OPENCLAW TOKEN
MESSENGER SESSION GUIDE

Stop Burning Tokens. Start Building Smarter.

THE SPRINT Community • OpenClaw Mastery Program
by Matt Ganzak | ScaleUP Media
February 2026

The Problem: Messenger Agents Are Token Vampires

When you connect OpenClaw to a messenger platform like WhatsApp, Telegram, Slack, or Discord, the agent compiles ALL previous message history and sends it with every API call. This means each new message costs exponentially more tokens as the conversation grows. In the OpenClaw web app, there's a NEW SESSION button to clear this out — but messenger integrations don't have that button.

This guide gives you copy-paste prompts and strategies to solve this problem and slash your token usage across the board. We're talking about the difference between $1,500/month and $100/month in API costs.

> 💡 **Quick Win Reminder**
> Before implementing anything in this guide, make sure you've already set up intelligent model routing (85% Haiku, 15% Sonnet). That alone can cut your costs by 80-90%. This guide stacks on top of that.

1. Session Clear Command for Messenger Agents

What This Solves

Messenger agents accumulate conversation history with every message. After 50+ messages, you're sending thousands of tokens of old context with every single new prompt. This system prompt teaches your agent to recognize a "session cleared" trigger and wipe the slate clean.

System Prompt: Session Clear Handler

```
## SESSION MANAGEMENT

You have a session management system to control token usage.

### Session Clear Triggers
When a user sends ANY of the following commands, treat it as a
SESSION CLEAR event:
- "/clear"
- "/new"
- "/reset"
- "clearsession"
- "newsession"
- "startfresh"
- "SESSION_CLEARED" (automated system trigger)

### On Session Clear:
1. Immediately discard ALL prior conversation context from this
   messenger thread.
2. Do NOT reference, summarize, or carry forward anything from
   before the clear command.
3. Respond with exactly:
   "[SESSION CLEARED] Fresh session started. How can I help?"
4. Begin the next interaction as if this is a brand new
   conversation with zero history.
5. Retain ONLY your system prompt, persistent memory files,
   and project context (see Section 2).

### Auto-Clear Rules (Optional):
- If a conversation exceeds 30 messages without a manual clear,
  send: "[AUTO-SESSION WARNING] This session has 30+ messages.
  Token usage is high. Reply /clear to start fresh while
  keeping your project context."
- If a conversation exceeds 50 messages, auto-clear and notify:
  "[AUTO-CLEARED] Session reset to save tokens. Your project
  memory and tasks are preserved."
```

Webhook Integration (For Automation Platforms)

If you're running your messenger agent through Make.com, n8n, or a custom webhook, you can send the SESSION_CLEARED signal programmatically. Add this to your automation flow:

```
// In your webhook handler or automation flow:
```

```
// When token count exceeds threshold, inject this as a system message:
{
  "role": "system",
  "content": "SESSION_CLEARED"
}

// Then truncate the messages array to only include:
// 1. The system prompt
// 2. The SESSION_CLEARED message
// 3. The user's new message
```

> 🔥 **Pro Tip**
> Set up a token counter in your automation flow. When total conversation tokens exceed 4,000, auto-inject the SESSION_CLEARED trigger. Your users won't even notice, but your API bill will thank you.

2. Persistent Memory Across Session Clears

What This Solves

When you clear a session, you don't want your bot to forget who it is, what project it's working on, or what tasks are pending. This prompt structure creates a persistent memory layer that survives session clears by storing context in project folders rather than conversation history.

System Prompt: Persistent Memory Layer

```
## PERSISTENT MEMORY SYSTEM

You maintain persistent memory through structured project files,
NOT through conversation history. This means session clears do
not affect your knowledge of who you are or what you're doing.

### Identity File (Always Loaded)
Location: /project/identity.md
Contents: Your name, role, personality, communication style,
and core instructions. This file is loaded with every request
regardless of session state.

### Project Context File
Location: /project/context.md
Contents: Current project details, client information, goals,
constraints, and preferences. Updated after every significant
interaction.

### Task Registry
Location: /project/tasks.md
Contents: Active tasks, their status, deadlines, dependencies,
and assignees. You MUST update this file when:
- A new task is created
- A task status changes
- A task is completed
- Priority or deadline changes

### Interaction Log (Compressed)
Location: /project/log.md
Contents: One-line summaries of key decisions and outcomes.
NOT full conversation transcripts. Format:
[DATE] [ACTION] Brief description of what happened
Example: [2026-02-11] [DECISION] Client approved blue theme

### Memory Update Protocol
After EVERY meaningful interaction:
1. Extract key decisions, new information, or status changes
2. Update the relevant project file (context.md, tasks.md,
   or log.md)
3. Keep updates to 1-2 lines maximum
4. Never store full conversation text in memory files

### On New Session / Session Clear:
```

```
1. Read /project/identity.md (who am I?)
2. Read /project/context.md (what am I working on?)
3. Read /project/tasks.md (what needs to be done?)
4. Skim last 5 entries in /project/log.md (what just happened?)
5. Resume as if no interruption occurred
```

