Prompt Engineering / AI Agents

Given a task and available tools, recommend which tool(s) the agent should use and in what order.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Agent Tool Use, Function Calling, Workflow Design
Updated: May 2026
Why This Prompt Exists
Agents fail not because they're incapable, but because they pick the wrong tool — or use tools in the wrong order.

You get:

- agents using expensive tools when cheap ones would work
- agents calling tools in sequence when parallel would be faster
- agents missing necessary tool calls (incomplete execution)
- agents stuck in loops calling the same tool repeatedly
- no systematic way to optimize tool selection for a task

But tool selection follows patterns:

- single tool: the task can be completed with one tool call
- sequential tools: tool A's output is input to tool B
- parallel tools: multiple tools can be called simultaneously
- conditional tools: tool selection depends on intermediate results
- fallback tools: if primary tool fails, use alternative

Without optimization, agents waste time, tokens, and money.

This prompt recommends optimal tool selection strategies for any task.

The Prompt

```
Assume the role of an agent tooling architect who optimizes tool selection.

Your task is to recommend which tools an agent should use and in what order.

Generate:

1. TASK ANALYSIS
   - Task description
   - Inputs available
   - Desired output
   - Constraints (latency, cost, accuracy)

2. AVAILABLE TOOLS (per tool)
   - Tool name and description
   - Inputs required
   - Outputs produced
   - Cost (token or API cost)
   - Latency (fast/medium/slow)
   - Success rate (if known)

3. TOOL SELECTION RECOMMENDATION

| Step | Tool | Why this tool | Alternative if fails |
```

```
|------|------|---------------|---------------------|
| 1 | [Tool A] | [reason] | [Fallback] |
| 2 | [Tool B] | [reason] | [Fallback] |
| ... | ... | ... | ... |

4. EXECUTION PATTERN
   - Sequential (Tool A → Tool B)
   - Parallel (Tool A and Tool B simultaneously)
   - Conditional (Tool A, then decide based on result)
   - Loop (repeat Tool A until condition met)

5. OPTIMIZATION NOTES
   - What to monitor (timeouts, error rates)
   - When to use alternative tools
   - How to chain outputs

6. READY-TO-USE TOOL SELECTION PROMPT
   - A prompt snippet that encodes this selection logic

INPUTS:

Task description:
[E.G., "Search the web for recent news about AI, then summarize the
top 3 articles"]

Available tools (list with descriptions):
[E.G., "web_search(query) – returns 10 URLs; scrape(url) – returns
page text; summarize(text) – returns 3 sentences"]
```

```
Constraints:
[E.G., "Under $0.10 per task, under 10 seconds"]

Agent framework:
[FUNCTION CALLING / REACT / PLAN-EXECUTE / OTHER]

RULES:
- Prefer cheap tools when accuracy is sufficient (don't overpay)
- Prefer parallel execution when tools are independent (faster)
- Always include fallback tools (primary may fail)
- Flag if the task requires a tool you don't have (gap analysis)
- Recommend monitoring for tools with high failure rates
```

How To Use It

- Run this before implementing any agent workflow — design tool selection intentionally.
- Include cost and latency in your tool specifications (agents need this to optimize).
- Always include fallbacks — no tool works 100% of the time.
- Test the recommended pattern on 10+ tasks before deploying.
- Re-optimize when new tools become available or costs change.

Example Input

Task description:
“Given a customer question, search our knowledge base, find the most relevant article, and generate a response.”

Available tools:
“kb_search(query) — returns 5 article titles and IDs — fast, cheap. get_article(id) — returns full article text — medium cost. generate_response(question, article) — returns answer —

cheap. llm_direct(question) — answers without KB — cheap but may hallucinate."

Constraints:
"Under 5 seconds, accurate responses preferred over cheap"

Why It Works
Most agents use a fixed tool selection strategy — "try the first tool, then the next" — which is rarely optimal.

This framework improves outcomes by forcing:

- tool capability analysis (what each tool actually does)
- cost/latency specification (trade-offs matter)
- execution pattern selection (sequential, parallel, conditional)
- fallback planning (what to do when primary fails)
- monitoring recommendations (what to track)

Great tool selection doesn't just list tools — it orchestrates them for efficiency and reliability.

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 Agent Reflection Prompt