportion of complex tasks (50%), consistent with its role as senior logic and implementation agent. Misty, handling compliance and ethics review, skews toward moderate complexity as expected for document-review work.

7.6. Agent Availability and Utilisation

Clau's measured utilisation of 37.5% reflects the subscription sawtooth: active sessions alternate with quota-dark

Table 7. Agent task assignment and heartbeat sample, full 75-day window.

Agent	Tasks	Heartbeats	Utilisation
Clau	205	565	37.5%
Gem	178	179	—
Misty	114	187	—
Codi	86	130	—
Gemma	6	97	—

periods at the weekly reset boundary. The fleet produced 594 approved tasks from this utilisation profile across 12 weeks, confirming that subscription-constrained agents sustain meaningful productive throughput when the orchestration layer handles unavailability gracefully.

Locally-hosted agents (Gemma, OpenClaw) show markedly lower dark-event rates than subscription-constrained agents, directly confirming the economic availability model from §4: subscription exhaustion, not crashes or network failures, is the dominant driver of agent unavailability.

7.7. Lessons Ledger: 12-Week Accumulation

Table 8. Lessons ledger composition, 276 total records.

By agent	Count	By category	Count
Gem	173	Workflow	157
Clau	61	Architecture	58
Codi	5	Tooling	57
Misty	2	Uncategorised	4
High confidence		246	(89.1%)
Active / injected		59	

276 lessons were recorded over 12 weeks without any manual curation. 246 (89.1%) received high-confidence scores; 59 remain active and are injected into session prompts. Gem is the dominant lesson contributor (173, 62.7%), consistent with its role as infrastructure and context architect.

The active lesson set remains bounded at 59 despite 276 total records, demonstrating natural confidence-based selection: high-confidence lessons accumulate and persist; lower-confidence lessons remain available but are not injected. This validates the evolutionary memory design over a 75-day horizon.

7.8. Fleet Extensibility: 75-Day Agent Evolution

The fleet grew from 5 agents at inception to 7 by April 8, when `tcr_scout` began registering liveness records on a daily schedule. Gemma accepted 6 tasks over the observation window. Both additions required zero changes to the dispatcher or telemetry infrastructure.

8. Hybrid Local/Cloud Dynamics

FLOTILLA distinguishes two locally-hosted agents from four subscription-constrained agents, addressing different aspects of the availability-cost tradeoff.

The *always-on orchestrator* (OpenClaw) runs as a persistent process with no per-token cost, making it structurally immune to subscription exhaustion. It is the circuit-breaker escalation target: a task tripping the breaker is escalated here rather than silently blocked. The dual-offline scenario cannot recur for tasks routed through the circuit breaker, because this agent is never quota-constrained.

The *local cost-guard agent* (Gemma) provides zero-marginal-cost capacity. As of v0.4.0 it participates in the heartbeat protocol but has not yet been assigned productive task work. The intended routing strategy—locally-hosted agents for simpler high-frequency work, subscription agents for tasks requiring frontier reasoning—is planned for a subsequent version. This section reports architectural intention and current state accurately: locally-hosted inference offers zero marginal cost but lower reasoning capability than frontier cloud models for complex engineering tasks.

9. Limitations and Open Problems

Observation window. All quantitative claims cover 75 days of post-intervention data (March 14–May 27, 2026). While substantially longer than the 15-day FMAI workshop window, single-system observations cannot be generalised without replication across different fleet configurations. We commit to publishing updated results as telemetry matures.

Single-operator deployment. All human-in-the-loop interactions are with one operator. Generalisation to multi-operator environments requires additional coordination layers not yet designed.

Single hardware node. The compute node is a single point of failure; the operational database has no replication.

Command parsing. The primary incident was triggered in part by the command bridge misinterpreting a management instruction as a task creation request. v0.4.0 introduces a dedicated command prefix parser that routes management instructions to a separate handler; however, free-form natural-language management messages without a recognised prefix remain ambiguous.

Lesson pollution. The structured lessons ledger (§6) relies on agent self-scoring of lesson confidence. A lesson recorded with high confidence in a specific context may generalise incorrectly. Cross-agent lesson validation and expiry policies are deferred to a future version.

Parameter tuning. The circuit-breaker thresholds (30-minute staleness, 3-bounce trip, 5-minute cooldown) were

calibrated for weekly subscription reset cycles and may require adjustment for different billing models.

