DealNews Engineering

What We Learned About Prompt Caching in Production

A talk on a small structural change
that cut our AI inference costs by 97%

dealnews.com

@DealNews

Who We Are and What We Built

DealNews

Affiliate-driven media company. We publish deals and buying guides. Content velocity and quality directly drive revenue.

The AI Pipeline

We built an AI pipeline to help our writers generate and vet content fields — titles, descriptions, categories, and metadata. It also validates that products and images don't contain banned items.

Humans First

Writers retain final editorial authority. The AI suggests, they decide. This matters for quality and for trust.

Where We Started

AI inference gets expensive fast

$0.18

per offer

starting cost

- One model handling all tasks
- Running at volume, costs compound fast
- We needed to understand where the money was going
- Prompt caching turned out to be the biggest single lever

How Prompt Caching Works

Call 1 — full price

System Prompt (static context)

Paid in full — written to cache

User Prompt (variable data)

Paid in full

Call 2+ — cache hit

System Prompt (from cache)

~75-90% discount on these tokens

User Prompt (variable data)

Paid in full

The cache only works on a stable prefix. If the first thing in your prompt changes every call, the cache never hits.

The Wrong Way — Variable data at the top kills caching

```
curl https://api.anthropic.com/v1/messages \
  -H "x-api-key: $ANTHROPIC_API_KEY" \
  -H "anthropic-version: 2023-06-01" \
  -d '{
    "model": "claude-sonnet-4-6",
    "messages": [
      {
        "role": "user",
        "content": "Process this offer: OFFER_TITLE_HERE\n\n
          You are an expert content editor for an e-commerce site.
          Always follow AP style. Be concise and accurate.
          Category list: Electronics, Clothing, Home & Garden..."
      }
    ]
```

Variable offer data is the very first token — the prefix never matches, cache never hits, full price every call

Why Variable Data at the Top Breaks the Cache

Call 1	`"Process this offer: Sony TV…"`	`…static instructions…`
	CACHE MISS	
Call 2	`"Process this offer: Nike Shoes…"`	`…static instructions…`
	CACHE MISS	
Call 3	`"Process this offer: Instant Pot…"`	`…static instructions…`
	CACHE MISS	

Every call looks different from the start. The provider has nothing stable to cache against.

The Right Way — Static context first, variable data last

```
curl https://api.anthropic.com/v1/messages \
  -H "x-api-key: $ANTHROPIC_API_KEY" \
  -H "anthropic-version: 2023-06-01" \
  -H "anthropic-beta: prompt-caching-2024-07-31" \
  -d '{
    "model": "claude-sonnet-4-6",
    "system": [{ "type": "text", "text": "You are an expert content
      editor for an e-commerce site. Always follow AP style. Be
      concise and accurate. Category list: Electronics, Clothing...",
      "cache_control": { "type": "ephemeral" } }],
    "messages": [{
      "role": "user",
      "content": "Process this offer: OFFER_TITLE_HERE"
    }]
```

System prompt is now stable across every call — prefix matches, cache hits, big savings

Now the Cache Has Something to Work With

Call 1	"You are an expert editor…"	"Process this offer: Sony TV"
CACHE WRITE		
Call 2	"You are an expert editor…"	"Process this offer: Nike Shoes"
CACHE HIT		
Call 3	"You are an expert editor…"	"Process this offer: Instant Pot"
CACHE HIT		

The prefix is identical every time. Call 1 pays to write the cache. Every call after reads from it at a fraction of the cost.

Real Numbers From Production

Claude Sonnet 4.6 — actual log entry

```
input_tokens:          1,519
cached_tokens:         1,473
cache_write_tokens:    43
output_tokens:         117
cost:                  $0.00237
```

97%
Cache Hit Rate

$0.0063
Without Caching

$0.0024
With Caching

63%
Savings Per Call

1,473 of 1,519 input tokens served from cache. Only 43 new tokens written. Output tokens always billed at full price.

Gemini 3.1 Flash-Lite — Cold Cache vs Warm Cache

Two calls, 26 seconds apart, same prompt

Cold Cache — 05:29:00

input_tokens:	10,396
cached_tokens:	0
output_tokens:	30
cost:	$0.002644

Warm Cache — 05:29:26

input_tokens:	10,394
cached_tokens:	8,164
output_tokens:	24
cost:	$0.000798

78% cache hit rate on the warm call | 70% cost reduction | $0.002644 → $0.000798

Where We Are

94.5% cost reduction from where we started

$0.00994

per offer

current cost

- Two models: Sonnet 4.6 for copy, Gemini Flash-Lite for everything else
- Prompt caching in place across both providers
- Sonnet cache hit rate: 97% in production logs
- Gemini cache hit rate: 78% once warm

Provider Differences Worth Knowing

Anthropic

Claude Sonnet 4.6

Explicit

- Must opt in with cache_control parameter
- You control what gets cached and TTL
- Charged for cache writes
- Cache reads: ~90% cheaper than full price
- 97% cache hit rate in our production logs
- Minimum prompt size: 1,024 tokens

Google

Gemini 3.1 Flash-Lite

Automatic

- Prefix caching is automatic — no opt-in needed
- No cache write charges
- Cache reads: ~90% cheaper than full price
- 78% cache hit rate once warm in our logs
- First call after cold cache pays full price
- Minimum prompt size: 4,096 tokens

OpenAI

GPT-5.4 mini

Automatic

- Prefix caching is automatic — no opt-in needed
- No cache write charges
- Cache reads: ~50% cheaper than full price
- Get your prompt structure right and it just works
- No special parameters required
- Minimum prompt size: 1,024 tokens

Three Things to Take Home

1

Prompt structure matters as much as model selection

Variable data at the top of your prompt kills caching. Static context in the system prompt, variable data in the user prompt. That is it.

2

Know your provider's caching model

OpenAI and Google do it automatically. Anthropic requires explicit cache_control parameters and charges for cache writes. Both are worth using.

3

The change is smaller than you think

We updated a handful of prompts in an afternoon. No refactor. The content did not change, just where it lived. Cost dropped 97%.