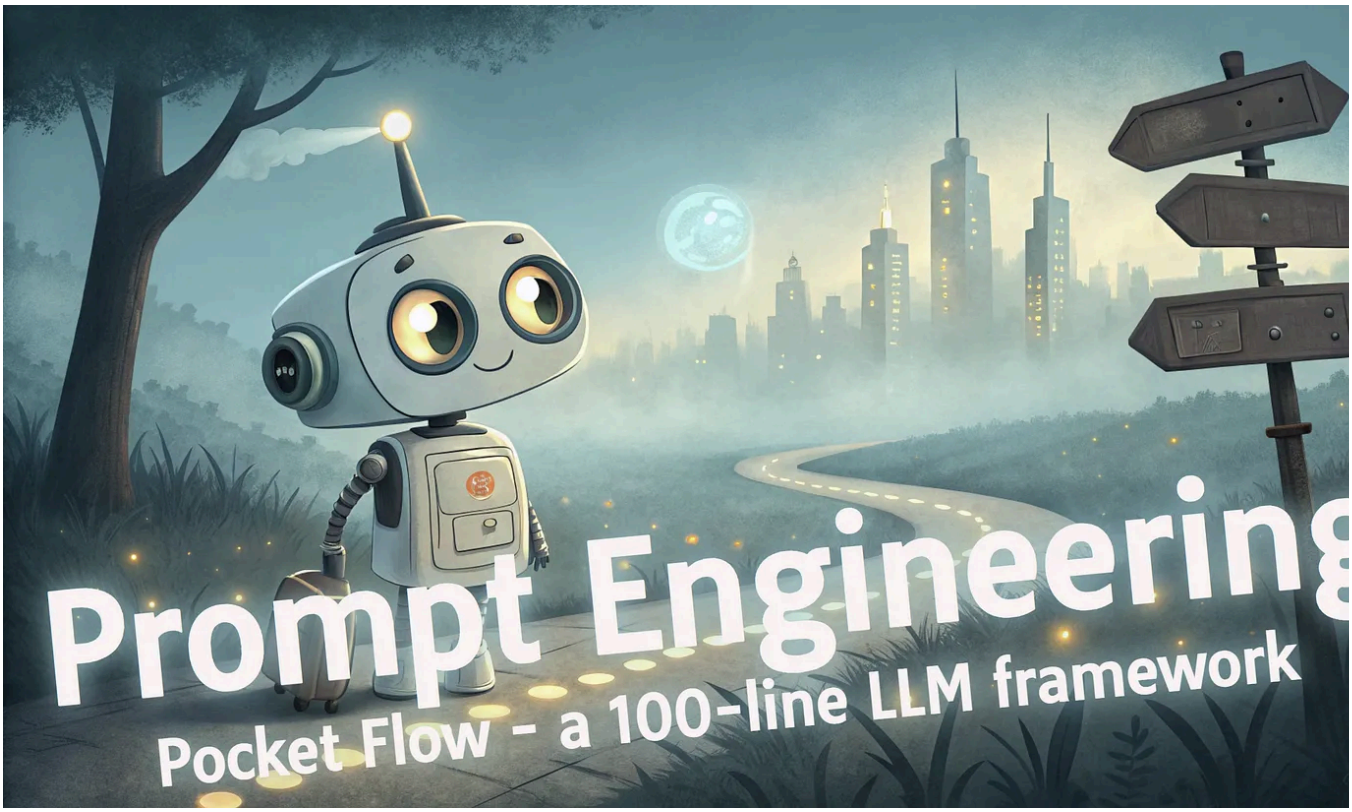
LLM Agents & Prompt Engineering: A Warrior's Guide to Clear Commands

ZACHARY HUANG
JUL 29, 2025



> *You've forged a legendary warrior with the perfect arsenal and taught them to navigate dungeon. But what happens when your commands are as clear as mud? "Attack the thi you shout. Which thing? With what weapon? Your warrior stands confused. In this gui you'll master the final art—giving crystal-clear battle commands that turn hesitation i decisive action.*

1. Introducti

Picture this: You've s us posts, you've given them a razor-sharp sword (perfect actions). You've taught them

carry a lightweight but complete backpack (smart context management), and they' standing right in front of the treasure chest. Victory is one command away.

> "Get the treasure!" you command confidently.

