Slash Your AI Costs With a Token Audit

February 5, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Your AI API bill is bloated because you're sending tokens you don't need to send. Run the 15-minute audit below, drop in the 50-line Python logger, and most people cut 40-60% on the first pass. One common culprit: a 2,000-token system prompt costs $180/month at Sonnet rates across 1,000 daily calls.

Related: Cost to Run LLMs Locally · OpenClaw Token Optimization · Planning Tool

Your API bill is too high. You already know this. What you probably don't know is where the waste is hiding, and it's almost never the obvious stuff.

This guide walks you through finding the leaks and plugging them. Fifteen minutes, a Python logger, and some configuration changes.

Step 1: Check Your Dashboard (5 Minutes)

Before touching any code, look at what you're actually spending.

Pull up your usage page: Anthropic | OpenAI | DeepSeek

Look at the last 7 days and answer three questions:

1. **Which model is eating the most money?** If 80% of your spend is on Opus but only 20% of your tasks need it, that's your single biggest fix.
2. **What's your average input size?** If it's over 2,000 tokens and you're not using prompt caching, stop reading and enable caching right now. Come back after.
3. **Are there any spending spikes?** A sudden jump usually means a runaway loop, an agent retrying, or conversation history that ballooned overnight.

This alone tells most people where 60% of the problem lives.

Step 2: Find the Four Hidden Drains

These are the costs that don't show up in your prompt but absolutely show up on your bill.

System Prompts: A Tax on Every Request

Your system prompt gets re-sent as input tokens on every API call. It's not "remembered" between calls.

A 2,000-token system prompt across 1,000 calls/day:

- **2M extra input tokens/day**
- At Sonnet rates ($3/MTok): **$6/day = $180/month**
- At Opus 4 rates ($15/MTok): **$30/day = $900/month**

That's before your actual questions even start.

Cut your system prompt to the minimum that works. A 500-token prompt instead of 2,000 saves $135/month at Sonnet rates. Or use prompt caching to drop that cost by 90%.

Conversation History: The Snowball

Multi-turn conversations resend every previous message as input on every call. Turn 1 sends your message. Turn 10 sends all 10 messages. Turn 20 sends all 20.

A 20-turn conversation with 500 tokens per turn doesn't cost 10,000 tokens. It costs ~105,000 tokens (the triangular sum). That's 10x what most people expect.

Summarize older turns instead of sending full text. Or just start fresh conversations more often.

Tool Definitions: Hidden Padding

If you're using function calling, every API call includes hidden tokens for tool infrastructure — even when no tools fire.

Anthropic documents this:

Tool Configuration	Hidden Tokens Per Request
`auto` or `none` tool choice	346 tokens
`any` or specific tool choice	313 tokens
Each tool definition (avg)	~150 tokens

Tool Configuration	Hidden Tokens Per Request
Bash tool	245 tokens
Text editor tool	700 tokens
Computer use	466-499 tokens + 735 per tool

An agent with 5 tools adds ~1,100 tokens to every request (346 base + 750 for definitions). Invisible in your prompt, very real in your bill.

OpenAI's function calling has similar overhead — tool schemas are serialized into the system prompt.

Strip out tools you don't need on each call. A request that just needs a text response doesn't need 5 tool definitions attached.

OpenAI Reasoning Tokens: The Invisible Output Tax

This one is specific to OpenAI's o-series models (o1, o3, o3-mini, o4-mini) and it's easy to miss entirely.

These models generate internal "reasoning tokens," a chain-of-thought used to work through the problem. You pay for them as output tokens (the most expensive type), but they never appear in the response. You can't cap them either.

A request that returns 500 visible tokens might consume 2,000+ total output tokens. At o1 rates ($60/MTok output), that's the difference between $0.03 and $0.12 per request.

Check `completion_tokens_details.reasoning_tokens` in every o-series response. The `reasoning.effort` parameter (low/medium/high) gives you partial control. Set it to `low` for simple tasks.

