Prompt Engineering / Prompt Optimization

Reduce prompt token count while preserving or improving performance on key metrics.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Cost Reduction, Latency Optimization, Context Preservation
Updated: May 2026
Why This Prompt Exists
Long prompts cost more, run slower, and leave less room for context. Most prompts have 30-50% waste.

You get:

- paying for tokens you don’t need (every API call adds up)
- slower responses from verbose prompts
- context window filled with instructions instead of useful information
- no systematic way to shorten prompts without breaking them
- prompts that grow over time (never edited, only added to)

But token waste is identifiable:

- redundant phrasing: “please kindly summarize” → “summarize”
- verbose examples: 3 similar examples → 1 representative
- unnecessary framing: “I want you to act as…” → often deletable
- implicit instructions: model already knows to do this
- formatting waste: extra line breaks, indentation, comments
- repeated constraints: same instruction stated multiple ways

Without optimization, you waste money and speed.

This prompt compresses prompts while preserving performance.

The Prompt

```
Assume the role of a token efficiency engineer who compresses prompts.

Your task is to reduce prompt token count while maintaining
performance.

Generate:

1. ORIGINAL PROMPT STATISTICS
   - Character count
   - Estimated token count
   - Number of examples

2. WASTE ANALYSIS

| Category | Original | Suggested Replacement | Tokens saved |
|----------|----------|-----------------------|--------------|
| Redundant phrasing | [text] | [shorter text] | [N] |
| Verbose examples | [example] | [condensed example] | [N] |
| Unnecessary framing | [text] | [delete] | [N] |
| Formatting waste | [spaces/breaks] | [standardized] | [N] |

3. OPTIMIZATION STRATEGY
   - Remove polite fluff ("please", "kindly", "I would like you to")
   - Shorten role descriptions ("act as a helpful assistant" → "you
are a helpful assistant")
```

```
   - Condense examples (remove redundant parts)
   - Merge duplicate instructions
   - Use symbols instead of words ("→" instead of "therefore")

4. COMPRESSED PROMPT
   - Full compressed version

5. PERFORMANCE PRESERVATION CHECK
   - Does the compressed prompt produce the same output on test
inputs? (Yes/No — test required)
   - Which test inputs should be used to verify?

6. COMPRESSION RESULTS
   - Original token count: [X]
   - Compressed token count: [Y]
   - Tokens saved: [Z] ([percentage]%)
   - Estimated cost savings per 1K calls: [$]

7. DEPLOYMENT RECOMMENDATION
   - Safe to deploy (passes performance tests)
   - Test more before deploying (unclear if performance preserved)
   - Keep original (compression broke something important)

INPUTS:

Original prompt:
[PASTE THE PROMPT]

Performance metrics to preserve:
```

```
[E.G., "Must maintain >90% accuracy on test set"]

Test inputs (to verify compression):
[PASTE 5-10 REPRESENTATIVE INPUTS]

Model:
[GPT-4 / CLAUDE / GEMINI]

Compression aggressiveness:
[CONSERVATIVE (only obvious waste) / AGGRESSIVE (shorten everything) /
EXTREME (abbreviations)]

RULES:
- Test compressed prompt before deploying (some compressions break
prompts)
- Prioritize removing redundant instructions over condensing examples
- Flag if compression would require removing necessary examples
- Note that some prompts are already optimal (compression not always
possible)
- For extreme compression, consider using a smaller model instead
```

How To Use It

- Run this on any prompt that will be used at scale — token savings add up fast.
- Test the compressed prompt on your full test set before deploying.
- Don’t sacrifice performance for token savings (the trade-off isn’t worth it).
- Re-compress periodically — prompts tend to grow over time.
- For high-volume prompts, even 10% token savings is meaningful.

Example Input

Original prompt:
“Hello, I would like you to please act as a helpful customer service assistant. Your job is to read the following customer message and then determine if the customer is angry, neutral, or happy. Please only respond with one word: ANGRY, NEUTRAL, or HAPPY. Do not add any other text. Do not explain your reasoning. Just the one word. Here is the customer message: {{message}}”

Performance metrics to preserve:
“Must maintain same classification accuracy”

Test inputs:
“5 customer messages of varying sentiment”

Why It Works
Most prompts are written for clarity first — which is good — but never compressed for production — which is wasteful.

This framework improves outcomes by forcing:

- waste identification (what can be removed?)
- optimization strategy (systematic shortening)
- performance preservation check (did we break it?)
- token savings calculation (quantified benefit)
- deployment recommendation (safe or not?)

Great token efficiency optimization doesn’t sacrifice quality — it removes waste while preserving everything that matters.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, debugging frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also Failure-Driven Refiner