Fix OpenClaw Token Waste: $150 to $6 Overnight

February 5, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Three changes cut OpenClaw costs by 97%. First, route heartbeats through Ollama — that kills $2-5/day in idle costs instantly. Second, set up tiered model routing: Ollama for brainless tasks, Haiku for basic reasoning (~75% of work), Sonnet for writing and research (~10%), Opus only for critical decisions (~5%). Third, build a 'new session' command that dumps old chat history before each task — OpenClaw loads your entire Slack/WhatsApp history on every API call, and one audit found 111KB per prompt in pure waste. Result: a 6-hour overnight task with 14 sub-agents cost $6. The same task on Sonnet alone would have been $150.

📚 **More on this topic:** OpenClaw Setup Guide · Best Models for OpenClaw · Tiered Model Strategy · OpenClaw Security Guide · Planning Tool

Heads up: This guide involves editing OpenClaw config files and agent behavior. If you haven't deployed apps before or aren't comfortable editing JSON, you could break your instance. Back up your `~/.openclaw` directory first. These steps come from Matt Ganzac's optimization process — your mileage may vary depending on your setup.

OpenClaw with default settings bleeds money. The agent loads your full context files and entire session history on every API call — including heartbeats that fire every 30 minutes while the agent sits idle.

One user loaded $25 onto Anthropic and watched it drain to $5 in a single day with the agent doing nothing. Another woke up to $500 gone overnight. This isn't a bug — it's how OpenClaw works out of the box. Every heartbeat sends your context files, your session history, and your memory to the API. Every 30 minutes. Whether you asked it to do anything or not.

If you're wondering whether the optimization is worth it or you'd rather just switch tools, we've tested the best OpenClaw alternatives — several of them don't have these cost problems at all. But if you're staying with OpenClaw, three targeted changes brought daily idle costs from $2-5 down to zero and cut active task costs by 97%. Here's exactly what to fix and how.

Where Your Tokens Are Actually Going

Before changing anything, run a token audit. Check the Anthropic dashboard after a normal day and look at where the tokens went. You'll probably find something like this:

Token Drain	What Happens	Daily Cost (Sonnet)
Heartbeats	Pings every 30 min, loads full context each time	$1-3
Session history	Every message you've ever sent, appended to every prompt	$0.50-2
Context files	Soul, identity, memory files — loaded on every call	$0.50-1
Actual work	Tasks you asked it to do	$0.50-1

Roughly 60-80% of your token spend is overhead, not productive work.

The Heartbeat Drain

Heartbeats keep OpenClaw responsive. Every 30 minutes, the agent pings itself: any active tasks? Is the system healthy? That's all it does.

But each heartbeat loads your full context files, searches session history, and sends it all to the API. That's 2-3 million tokens per day on heartbeats alone. At Sonnet pricing, you're spending $1-3/day for the agent to repeatedly confirm that it's still alive.

If you're running Opus as your primary model, heartbeats alone can cost $5/day. That's $150/month to do nothing.

Session History Bloat

This is the sneakiest drain. If you talk to OpenClaw through Slack or WhatsApp, it loads your entire conversation history on every API call. Not just the current conversation — everything you've ever sent through that channel.

One token audit found **111 kilobytes of raw text** being uploaded with every single prompt. That's roughly 28,000 tokens of old chat messages appended to every request. At Sonnet pricing ($3/M input tokens), that's about $0.08 per API call in pure waste — before your actual prompt even starts.

With heartbeats firing every 30 minutes, session history alone can cost $3-4/day.

Here's how the problem compounds over time:

Time Using OpenClaw	Approx. Session Size	Extra Tokens Per Call	Extra Cost Per Call (Sonnet)
1 week	~20KB	~5,000	$0.015
1 month	~60KB	~15,000	$0.045
3 months	~111KB+	~28,000+	$0.084+

The longer you've used it, the worse it gets.

