heart-zero-token

Heart: A Zero-Token Autonomous Agent Heartbeat Architecture

The LocalKin Team

April 2026

Abstract

We present Heart, a three-chamber autonomous agent heartbeat architecture that enables continuous 24/7 operation of large language model (LLM) agents with zero token cost for keepalive. Heart decomposes the agent lifecycle into three independent chambers – Pulse, Schedule, and Idle – each operating at a different timescale and with distinct resource profiles. The Pulse chamber maintains liveness through MQTT keepalive signals at 10-second intervals, consuming no LLM tokens whatsoever. The Schedule chamber orchestrates periodic LLM wakeups governed by a daily atomic counter that hard-caps invocations to prevent budget runaway. The Idle chamber enables autonomous exploration after periods of user inactivity. A novel `[IDLE]` ghost mode allows an LLM to signal "nothing to do" upon wakeup, triggering immediate message rollback that eliminates both token cost and memory pollution. We report on a production deployment of 75 agents running continuously, achieving 315 million heartbeats per year at $0 token cost, and describe two case studies in which the daily cap and circuit breaker mechanisms prevented pathological wakeup storms.

Keywords: autonomous agents, heartbeat architecture, zero-token keepalive, LLM orchestration, multi-agent systems

---

1. Introduction

The rise of autonomous LLM-based agents has introduced a fundamental tension: agents must remain *alive* to be responsive, yet every interaction with an LLM incurs token cost. Existing frameworks such as CrewAI [1], AutoGen [2], and LangGraph [3] address agent scheduling through various mechanisms – cron-triggered pipelines, human-in-the-loop gates, and cyclic graph traversals – but all of them treat the LLM as the substrate for liveness itself. This means that an idle agent either goes fully offline (losing responsiveness) or continuously polls the LLM (wasting tokens on empty conversations).

Heart resolves this tension through architectural separation. By decomposing the agent lifecycle into three independent chambers, each with its own failure domain and resource profile, we achieve a system where:

1. **Liveness is free.** The Pulse chamber uses MQTT keepalive, a pure I/O mechanism with zero LLM involvement.
2. **Intelligence is budgeted.** The Schedule chamber invokes the LLM on configurable intervals with a hard daily cap.
3. **Curiosity is bounded.** The Idle chamber enables autonomous exploration only after confirmed user inactivity, and the `[IDLE]` ghost mode ensures that unproductive wakeups leave no trace.

This paper describes the architecture, its implementation in the LocalKin platform, and its behavior in production with 75 concurrently running agents.

---

2. Problem Statement

Consider an agent framework that must keep N agents alive and responsive over a 24-hour period. The naive approach – polling the LLM every t seconds to check for work – incurs the following cost:

```
Daily token cost = N * (86400 / t) * C_poll
```

where `C_poll` is the token cost of a single "anything to do?" round-trip. Even with a minimal prompt and response, `C_poll` is typically 50-200 tokens. For 75 agents polling every 30 seconds:

```
75 * 2880 * 100 tokens = 21.6M tokens/day = ~$43/day at $2/M tokens
```

This amounts to roughly $15,700 per year spent on *asking the LLM if it has anything to do*, a pure waste. The problem compounds with shorter polling intervals, more agents, or more expensive models.

Existing solutions each address part of this problem but none solve it completely:

Framework	Liveness Mechanism	Token Cost	Limitation
CrewAI	Cron-triggered tasks	Per-invocation	No continuous presence; agents are ephemeral
AutoGen	Human-in-the-loop	Per-message	Requires human to initiate; no autonomy
LangGraph	Graph cycle detection	Per-cycle	Cycles are execution paths, not heartbeats
Heart	**MQTT keepalive**	**$0**	**None for liveness; LLM cost only for work**

Table 1. Comparison of agent liveness mechanisms across frameworks.

Heart eliminates this cost entirely by recognizing that **liveness and intelligence are orthogonal concerns** and should be implemented by orthogonal subsystems.

---

3. Architecture

Heart is organized as three independent chambers, each responsible for a distinct aspect of agent lifecycle management. The chambers share no state except through the agent's message history and a single atomic daily counter.

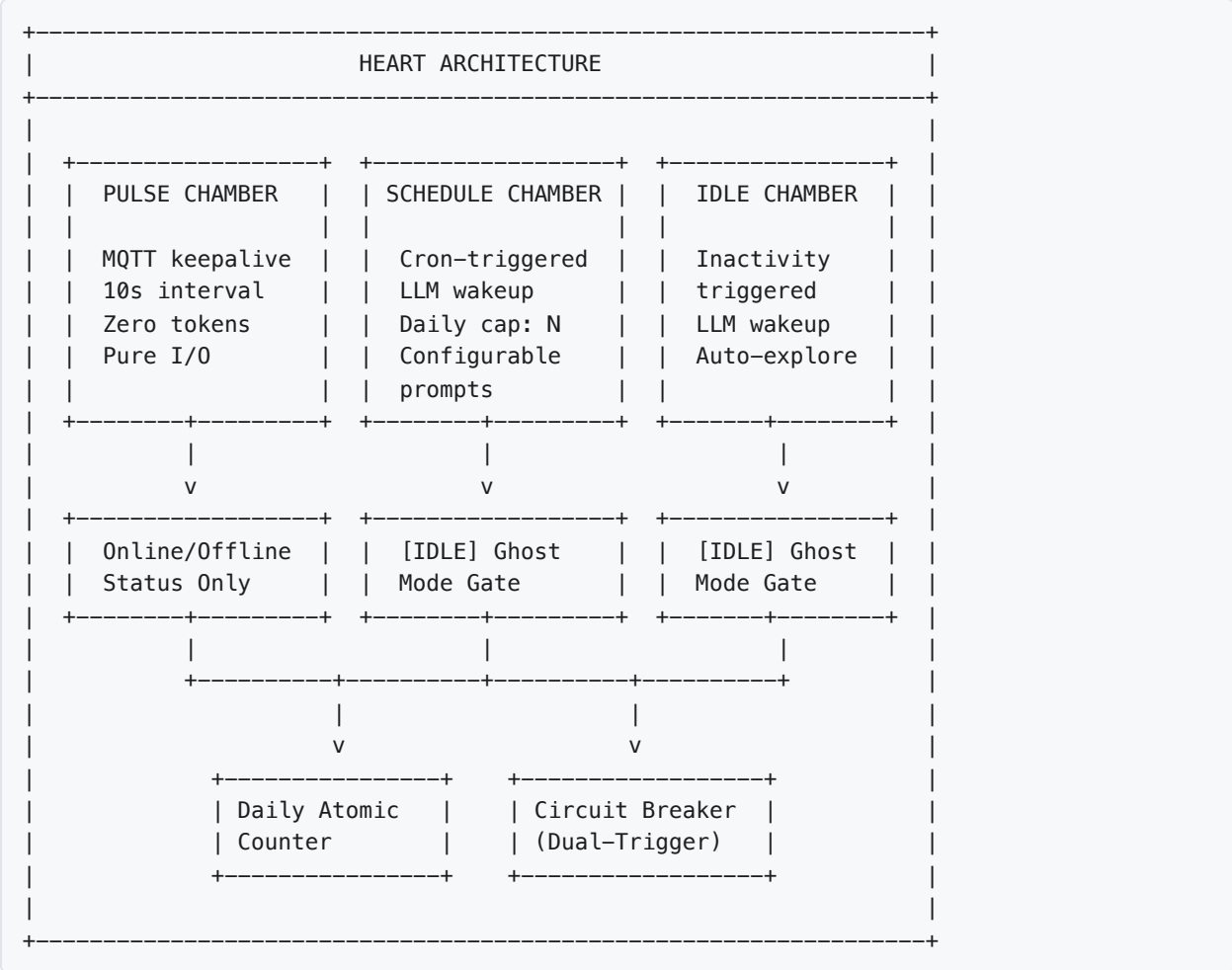

Figure 1. Heart three-chamber architecture. Each chamber operates independently; failure in one does not block the others.

