OpenClaw

Token Optimization Guide

Reduce Your AI Costs by 97%

From $1,500+/month to under $50/month

WHAT YOU'LL ACHIEVE

97% Token Reduction • 5 Minutes to Implement • No Complex Setup

Free Local Heartbeat • Smart Model Routing • Session Management

ScaleUP Media

@mattganzak

Introduction

If you've been running OpenClaw and watching your API bills climb, you're not alone. The default configuration prioritizes capability over cost, which means you're probably burning through tokens on routine tasks that don't need expensive models.

This guide covers four key optimizations that work together to slash your costs:

1. Session Initialization — Stop loading 50KB of history on every message
2. Model Routing — Use Haiku for routine tasks, Sonnet only when needed
3. Heartbeat to Ollama — Move your heartbeat checks to a free local LLM
4. Rate Limits & Budgets — Prevent runaway automation from burning tokens

> **Why This Matters**
>
> Each optimization targets a different cost driver. Combined, they take you from $1,500+/month down to $30-50/month. That's money you can reinvest in actually building things.

Overall Cost Impact

Time Period	Before	After
Daily	$2-3	**$0.10**
Monthly	$70-90	**$3-5**
Yearly	$800+	**$40-60**

Part 1: Session Initialization

THE PROBLEM

Your agent loads 50KB of history on every message. This wastes 2-3M tokens per session and costs $4/day. If you're using third-party messaging or interfaces that don't have built-in session clearing, this problem compounds fast.

The Solution

Add this session initialization rule to your agent's system prompt. It tells your agent exactly what to load (and what NOT to load) at the start of each session:

```
SESSION INITIALIZATION RULE:

On every session start:
1. Load ONLY these files:
   - SOUL.md
   - USER.md
   - IDENTITY.md
   - memory/YYYY-MM-DD.md (if it exists)

2. DO NOT auto-load:
   - MEMORY.md
   - Session history
   - Prior messages
   - Previous tool outputs

3. When user asks about prior context:
   - Use memory_search() on demand
   - Pull only the relevant snippet with memory_get()
   - Don't load the whole file

4. Update memory/YYYY-MM-DD.md at end of session with:
   - What you worked on
   - Decisions made
   - Leads generated
   - Blockers
   - Next steps

This saves 80% on context overhead.
```

Why This Works

- Session starts with 8KB instead of 50KB
- History loads only when asked

- Daily notes become your actual memory
- Works with any interface — no built-in session clearing needed

Results: Before & After

❌ BEFORE	✓ AFTER
50KB context on startup	8KB context on startup
2-3M tokens wasted per session	Only loads what's needed
$0.40 per session	$0.05 per session
History bloat over time	Clean daily memory files
No session management	Works with any interface

Part 2: Model Routing

Out of the box, OpenClaw typically defaults to using Claude Sonnet for everything. While Sonnet is excellent, it's overkill for tasks like checking file status, running simple commands, or routine monitoring. Haiku handles these perfectly at a fraction of the cost.

Step 1: Update Your Config

Your OpenClaw config file is located at:

```
~/.openclaw/openclaw.json
```

Add or update your config with these model settings:

```
{
  "agents": {
    "defaults": {
      "model": {
        "primary": "anthropic/claude-haiku-4-5"
      },
      "models": {
        "anthropic/claude-sonnet-4-5": {
          "alias": "sonnet"
        },
        "anthropic/claude-haiku-4-5": {
          "alias": "haiku"
        }
      }
    }
  }
}
```

What This Does

Sets Haiku as your default model (fast and cheap) and creates easy aliases so your prompts can say "use sonnet" or "use haiku" to switch models on-demand.

Step 2: Add Routing Rules to System Prompt

```
MODEL SELECTION RULE:

Default: Always use Haiku
Switch to Sonnet ONLY when:
- Architecture decisions
- Production code review
- Security analysis
```

```
- Complex debugging/reasoning
- Strategic multi-project decisions

When in doubt: Try Haiku first.
```

Results: Before & After

❌ BEFORE	✓ AFTER
Sonnet for everything	Haiku by default
$0.003 per 1K tokens	$0.00025 per 1K tokens
Overkill for simple tasks	Right model for the job
$50-70/month on models	$5-10/month on models

Part 3: Heartbeat to Ollama

OpenClaw sends periodic heartbeat checks to verify your agent is running and responsive. By default, these use your paid API — which adds up fast when you're running agents 24/7. The solution? Route heartbeats to a free local LLM using Ollama.