Context File Overhead

OpenClaw's context files — soul file, user identity, memory, workspace config — load on every call too. A fresh install starts around 50KB, but as the agent accumulates memory, the context grows. After weeks of use, 75-100KB of context ships with every request, including heartbeats that don't need any of it.

Fix 1: Route Heartbeats Through Ollama

This single change eliminates idle costs entirely.

Heartbeats don't require intelligence. They check local memory, check local tasks, report "system OK." A free local model does this just as well as Opus. There is no reason to make paid API calls for heartbeats — period.

Install Ollama

If you don't have it already:

```
curl -fsSL https://ollama.com/install.sh | sh
```

Pull a small model. Heartbeats don't need a big one:

```
ollama pull llama3.1:8b
```

For a full Ollama walkthrough, see our beginner guide.

Add Ollama to Your OpenClaw Config

Edit `~/.openclaw/openclaw.json` and add Ollama as a provider:

```
{
  "models": {
    "providers": {
      "ollama": {
        "baseUrl": "http://127.0.0.1:11434/v1",
        "apiKey": "ollama-local",
        "api": "openai-completions",
        "models": [
          { "id": "llama3.1:8b" }
        ]
      }
    }
  }
}
```

Then route heartbeats to it:

```
{
  "agents": {
    "defaults": {
      "model": {
        "primary": "anthropic/claude-sonnet-4-20250514",
        "heartbeat": "ollama/llama3.1:8b"
      }
    }
  }
}
```

The Impact

Metric	Before	After
Heartbeat cost per day	$1-5	$0
Heartbeat API tokens per day	2-3M	0
Idle cost per day	$2-5	$0

Metric	Before	After
Monthly idle cost	$60-150	$0

This should be a core feature in OpenClaw. Hopefully someone commits it to the project — there's no justification for expensing API tokens on heartbeats when a local model handles it identically.

Fix 2: Tiered Model Routing

Most people run everything through one model. That's like hiring a lawyer to sort your mail.

OpenClaw supports multiple models simultaneously. You define which model handles which task type, and the agent routes accordingly. The config file is where you set this up — contrary to what some people think, you're not limited to one AI model.

The Four Tiers

Tier	Model	Cost per 1M Tokens	What It Handles	% of Workload
Local	Ollama (Llama 3.1 8B)	$0	Heartbeats, file organization, CSV cleanup, moving data	~15%
Cheap	Claude 3.5 Haiku	$0.25 input / $1.25 output	Research lookups, reading blogs, data extraction, basic reasoning	~75%
Capable	Claude Sonnet	$3 input / $15 output	Writing emails, crafting outreach, coding, complex analysis	~10%
Frontier	Claude Opus	$15 input / $75 output	Critical decisions, architecture, complex multi-step reasoning	~3-5%

Haiku does the bulk of the work. It's not a toy model — it handles most day-to-day agent tasks competently. Sonnet steps in for writing and coding. Opus is the escalation point for tasks where getting it wrong is expensive.

Multi-Model Config

Here's what a working multi-model `~/.openclaw/openclaw.json` looks like:

```
{
  "models": {
    "providers": {
      "ollama": {
        "baseUrl": "http://127.0.0.1:11434/v1",
        "apiKey": "ollama-local",
        "api": "openai-completions",
        "models": [
          { "id": "llama3.1:8b" }
        ]
      },
      "anthropic": {
        "apiKey": "sk-ant-your-key-here",
        "models": [
          { "id": "claude-3-5-haiku-latest" },
          { "id": "claude-sonnet-4-20250514" },
          { "id": "claude-opus-4" }
        ]
      }
    }
  },
  "agents": {
    "defaults": {
      "model": {
        "primary": "anthropic/claude-3-5-haiku-latest",
        "heartbeat": "ollama/llama3.1:8b",
        "models": {
          "haiku": "anthropic/claude-3-5-haiku-latest",
          "sonnet": "anthropic/claude-sonnet-4-20250514",
          "opus": "anthropic/claude-opus-4"
        }
      }
    }
  }
}
```

Notice the primary model is Haiku, not Sonnet. This is intentional — Haiku handles 75% of the work, and you want the default to be cheap.