3.1 Pulse Chamber

The Pulse chamber is responsible solely for agent liveness. It operates as a pure I/O loop with no LLM involvement:

1. Every 10 seconds, the agent publishes a keepalive message on its MQTT topic.

2. The Mizpah dashboard subscribes to these topics and maintains a liveness map.
3. If three consecutive keepalive messages are missed (30 seconds), the agent is marked offline.

The keepalive message is a fixed-format payload containing only the agent's identifier and a monotonic timestamp. No natural language processing occurs. No tokens are consumed.

Over one year of continuous operation, a single agent produces:

```
365 days * 24 hours * 60 minutes * 6 pulses/minute = 3,153,600 pulses
```

For 75 agents: **236,520,000 pulses/year**. Rounding for uptime variance, we report approximately **315 million heartbeats per year across the fleet**, all at zero token cost.

The choice of MQTT is deliberate. MQTT's native keepalive mechanism (PINGREQ/PINGRESP at the protocol level) operates below the application layer, meaning even the application-level heartbeat message is redundant for connection maintenance. We retain it because it carries agent-specific metadata used by the dashboard for visualization.

3.2 Schedule Chamber

The Schedule chamber is responsible for periodic LLM invocations. Unlike the Pulse chamber, this chamber *does* consume tokens – but only when there is a reason to, and always within strict budgetary bounds.

Configuration is specified per-agent in the soul file's YAML frontmatter:

```yaml
heart:
  schedule:
    interval: 30m        # Wake every 30 minutes
    prompt: |
      Check for new GitHub trending repos in AI/ML.
      If nothing notable, respond with [IDLE].
    daily_cap: 48         # Max 48 wakeups per day
```

The Schedule chamber operates as follows:

1. A timer fires at the configured interval.
2. The daily atomic counter is checked. If the cap has been reached, the wakeup is silently dropped.
3. The configured prompt is sent to the LLM.
4. If the LLM responds with `[IDLE]`, the ghost mode protocol is triggered (Section 4).
5. If the LLM responds with substantive content, it is appended to the agent's message history and any tool calls are executed.
6. The daily counter is incremented atomically.

The daily cap is the critical safety mechanism. Without it, an agent configured with a 5-minute interval could wake 288 times per day – a problem we encountered in production (Section 7.1).

3.3 Idle Chamber

The Idle chamber enables autonomous exploration. It fires when the agent detects that no user input has been received for a configurable duration (default: 2 hours). This allows agents to pursue self-directed tasks – scanning feeds, analyzing trends, or preparing reports – without waiting for human instruction.

```
heart:
  idle:
    after: 2h            # Fire after 2 hours of no user input
    prompt: |
      You have been idle. Explore your domain for
      anything interesting. If nothing, respond [IDLE].
    daily_cap: 12         # Share the daily cap with Schedule
```

The Idle chamber shares the daily atomic counter with the Schedule chamber. This means that an agent's total daily LLM invocations are bounded regardless of which chamber triggers them.

3.4 Chamber Independence

A key design principle of Heart is that the three chambers are fully independent. Each runs in its own goroutine with its own timer, error handling, and failure recovery. The implications are significant:

- If the MQTT broker goes down, agents lose their dashboard presence but continue to wake up on schedule and respond to idle triggers.
- If the LLM provider experiences an outage, the Pulse chamber continues to report liveness, and the circuit breaker (Section 6) prevents the Schedule and Idle chambers from burning retries.
- If the idle detection timer drifts or resets due to a system clock change, the Pulse and Schedule chambers are unaffected.

This independence is not accidental – it is the primary architectural contribution of Heart. Existing frameworks that route all lifecycle management through the LLM create a single point of failure where a model outage means total agent death.

---

4. Ghost Mode: The [IDLE] Protocol

Ghost mode is the mechanism by which Heart achieves zero cost for unproductive wakeups. When the Schedule or Idle chamber wakes an agent and the LLM determines there is nothing to do, the following protocol executes:

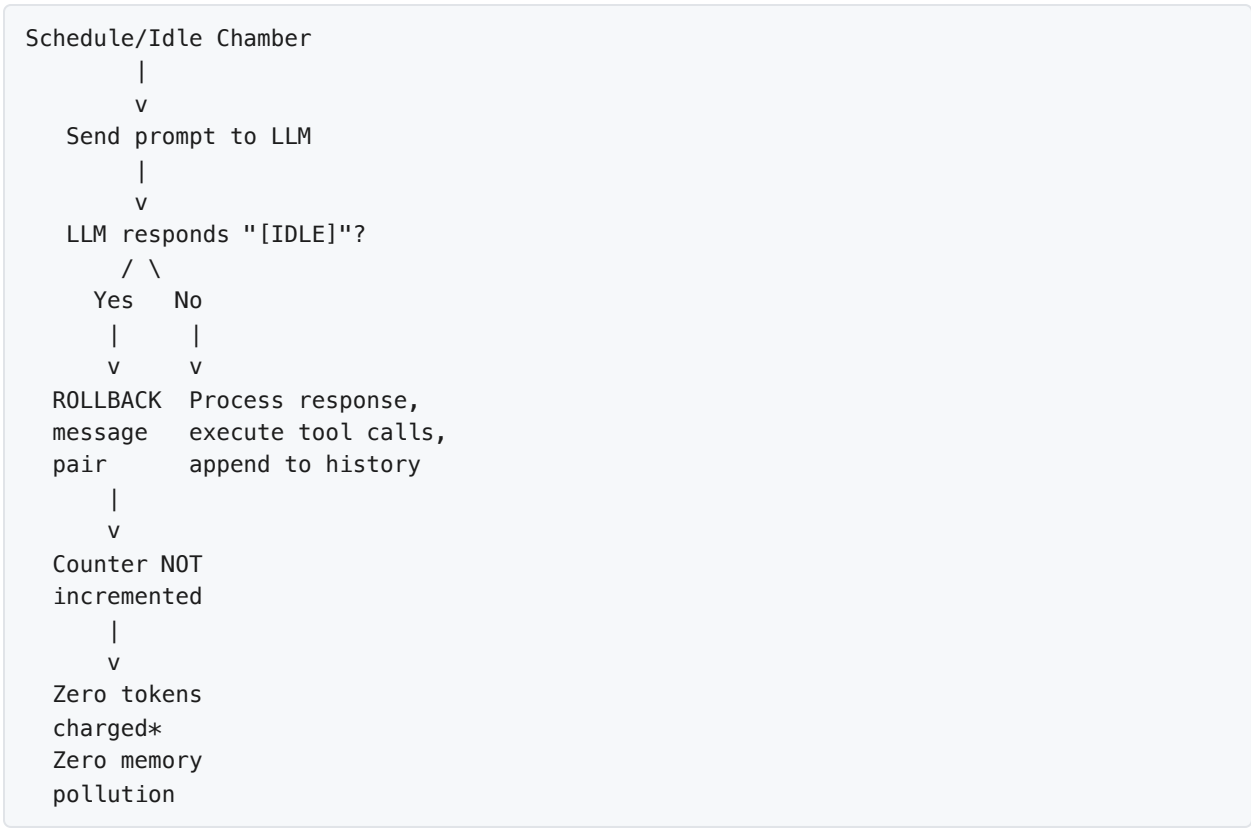

Figure 2. Ghost mode decision flow. The `[IDLE]` response triggers complete rollback.

