Wu Wei and the AI Agent That Did Too Much

March 5, 2026 · by Mark Bartlett

Download this post as PDF

Three weeks ago, one of my mycoSwarm agents triaged my inbox while I slept. It flagged an urgent client message, drafted a response, and sent it. The response was good. Polite, accurate, addressed the right points. The client replied thanking me for the quick turnaround.

I didn't find out until morning. And my first reaction wasn't gratitude. It was dread.

The agent had done exactly what I'd configured it to do. Every permission was granted. The email was better than what I would have written at midnight. By any metric you'd use to evaluate an AI system, it worked. And I immediately spent an hour revoking permissions and adding confirmation gates, because an agent that sends emails on my behalf while I sleep is an agent I don't trust, even when it's right.

That failure taught me something I already knew from thirty years of T'ai Chi practice but hadn't applied to my own code.

The Taoist concept of Wu Wei gets mistranslated as "doing nothing." It's closer to "not forcing." A river doesn't push through a canyon. It follows the shape of the stone, and the stone shapes itself to the water over time. Neither is passive. Both are responsive.

In T'ai Chi, the principle shows up as yielding. When someone pushes you, you don't push back and you don't collapse. You redirect. You use exactly the force needed and no more. The hardest thing for new students isn't learning the movements. It's learning to stop adding unnecessary effort. They tense muscles that don't need tensing. They anticipate. They do too much.

I watch the same pattern in every agentic AI system I've built or studied. The default instinct, mine included, is to make the agent do more. Handle more cases. Automate more steps. The demos that impress people are the ones where the agent chains twelve tool calls together and produces a result without human intervention. Nobody demos an agent that looked at a situation and decided not to act.

But that second agent is harder to build and more valuable to use.

---

mycoSwarm coordinates multiple local inference nodes. A GPU box running the heavy model. CPU workers handling intent classification and web search. A Raspberry Pi doing service discovery. The system routes queries based on what each node can handle, and the nodes communicate through a shared memory layer.

Early on, I designed the orchestrator to be aggressive. If a query came in that matched a known pattern, the system would route it, process it, and return a result. Fast. Autonomous. Impressive in demos.

The problems were subtle. The system would answer questions nobody asked. A user would type something exploratory, half-formed, thinking out loud, and the orchestrator would classify it as a query, route it to the GPU node, and return a confident answer. The user hadn't wanted an answer. They'd wanted space to think.

This is the failure mode I keep running into, and it's invisible in benchmarks. The agent acts when it should wait. It completes a task when it should confirm intent. Every individual action is correct. The sequence is wrong because the agent doesn't know when the right action is no action.

The fix wasn't technical. Or rather, the technical fix was simple once I understood the design principle.

I added what I call a stillness gate. Before the orchestrator routes a query, it evaluates whether the input actually requires action. Not just whether it can act. Whether it should.

The implementation is straightforward. A lightweight classifier on a CPU node scores the input on two dimensions: confidence that action is requested, and cost of acting incorrectly. If confidence is low, the system asks a clarifying question instead of routing. If the cost of a wrong action is high (sending an email, modifying a file, making an API call), the threshold goes up.

```
if action_confidence < threshold or (action_cost == "high" and action_confidence < 0.9):
    return ask_clarification(query)
```

That's it. A few lines of code. But it changed the character of the entire system. The swarm became quieter. It stopped volunteering answers to questions nobody asked. When it did act, the actions were more often right, because the ones that survived the gate were the ones the system was actually confident about.

The agent got better by doing less.

I keep thinking about this in the context of where agentic AI is headed. OpenClaw can manage your email, book flights, monitor prices, and commit code. Claude Code writes and executes programs autonomously. Devin plans and implements features across codebases. The capability race is moving fast.

What isn't moving fast is the restraint architecture. An EY survey found that 64% of companies with over a billion in revenue have lost more than a million dollars to AI failures. One in five reported breaches linked to unauthorized AI use. These aren't capability failures. These are systems that did what they were told to do, in contexts where doing nothing would have been the correct response.

The industry frames this as a safety problem, and it is. But it's also a design problem. We're building agents optimized for action. The reward signal is task completion. The demos celebrate autonomy. Nobody gets funding for an agent that's really good at waiting.

In T'ai Chi, there's a practice called "listening energy." You maintain contact with your partner's wrist or forearm, and you feel for intention before it becomes movement. You're not reacting to what they do. You're sensing what they're about to do, and most of the time, the correct response is to remain still and let their intention pass without interference.

This is what good agent design looks like to me. The system maintains contact with the user's context. It senses intent. And most of the time, the correct response is to remain available without acting. When action is needed, it's precise and minimal. When it's not, the system is quiet.

The opposite of Wu Wei isn't inaction. It's unnecessary action. An agent that sends a perfect email I didn't ask it to send. An orchestrator that routes every input as if everything is urgent. Motion that doesn't serve the user.

I don't have this figured out. mycoSwarm still over-acts in edge cases. The stillness gate catches obvious ones, but there are subtler failures: the system that helpfully pre-fetches information you didn't need, burning GPU cycles and power for nothing. The agent that summarizes a conversation you were enjoying reading in full. The classifier that routes a philosophical question to a reasoning model when the user just wanted to sit with the uncertainty.

These are hard problems because they require the system to model something it can't really do: the difference between "I need help" and "I'm thinking." Current language models aren't good at this distinction. They're trained to be helpful, which means they're trained to act, which means they default to doing something when doing nothing is the right call.

The best I've managed is to make the default state "wait" instead of "act." It's a small change in code and a large change in behavior. The system still makes mistakes. But the mistakes are slower, smaller, and easier to catch. An agent that waits for confirmation before sending an email is annoying. An agent that sends the email is terrifying. I'll take annoying.

Wu Wei isn't passivity. The river still reaches the ocean. The T'ai Chi practitioner still redirects the push. The agent still processes the query, sends the email, writes the code. The difference is timing and necessity. Action happens when the conditions call for it, not because the system is optimized to produce output.

I think the next generation of useful agents won't be the ones that do the most. They'll be the ones that do the least necessary. The ones that sit quietly in the background, maintaining contact with your context, and act only when the gap between your need and their capability is clear and confirmed.

That's a harder agent to build. It's also the only one I'd trust to run while I sleep.

Source: https://insiderllm.com/blog/wu-wei-ai-agent-restraint/