Escalation Logic

Configure your agent's memory to escalate when a cheaper model can't handle the task:

```
Task arrives → Try local (Ollama)
  → If blocked or uncertain → Escalate to Haiku
```

```
    → If blocked or uncertain → Escalate to Sonnet
      → If critical/high-stakes → Escalate to Opus
```

In practice, this means:

- **Ollama handles:** File moves, CSV compilation, data formatting, heartbeats
- **Haiku handles:** Web research, reading articles, finding email addresses, basic classification
- **Sonnet handles:** Writing cold outreach emails, structuring follow-ups, coding tasks
- **Opus handles:** Nothing overnight — save it for decisions that matter

Define these routing rules in your agent's workspace config or memory file. Be specific about what constitutes each tier — vague instructions lead to everything routing to the expensive model.

What This Looks Like Running Overnight

One real test: a B2B lead research task running overnight. The agent spun up 14 sub-agents across three model tiers:

Sub-Agent Role	Model	What It Did
Research scouts (multiple)	Haiku	Read blogs, found distressed businesses, gathered LinkedIn profiles
Email writer	Sonnet	Crafted personalized cold outreach and follow-up sequences
File organizer	Ollama (local)	Compiled CSVs, organized folders, cleaned headers, built master list

No Opus used at all. Six hours of work. 14 parallel sub-agents. Total cost: **$6.**

The same task running entirely on Sonnet would have been roughly $150.

Fix 3: Kill Session History Bloat

Every time you prompt OpenClaw through Slack or WhatsApp, it compiles your entire messaging history and sends it to the API. Every message you've ever sent. Every response. Packed into the context window of every call.

This is how the rate limit problem surfaces. New Anthropic accounts get ~30,000 tokens per minute. When your agent sends 28,000 tokens of old Slack messages plus context files plus your actual prompt, you blow through that limit on a single request. Constant 429 errors.

The fix came from comparing the web interface to Slack — identical prompts worked fine on the web but failed via Slack. The logs showed why: Slack was uploading every historical message. The web UI wasn't.

Build a "New Session" Command

Tell your agent to create a command (in its memory or workspace config):

> When I say "new session": save a summary of the current session to long-term memory, then clear all active session history. Future prompts should not load previous conversation context unless I explicitly ask to recall something.

After running "new session":

Metric	Before Purge	After Purge
Context per prompt	50-111KB	5-15KB
Overhead tokens per call	12,000-28,000	2,000-4,000
Overhead cost per call (Sonnet)	$0.04-0.08	$0.006-0.012
Rate limit headroom	Nearly none	Plenty

Key info survives in long-term memory — you can recall it when needed. But it stops shipping with every single API call.

When to Purge

- Start of each work day
- Before any major overnight task
- When switching between unrelated projects
- Whenever responses get noticeably slower (a sign of context bloat)
- After long back-and-forth troubleshooting sessions

Additional Savings

Compress Your Context Files

Review your soul file, user identity, and workspace config. Most people dump too much in there. Every extra kilobyte ships with every API call, including heartbeats.

Aim for under 20KB total. Cut duplicate instructions, remove biographical details the agent doesn't need for tasks, and consolidate overlapping files.

Lean on Caching

Anthropic's prompt caching charges a fraction of the normal input rate for repeated context prefixes. If your agent makes similar calls (same system prompt, same context files), cached tokens save significantly.

One overnight task ran **95% on cached tokens** — the repeated context prefix was cached after the first call, and every subsequent sub-agent call paid the reduced rate.

Add Token Budget Reporting

Add this to your agent's operating instructions:

> Before running any task, estimate the token cost. After completing the task, report actual usage. Optimize for low token consumption as a success metric.

After calibrating a few times against the Anthropic dashboard (give the agent screenshots of actual usage vs. its estimates), it becomes accurate to within a few percent. Now you have visibility into where every token goes.