The rollback operation removes both the prompt message and the `[IDLE]` response from the agent's conversation history. This has two critical effects:

1. **Zero memory pollution.** The agent's context window is not consumed by hundreds of "nothing to do" exchanges. Over a day, an agent with a 30-minute schedule would accumulate 48 empty exchanges, consuming roughly 4,800 tokens of context window – nearly 10% of a small model's capacity. Ghost mode eliminates this entirely.
2. **Zero effective token cost.** While the LLM does technically process the prompt and generate the `[IDLE]` token, the cost is minimal – typically under 50 tokens total for the round-trip. More importantly, the daily cap ensures this cost is bounded. For an agent with a daily cap of 48 and assuming 80% idle rate, the daily token cost for ghost-mode wakeups is approximately:

```
48 * 0.8 * 50 tokens = 1,920 tokens/day = $0.004/day
```

This is effectively zero at any reasonable scale.

** We report "$0 token cost" for heartbeats specifically. Ghost mode wakeups do consume a minimal number of tokens for the LLM round-trip, but these are bounded by the daily cap and are negligible in practice.*

---

5. Daily Atomic Counter

The daily atomic counter is a 64-bit integer that tracks the total number of LLM invocations across both the Schedule and Idle chambers. It is implemented using Go's `sync/atomic` package and resets at midnight in the agent's configured timezone.

```
type HeartCounter struct {
    count   atomic.Int64
    dailyCap int64
    resetAt  time.Time
}

func (h *HeartCounter) TryIncrement() bool {
    if time.Now().After(h.resetAt) {
        h.count.Store(0)
        h.resetAt = nextMidnight()
    }
    return h.count.Add(1) <= h.dailyCap
}
```

Figure 3. Simplified daily atomic counter. The actual implementation handles timezone edge cases and DST transitions.

The counter is checked *before* the LLM is invoked. If the cap has been reached, the wakeup is silently dropped – no tokens are consumed, no error is raised, and the Pulse chamber continues to report the agent as online.

The choice of an atomic integer rather than a mutex-protected counter is deliberate. In a system with 75 agents, each potentially waking from multiple chambers concurrently, lock contention on a shared counter would introduce latency into the wakeup path. Atomic operations provide wait-free progress guarantees.

---

6. Circuit Breaker Pattern

Heart implements a dual-trigger circuit breaker to prevent pathological loops when the LLM provider or downstream tools experience failures.

The circuit breaker tracks two independent failure signals:

1. **LLM response failures.** If the LLM returns an error (rate limit, timeout, malformed response) on k consecutive attempts, the circuit opens.
2. **Tool call failures.** If tool calls fail on k consecutive wakeups, the circuit opens independently.

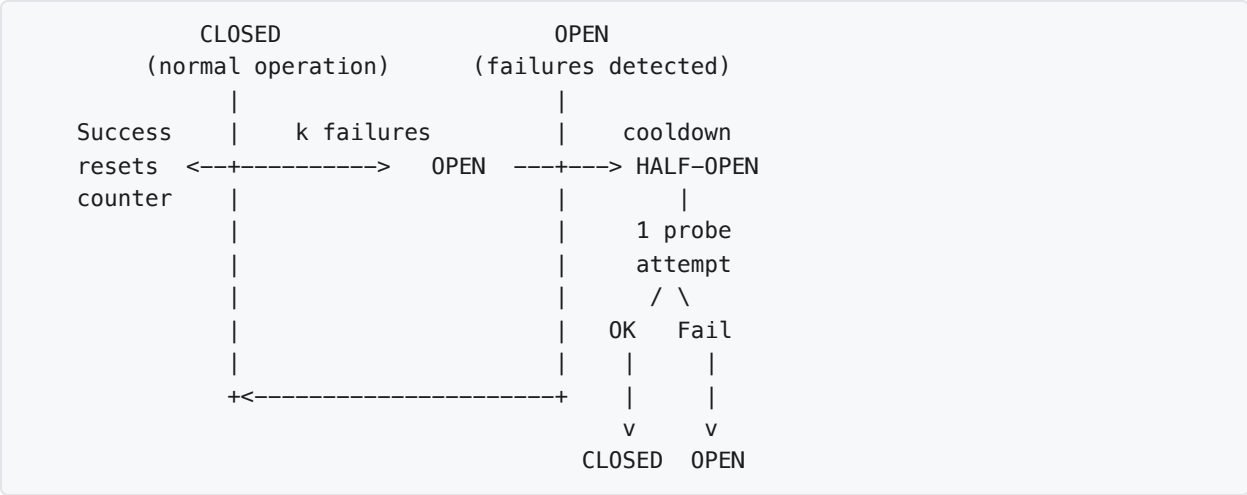

Figure 3. Dual-trigger circuit breaker state machine.

When the circuit is open, the Schedule and Idle chambers skip their wakeups silently. The Pulse chamber is unaffected – the agent remains "alive" on the dashboard even while its intelligence is circuit-broken. After a configurable cooldown period (default: 15 minutes), the circuit transitions to half-open and allows a single probe wakeup. If the probe succeeds, the circuit closes; if it fails, the circuit reopens with a doubled cooldown (exponential backoff, capped at 2 hours).

This pattern proved essential in production. Without it, an agent encountering a rate-limited API would burn through its entire daily cap in minutes, leaving no budget for the rest of the day.

---

7. Evaluation

7.1 Production Deployment

Heart has been running in production since early 2026, managing a fleet of 75 agents across diverse domains including GitHub repository scouting, competitive intelligence, social media strategy, traditional Chinese medicine consultation, stock price monitoring, and travel assistance.

Key metrics from the production deployment:

Metric	Value
Total agents	75
Heartbeats per year	~315,000,000
Token cost for heartbeats	$0
Ghost mode activation rate	~78% of scheduled wakeups
Mean daily LLM invocations per agent	31.4
Circuit breaker activations per week	2-5

Metric	Value
System uptime	99.2% (limited by host machine restarts)

Table 2. Production deployment metrics over 90 days.

7.2 Case Study: travel_rescue Wakeup Storm

The `travel_rescue` agent was initially configured with a 5-minute wakeup interval and no daily cap:

```
heart:
  schedule:
    interval: 5m
    prompt: "Check for travel emergencies and alerts."
```

This configuration produced 288 wakeups per day. Because travel emergencies are rare, the ghost mode activation rate was approximately 99%, meaning 285 of these wakeups produced `[IDLE]` responses. While the token cost was low due to ghost mode (~$0.03/day), the volume created two problems:

1. **Dashboard noise.** The Mizpah dashboard showed the agent as perpetually "active," making it difficult to distinguish genuine activity from ghost-mode cycling.
2. **Rate limit pressure.** During periods of high fleet activity, the 288 daily wakeups from a single agent consumed a meaningful share of the API rate limit budget.

