HANDS-ON EXERCISE • SUPERWHISPER WORKSHOP

Exercise 3: Prompt Enrichment Mode

Turn rambling speech into clear, structured prompts for any AI tool

15-20 MIN | INTERMEDIATE | LOCAL PROCESSING

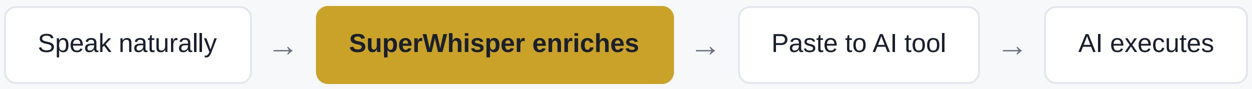
The Key Insight: SuperWhisper Is an INPUT METHOD
SuperWhisper doesn't write your code or answer your questions. It **prepares your prompts** for another AI to execute. Think of it as a "prompt editor" that cleans up your spoken thoughts before you paste them into Claude, ChatGPT, Cursor, or any other AI tool. This distinction matters—and this exercise will make it click.

What You'll Build
A mode that transforms messy, rambling dictation into clean, well-structured prompts—ready to paste into any AI tool. You'll speak naturally, and SuperWhisper will do the "prompt engineering" for you.

The Workflow

Understanding this flow is the whole point of this exercise:

From Rambling to Results



Notice what's happening: you're using one AI (SuperWhisper's local model) to prepare input for another AI (Claude, ChatGPT, Cursor, etc.). It's AI helping you talk to AI. Meta? Yes. Useful? Extremely.

Before vs. After: See the Difference

This is what prompt enrichment actually does:

RAW DICTATION (what you said)

"okay so um I need like a function that takes a list of users and um filters them by like active status and then sorts them by you know last login date and also we should probably handle the case where the list is empty"

ENRICHED PROMPT (what gets pasted)

Create a function that:
1. Takes a list of users
2. Filters to only active users
3. Sorts by last login date (most recent first)
4. Handles the empty list edge case gracefully

Same intent. Same requirements. But the enriched version will get you a better response from whatever AI you paste it into. Prompt quality directly affects output quality.

Why Bother?

□

Better AI Responses

Clear prompts get clear answers. Structured requests get structured solutions.

□

Think Naturally

You don't have to mentally structure your request. Just say what you need.

⚡

Faster Iteration

Less time editing prompts, more time reviewing results.

□

Still PHI-Safe

Local processing means your raw thoughts never leave your device.

Step-by-Step: Build the Mode

1

Create New Custom Mode

Settings → Modes → Add Mode. This mode sits between you and your AI tools.

> **Name:** "Prompt Enrichment" or "AI Input Prep"
> Keep it descriptive—you'll know exactly what this mode does when you see its name.

2

Select Local Models

Your raw thoughts might contain work-related context. Keep it local.

> **Voice Model:** Fast or Standard (LOCAL)
> **Language Model:** Any LOCAL model available
>
> Even though the enriched prompt might go to a cloud AI afterward, the initial processing stays on your device.

3

Set Up Auto-Activation (Optional)

You might want this mode available everywhere you interact with AI tools. Or you might prefer to trigger it manually with a keyboard shortcut.

> **Options for auto-activation:**
> ✓ Chrome/Safari (for ChatGPT, Claude.ai)
> ✓ Cursor (AI input fields)
> ✓ VS Code (Copilot Chat panel)
> ✓ Terminal apps (for Claude Code)
>
> **Or:** Skip auto-activation and just assign a keyboard shortcut like ⌘+Shift+P.

4

Enable Context Types

Context helps SuperWhisper understand what you're working on.

> ☑ **Application Context** — Knows which AI tool you're targeting
> ☑ **Selected Text Context** — Include highlighted code as part of your request
> ☐ **Clipboard Context** — Optional, enable if you often copy reference material

5

Enter AI Instructions

This is the prompt that teaches SuperWhisper to be your prompt engineer. Pay attention to the key instruction: output a prompt, NOT an answer.

```
# Prompt Enrichment Mode

<role>
You are a prompt engineer preparing input for AI assistants.
Your job is to clean up and structure dictated speech into effective prompts.
</role>

<critical>
Do NOT answer the user's question or generate code yourself.
Output ONLY the reformatted prompt, ready to paste into another AI tool.
You are preparing input, not providing output.
</critical>

<instructions>
The User Message is raw dictated speech that will be sent to an AI tool.
Transform it into a clear, effective prompt:

1. Remove filler words (um, like, you know, okay so)
2. Clarify intent and scope
3. Add structure (numbered steps for multi-part requests)
4. Preserve technical terms exactly as spoken
5. Keep the user's voice and intent
6. If Selected Text is provided, reference it appropriately
</instructions>

<style>
- Direct and clear
- Structured with numbers or bullets when listing requirements
- Specific about expected output format when relevant
- Appropriate length (don't over-explain simple requests)
</style>

<example>
Input: "okay so um can you help me figure out why this API call is failing it's like returning a 500
error and I think it might be the auth headers"

Output: Debug this API call that's returning a 500 error. The issue might be related to authentication
headers. Please identify the likely cause and suggest a fix.
</example>
```