Your warrior looks at you. Looks at the chest. Looks back at you. Then proceeds tc attack the chest with their sword, smashing it to pieces and destroying the treasur inside.

> "What are you DOING?!" you scream.

The warrior shrugs. "You said 'get' the treasure. I thought you meant destroy the c to get to it. Maybe you should have said 'carefully open the chest and retrieve the treasure inside.'"

This is the final challenge in building effective LLM agents. You can have the mos sophisticated graph structure, the deadliest arsenal of tools, and the smartest cont management—but if your commands are vague, your agent becomes a confused m

The good news? **Prompt engineering isn't magic.** It's not about finding secret incantations that "unlock" your LLM's hidden powers. It's simply the art of giving clear, unambiguous instructions. Think of it as the final sharpening of your blade- the forging itself, but the careful honing that turns a good sword into a legendary

---

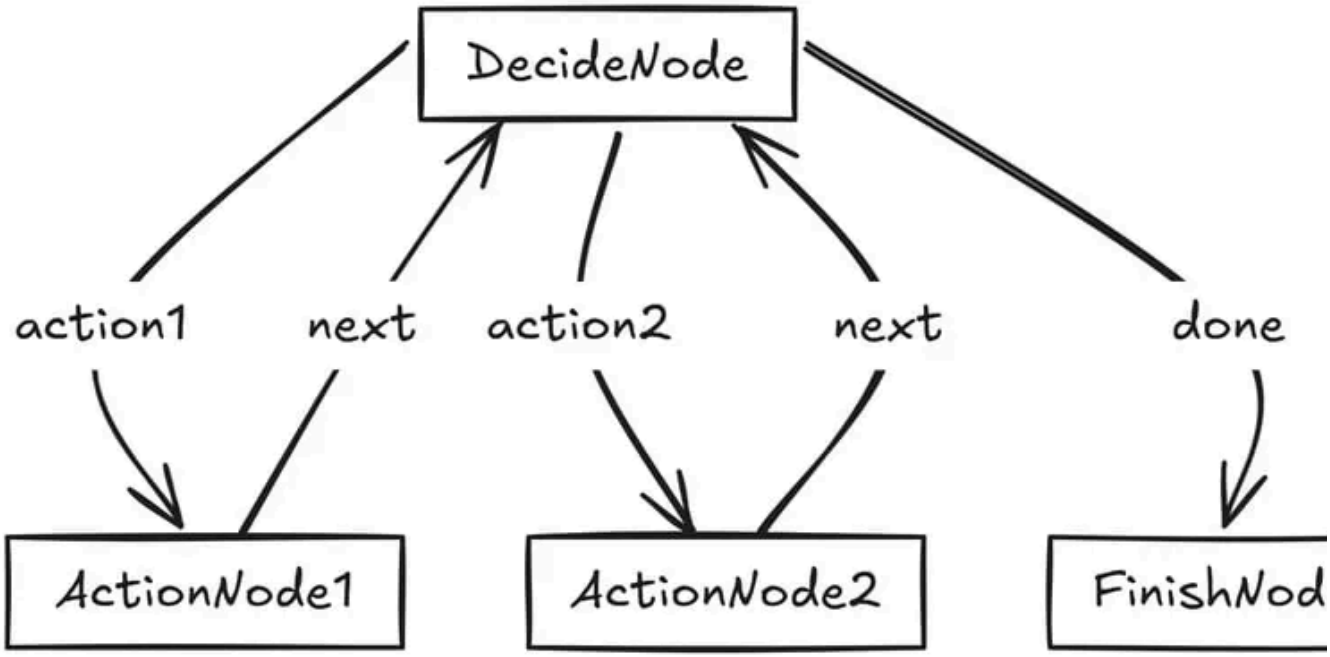
In this guide, we'll strip away the mystique and show you three simple laws that will trans your prompts from c warrior won't just

2. How Agents Work: The Battle Plan Recap

Before we sharpen our commands, let's quickly remind ourselves how our warrior actually operates. As we discovered in our previous adventures, every LLM agent—matter how complex it appears—follows a devastatingly simple loop:

Assess → Strike → Repeat

In the **PocketFlow** framework, this loop manifests as a graph:



The `DecideNode` is our warrior's brain—the battle tactician that looks at the situation and chooses the next move. But here's the crucial part: **how does it think**

The answer is simpler than you might expect. The entire "thinking" process is just prompt. Here's the general structure inside any DecideNode:

```
prompt = f"""
### YOUR TASK
{task_instructio

### CURRENT STATI
{context}

### AVAILABLE AC
{action_space}
```

```
### YOUR NEXT MOVE
Based on your task and current state, choose the best action.
Format your response as:

```yaml
thinking: |
    <your reasoning>
action: <chosen action>
parameters: <any required parameters>
```"""
```

This is the prompt we're going to improve.