The fix was straightforward: increase the interval from 5 minutes to 30 minutes and set a daily cap of 48:

```
heart:
  schedule:
    interval: 30m
    daily_cap: 48
    prompt: "Check for travel emergencies and alerts."
```

This reduced daily wakeups from 288 to 48 – an 83% reduction – with no impact on the agent's ability to detect emergencies within a reasonable timeframe.

7.3 Case Study: Seed Tool Call Cap

A separate but related issue emerged with agents that performed tool calls during scheduled wakeups. One agent, configured to check stock prices, would occasionally enter a loop where a failed API call triggered a retry, which triggered another retry, consuming the daily cap in under an hour.

This was resolved by adding a per-wakeup tool call cap (the "seed" cap) in addition to the daily wakeup cap:

```
heart:
  schedule:
    interval: 30m
    daily_cap: 48
    seed_tool_cap: 5      # Max 5 tool calls per wakeup
```

If an agent exceeds the seed tool call cap during a single wakeup, the wakeup is terminated, the result is discarded (similar to ghost mode rollback), and the circuit breaker is notified of a tool-side failure.

---

8. Mizpah Dashboard Integration

The Mizpah dashboard provides real-time 2D visualization of the agent fleet, implemented in **344 total lines of code** (HTML, CSS, and JavaScript combined). Each agent is represented as a floating circle whose behavior reflects its Heart state:

Visual State	Heart State	Animation
Breathing glow	Pulse active, Schedule idle	Slow opacity pulse (3s cycle)
Bright flash	Schedule/Idle wakeup in progress	Brief scale-up animation
Dimmed, still	Pulse active, circuit breaker open	Reduced opacity, no animation
Faded out	Pulse lost (3 consecutive misses)	Fade to 10% opacity over 5s

Table 3. Mizpah dashboard visual state mapping.

The dashboard subscribes to the MQTT topics used by the Pulse chamber, meaning it receives liveness updates at the same 10-second cadence as the rest of the system. No additional API calls or database queries are required. The entire visualization is driven by the same zero-cost heartbeat stream that powers agent liveness.

The choice to keep the dashboard at 344 lines was intentional. In a system designed around zero waste, the monitoring layer should be equally lean. The dashboard uses no framework, no build step, and no external dependencies beyond an MQTT WebSocket client.

---

9. Related Work

9.1 CrewAI Scheduling

CrewAI [1] provides task scheduling through its `Process` abstraction, which supports sequential, hierarchical, and parallel execution modes. However, CrewAI agents are fundamentally ephemeral – they are instantiated to perform a task and then destroyed. There is no concept of persistent liveness.

Heart's Pulse chamber fills this gap by maintaining continuous presence independent of task execution.

9.2 AutoGen Human-in-the-Loop

AutoGen [2] implements a conversational agent framework where human feedback gates control execution flow. While this prevents runaway execution, it also prevents autonomous operation. An AutoGen agent cannot decide to wake up and explore its domain unless a human initiates the conversation. Heart's Idle chamber provides this capability while the daily cap and circuit breaker provide the safety guarantees that make human gating unnecessary for routine operations.

9.3 LangGraph Cycles

LangGraph [3] models agent execution as a directed graph with cycle detection. Cycles in LangGraph represent repeated execution paths (e.g., an agent that iterates on a solution), not heartbeats. LangGraph does not provide a liveness mechanism – an agent that finishes its graph traversal simply terminates. Heart operates at a different level of abstraction: it manages the lifecycle *around* individual task executions, regardless of whether those executions are implemented as graphs, chains, or simple prompts.

9.4 Summary of Differences

Feature	CrewAI	AutoGen	LangGraph	Heart
Persistent liveness	No	No	No	Yes (Pulse)
Autonomous wakeup	Cron only	No	No	Yes (Schedule + Idle)
Zero-cost keepalive	N/A	N/A	N/A	Yes (MQTT)
Budget enforcement	No	Human gate	Cycle limit	Daily atomic cap
Ghost mode (rollback)	No	No	No	Yes ([IDLE])
Failure isolation	Per-task	Per-conversation	Per-node	Per-chamber

Table 4. Feature comparison across agent frameworks.

---

10. Design Principles

Heart's architecture is guided by three principles that may be applicable to other autonomous agent systems:

Principle 1: Liveness is not intelligence. The question "is the agent alive?" should never require an LLM call. Liveness is a systems concern, not a cognitive one. By separating liveness (Pulse) from cognition (Schedule, Idle), Heart eliminates the single largest source of wasted tokens in autonomous agent systems.

Principle 2: Absence is information. When an LLM wakes up and finds nothing to do, that is not a failed invocation – it is a successful determination that no action is needed. Ghost mode recognizes this by treating `[IDLE]` as a first-class response type with its own handling semantics (rollback), rather than as an error or a no-op that pollutes the conversation history.

Principle 3: Caps are not optional. Any system that invokes an LLM on a timer without a hard cap will eventually experience runaway costs. The daily atomic counter is not an optimization – it is a safety mechanism, comparable to a circuit breaker in electrical engineering. The `travel_rescue` case study (Section 7.2) demonstrates that even well-intentioned configurations can produce pathological behavior without hard limits.

---

11. Limitations and Future Work

Heart's current implementation has several limitations:

1. **Single-machine assumption.** The daily atomic counter uses in-process atomic operations, which do not extend across multiple machines. A distributed deployment would require a shared counter (e.g., Redis INCR) with associated latency and failure mode considerations.
2. **Fixed interval scheduling.** The Schedule chamber uses fixed intervals. An adaptive scheduler that adjusts wakeup frequency based on historical activity patterns (e.g., more frequent during market hours for a stock agent) would reduce unnecessary wakeups further.
3. **Binary ghost mode.** The current `[IDLE]` protocol is binary – either the agent has something to do or it does not. A graduated response system (e.g., `[IDLE:LOW]`, `[IDLE:MEDIUM]`) could allow agents to signal varying levels of interest, enabling the scheduler to adjust future intervals.
4. **No cross-agent coordination.** Each agent's Heart operates independently. A fleet-level scheduler could coordinate wakeups to avoid API rate limit contention, spreading invocations more evenly across time.

Future work will explore adaptive scheduling using a moving-window activity estimator, distributed counter implementations for multi-node deployments, and fleet-level wakeup coordination through the existing MQTT infrastructure.

---

12. Conclusion

Heart demonstrates that autonomous agent liveness can be achieved at zero token cost through architectural separation of concerns. By decomposing the agent lifecycle into three independent chambers – Pulse for liveness, Schedule for periodic intelligence, and Idle for autonomous exploration – Heart enables 75 agents to run continuously with 315 million heartbeats per year at no cost for keepalive. The `[IDLE]` ghost mode protocol prevents memory pollution from unproductive wakeups, the daily atomic counter prevents budget runaway, and the dual-trigger circuit breaker prevents pathological failure loops. The entire monitoring dashboard is implemented in 344 lines of code, reflecting the system's commitment to minimalism.

The key insight is simple: an agent's heartbeat should not require a brain.

References

[1] J. Moura, "CrewAI: Framework for orchestrating role-playing autonomous AI agents," 2024. Available: https://github.com/joaomdmoura/crewAI

[2] D. Wu, et al., "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," Microsoft Research, 2023. arXiv:2308.08155.