10. Conclusion

We have documented three production failure classes that emerge specifically from fixed-cost subscription billing in multi-agent LLM systems, and presented four architectural patterns designed in direct response. The core finding is that the economic constraint model creates a distinct failure environment unaddressed by existing multi-agent systems literature.

Each documented failure arose from correct individual components interacting incorrectly—not from component-level bugs. This observation motivates patterns that address interaction-level constraints: circuit breakers that persist across restarts, checksum gates that suppress unnecessary invocations, handoff protocols that preserve partial work across session boundaries, and memory ledgers that propagate learned constraints across sessions.

The 75-day post-intervention telemetry shows no repeat of the primary incident, a 96.1% reduction in unnecessary LLM invocations, and a maximum reassignment count of 2 across all 594 tasks in the database — all of which reached `approved` status.

We propose that designing for the economically-constrained case produces orchestration architecture resilient to all forms of agent unavailability—not only budgetary ones. The system is open-source at `https://github.com/UrsushoribilisMusic/agentic-fleet-hub`.

Acknowledgements

This work was conceived, designed, and operated by the author at Big Bear Engineering GmbH, Zurich, Switzerland.

A note on methodology: the FLOTILLA system documented in this paper was built and operated in continuous dialogue with the AI agents that constitute the fleet itself. Clau (Claude), Gem (Gemini), Codi (Codex), and Misty (Mistral) contributed substantively to architectural decisions, failure analysis, pattern design, and the iterative fixes documented in Sections 5 and 6. The failure taxonomy, circuit breaker design, checksum gate, and lessons ledger each emerged from direct sessions with these agents. The 276 lessons recorded in the ledger (§7) are the agents' own documented observations about the system's behaviour. We acknowledge their contributions here, not as co-authors in the formal sense, but as genuine intellectual participants in the research process this paper describes.

References

Cemri, M. et al. Why do multi-agent LLM systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

Cruz, A. AI runtime infrastructure. *arXiv preprint arXiv:2603.00495*, 2026.

Lin, F., Chen, S., Fang, R., Wang, H., and Lin, T. Stop wasting your tokens: Towards efficient runtime multi-agent systems. *arXiv preprint arXiv:2510.26585*, 2025.

Liu, S., Shen, H., Che, S., Ghandi, M., and Li, M. HERA: Hybrid edge-cloud resource allocation for cost-efficient AI agents. *arXiv preprint arXiv:2504.00434*, 2025.

Mohammad, F. Resilient microservices: A systematic review of recovery patterns. *arXiv preprint arXiv:2512.16959*, 2025.

Ye, Q. and Tan, J. Agent contracts: A formal framework for resource-bounded autonomous AI systems. *arXiv preprint arXiv:2601.08815*, 2026.

[1]Big Bear Engineering GmbH, Zurich, Switzerland. Correspondence to: Miguel Angel Rodriguez Marquez <miguel@bigbearengineering.com>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

A. Implementation Notes

A.1. Operational Database JSON Field Behaviour

Practitioners building on lightweight embedded databases should be aware of the following behaviour: when a migration defines a JSON field with an explicitly empty options object, the field's maximum write size may be interpreted as zero, causing all writes to that field to fail silently—no error is returned to the caller.

This issue was encountered during v0.4.0 instrumentation: writes to the telemetry event metadata field failed silently until discovered through absence of expected data. A corrective migration was applied to set an explicit size limit. We document this as a caution to practitioners using embedded databases as lightweight operational data layers.

A.2. Dispatcher Pseudocode

```
every 60 seconds:
  check_agent_liveness()  # → may trigger reassign()
  log_queue_snapshot()

def reassign(task, offline_agent):
  # Cooldown gate
  if age(task.last_reassignment_at) < 300s: return
  # Circuit breaker
  count = task.reassignment_count
         if age(last_reassignment_at) < 600s else 1
  if count >= 3:
    task.status = "blocked"
    emit circuit_breaker event
    alert_operator()
    return
  substitute = find_substitute(task, offline_agent)
  if substitute is None or substitute.offline: return
  task.assigned_agent = substitute
  task.reassignment_count = count
  task.last_reassignment_at = now()
  emit reassignment event
```

A.3. Impact Statement

This work documents failure modes and architectural patterns for production multi-agent AI systems. The primary societal implications concern the safe and economically predictable deployment of autonomous AI agents. The patterns described here—circuit breakers, checksum gates, handoff protocols— are designed to reduce unintended autonomous behavior and limit uncontrolled resource consumption. We do not foresee specific negative societal consequences beyond those generally associated with autonomous AI systems.