Notice how this universal template has three main components:

1. **Task Instruction**: What the agent is trying to accomplish
2. **Context**: The current state (from the `shared` store we discussed)
3. **Action Space**: The available tools (the arsenal we forged)

We've already mastered forging the perfect **action space** (giving our warrior the ri weapons) and managing **context** (keeping the backpack light and relevant). Now it time to focus on that final piece: the **task instruction**—the actual commands we g our warrior.

Look at it carefully. Everything the agent knows comes from these three sections. since we've already perfected the action space and context management in our previous guides, the quality of your agent now hinges on that task instruction—th specific commands and guidelines you provide.

3. The Myth

First, let's clear the a id internet is full of hyp e potential. People sha ing "let's think step by step or you are an expert will transform your results.

The reality is much simpler and, frankly, more empowering. Effective prompt engineering is not about tricking the LLM or finding secret incantations. It's abou **clarity and structure**.

Your goal is simple: write an instruction manual so clear, so unambiguous, that th LLM can't possibly misinterpret it. It's not magic—it's just good communication.

Here's the most important principle for any agent builder:

> **The 90/10 Rule: Spend 90% of your time designing actions and context. Prompting is the final 10%.**

Think of it this way:

- **Actions** (**The Arsenal**): If your warrior doesn't have the right weapons, no amo of yelling will help
- **Context** (**The Backpack**): If your warrior is carrying 500 pounds of junk, they c move effectively
- **Prompts** (**The Commands**): Only after the first two are solid should you polish your instructions

If your agent is failing, resist the urge to immediately tweak prompts. Instead, ask

1. Does it have the right tools for this job?
2. Is its context clear and focused?
3. Only then: Are my instructions clear?

This is why prompt engineering is the "sharpening, not the forging." You can't sharpen a blade that doesn't exist. You can't polish a sword made of rubber. But or you have a well-forge all difference.

Let's learn how to sh

4. The 3 Sacred Laws of Command

Now that we understand prompt engineering is about clarity, not magic, let's dive the three fundamental techniques that will transform your vague mumbles into crystal-clear battle commands.

4.1. Law #1: Demand the Battle Plan (Chain of Thought

Imagine two different commanders:

Commander A: "Attack the fortress!"

Commander B: "First, tell me your plan for attacking the fortress. Then execute it

Which warrior is more likely to succeed? The one who charges blindly, or the one thinks through their approach first?

This is the power of Chain of Thought (CoT). By explicitly asking your agent to ex its reasoning BEFORE taking action, you dramatically improve its decision quality like the difference between a warrior who swings wildly and one who calculates ea strike.

Before (No Chain of Thought):

```
prompt = f"""### YOUR TASK
{task_instruction}

### YOUR NEXT MOVE
Choose an action and provide parameters."""
```

After (With Chain of Thought):

```
prompt = f"""###
{task_instructio

### YOUR NEXT MO\
First, explain yo
Then choose an action and provide parameters.
```

```
Format your response as:
```yaml
thinking: |
    <your step-by-step reasoning process>
action: <chosen action>
parameters: <required parameters>
```"""
```

The difference is profound. With the second prompt, your agent might think:

```
response = """thinking: |
    1. I need to find grocery spending for last month
    2. First, I should identify what "last month" means (get current
date)
    3. Then search for transactions in that date range
    4. Filter for grocery-related merchants
    5. Sum up the amounts
    Starting with getting the current date...
action: get_current_date
..."""
```

Without CoT, it might have jumped straight to searching transactions without properly defining the date range. **Chain of Thought forces your warrior to strateg before striking.**

> **Pro Tip:** If you're using a thinking model (like o1 or Claude with thinking mod you get this for free! These models automatically think through problems step- step without explicit prompting. It's like having a warrior who's already trained strategize before every move.

4.2. Law #2: S Specific)

Your warrior stands

Poetic: "Choose wisely, for one path leads to glory."