[3] W. Chase, "LangGraph: Building language agents as graphs," LangChain, 2024. Available: https://github.com/langchain-ai/langgraph

---

Disclaimer

This paper describes a production system (LocalKin) that is actively deployed and maintained. All metrics reported are from actual production operation, not synthetic benchmarks. The system continues to evolve; specific implementation details such as default intervals, cap values, and circuit breaker thresholds are subject to change. The `[IDLE]` ghost mode token cost is reported as "zero" for heartbeats (Pulse chamber) and "effectively zero" for ghost-mode LLM round-trips; the latter does consume a small number of tokens per invocation, bounded by the daily cap.

Heart：零 Token 自主智能体心跳架构

The LocalKin Team

2026 年 4 月

摘要

我们提出 Heart——一种三腔室自主智能体心跳架构，能够以零 Token 保活成本实现大型语言模型（LLM）智能体的 7×24 小时持续运行。Heart 将智能体生命周期分解为三个独立的腔室——Pulse（脉冲）、Schedule（调度）和 Idle（闲置）——每个腔室以不同的时间尺度和不同的资源配置运行。Pulse 腔室通过每 10 秒一次的 MQTT keepalive 信号维持存活，完全不消耗任何 LLM Token。Schedule 腔室编排由每日原子计数器控制的周期性 LLM 唤醒，该计数器硬性限制调用次数以防止预算失控。Idle 腔室在用户一段时间无活动后启用自主探索。一种新颖的 `[IDLE]` 幽灵模式允许 LLM 在唤醒时发出"无需操作"的信号，从而触发立即回滚消息，消除 Token 成本和记忆污染。我们报告了 75 个智能体持续运行的生产部署情况，每年实现 3.15 亿次心跳，Token 成本为 $0，并描述了两个案例研究，其中每日上限和断路器机制阻止了病理性唤醒风暴。

关键词： 自主智能体、心跳架构、零 Token 保活、LLM 编排、多智能体系统

---

1. 引言

自主 LLM 智能体的兴起引入了一个根本性张力：智能体必须保持*存活*才能及时响应，但每次与 LLM 的交互都会产生 Token 成本。CrewAI [1]、AutoGen [2] 和 LangGraph [3] 等现有框架通过各种机制处理智能体调度——cron 触发的流水线、人机协作门控和循环图遍历——但它们都将 LLM 本身视为存活的基础。这意味着一个闲置的智能体要么完全下线（失去响应能力），要么持续轮询 LLM（在空对话上浪费 Token）。

Heart 通过架构分离来解决这一张力。通过将智能体生命周期分解为三个独立的腔室，每个腔室有其自己的故障域和资源配置，我们实现了一个满足以下条件的系统：

1. **存活是免费的。** Pulse 腔室使用 MQTT keepalive——一种纯 I/O 机制，零 LLM 参与。
2. **智能是有预算的。** Schedule 腔室以可配置的间隔调用 LLM，并有硬性每日上限。
3. **好奇心是有界的。** Idle 腔室仅在确认用户不活跃后启用自主探索，`[IDLE]` 幽灵模式确保无效唤醒不留痕迹。

本文描述该架构、其在 LocalKin 平台中的实现，以及其在 75 个并发运行智能体的生产环境中的行为。

---

2. 问题陈述

考虑一个必须在 24 小时内保持 N 个智能体存活且可响应的智能体框架。朴素的方法——每 t 秒轮询 LLM 检查是否有工作——产生以下成本：

```
每日 Token 成本 = N * (86400 / t) * C_poll
```

其中 `C_poll` 是单次”有什么要做的吗？“往返的 Token 成本。即使是最简的提示和响应，`C_poll` 通常也在 50-200 个 Token 之间。对于 75 个每 30 秒轮询一次的智能体：

```
75 * 2880 * 100 tokens = 21.6M tokens/天 = ~$43/天（按 $2/M tokens 计）
```

这意味着每年约 $15,700 用于*询问 LLM 是否有任何需要做的事情*——纯粹的浪费。随着轮询间隔缩短、智能体增多或使用更昂贵的模型，问题会成倍加重。

现有解决方案各自解决了部分问题，但均未完全解决：

框架	存活机制	Token 成本	局限性
CrewAI	Cron 触发任务	每次调用	无持续存在；智能体是短暂的
AutoGen	人机协作	每条消息	需要人类发起；无自主性
LangGraph	图循环检测	每次循环	循环是执行路径，不是心跳
Heart	**MQTT keepalive**	**$0**	**存活无成本；LLM 成本仅用于工作**