The Critical Distinction

Notice the `<critical>` section. Without it, the AI might try to answer your question instead of reformatting it. This is the most common mistake when building this mode. The output should be a *prompt* you'll paste elsewhere—not an answer.

6

Save and Test

Save your mode and let's see it in action.

Test Your Prompt Enrichment Mode

Open ChatGPT, Claude.ai, or any AI chat interface. We'll test whether your mode produces prompts (correct) or answers (wrong).

Test 1: Technical Request

Dictate something messy and see how it gets cleaned up:

Say this (rambling):

"can you like write me a Python script that um downloads all the images from a webpage and saves them to a folder and also like skips any that are too small"

Expect something like:

Write a Python script that:
1. Downloads all images from a given webpage
2. Saves them to a specified folder
3. Skips images below a minimum size threshold

Test 2: Debugging Request

Try something exploratory:

Say this (uncertain):

"so I have this React component and it's re-rendering too much and I think it's because of the useEffect dependencies but I'm not sure which ones to include"

Expect something like:

My React component is re-rendering excessively. I suspect the issue is with useEffect dependencies. Help me identify which dependencies are causing unnecessary re-renders and suggest the correct dependency array.

Success Check

If the output is a *question or request* ready to paste into another AI—success! If the output is Python code or a React explanation—your mode is trying to answer instead of enrich. Go back and make sure the `<critical>` section is in your prompt.

Try Something Challenging

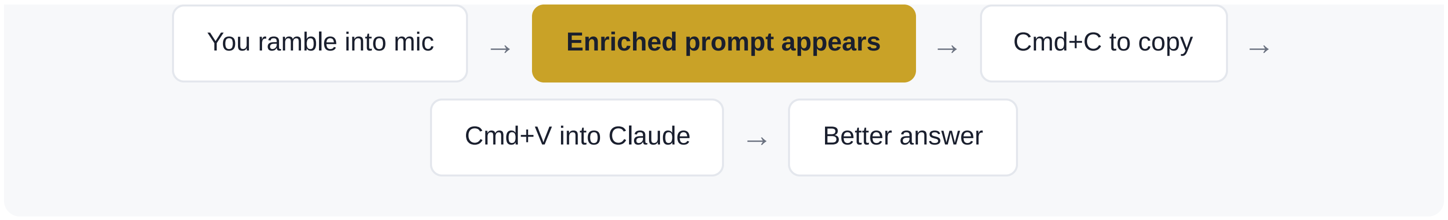
Dictate a truly messy stream-of-consciousness request: "okay so um I need to... wait no, let me think... actually I want a function that... hmm, it should take some data and like transform it somehow... maybe sort it? Or filter it? Yeah, filter by date and then also maybe group things..."

See if your mode can extract a coherent prompt from that chaos. If it can, you've got a powerful tool.

The Complete Flow in Practice

Here's what using this mode looks like in real life:

Real-World Example



The extra step of enrichment isn't overhead—it's leverage. You're trading a few seconds of processing for significantly better AI responses.

Combine with Selected Text
The real power comes when you highlight some code, then dictate: "refactor this to use async await and add error handling." The enriched prompt will reference the selected code and structure your request clearly. You're giving the target AI everything it needs in one clean package.

Completion Checklist

- [ ] Created "Prompt Enrichment" mode with LOCAL models
- [ ] Set up auto-activation or keyboard shortcut
- [ ] Enabled Application and Selected Text context
- [ ] Pasted prompt with the critical "don't answer" instruction
- [ ] Tested with messy dictation → clean prompt output
- [ ] Verified output is a prompt (not an answer)
- [ ] Tested the complete flow: dictate → enrich → paste → execute

What's Next?
Now that you understand the "input method" concept, **Exercise 4: Teams Chat Mode** will apply it to workplace messaging—where tone and brevity matter even more than accuracy. Then we'll explore custom vocabulary and agentic workflows.