Image Tokens: Add Up Faster Than You Think

Provider	1024x1024 Image	Cost at Mid-Tier
Claude	~1,398 tokens	$0.004 (Sonnet)
GPT-4o	~765 tokens	$0.002
Gemini	~1,290 tokens	$0.0004 (2.0 Flash)

Tiny per image. But 1,000 screenshots/day adds $4/day on Claude before any text.

Step 3: Add the Logger (10 Minutes)

Stop guessing. Drop this into your codebase and log every call with its actual cost.

```python
import json
from datetime import datetime, timezone

# Pricing per million tokens (update when prices change)
PRICING = {
    # Anthropic
    "claude-opus-4": {"input": 15.0, "output": 75.0,
                      "cache_read": 1.50, "cache_write": 18.75},
    "claude-sonnet-4": {"input": 3.0, "output": 15.0,
                        "cache_read": 0.30, "cache_write": 3.75},
    "claude-haiku-3.5": {"input": 0.80, "output": 4.0,
                         "cache_read": 0.08, "cache_write": 1.00},
    # OpenAI
    "gpt-4o": {"input": 2.5, "output": 10.0, "cached_input": 1.25},
    "gpt-4o-mini": {"input": 0.15, "output": 0.60, "cached_input": 0.075},
    "o3-mini": {"input": 1.10, "output": 4.40, "cached_input": 0.55},
    # DeepSeek
    "deepseek-chat": {"input": 0.27, "output": 1.10, "cached_input": 0.07},
    "deepseek-reasoner": {"input": 0.55, "output": 2.19, "cached_input": 0.14},
}

def log_anthropic(response, model, label=""):
    """Log an Anthropic API response. Returns cost in USD."""
    u = response.usage
    p = PRICING.get(model, {})
    cache_write = getattr(u, "cache_creation_input_tokens", 0)
    cache_read = getattr(u, "cache_read_input_tokens", 0)

    cost = (
        u.input_tokens * p.get("input", 0)
        + cache_write * p.get("cache_write", 0)
        + cache_read * p.get("cache_read", 0)
        + u.output_tokens * p.get("output", 0)
    ) / 1_000_000

    entry = {
        "ts": datetime.now(timezone.utc).isoformat(),
        "provider": "anthropic", "model": model, "label": label,
        "input": u.input_tokens, "cache_write": cache_write,
        "cache_read": cache_read, "output": u.output_tokens,
        "cost_usd": round(cost, 6),
    }
    _append(entry)
    return entry

def log_openai(response, model, label=""):
    """Log an OpenAI API response. Returns cost in USD."""
    u = response.usage
    p = PRICING.get(model, {})
    cached = 0
    reasoning = 0
    if u.prompt_tokens_details:
        cached = getattr(u.prompt_tokens_details, "cached_tokens", 0)
    if u.completion_tokens_details:
        reasoning = getattr(u.completion_tokens_details,
                            "reasoning_tokens", 0)

    uncached_input = u.prompt_tokens - cached
    cost = (
        uncached_input * p.get("input", 0)
        + cached * p.get("cached_input", p.get("input", 0))
        + u.completion_tokens * p.get("output", 0)
    ) / 1_000_000

    entry = {
        "ts": datetime.now(timezone.utc).isoformat(),
        "provider": "openai", "model": model, "label": label,
        "input": u.prompt_tokens, "cached": cached,
        "output": u.completion_tokens, "reasoning": reasoning,
        "cost_usd": round(cost, 6),
    }
    _append(entry)
    return entry

LOG_FILE = "token_log.jsonl"

def _append(entry):
    with open(LOG_FILE, "a") as f:
        f.write(json.dumps(entry) + "\n")

def daily_summary():
    """Print cost summary grouped by model."""
    totals = {}
    with open(LOG_FILE) as f:
        for line in f:
            e = json.loads(line)
            model = e["model"]
            totals.setdefault(model, {"calls": 0, "cost": 0.0})
            totals[model]["calls"] += 1
            totals[model]["cost"] += e["cost_usd"]
    print(f"{'Model':<25} {'Calls':>6} {'Cost':>10}")
    print("-" * 43)
    grand = 0
    for model, data in sorted(totals.items(), key=lambda x: -x[1]["cost"]):
        print(f"{model:<25} {data['calls']:>6} ${data['cost']:>9.4f}")
        grand += data["cost"]
    print("-" * 43)
    print(f"{'TOTAL':<25} {'':>6} ${grand:>9.4f}")
```