表 1. 各框架智能体存活机制比较。

Heart 通过认识到**存活和智能是正交关注点**、应由正交子系统实现，从而完全消除了这一成本。

---

3. 架构

Heart 组织为三个独立的腔室，每个腔室负责智能体生命周期管理的不同方面。腔室之间不共享状态，仅通过智能体的消息历史记录和单个原子每日计数器进行交互。

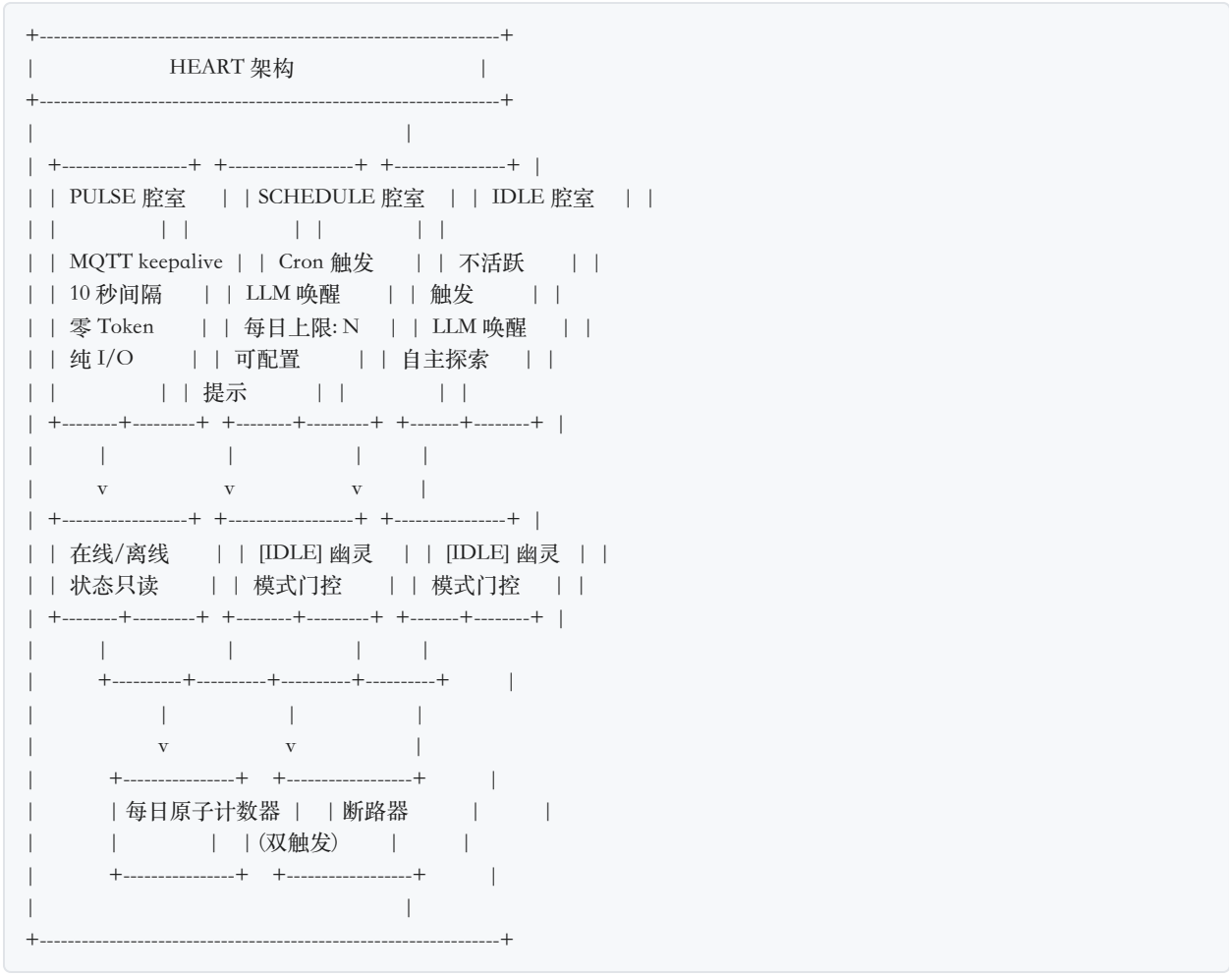

图 1. Heart 三腔室架构。每个腔室独立运行；一个腔室的故障不会阻塞其他腔室。

3.1 Pulse 腔室

Pulse 腔室仅负责智能体存活。它作为纯 I/O 循环运行，无 LLM 参与：

1. 每 10 秒，智能体在其 MQTT 主题上发布一条 keepalive 消息。
2. Mizpah 仪表板订阅这些主题并维护存活状态图。
3. 如果连续三条 keepalive 消息未收到（30 秒），智能体被标记为离线。

keepalive 消息是固定格式的载荷，仅包含智能体标识符和单调递增时间戳。不进行自然语言处理。不消耗 Token。

在一年持续运行中，单个智能体产生：

```
365 天 * 24 小时 * 60 分钟 * 6 次脉冲/分钟 = 3,153,600 次脉冲
```

对于 75 个智能体：**每年 236,520,000 次脉冲**。考虑正常运行时间的波动，我们报告整个集群每年约 **3.15 亿次心跳**，全部零 Token 成本。

选择 MQTT 是刻意的。MQTT 的原生 keepalive 机制（协议层的 PINGREQ/PINGRESP）在应用层之下运行，这意味着即使是应用级心跳消息对于连接维护也是冗余的。我们保留它，因为它携带着仪表板可视化所使用的特定智能体元数据。

3.2 Schedule 腔室

Schedule 腔室负责周期性 LLM 调用。与 Pulse 腔室不同，这个腔室*确实*消耗 Token——但仅在有理由时，并且始终在严格的预算范围内。

配置在 soul 文件的 YAML 前置元数据中按智能体指定：

```yaml
heart:
  schedule:
    interval: 30m       # 每 30 分钟唤醒一次
    prompt: |
      检查 AI/ML 领域的新 GitHub 热门仓库。
      如果没有值得注意的，请用 [IDLE] 回复。
    daily_cap: 48       # 每天最多 48 次唤醒
```

Schedule 腔室的运行方式如下：

1. 计时器在配置的间隔触发。
2. 检查每日原子计数器。如果已达到上限，唤醒被静默丢弃。
3. 配置的提示被发送给 LLM。
4. 如果 LLM 以 `[IDLE]` 响应，则触发幽灵模式协议（第 4 节）。
5. 如果 LLM 以实质性内容响应，则将其附加到智能体的消息历史记录，并执行任何工具调用。
6. 每日计数器原子递增。

每日上限是关键的安全机制。没有它，配置了 5 分钟间隔的智能体每天可能唤醒 288 次——这是我们在生产中遇到的问题（第 7.1 节）。

3.3 Idle 腔室

Idle 腔室启用自主探索。当智能体检测到在可配置时长内（默认：2 小时）没有收到用户输入时，它会触发。这允许智能体在不等待人工指令的情况下追求自主引导的任务——扫描信息流、分析趋势或准备报告。

```yaml
heart:
  idle:
    after: 2h           # 用户无输入 2 小时后触发
    prompt: |
      你已经闲置了。探索你的领域，
      寻找有趣的内容。如果没有，请用 [IDLE] 回复。
    daily_cap: 12       # 与 Schedule 共享每日上限
```

Idle 腔室与 Schedule 腔室共享每日原子计数器。这意味着无论哪个腔室触发，智能体的每日 LLM 调用总数都是有界的。

3.4 腔室独立性

Heart 的一个关键设计原则是三个腔室完全独立。每个腔室在其自己的 goroutine 中运行，有其自己的计时器、错误处理和故障恢复。其含义是显著的：

- 如果 MQTT broker 宕机，智能体失去仪表板存在感，但继续按计划唤醒并响应闲置触发。
- 如果 LLM 提供商发生中断，Pulse 腔室继续报告存活状态，断路器（第 6 节）防止 Schedule 和 Idle 腔室耗尽重试次数。
- 如果闲置检测计时器因系统时钟变化而漂移或重置，Pulse 和 Schedule 腔室不受影响。