Step 1: Install Ollama

If you don't already have Ollama installed, grab it from ollama.ai or run:

```
# macOS / Linux
curl -fsSL https://ollama.ai/install.sh | sh

# Then pull a lightweight model for heartbeats
ollama pull llama3.2:3b
```

> **Why llama3.2:3b?**
> It's lightweight (2GB), fast, and handles complex context better than 1b for production use

Step 2: Configure OpenClaw for Ollama Heartbeat

Update your config at ~/.openclaw/openclaw.json to route heartbeats to Ollama:

```
{
  "agents": {
    "defaults": {
      "model": {
        "primary": "anthropic/claude-haiku-4-5"
      },
      "models": {
        "anthropic/claude-sonnet-4-5": {
          "alias": "sonnet"
        },
        "anthropic/claude-haiku-4-5": {
          "alias": "haiku"
        }
      }
    }
  },
  "heartbeat": {
    "every": "1h",
    "model": "ollama/llama3.2:3b",
    "session": "main",
    "target": "slack",
    "prompt": "Check: Any blockers, opportunities, or progress updates needed?"
  }
```

Configuration Options

Option	Description
`interval`	Seconds between heartbeat checks (60 = once per minute)
`provider`	Set to "ollama" to use local LLM instead of paid API
`model`	Any Ollama model (llama3.2:1b is fast and tiny)
`endpoint`	Ollama's local API (default: http://localhost:11434)

Step 3: Verify Ollama is Running

```
# Make sure Ollama is running
ollama serve

# In another terminal, test the model
ollama run llama3.2:3b "respond with OK"

# Should respond quickly with "OK" or similar
```

Results: Before & After

❌ BEFORE	✓ AFTER
Heartbeats use paid API	Heartbeats use free local LLM
1,440 API calls/day (every minute)	Zero API calls for heartbeats
$5-15/month just for heartbeats	$0/month for heartbeats
Adds to rate limit usage	No impact on rate limits

Part 4: Rate Limits & Budget Controls

Even with model routing and optimized sessions, runaway automation can still burn through tokens. These rate limits act as guardrails to protect you from accidental cost explosions.

Add to Your System Prompt

```
RATE LIMITS:

- 5 seconds minimum between API calls
- 10 seconds between web searches
- Max 5 searches per batch, then 2-minute break
- Batch similar work (one request for 10 leads, not 10 requests)
- If you hit 429 error: STOP, wait 5 minutes, retry

DAILY BUDGET: $5 (warning at 75%)
MONTHLY BUDGET: $200 (warning at 75%)
```

Limit	What It Prevents
5s between API calls	Rapid-fire requests that burn tokens
10s between searches	Expensive search loops
5 searches max, then break	Runaway research tasks
Batch similar work	10 calls when 1 would do
Budget warnings at 75%	Surprise bills at end of month

Results: Before & After

❌ BEFORE	✓ AFTER
No rate limiting	Built-in pacing
Agent makes 100+ calls in loops	Controlled, predictable usage
Search spirals burn $20+ overnight	Max exposure capped daily
No budget visibility	Warnings before limits hit

Part 5: Workspace File Templates

Create these files in your workspace. They provide the essential context your agent needs while keeping the token footprint minimal.

SOUL.md Template

This file defines your agent's core principles and operating rules:

```
# SOUL.md

## Core Principles

[YOUR AGENT PRINCIPLES HERE]

## How to Operate

See OPTIMIZATION.md for model routing and rate limits.

## Model Selection

Default: Haiku
Switch to Sonnet only for: architecture, security, complex reasoning

## Rate Limits

5s between API calls, 10s between searches, max 5/batch then 2min break
```

USER.md Template

This file gives your agent context about you and your goals:

```
# USER.md

- **Name:** [YOUR NAME]
- **Timezone:** [YOUR TIMEZONE]
- **Mission:** [WHAT YOU'RE BUILDING]

## Success Metrics

- [METRIC 1]
- [METRIC 2]
- [METRIC 3]
```

Keep It Lean

Resist the urge to add everything to these files. Every line costs tokens on every request. Include only what the agent absolutely needs to make good decisions.

