Prompt Engineering / Chain-of-Thought

Have the model verify its own reasoning by restating, checking assumptions, and testing edge cases.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Self-Correction, Quality Assurance, High-Stakes Reasoning
Updated: May 2026
Why This Prompt Exists
Models make mistakes. But they can also catch their own mistakes — if you ask them to verify.

You get:

- answers that seem right but are wrong (because no one checked)
- reasoning steps that contain hidden errors (never caught)
- assumptions that are false (but never questioned)
- edge cases that break the reasoning (never considered)
- missed opportunities for self-correction

But verification loops catch errors:

- restatement: model restates the problem in its own words (checks understanding)
- assumption check: model lists and verifies each assumption
- step verification: model checks each reasoning step for errors
- edge case testing: model tests the answer on edge cases
- alternative path: model solves using different method to verify

Without verification, errors survive.

This prompt creates a self-verification loop that catches mistakes before final answer.

The Prompt

```
Assume the role of a quality assurance engineer who verifies
reasoning.

Your task is to create a verification loop that checks a model's own
reasoning.

Generate:

1. INITIAL REASONING
   - The model's first attempt at solving the problem

2. VERIFICATION LOOP STRUCTURE
   - Step 1: Restate the problem in your own words (confirm
understanding)
   - Step 2: List all assumptions (explicitly)
   - Step 3: Verify each assumption (is it justified?)
   - Step 4: Check each reasoning step for errors (calculation, logic,
missing cases)
   - Step 5: Test answer on edge cases
   - Step 6: Solve using different method (if possible)
   - Step 7: Compare answers

3. VERIFICATION EXECUTION
   - Run the verification loop on the initial reasoning
   - Show each step's findings
```

4. ERRORS FOUND (if any)
   - Type of error (assumption, calculation, logic, missing case)
   - Location
   - Correction

5. FINAL VERIFIED ANSWER
   - The answer after verification (may be same as initial, or corrected)

6. CONFIDENCE SCORE (after verification)
   - 1-10, with rationale

7. VERIFICATION PROMPT (ready to use)
   - A copy-paste prompt that adds verification to any reasoning task

INPUTS:

Initial reasoning and answer:
[PASTE THE MODEL'S FIRST ATTEMPT]

Problem statement:
[PASTE THE ORIGINAL PROBLEM]

Model:
[GPT-4 / CLAUDE / GEMINI]

Verification depth:
[LIGHT (key steps only) / STANDARD / EXHAUSTIVE (all assumptions)]

```
RULES:
- Verification must be separate from initial reasoning (don't rely on first attempt)
- If verification finds an error, the model must correct it before final answer
- Flag when verification passes but answer is still wrong (limitation of self-verification)
- Use exhaustive verification for high-stakes problems
- Save verification traces to learn what errors the model commonly makes
```

How To Use It

- Use this for any problem where wrong answers have high cost.
- Don’t rely on self-verification alone — it catches some errors but not all.
- Run verification on a sample of outputs to estimate error rate.
- Use the verification prompt as a wrapper around existing reasoning prompts.
- Save verification failures to improve your prompts (errors reveal prompt weaknesses).

Example Input

Initial reasoning and answer:
“Step 1: The store has 25% off. Step 2: The item is $100. Step 3: 25% of 100 is $25. Step 4: So final price is $75. Answer: $75.”

Problem statement:
“The store has 25% off all items. You also have a $20 coupon that cannot be combined with other offers. Which discount should you use, and what is the final price of a $100 item?”

Why It Works
Most models produce answers without checking their own work — and humans rarely

double-check thoroughly.

This framework improves outcomes by forcing:

- problem restatement (checks understanding)
- assumption verification (catches hidden false assumptions)
- step-by-step checking (finds calculation or logic errors)
- edge case testing (finds where answer breaks)
- alternative method verification (cross-checks with different approach)

Great verification loops don’t eliminate errors — they catch most of them before final answer.

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