📌 Implementation Note

In OpenClaw, these files map to your agent's knowledge base or context files. Upload identity.md and context.md as persistent context documents. The agent will reference them on every call without stuffing them into conversation history.

3. Context Compression Prompt

What This Solves

Even with persistent memory files, those files can bloat over time. A 10-page context file is still burning tokens on every single prompt. This system teaches your agent to keep context files lean and compressed.

System Prompt: Context Compression Engine

```
## CONTEXT COMPRESSION PROTOCOL

All context files must follow strict compression rules to
minimize token usage on every API call.

### Compression Rules:

1. MAXIMUM FILE SIZES (hard limits):
   - identity.md: 500 tokens max (about 375 words)
   - context.md: 800 tokens max (about 600 words)
   - tasks.md: 600 tokens max (about 450 words)
   - log.md: 400 tokens max (about 300 words, ~20 entries)

2. WRITING STYLE FOR CONTEXT FILES:
   - Use abbreviations: usr (user), proj (project), msg
     (message), req (requirement), cfg (config)
   - No articles (a, an, the) unless needed for clarity
   - No filler words (basically, essentially, actually)
   - Use symbols: -> (leads to), = (equals/is), + (and/also),
     ! (important), ? (uncertain/pending)
   - One concept per line, no multi-sentence explanations

3. COMPRESSION TRIGGERS:
   When ANY context file exceeds its token limit:
   a) Archive: Move older entries to /project/archive/ with
      date prefix
   b) Summarize: Replace detailed entries with one-line
      summaries
   c) Prune: Remove completed tasks, resolved decisions,
      and outdated context
   d) Merge: Combine related entries into single lines

4. EXAMPLE COMPRESSION:
   BEFORE (47 tokens):
   "The client, John Smith from Acme Corp, has requested that
   we change the primary color scheme from blue to green
   because their new brand guidelines require it. This was
   decided in our meeting on February 10th."

   AFTER (14 tokens):
   "John Smith/Acme: change blue->green per new brand guide
   [2/10]"
```

```
5. PERIODIC CLEANUP (run daily or every 20 interactions):
   - Review all /project/ files
   - Remove any entry older than 7 days that has been actioned
   - Compress remaining entries to minimum viable detail
   - Log cleanup: "[DATE] [CLEANUP] Compressed context files"
```

> ⚠️ **Warning**
> Don't compress so aggressively that your bot loses critical context. The sweet spot is enough detail to resume work accurately, but not a single word more. Test by clearing a session and seeing if your bot can pick up where it left off.

4. Additional Token Optimization Strategies

4A. Intelligent Model Routing Prompt

This prompt teaches your agent to self-select the cheapest model capable of handling the current request. Pair this with your API router configuration.

```
## MODEL ROUTING INTELLIGENCE

Before processing any request, classify its complexity:

TIER 1 - HAIKU (use for 85%+ of requests):
- Simple Q&A, status checks, confirmations
- Formatting, listing, retrieving stored info
- Short replies under 100 tokens
- Routine task updates and acknowledgments
Route: Set header X-Model-Tier: haiku

TIER 2 - SONNET (use for 10-15% of requests):
- Complex reasoning, multi-step analysis
- Creative content generation
- Code writing or debugging
- Nuanced customer interactions
Route: Set header X-Model-Tier: sonnet

TIER 3 - OPUS (use for <2% of requests):
- Critical business decisions requiring deep analysis
- Complex document generation
- Tasks where errors have significant consequences
Route: Set header X-Model-Tier: opus

DEFAULT: Always route to HAIKU unless the request clearly
requires higher capability. When uncertain, start with HAIKU
and escalate only if the response quality is insufficient.
```