General: "Take the left door. It leads to the armory. Avoid the right door—it's trap with poison darts."

Which instruction leads to success? The specific one, every time.

The biggest prompt engineering mistake is using vague, open-ended language wh you actually have specific requirements. Let's look at common examples:

Vague → Specific Transformations:

- ❌ "Search for relevant information"
- ✅ "Search for peer-reviewed studies published after 2020 about renewable en costs"
- ❌ "Summarize this appropriately"
- ✅ "Create a 3-paragraph summary with: (1) main finding, (2) methodology, (3) implications"
- ❌ "Handle errors gracefully"
- ✅ "If a search returns no results, try broadening the query by removing adjectives. If it still fails, return 'No information found' and explain what was searched"

The Power of Constraints:

Constraints aren't limitations—they're guideposts that lead to better results. Cons these improvements:

```
# Vague prompt
prompt_vague = f'
Research the top

# Specific prompt
prompt_specific =
Research electric vehicles with these constraints:
```

```
- Focus on: battery technology and charging infrastructure
- Time period: 2022-2024 developments only
- Sources: Prioritize industry reports and government data
- Length: Provide 3-5 key findings, each in 1-2 sentences
- Format: Bullet points with source citations"""
```

The specific version removes all ambiguity. Your warrior knows exactly what victo looks like.

4.3. Law #3: Show, Don't Just Tell (Few-Shot Examples

Sometimes, even the clearest words aren't enough. This is when you need to demonstrate, not just describe. Think of it as the difference between explaining sv techniques with words versus actually showing the movements.

Few-shot examples are perfect when you need:

- A specific writing style
- A particular output format
- Complex patterns that are hard to describe

Without Examples (Frustrating):

```
prompt = f"""### TASK
Write a customer support response in our company's friendly, solution-
focused style

<context and customer message here>
..."""
```

Your agent thinks: "V ne

With Examples (Cry

```
prompt = f"""### TASK
Write a customer support response following our style:

Example 1:
Customer: "My order hasn't arrived and it's been a week!"
Response: "I completely understand your frustration - a week is
definitely too long to wait! Let me track that down for you right away
I can see your order #12345 left our warehouse on Monday. I'll contact
the carrier now and have an update for you within 2 hours. In the
meantime, I've credited your account with 20% off your next order for
the inconvenience."

Example 2:
Customer: "This product broke after one day"
Response: "Oh no! That's definitely not the experience we want you to
have. I'm starting a replacement order for you right now - it'll ship
today with express delivery at no charge. No need to return the broken
item. Your new one should arrive by Thursday. Is there anything else I
can help make right?"

Now respond to this customer following the same style:
Customer: {customer_message}"""
```

Notice how the examples teach the style without explicit rules:

- Start with empathy/acknowledgment
- Take immediate action (no "we'll look into it")
- Provide specific next steps and timelines
- Offer something extra for the inconvenience
- End with an open offer to help more

When to Use Exampl

- **Writing style**: "N
- **Response forma**
- **Tone matching**: es
- **Edge cases**: Show how to handle tricky situations

Examples are particularly powerful because they bypass the ambiguity of language Instead of trying to define "friendly but professional," you simply show what it loc like in action.

6. Conclusion: From Confused Grunt to Elite Soldier

And so, our journey is complete. We've transformed a simple graph into a legenda warrior:

1. **The Graph**: Every agent is just Assess → Strike → Repeat
2. **The Arsenal**: Sharp, purposeful tools for every task
3. **The Context**: A light backpack with only what's needed
4. **The Commands**: Crystal-clear orders that eliminate confusion

Remember the 90/10 rule: If your agent fails, check the arsenal and context first. Prompt engineering is just the final polish. But when you need that polish, your th laws are:

- Demand the battle plan (Chain of Thought)
- Speak like a general (Be Specific)
- Show, don't tell (Few-Shot Examples)

Your warrior no longer smashes treasure chests when you meant "open them carefully." They understand. They execute. They win.

The next time someone shares "secret prompt tricks," smile knowingly. You understand it's not about magic—it's about clarity.

Ready to build agents *d s*

Thanks for reading Pocket Flow! Subscribe for free to receive new posts and support my work.

12 Likes · 1 Restack

← Previous

Discussion about this post

Comments Restacks

Write a comment...