How to Use It

```
import anthropic

client = anthropic.Anthropic()
response = client.messages.create(
    model="claude-sonnet-4",
    max_tokens=1024,
    messages=[{"role": "user", "content": "Explain quicksort"}],
)
log_anthropic(response, "claude-sonnet-4", label="quicksort-explainer")
```

```
from openai import OpenAI

client = OpenAI()
response = client.chat.completions.create(
    model="gpt-4o",
    messages=[{"role": "user", "content": "Explain quicksort"}],
)
log_openai(response, "gpt-4o", label="quicksort-explainer")
```

Run `daily_summary()` after a week. The output shows exactly where your money went.

Step 4: Enable Prompt Caching

If you’re sending the same system prompt, tool definitions, or context on repeated calls, caching drops those tokens to 10% of base cost. Nothing else in this guide saves you as much for as little work.

Anthropic

Mark static content with `cache_control`:

```
response = client.messages.create(
    model="claude-sonnet-4",
    max_tokens=1024,
    system=[
        {
            "type": "text",
            "text": "Your long system prompt here...",
            "cache_control": {"type": "ephemeral"},
        }
    ],
    messages=[{"role": "user", "content": "Your question"}],
)
```

First call writes the cache at 1.25x input rate. Every call within 5 minutes reads at 0.1x – a 90% discount.

Real numbers with a 10,000-token system prompt at Sonnet rates:

	Without Caching	**With Caching (after first call)**
System prompt cost per call	$0.030	$0.003
1,000 calls/day	$30.00	$3.00 + $0.0375 cache write
Monthly	**$900**	**$91**

OpenAI

Automatic for prompts over 1,024 tokens. No code changes. Cached tokens bill at 50% (less aggressive than Anthropic's 90%, but free to enable).

DeepSeek

Also automatic. Repeated prefixes hit cache pricing server-side. V3 drops from $0.27 to $0.07/MTok on cache hits – 74% off without any configuration.

Step 5: Route Tasks to Cheaper Models

Running everything through one model is the most common waste pattern. Different tasks need different horsepower.

Task Type	Model	Cost per 1K Calls (avg 500-token response)
Classification, routing	DeepSeek V3 (cached)	$0.04
Simple Q&A, formatting	Haiku 3.5	$2.40
Standard generation	Sonnet 4	$9.00
Complex reasoning	Opus 4.5/4.6	$14.50
Maximum capability	Opus 4	$45.00

The cheapest frontier-class token (DeepSeek V3 cached) costs $0.07/MTok. The most expensive (Opus 4 output) costs $75/MTok. That's a 1,071x spread. If you're using one model for everything, you're overpaying for simple tasks or underperforming on hard ones.

Step 6: Read the Usage Object

Every API response tells you exactly what you were charged. Here's how to read it.

Claude (Anthropic)

```
{
  "usage": {
    "input_tokens": 50,
    "cache_creation_input_tokens": 1500,
    "cache_read_input_tokens": 18000,
    "output_tokens": 393
  }
}
```

The gotcha: `input_tokens` is NOT your total input. It's only the tokens after the last cache breakpoint. Total input = `input_tokens + cache_creation_input_tokens + cache_read_input_tokens`. If you're caching and only tracking `input_tokens`, you're undercounting.

OpenAI

```
{
  "usage": {
    "prompt_tokens": 1250,
    "completion_tokens": 500,
    "total_tokens": 1750,
    "prompt_tokens_details": {
      "cached_tokens": 1000
    },
    "completion_tokens_details": {
      "reasoning_tokens": 200
    }
  }
}
```

Watch `reasoning_tokens` on o-series models. That's invisible output you're paying full price for. For streaming, add `stream_options: {"include_usage": true}` or you won't get the usage object at all.