Part 6: Prompt Caching

90% Token Discount on Reused Content

THE PROBLEM

Your system prompt, workspace files (SOUL.md, USER.md), and reference materials get sent to the API with every single message. If your system prompt is 5KB and you make 100 API calls per week, that's 500KB of identical text being re-transmitted and re-processed every week. With Claude, you're paying full price for every copy.

THE SOLUTION

Prompt caching (available on Claude 3.5 Sonnet and newer) charges only 10% for cached tokens on re-use and 25% for cache writes. For static content you use repeatedly, this cuts costs by 90%.

How Prompt Caching Works

When you send content to Claude:

5. **First request:** Full price (1 token = $0.003)
6. **Claude stores it in cache:** Marked for reuse
7. **Subsequent requests (within 5 minutes):** 90% discount ($0.00003 per token)

What This Means

A 5KB system prompt costs ~$0.015 on first use, then $0.0015 on each reuse. Over 100 calls/week, you save ~$1.30/week just on system prompts.

Step 1: Identify What to Cache

✓ CACHE THESE	❌ DON'T CACHE
System prompts (rarely change)	Daily memory files (change frequently)
SOUL.md (operator principles)	Recent user messages (fresh each session)
USER.md (goals and context)	Tool outputs (change per task)
Reference materials (pricing, docs, specs)	
Tool documentation (rarely updated)	
Project templates (standard structures)	

Step 2: Structure for Caching

OpenClaw automatically uses prompt caching when available. To maximize cache hits, keep static content in dedicated files:

```
/workspace/
  ├── SOUL.md                    ← Cache this (stable)
  ├── USER.md                    ← Cache this (stable)
  ├── TOOLS.md                   ← Cache this (stable)
  ├── memory/
  │   ├── MEMORY.md              ← Don't cache (frequently updated)
  │   └── 2026-02-03.md          ← Don't cache (daily notes)
  └── projects/
      └── [PROJECT]/REFERENCE.md  ← Cache this (stable docs)
```

Step 3: Enable Caching in Config

Update ~/.openclaw/openclaw-config.json to enable prompt caching:

```
{
  "agents": {
    "defaults": {
      "model": {
        "primary": "anthropic/claude-haiku-4-5"
      },
      "cache": {
        "enabled": true,
        "ttl": "5m",
        "priority": "high"
      },
      "models": {
        "anthropic/claude-sonnet-4-5": {
          "alias": "sonnet",
          "cache": true
        },
        "anthropic/claude-haiku-4-5": {
          "alias": "haiku",
          "cache": false
        }
      }
    }
  }
}
```

Note

Caching is most effective with Sonnet (better reasoning tasks where larger prompts are justified). Haiku's efficiency makes caching less critical.

Configuration Options

Option	Description
`cache.enabled`	true/false — Enable prompt caching globally
`cache.ttl`	Time-to-live: "5m" (default), "30m" (longer sessions), "24h"
`cache.priority`	"high" (prioritize caching), "low" (balance cost/speed)
`models.cache`	true/false per model — Sonnet recommended, Haiku optional

Step 4: Cache Hit Strategy

To maximize cache efficiency:

1. Batch requests within 5-minute windows

- Make multiple API calls in quick succession
- Reduces cache misses between requests

2. Keep system prompts stable

- Don't update SOUL.md mid-session
- Changes invalidate cache; batch them during maintenance windows

3. Organize context hierarchically

- Core system prompt (highest priority)
- Stable workspace files
- Dynamic daily notes (uncached)

4. For projects: Separate stable from dynamic

- product-reference.md (stable, cached)
- project-notes.md (dynamic, uncached)
- Prevents cache invalidation from note updates

Real-World Example: Outreach Campaign

You're running 50 outreach email drafts per week using Sonnet (reasoning + personalization).

