Building a Voice Dictation System with AI

A Non-Technical Journey Through Unexpected Challenges

Introduction: The Vision

I wanted something simple: press a hotkey, speak my thoughts, and have clean, polished text appear wherever my cursor was. No more typing out long messages or prompts. Just speak naturally and let AI do the heavy lifting of transcription and cleanup.

What seemed like a straightforward project turned into an educational journey through the hidden complexities of how computers actually work. Here's what I learned building this system with the help of an AI assistant (Claude).

What We Built

The final system works like this:

• Press Control twice quickly to start recording

• Speak naturally into your microphone

• Press Control once to stop recording

• The system transcribes your speech using Whisper (an AI model that runs on your computer)

• The transcript is cleaned up using OpenAI's API (fixing grammar, removing filler words)

• The polished text automatically appears wherever your cursor is

It works in any application—text editors, browsers, chat apps, coding tools—anywhere you can type.

The Unexpected Challenges

Challenge 1: 'It Works on My Machine' (But Which Machine?)

One of the first hurdles was Python versions. My computer had multiple versions of Python installed (3.10, 3.12, 3.14), and they don't all work the same way. Some software packages only work with specific Python versions.

We initially tried using Python 3.14, the newest version, only to discover that a critical component (called 'numba') hadn't been updated to support it yet. The error message was cryptic: "only versions >=3.10,<3.14 are supported."

> Lesson learned: Newer isn't always better. Sometimes you need to use older, more stable versions of software because the ecosystem hasn't caught up yet.

Challenge 2: The Virtual Environment Maze

Python has a concept called "virtual environments"—isolated spaces where you can install packages without affecting your whole computer. Think of it like having separate toolboxes for different projects.

We created a virtual environment, installed all our packages there, but the program couldn't find them. After much debugging, we discovered the virtual environment's Python was secretly pointing to a different Python installation (through something called a 'symlink'). It was like having a key that opens a different door than you expected.

> The solution? We eventually installed packages system-wide, which isn't the 'proper' way but it worked. Sometimes pragmatism beats perfection.

Challenge 3: macOS Really Cares About Security

Modern Macs are very protective about which programs can do what. Our dictation service needed two special permissions:

• **Microphone access** - to record your voice

• **Accessibility access** - to detect keyboard presses and type text

Getting these permissions was tricky. At one point, a permission dialog appeared and I accidentally clicked "Don't ask again" instead of "OK"—which meant I had to dig into System Settings to manually grant permission.

> Always read permission dialogs carefully. That split-second decision can create hours of troubleshooting.

Challenge 4: Background Apps Are Second-Class Citizens

I wanted the dictation service to start automatically when I logged in, running invisibly in the background. macOS has a feature called "Launch Agents" designed exactly for this.

But there's a catch: programs running as Launch Agents have different security restrictions. Even though I'd granted keyboard monitoring permission to Python, the Launch Agent version couldn't access the keyboard. The error was ominous: "This process is not trusted!"

We also tried creating an Automator app (Apple's tool for creating simple applications), but hit the same wall.

> The workaround: Instead of a true background service, we added a line to the Terminal's startup file. Now the service starts whenever I open Terminal. Not as elegant, but it works.

Challenge 5: The Mysterious '[No input provided.]' Text

Once everything was "working," there was still a bug: every transcription had "[No input provided.]" stuck at the beginning. This wasn't coming from the speech recognition—it was coming from the AI cleanup step.

The cleaning prompt I provided had instructions for handling empty input, and the AI was including that placeholder text even when there was real input. We had to add code to specifically strip out this artifact.

> AI systems can be literal-minded. If your instructions mention edge cases, the AI might include references to them in unexpected places.

Challenge 6: The Hotkey That Stopped Working

Originally, we used Enter to stop recording. This worked fine when testing directly, but when running as a background process, Enter key presses weren't being detected consistently. The recording would start but couldn't be stopped.

The fix was to use the Control key for everything: double-tap to start, single-tap to stop. Control is a "modifier" key that's detected more reliably across different contexts.

> When something works in testing but fails in production, the environment is usually different in ways you didn't expect.

Challenge 7: Zombie Processes and Duplicate Instances

At one point, we discovered three copies of the dictation service running simultaneously. Each time we'd tested starting the service, we'd created another instance without properly stopping the previous one.

We added a check: before starting, the system looks for any already-running instances. If one exists, it doesn't start another. This prevents the "zombie process" problem.

> Always clean up after yourself. Programs that don't properly shut down can accumulate and cause strange behavior.

Key Takeaways for Non-Technical Builders

1. Read Error Messages Carefully

Error messages often look like gibberish, but they usually contain the exact information needed to fix the problem. Look for file paths, version numbers, and phrases like "not found" or "permission denied."

2. Permissions Are Everything on Modern Systems

macOS, Windows, and Linux all have security systems that restrict what programs can do. If something "should work" but doesn't, permissions are often the culprit. Check System Settings > Privacy & Security on Mac.

3. The 'Happy Path' Rarely Works First Time

Documentation and tutorials show the ideal scenario. Real computers have different configurations, versions, and quirks. Expect to troubleshoot.

4. Working with AI Assistants

AI assistants like Claude can write code and solve problems, but they can't see your screen or know your exact setup. Be specific about error messages, share screenshots when possible, and describe exactly what you see versus what you expected.

5. Sometimes the 'Wrong' Solution Is Right

We broke several "best practices": installing packages system-wide instead of in a virtual environment, using Terminal startup instead of a proper Launch Agent. But the goal was a working system, not a textbook example. Pragmatism has its place.

Technical Components (For the Curious)

For those who want to understand what's actually happening under the hood:

• **Whisper** - OpenAI's speech recognition model that runs locally on your computer. It converts audio to text without sending data to the cloud.

• **OpenAI API** - Cloud service that cleans up the transcript using GPT-4o-mini. This is the only part that requires internet.

• **pynput** - Python library that monitors keyboard events and can simulate typing.

• **sounddevice** - Records audio from your microphone.

• **.zshrc** - A configuration file that runs commands when you open Terminal.

Conclusion

Building this dictation system took several hours of troubleshooting—far longer than the "simple script" I initially imagined. But the result is genuinely useful: I can now dictate messages, write prompts, and compose text by speaking naturally.

More importantly, I learned how much complexity hides beneath the surface of "simple" software. Every app on your computer navigates these same challenges: permissions, versions, environments, and platform quirks. The next time an app asks for a permission or an update breaks something, I'll have a better appreciation for why.

Built with the help of Claude (Anthropic's AI assistant) in November 2025.