DeepSeek

Follows OpenAI's format. Caching is server-side and automatic – check the dashboard for cache hit rates.

Current Pricing Reference

Prices as of early 2026, per million tokens.

Claude (Anthropic)

Model	Input	Output	Cache Read	Cache Write (5min)
Opus 4	$15.00	$75.00	$1.50	$18.75
Opus 4.5/4.6	$5.00	$25.00	$0.50	$6.25
Sonnet 4/4.5	$3.00	$15.00	$0.30	$3.75
Haiku 3.5	$0.80	$4.00	$0.08	$1.00

Model	Input	Output	Cache Read	Cache Write (5min)
Haiku 4.5	$1.00	$5.00	$0.10	$1.25

Batch API: 50% off everything. Long context (>200K tokens): input doubles, output goes to 1.5x.

OpenAI

Model	Input	Cached Input	Output
GPT-4o	$2.50	$1.25	$10.00
GPT-4o-mini	$0.15	$0.075	$0.60
GPT-4.1	$2.00	$0.50	$8.00
o3	$2.00	$0.50	$8.00
o3-mini	$1.10	$0.55	$4.40
o1	$15.00	$7.50	$60.00

Batch API: 50% off all models, results within 24 hours.

DeepSeek

Model	Input (Cache Hit)	Input (Cache Miss)	Output
DeepSeek-V3	$0.07	$0.27	$1.10
DeepSeek-R1	$0.14	$0.55	$2.19

Off-peak discounts (16:30-00:30 GMT): up to 75% off R1, 50% off V3.

Google Gemini

Model	Input	Output	Context Window
Gemini 2.5 Pro	$1.25	$10.00	1M
Gemini 2.5 Flash	$0.30	$2.50	1M
Gemini 2.0 Flash	$0.10	$0.40	1M

When Local Models Win on Cost

At high enough volume, running your own hardware wipes out per-token costs.

Daily Volume	API Cost (Sonnet)	API Cost (DeepSeek V3)	Local Cost (RTX 3090)
100K tokens	$0.90	$0.07	~$0.15 (electricity)
1M tokens	$9.00	$0.68	~$0.15
10M tokens	$90.00	$6.80	~$0.30
100M tokens	$900.00	$68.00	~$0.50

At 1M tokens/day, DeepSeek is cheaper than your own hardware. At 10M/day, local starts winning. At 100M/day, local is 136x cheaper than Sonnet.

Break-even for a $750 used RTX 3090 setup: about 3 months at 1M tokens/day vs Sonnet, or 12-18 months at 10M/day vs DeepSeek.

The tradeoff is quality. Local models (Qwen 3 32B, Llama 3.3 70B) are good but not Sonnet-good for complex tasks. For classification, formatting, extraction, and embeddings, they match API quality at zero marginal cost. Full hardware and electricity math in our cost to run LLMs locally guide.

Quick-Reference: Cuts Ranked by Effort

What to Do	Effort	Typical Savings
Enable prompt caching	Low (one field)	50-90% on repeated content
Use batch API	Low (change endpoint)	50% on async work
Trim system prompt	Medium (rewrite it)	20-40% on input costs
Route tasks to cheaper models	Medium (add routing)	40-70% overall
Truncate conversation history	Medium (add summarization)	30-60% on multi-turn
Remove unused tool definitions	Low (delete lines)	5-15% on tool-heavy calls
Set reasoning effort to low	Low (one parameter)	30-50% on o-series output
Use DeepSeek for simple tasks	Low (swap model)	80-95% vs Claude/OpenAI

What to do right now

The difference between a $500/month AI bill and a $50/month bill usually isn't using less AI. It's not paying for tokens you didn't need to send.

Check your dashboard. Add the logger. Enable caching. Route cheap tasks to cheap models. Most people who do all four cut their bill in half within a month.

Source: https://insiderllm.com/guides/token-audit-guide/