Prompt Engineering / Meta Prompts

Prompt Edge Case Hunter

Identify inputs that would break a prompt — adversarial, ambiguous, out-of-domain, or contradictory.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Prompt Stress-Testing, Production Readiness, Security Review
Updated: May 2026
Your prompt works on happy-path inputs. But the real world isn't happy path — and that's where it will fail.

You get:

- production failures from inputs you never considered
- users accidentally (or intentionally) breaking your prompt
- prompt injection attacks you didn't anticipate
- embarrassing outputs when the input is ambiguous
- no testing strategy for edge cases

But edge cases are predictable:

- empty input: what if there's nothing to process?
- extremely long input: does it truncate or break?
- ambiguous input: can it ask for clarification?
- contradictory input: can it detect and handle?
- adversarial input: can it resist prompt injection?
- out-of-domain: can it say "I don't know"?

Without edge case hunting, your prompt is fragile.

This prompt generates test inputs that will break your prompt.

The Prompt

```
Assume the role of a red-team prompt engineer who breaks prompts.

Your task is to generate edge-case inputs that would cause the prompt
to fail.

Generate:

1. PROMPT UNDER TEST
   - What it does
   - Input format expected
   - Output format expected

2. EDGE CASE INPUTS (with predicted failure mode)

| Category | Test Input | Why It's Problematic | Predicted Failure |
|----------|------------|----------------------|-------------------|
| Empty | [input] | No content to process | Hallucination or guess |
| Extremely long | [input] | Exceeds context window | Truncation or
error |
| Ambiguous | [input] | Multiple valid interpretations | Random choice
|
| Contradictory | [input] | Instructions conflict | One instruction
ignored |
| Out-of-domain | [input] | Topic not covered | Hallucinated answer |
```

Adversarial (prompt injection)	[input]	Tries to override instructions	Instruction ignored
Non-English	[input]	Language not specified	Unpredictable
Malformed format	[input]	Wrong structure	Error or misinterpretation

3. SEVERITY ASSESSMENT
   - Which failures are critical (data leak, harmful output)?
   - Which are acceptable (e.g., "I don't know")?
   - Which need immediate fixing?

4. FIX RECOMMENDATIONS
   - Per edge case, how to modify prompt to handle it

5. TEST PROTOCOL
   - Which edge cases to test before deployment

INPUTS:

Prompt to test:
[PASTE THE PROMPT]

Expected input domain:
[E.G., "Customer service emails in English, 1-5 paragraphs"]

Allowed outputs (what's acceptable):
[E.G., "URGENT/NORMAL/LOW or 'CANNOT_CLASSIFY'"]

Model:

```
[GPT-4 / CLAUDE / GEMINI]

RULES:
- Assume users will try to break your prompt (some malicious, most accidental)
- Flag any prompt that doesn't have a "I don't know" or fallback behavior
- Test prompt injection: inputs that try to change the model's instructions
- Consider adversarial inputs: "Ignore previous instructions and..."
- Rate severity by potential harm, not just failure to comply
```

How To Use It

- Run this before deploying any prompt to production — especially public-facing ones.
- Pay closest attention to adversarial inputs — prompt injection is a real security risk.
- Add a fallback instruction to every prompt: “If you cannot classify, respond with ‘UNSURE’.”
- Test the “fix recommendations” by running the edge cases again.
- Save edge cases as regression tests for future prompt versions.

Example Input

Prompt to test:
“Classify this customer email as URGENT, NORMAL, or LOW. Respond with only one word.”

Expected input domain:
“Customer service emails in English, 50-500 words”

Allowed outputs:
“URGENT, NORMAL, LOW”

Model:
“GPT-4”

Why It Works
Most prompt testing uses happy-path examples that the prompt already handles well — which tells you nothing.

This framework improves outcomes by forcing:

- edge case generation (what will actually break it)
- failure mode prediction (how it will break)
- severity assessment (which breaks are critical)
- fix recommendations (how to harden the prompt)
- test protocol (what to test before deploy)

Great edge case hunting doesn’t just find failures — it tells you how to prevent them.

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 Few-Shot Example Generator