Prompt Engineering / Chain-of-Thought

Find the shortest valid reasoning chain for a problem — reduces tokens while maintaining accuracy.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Cost Optimization, Latency Reduction, Efficient Reasoning
Updated: May 2026
Why This Prompt Exists
Chain-of-thought is powerful but expensive. Many reasoning chains are longer than necessary — adding tokens, latency, and cost without improving accuracy.

You get:

- reasoning chains with redundant or unnecessary steps
- paying for tokens that don’t improve the answer
- slower responses from overly verbose reasoning
- context window filled with fluff instead of useful reasoning
- no systematic way to find the minimal effective chain

But minimal chains have properties:

- necessary steps: cannot be removed without breaking correctness
- redundant steps: can be removed with no impact
- compressible steps: can be stated more concisely
- parallelizable steps: can be reordered or combined
- implicit steps: don’t need to be stated (model can infer)

Without optimization, you waste tokens.

This prompt finds the minimal reasoning chain that still produces correct answers.

The Prompt

```
Assume the role of a reasoning efficiency engineer who compresses
reasoning chains.

Your task is to find the minimal set of steps needed to solve a
problem correctly.

Generate:

1. ORIGINAL REASONING CHAIN
   - All steps as originally written

2. STEP NECESSITY ANALYSIS

| Step | Content | Necessary? (Yes/No) | Why |
|------|---------|---------------------|-----|
| 1 | [step] | Yes/No | [reason] |
| 2 | [step] | Yes/No | [reason] |
| ... | ... | ... | ... |

3. REDUNDANT STEPS (can be removed)
   - List with rationale

4. COMPRESSIBLE STEPS (can be shortened)
   - Original vs. compressed version
```

5. MINIMAL REASONING CHAIN
   - Only necessary steps, each stated concisely

6. VALIDATION
   - Does the minimal chain still produce the correct answer? (Yes/No – test required)
   - Token savings: [X] tokens saved ([Y]% reduction)
   - Accuracy risk: (Low / Medium / High)

7. DEPLOYMENT RECOMMENDATION
   - Use minimal chain for production (if low risk)
   - Keep original for complex cases
   - Test on edge cases before deploying

INPUTS:

Original reasoning chain (full):
[PASTE THE REASONING WITH STEPS]

Problem type:
[ARITHMETIC / LOGIC / COMMON SENSE / PLANNING / OTHER]

Correct answer (for validation):
[THE KNOWN CORRECT ANSWER]

Model:
[GPT-4 / CLAUDE / GEMINI]

RULES:

```
- A step is necessary if removing it changes the answer or breaks the reasoning
- A step is redundant if the model can infer it without explicit statement
- Compressed steps must preserve meaning (no shortcuts that change the logic)
- Test the minimal chain before deploying (some necessary steps are not obvious)
- Flag if minimal chain reduces accuracy for edge cases
```

How To Use It

- Run this on reasoning chains that will be used at high volume — token savings add up fast.
- Test the minimal chain on at least 10 edge cases before deploying.
- Don’t over-optimize for low-stakes tasks (saving tokens isn’t worth accuracy risk).
- Save the minimal chain as a template for similar problems.
- Re-optimize when you switch to a new model (different models need different steps).

Example Input

Original reasoning chain:
“Step 1: The problem asks for 15% of 280. Step 2: 15% means 15 per 100. Step 3: So 15% of 280 equals (15/100) × 280. Step 4: 15/100 simplifies to 0.15. Step 5: 0.15 × 280 = (0.15 × 200) + (0.15 × 80). Step 6: 0.15 × 200 = 30. Step 7: 0.15 × 80 = 12. Step 8: 30 + 12 = 42. Step 9: Therefore, 15% of 280 is 42.”

Problem type:
“ARITHMETIC”

Correct answer:

“42”

Why It Works
Most reasoning chains are written for clarity — which is good for learning — but inefficient for production.

This framework improves outcomes by forcing:

- step necessity analysis (what actually matters)
- redundancy identification (what can be removed)
- step compression (what can be shortened)
- validation testing (does it still work?)
- token savings calculation (quantified benefit)

Great minimal step optimization doesn’t sacrifice accuracy — it removes waste while preserving correctness.

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 Why This Prompt Exists