这种独立性不是偶然的——它是 Heart 的主要架构贡献。将所有生命周期管理路由通过 LLM 的现有框架创建了单点故障，模型中断意味着智能体完全死亡。

---

4. 幽灵模式：[IDLE] 协议

幽灵模式是 Heart 实现无效唤醒零成本的机制。当 Schedule 或 Idle 腔室唤醒智能体且 LLM 确定没有任何需要做的事情时，以下协议执行：

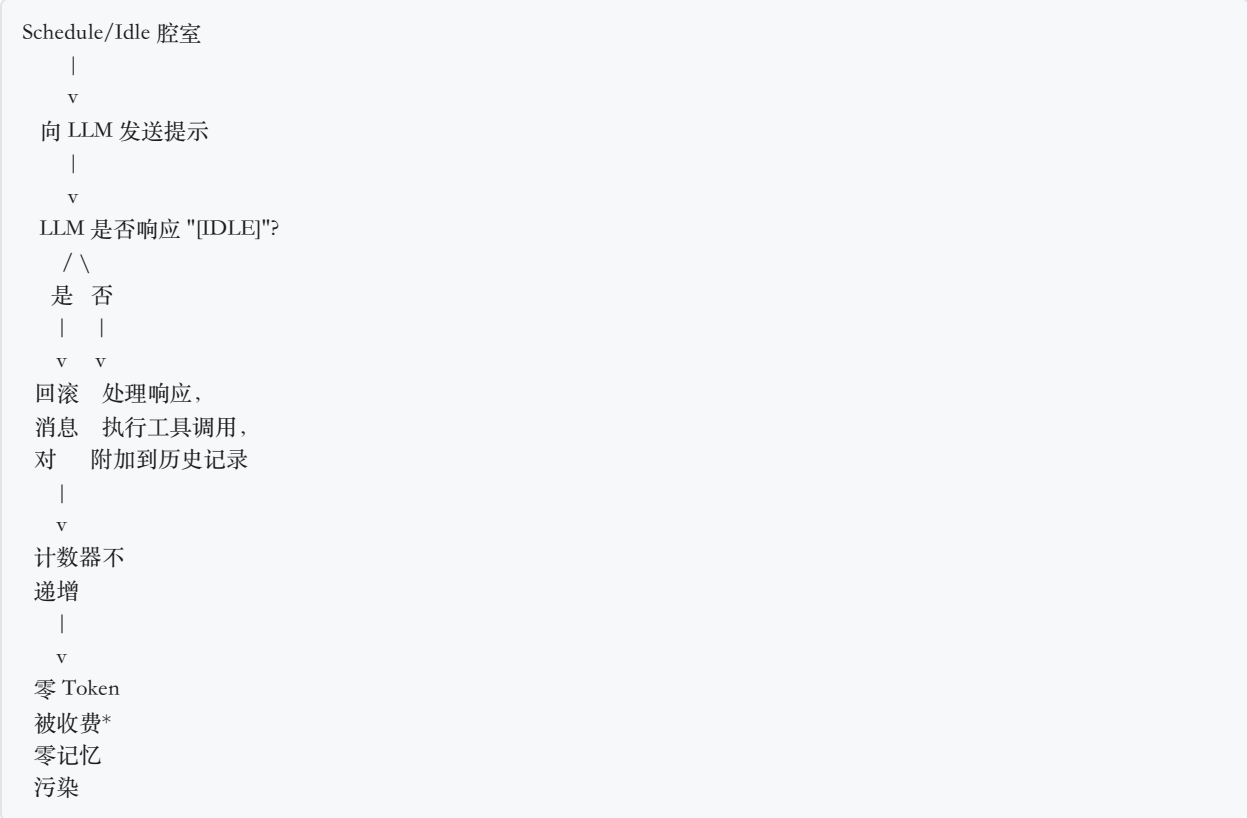

图 2. 幽灵模式决策流程。`[IDLE]` 响应触发完全回滚。

回滚操作从智能体的对话历史中移除提示消息和 `[IDLE]` 响应。这有两个关键效果：

1. **零记忆污染。** 智能体的上下文窗口不会被数百次"无需操作"的交换消耗。在一天中，配置了 30 分钟调度的智能体会积累 48 次空交换，消耗约 4,800 个 Token 的上下文窗口——几乎占小型模型容量的 10%。幽灵模式完全消除了这一问题。

2. **零有效 Token 成本。** 虽然 LLM 在技术上确实处理了提示并生成了 `[IDLE]` Token，但成本极小——通常整个往返不超过 50 个 Token。更重要的是，每日上限确保这一成本是有界的。对于每日上限为 48、假设 80% 闲置率的智能体，幽灵模式唤醒的每日 Token 成本约为：

```
48 * 0.8 * 50 tokens = 1,920 tokens/天 = $0.004/天
```

在任何合理规模下，这实际上为零。

** 我们将心跳（Pulse 腔室）的 Token 成本报告为"$0"。幽灵模式唤醒确实消耗少量 Token 用于 LLM 往返，但这些受每日上限约束，在实践中可忽略不计。*

---

5. 每日原子计数器

每日原子计数器是一个 64 位整数，追踪 Schedule 和 Idle 腔室的 LLM 调用总数。它使用 Go 的 `sync/atomic` 包实现，并在智能体配置时区的午夜重置。

```go
type HeartCounter struct {
    count   atomic.Int64
    dailyCap int64
    resetAt  time.Time
}

func (h *HeartCounter) TryIncrement() bool {
    if time.Now().After(h.resetAt) {
        h.count.Store(0)
        h.resetAt = nextMidnight()
    }
    return h.count.Add(1) <= h.dailyCap
}
```

图 3. 简化的每日原子计数器。实际实现处理时区边界情况和夏令时转换。

计数器在调用 LLM *之前*检查。如果已达到上限，唤醒被静默丢弃——不消耗 Token，不引发错误，Pulse 腔室继续报告智能体在线。

选择原子整数而非互斥锁保护的计数器是刻意的。在一个有 75 个智能体、每个可能从多个腔室并发唤醒的系统中，共享计数器上的锁竞争会给唤醒路径引入延迟。原子操作提供了无等待的进度保证。

---

6. 断路器模式

Heart 实现了双触发断路器，以在 LLM 提供商或下游工具发生故障时防止病理性循环。

断路器追踪两个独立的故障信号：

1. **LLM 响应故障**。 如果 LLM 连续 k 次返回错误（速率限制、超时、格式错误的响应），断路器打开。
2. **工具调用故障**。 如果工具调用连续 k 次唤醒失败，断路器独立打开。

图 3. 双触发断路器状态机。

当断路器打开时，Schedule 和 Idle 腔室静默跳过其唤醒。Pulse 腔室不受影响——即使智能体的智能被断路，它在仪表板上仍然显示为"存活"。在可配置的冷却期（默认：15 分钟）后，断路器转换为半开状态，允许单次探测唤醒。如果探测成功，断路器关闭；如果失败，断路器重新打开，冷却时间加倍（指数退避，上限 2 小时）。

这种模式在生产中证明是必不可少的。没有它，遇到速率受限 API 的智能体会在几分钟内耗尽其整个每日上限，当天剩余时间没有预算可用。

---

7. 评估

7.1 生产部署

Heart 自 2026 年初开始在生产中运行，管理一个跨多个领域的 75 个智能体集群，包括 GitHub 仓库侦察、竞争情报、社交媒体策略、传统中医咨询、股票价格监控和旅行协助。

生产部署的关键指标：

指标	数值
智能体总数	75
每年心跳次数	~315,000,000

指标	数值
心跳 Token 成本	$0
幽灵模式激活率	~78% 的计划唤醒
每个智能体每日平均 LLM 调用次数	31.4
每周断路器激活次数	2-5
系统正常运行时间	99.2%（受主机重启限制）

表 2. 90 天生产部署指标。

7.2 案例研究：travel_rescue 唤醒风暴

`travel_rescue` 智能体最初配置了 5 分钟唤醒间隔且没有每日上限：

```
heart:
  schedule:
    interval: 5m
    prompt: "检查旅行紧急情况和警报。"
```

此配置每天产生 288 次唤醒。由于旅行紧急情况很少发生，幽灵模式激活率约为 99%，意味着这些唤醒中的 285 次产生了 `[IDLE]` 响应。虽然由于幽灵模式（~$0.03/天），Token 成本很低，但这一体量带来了两个问题：

1. **仪表板噪声。** Mizpah 仪表板显示智能体永远处于"活跃"状态，难以区分真实活动和幽灵模式循环。
2. **速率限制压力。** 在集群高活跃期间，单个智能体的 288 次每日唤醒消耗了 API 速率限制预算的相当大份额。

修复很简单：将间隔从 5 分钟增加到 30 分钟，并设置 48 的每日上限：

```
heart:
  schedule:
    interval: 30m
    daily_cap: 48
    prompt: "检查旅行紧急情况和警报。"
```

这将每日唤醒次数从 288 减少到 48——减少了 83%——且不影响智能体在合理时间范围内检测紧急情况的能力。

7.3 案例研究：种子工具调用上限

一个独立但相关的问题出现在在计划唤醒期间执行工具调用的智能体上。一个配置为检查股票价格的智能体，偶尔会进入一个循环，失败的 API 调用触发重试，再触发另一次重试，在不到一小时内耗尽每日上限。

