Prompt Engineering / AI Agents

Given an agent design, predict how it will fail — loop traps, tool misuse, goal misalignment, context overflow.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Agent QA, Pre-Deployment Testing, Robustness Engineering
Updated: May 2026
Why This Prompt Exists
Agents fail in predictable ways — but only if you think about failure before deployment. Most teams discover failures in production.

You get:

- agents stuck in infinite loops (call tool → get error → call same tool again)
- agents misusing tools (wrong parameters, wrong order)
- goal misalignment (agent optimizes for stated goal but violates constraints)
- context overflow (agent forgets early instructions after long conversation)
- no systematic failure testing before deployment

But failure modes are predictable:

- loop traps: agent repeats same action without progress
- tool misuse: agent calls tool incorrectly or unnecessarily
- goal drift: agent gradually□□ original objective
- context loss: agent forgets instructions after many turns
- resource exhaustion: agent exceeds token, time, or API limits
- edge case failure: agent works on 90% of inputs, fails on 10%

Without prediction, you deploy broken agents.

This prompt predicts how an agent design will fail before deployment.

The Prompt

```
Assume the role of an agent reliability engineer who predicts failure modes.

Your task is to identify how an agent design is likely to fail in production.

Generate:

1. AGENT DESIGN SUMMARY
   - Architecture (ReAct, Plan-Execute, etc.)
   - Tools available
   - Planning horizon
   - Memory strategy
   - Reflection/verification steps

2. PREDICTED FAILURE MODES (ranked by likelihood)

| Failure Mode | Description | Likelihood | Severity | Mitigation |
|--------------|-------------|------------|----------|------------|
| Loop trap | [how it will loop] | High/Med/Low | High/Med/Low | [how to prevent] |
| Tool misuse | [how it will misuse tools] | High/Med/Low | High/Med/Low | [how to prevent] |
| Goal misalignment | [how it will optimize wrong thing] | High/Med/Low | High/Med/Low | [how to prevent] |
```

| Context overflow | [when it will forget] | High/Med/Low | High/Med/Low | [how to prevent] |
| Edge case | [which inputs will break it] | High/Med/Low | High/Med/Low | [how to prevent] |

3. STRESS TEST PROMPTS
   - Specific inputs designed to trigger each failure mode
   - Example: To test loop trap, give ambiguous instruction that requires clarification

4. MONITORING RECOMMENDATIONS
   - What to log to detect this failure in production
   - Alert thresholds

5. REDESIGN RECOMMENDATIONS
   - How to modify the agent design to eliminate or reduce each failure mode

6. DEPLOYMENT READINESS
   - Ready to deploy (no high-likelihood critical failures)
   - Needs testing (high-likelihood but mitigatable)
   - Redesign required (critical failure mode likely)

INPUTS:

Agent design description:
[PASTE AGENT SYSTEM PROMPT OR ARCHITECTURE]

Task domain:

```
[E.G., "Customer support for e-commerce"]

Known edge cases from testing (if any):
[E.G., "Failed when user asked about refunds"]

Model:
[GPT-4 / CLAUDE / GEMINI]

RULES:
- Loop traps are the most common agent failure — test for them
explicitly
- Tool misuse often happens when tool descriptions are ambiguous
- Goal misalignment requires explicit constraints in the prompt
- Context overflow happens around 20-30 turns for most models
- Always test on edge cases before deployment (don't assume they'll
work)
```

How To Use It

- Run this before deploying any agent — even simple ones have failure modes.
- Loop traps are the most common — test by giving ambiguous instructions.
- Use the stress test prompts to actually test the agent (don’t just predict).
- Monitor for the predicted failures in production (log tool calls, track loops).
- Redesign to eliminate high-likelihood, high-severity failures before deployment.

Example Input

**Agent design description:**
“ReAct agent with web search and calculator tools. No reflection step. Planning horizon = 5 steps. No memory beyond current turn.”

**Task domain:**
“Research assistant — answers factual questions”

**Known edge cases from testing:**
“None yet — pre-deployment”

Why It Works
Most agent testing uses happy-path examples — which all agents pass. Failure testing requires thinking like an adversary.

This framework improves outcomes by forcing:

- failure mode enumeration (not just “it might fail”)
- likelihood and severity ranking (prioritize fixes)
- stress test generation (specific inputs to trigger failures)
- monitoring recommendations (detect failures in production)
- redesign guidance (how to fix the design)

Great failure prediction doesn’t just list what could go wrong — it tells you how to prevent it.

## 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 Planning Horizon Determiner