WITHOUT CACHING	WITH CACHING (BATCHED)
System prompt: 5KB × 50 = 250KB/week	System prompt: 1 write + 49 cached
Cost: $0.75/week	Cost: $0.016/week
50 drafts × 8KB = $1.20/week	50 drafts (~50% cache hits) = $0.60/week
Total: $1.95/week = $102/month	**Total: $0.62/week = $32/month**
	SAVINGS: $70/month

Results: Before & After

❌ BEFORE	✓ AFTER
System prompt sent every request	System prompt cached, reused
Cost: 5KB × 100 calls = $0.30	Cost: 5KB × 100 calls = $0.003
No cache strategy	Batched within 5-minute windows
Random cache misses	90% hit rate on static content
Monthly reused content: $100+	Monthly reused content: $10
Single project: $50-100/month	Single project: $5-15/month
Multi-project: $300-500/month	Multi-project: $30-75/month

Step 5: Monitor Cache Performance

Check cache effectiveness with session_status:

```
openclaw shell
session_status

# Look for cache metrics:
# Cache hits: 45/50 (90%)
# Cache tokens used: 225KB (vs 250KB without cache)
# Cost savings: $0.22 this session
```

Or query the API directly:

```
# Check your usage over 24h
curl https://api.anthropic.com/v1/usage \
  -H "Authorization: Bearer $ANTHROPIC_API_KEY" | jq '.usage.cache'
```

Metrics to Track

Metric	What It Means
Cache hit rate > 80%	Caching strategy is working
Cached tokens < 30% of input	System prompts are too large (trim)
Cache writes increasing	System prompt changing too often (stabilize)
Session cost -50% vs last week	Caching + model routing combined impact

Combining Caching with Other Optimizations

Caching multiplies the benefit of earlier optimizations:

Optimization	Before	After	With Cache
Session Init (lean context)	$0.40	$0.05	**$0.005**
Model Routing (Haiku default)	$0.05	$0.02	**$0.002**
Heartbeat to Ollama	$0.02	$0	**$0**
Rate Limits (batch work)	$0	$0	**$0**
Prompt Caching	$0	$0	**-$0.015**
COMBINED TOTAL	**$0.47**	**$0.07**	**$0.012**

When to NOT Use Caching

- **Haiku tasks (too cheap to cache):** Caching overhead > savings
- **Frequent prompt changes:** Cache invalidation costs more than caching saves
- **Small requests (< 1KB):** Caching overhead eats the discount
- **Development/testing:** Too many prompt iterations; cache thrashing

Best Practices Checklist

✓ Cache stable system prompts (SOUL.md, USER.md)
✓ Batch requests within 5-minute windows
✓ Keep reference docs in separate cached files
✓ Monitor cache hit rate (target: > 80%)
✓ Combine caching with model routing (Sonnet + cache = max savings)
✓ Update system prompts during maintenance windows, not live
✓ Document cache strategy in TOOLS.md for consistency

The Bottom Line

Prompt caching is effortless cost reduction. With minimal setup, you get 90% discounts on content you're already sending. Combined with the other five optimizations, you go from $1,500+/month to $30-50/month.

Verifying Your Setup

After making these changes, verify everything is working correctly:

Check Your Configuration

```
# Start a session
openclaw shell

# Check current status
session_status

# You should see:
# - Context size: 2-8KB (not 50KB+)
# - Model: Haiku (not Sonnet)
# - Heartbeat: Ollama/local
```

Signs It's Working

- Context size shows 2-8KB instead of 50KB+
- Default model shows as Haiku
- Heartbeat shows Ollama/local (not API)
- Routine tasks complete without switching to Sonnet
- Daily costs drop to $0.10-0.50 range

Troubleshooting

- Context size still large → Check session initialization rules are in system prompt
- Still using Sonnet for everything → Verify config.json syntax and path
- Heartbeat errors → Make sure Ollama is running (ollama serve)
- Costs haven't dropped → Check your system prompt is being loaded

Quick Reference Checklist

Use this checklist to make sure you've completed all the steps:

SESSION INITIALIZATION	
☐	Added SESSION INITIALIZATION RULE to system prompt
MODEL ROUTING	
☐	Updated ~/.openclaw/openclaw.json with model aliases
☐	Added MODEL SELECTION RULE to system prompt
HEARTBEAT TO OLLAMA	
☐	Installed Ollama and pulled llama3.2:1b
☐	Added heartbeat config pointing to Ollama
☐	Verified Ollama is running (ollama serve)
RATE LIMITS & WORKSPACE	
☐	Added RATE LIMITS to system prompt
☐	Created SOUL.md with core principles
☐	Created USER.md with your info
☐	Verified with session_status command

The Bottom Line

No complex setup. No file management scripts. Just smart config, clear rules in your system prompt, and a free local LLM for heartbeats. The intelligence is in the prompt, not the infrastructure.

Questions? DM me @mattganzak