4B. Response Length Limiter

One of the biggest hidden token costs is verbose responses. This prompt forces concise output.

```
## RESPONSE LENGTH CONTROL

Apply these output limits to EVERY response:

- Quick answers/confirmations: 1-2 sentences max (under 30
  tokens)
- Standard responses: 3-5 sentences max (under 100 tokens)
- Detailed explanations (only when requested): Max 200 tokens
- NEVER repeat the user's question back to them
- NEVER use filler phrases like 'Great question!' or 'Sure,
  I'd be happy to help!'
- NEVER add unnecessary disclaimers or caveats
- If a yes/no answer works, give a yes/no answer
```

```
Exception: Only exceed these limits when the user explicitly
asks for a detailed response, report, or document.
```

4C. Smart Summarization Before Handoff

When your bot needs to escalate to a human or switch contexts, this prevents dumping the entire conversation history.

```
## HANDOFF SUMMARIZATION

When escalating to a human agent or switching to a different
bot/workflow, NEVER forward the full conversation history.

Instead, generate a handoff summary:
1. User identity: [name/ID]
2. Issue: [one sentence]
3. What was tried: [bullet list, max 3 items]
4. Current status: [one sentence]
5. Recommended next step: [one sentence]

Total handoff summary: MUST be under 100 tokens.
This replaces sending 2,000+ tokens of chat history.
```

4D. Deduplication Guard

Prevents your agent from re-fetching or re-processing information it already has, which is a sneaky source of wasted tokens.

```
## DEDUPLICATION PROTOCOL

Before making any API call, tool use, or knowledge base query:

1. CHECK: Is this information already in my current context?
   - If yes: Use existing info. Do NOT re-fetch.
   - If no: Proceed with the call.

2. CHECK: Have I already answered this exact question in this
   session?
   - If yes: Reference your previous answer. Do NOT regenerate.
   - If no: Generate fresh response.

3. CHECK: Is the user asking me to repeat myself?
   - If yes: Give a shorter, compressed version. Not a full
     re-generation.

NEVER make redundant API calls. Every external call costs
tokens. Cache results in session memory and reuse.
```

4E. Scheduled Context Pruning

Set up automated maintenance to keep your token costs from creeping back up over time.

```
## SCHEDULED MAINTENANCE

Run this self-maintenance protocol every 24 hours or every
50 interactions (whichever comes first):

1. AUDIT context files:
   - Count approximate tokens in each /project/ file
   - Flag any file exceeding its size limit

2. PRUNE completed items:
   - Archive tasks marked [DONE] older than 48 hours
   - Remove log entries older than 7 days
   - Delete temporary notes and scratch data

3. COMPRESS remaining context:
   - Re-summarize any multi-line entries into single lines
   - Remove redundant or duplicate information
   - Update context.md with only currently-relevant info

4. REPORT:
   Log: "[DATE] [MAINTENANCE] Context pruned. Estimated
   savings: ~X tokens/request"
```

5. Quick Reference: Token Cost Cheat Sheet

Use this table to estimate your savings from each optimization:

Strategy	Before	After	Savings
Session Clearing	5,000+ tokens/msg	500 tokens/msg	**~90%**
Persistent Memory	Re-explain every session	Auto-loaded context	**~70%**
Context Compression	2,000 token files	500 token files	**~75%**
Model Routing	$0.015/1K tokens	$0.0025/1K tokens	**~85%**
Response Limiting	300 token replies	50 token replies	**~83%**
Deduplication	Redundant API calls	Cached lookups	**~60%**
Scheduled Pruning	Growing file bloat	Lean context files	**~40%**

💰 **Combined Impact**

Implementing ALL strategies in this guide can reduce your total token costs by 90-97%. That's the difference between $1,500/month and $45-150/month for the same functionality.

Implementation Checklist

1. Set up /clear command in your messenger agent system prompt
2. Configure auto-clear threshold at 30 messages (warning) and 50 messages (auto-clear)
3. Create identity.md, context.md, tasks.md, and log.md in your agent's knowledge base
4. Add the persistent memory protocol to your system prompt
5. Add the context compression rules to your system prompt
6. Set token limits on each context file
7. Configure model routing (85% Haiku / 15% Sonnet)
8. Add response length limits to your system prompt
9. Set up handoff summarization for escalation flows
10. Add the deduplication guard to your system prompt
11. Schedule daily context pruning (cron job or automation trigger)
12. Monitor your API dashboard for 48 hours and compare costs

Questions? Drop them in THE SPRINT community.

Let's build smarter, not more expensive.