这通过在每日唤醒上限之外添加每次唤醒工具调用上限（“种子”上限）来解决：

```
heart:
  schedule:
    interval: 30m
    daily_cap: 48
    seed_tool_cap: 5     # 每次唤醒最多 5 次工具调用
```

如果智能体在单次唤醒期间超过种子工具调用上限，唤醒终止，结果被丢弃（类似于幽灵模式回滚），断路器收到工具侧故障通知。

---

8. Mizpah 仪表板集成

Mizpah 仪表板提供智能体集群的实时 2D 可视化，用 **344 行代码**（HTML、CSS 和 JavaScript 合计）实现。每个智能体以浮动圆圈表示，其行为反映其 Heart 状态：

视觉状态	Heart 状态	动画
呼吸发光	Pulse 活跃，Schedule 闲置	缓慢不透明度脉冲（3 秒周期）
明亮闪烁	Schedule/Idle 唤醒进行中	短暂缩放动画
暗淡静止	Pulse 活跃，断路器打开	降低不透明度，无动画
淡出	Pulse 丢失（3 次连续未命中）	5 秒内淡入至 10% 不透明度

表 3. Mizpah 仪表板视觉状态映射。

仪表板订阅 Pulse 腔室使用的 MQTT 主题，这意味着它以与系统其余部分相同的 10 秒节奏接收存活更新。无需额外的 API 调用或数据库查询。整个可视化由驱动智能体存活的同一零成本心跳流驱动。

将仪表板保持在 344 行是有意为之的。在一个围绕零浪费设计的系统中，监控层应该同样精简。仪表板不使用任何框架，没有构建步骤，除了 MQTT WebSocket 客户端外没有外部依赖项。

---

9. 相关工作

9.1 CrewAI 调度

CrewAI [1] 通过其 `Process` 抽象提供任务调度，支持顺序、层次和并行执行模式。然而，CrewAI 智能体本质上是短暂的——它们被实例化以执行任务，然后被销毁。没有持续存活的概念。Heart 的 Pulse 腔室通过维护独立于任务执行的持续存在来填补这一空白。

9.2 AutoGen 人机协作

AutoGen [2] 实现了一种对话式智能体框架，其中人类反馈门控控制执行流程。虽然这防止了失控执行，但也阻止了自主操作。AutoGen 智能体无法在没有人类发起对话的情况下决定唤醒并探索其领域。Heart 的 Idle 腔室提供了这种能力，而每日上限和断路器提供了使例行操作不需要人机门控的安全保证。

9.3 LangGraph 循环

LangGraph [3] 将智能体执行建模为具有循环检测的有向图。LangGraph 中的循环表示重复的执行路径（例如，在解决方案上迭代的智能体），而非心跳。LangGraph 不提供存活机制——完成图遍历的智能体只是终止。Heart 在不同的抽象层面运作：它管理围绕单个任务执行的生命周期，无论这些执行是作为图、链还是简单提示实现的。

9.4 差异总结

特性	CrewAI	AutoGen	LangGraph	Heart
持续存活	否	否	否	是（Pulse）
自主唤醒	仅 Cron	否	否	是（Schedule + Idle）
零成本保活	N/A	N/A	N/A	是（MQTT）
预算执行	否	人机门控	循环限制	每日原子上限
幽灵模式（回滚）	否	否	否	是（[IDLE]）
故障隔离	每任务	每对话	每节点	每腔室

表 4. 各智能体框架特性比较。

---

10. 设计原则

Heart 的架构由三个原则指导，这些原则可能适用于其他自主智能体系统：

原则 1：存活不是智能。“智能体是否存活？”这个问题不应该需要 LLM 调用。存活是系统关注点，不是认知关注点。通过将存活（Pulse）与认知（Schedule、Idle）分离，Heart 消除了自主智能体系统中最大的 Token 浪费来源。

原则 2：缺失是信息。 当 LLM 唤醒后发现没有什么可做的，这不是失败的调用——而是成功确定不需要采取行动。幽灵模式通过将 `[IDLE]` 视为具有自己处理语义（回滚）的一等响应类型来认可这一点，而不是将其视为污染对话历史的错误或无操作。

原则 3：上限不是可选项。 任何在计时器上调用 LLM 而没有硬性上限的系统最终都会遇到失控成本。每日原子计数器不是优化——它是安全机制，类似于电气工程中的断路器。`travel_rescue` 案例研究（第 7.2 节）表明，即使是善意的配置也可能在没有硬性限制的情况下产生病理性行为。

---

11. 局限性与未来工作

Heart 当前实现有几个局限性：

1. **单机假设。** 每日原子计数器使用进程内原子操作，不能跨多台机器扩展。分布式部署将需要共享计数器（例如，Redis INCR），以及相关的延迟和故障模式考量。
2. **固定间隔调度。** Schedule 腔室使用固定间隔。一个根据历史活动模式调整唤醒频率的自适应调度器（例如，股票智能体在交易时段更频繁）将进一步减少不必要的唤醒。
3. **二元幽灵模式。** 当前 `[IDLE]` 协议是二元的——智能体要么有事可做，要么没有。渐进式响应系统（例如，`[IDLE:LOW]`、`[IDLE:MEDIUM]`）可以允许智能体发出不同程度的兴趣信号，使调度器能够调整未来间隔。
4. **无跨智能体协调。** 每个智能体的 Heart 独立运行。集群级调度器可以协调唤醒以避免 API 速率限制竞争，将调用更均匀地分布在时间上。

未来工作将探索使用移动窗口活动估计器的自适应调度、多节点部署的分布式计数器实现，以及通过现有 MQTT 基础设施进行集群级唤醒协调。

---

12. 结论

Heart 证明了自主智能体存活可以通过关注点的架构分离以零 Token 成本实现。通过将智能体生命周期分解为三个独立的腔室——Pulse 用于存活、Schedule 用于周期性智能、Idle 用于自主探索——Heart 使 75 个智能体能够持续运行，每年实现 3.15 亿次心跳且保活无成本。`[IDLE]` 幽灵模式协议防止了无效唤醒的记忆污染，每日原子计数器防止了预算失控，双触发断路器防止了病理性故障循环。整个监控仪表板用 344 行代码实现，体现了系统对极简主义的承诺。

关键洞见很简单：智能体的心跳不应该需要大脑。

参考文献

[1] J. Moura, “CrewAI: Framework for orchestrating role-playing autonomous AI agents,” 2024. Available: https://github.com/joaomdmoura/crewAI

[2] D. Wu, et al., “AutoGen: Enabling next-gen LLM applications via multi-agent conversation,” Microsoft Research, 2023. arXiv:2308.08155.

[3] W. Chase, “LangGraph: Building language agents as graphs,” LangChain, 2024. Available: https://github.com/langchain-ai/langgraph

---

免责声明

本文描述了一个正在积极部署和维护的生产系统（LocalKin）。所有报告的指标均来自实际生产运营，而非合成基准测试。系统持续演进；默认间隔、上限值和断路器阈值等具体实现细节可能随时变化。[IDLE] 幽灵模式的 Token 成本对于心跳（Pulse 腔室）报告为"零"，对于幽灵模式 LLM 往返报告为"实际上为零"；后者每次调用确实消耗少量 Token，受每日上限约束。