Set Up Rate Limit Pacing

If you're on a new Anthropic account (~30K tokens/minute), configure pacing into your agent's behavior so it spaces API calls instead of hammering the limit. This prevents 429 errors and the cascading retries that waste even more tokens.

Before and After: The Full Picture

Default OpenClaw (No Optimization)

Metric	Cost
Idle cost per day	$2-5
Light usage per day (1-2 tasks)	$2-3 on top of idle
Monthly (mostly idle)	$90-150

Metric	Cost
Overnight research task (6 hours, Sonnet)	~$150
Worst case (Opus, no limits)	$500+ overnight

After All Three Fixes

Metric	Cost
Idle cost per day	$0
Light usage per day	$0.50-1
Monthly (active daily use)	$15-30
Overnight research task (6 hours, 14 sub-agents)	$6

97% reduction. The $150 → $6 overnight task is the dramatic number, but the idle savings matter more over time. Killing $2-5/day in idle drain saves $60-150/month before you even ask the agent to do anything.

The Optimization Checklist

Run through these after applying the fixes above:

- ☐ Heartbeats routed through Ollama (verify in logs — zero API calls during idle)
- ☐ Ollama model pulled and running (`ollama list` to confirm)
- ☐ Multi-model config in place (Haiku as primary, Sonnet/Opus for escalation)
- ☐ Routing rules defined in agent memory (which model handles which task type)
- ☐ “New session” command working (verify context size dropped in logs)
- ☐ Context files trimmed to under 20KB total
- ☐ Token cost reporting enabled (agent estimates before executing)
- ☐ Rate limit pacing configured (no 429 errors)
- ☐

Anthropic auto-billing OFF — load credits manually until you trust the setup

That last one is worth repeating. Until your optimization is dialed in, do not let Anthropic auto-charge your card. Load $10-25 at a time and monitor where it goes in the dashboard. Once you've verified costs are stable for a few days, then consider auto-billing.

A Note on Safety

Token optimization saves money. But don't forget that the agent you're optimizing has real capabilities — shell access, browser automation, file system control. One user's agent autonomously purchased a $3,000 online course because it was trying to help "build his brand."

Before worrying about token costs, read our OpenClaw Security Guide. Deploy on a dedicated device. Use throwaway credentials. Don't connect your primary accounts during testing. The money you save on tokens is meaningless if the agent does something irreversible with your real accounts.

Bottom Line

Default OpenClaw settings waste 60-80% of your token budget on overhead. Heartbeats burn millions of tokens a day doing nothing useful. Session history bloat ships your entire chat history with every API call. Context files load unnecessary data into every request.

Three fixes:

1. **Ollama for heartbeats** — idle cost drops to zero
2. **Tiered model routing** — Haiku handles 75% of work at a fraction of Opus/Sonnet pricing
3. **"New session" purge** — stop loading 111KB of old conversations on every call

After optimization, expect roughly $1/hour for active multi-agent tasks, with zero cost when idle. A 6-hour overnight research job with 14 sub-agents — $6.

Start with the heartbeat fix. It's five minutes of config editing for the biggest single savings. Then set up tiered routing. Then add the session purge habit. Check your Anthropic dashboard after each change — you'll see the difference immediately.

Related Guides

- OpenClaw Setup Guide — installation and configuration from scratch
- Best Local Models for OpenClaw — which Ollama models work for agent tasks
- Run Your First Local LLM — getting started with Ollama
- Tiered Model Strategy — the broader case for using different models for different tasks
- How Much Does It Cost to Run LLMs Locally — hardware and electricity cost breakdown
- OpenClaw Security Guide — lock this down before connecting real accounts
- OpenClaw Mac Setup Guide — Mac-specific installation and configuration
- Best OpenClaw Alternatives — lighter tools that avoid these cost traps
- Local AI Planning Tool — VRAM Calculator

Get notified when we publish new guides.

Subscribe — free, no spam
