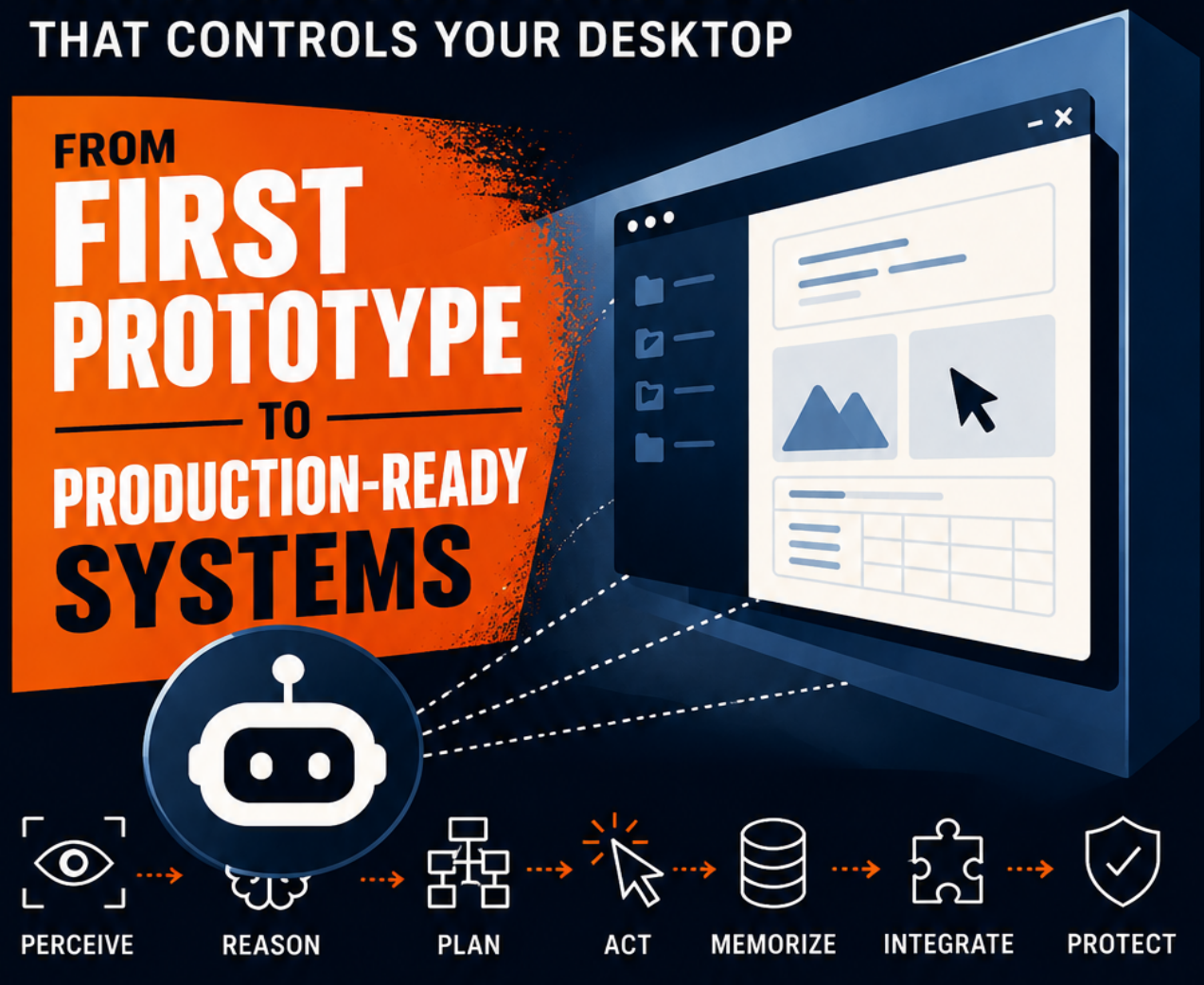
COMPUTER USE AGENTS

A PRACTICAL GUIDE TO BUILDING AI THAT CONTROLS YOUR DESKTOP

FROM FIRST PROTOTYPE TO PRODUCTION-READY SYSTEMS



CURTIS ZHANG

Computer Use Agents: A Practical Guide to Building AI That Controls Your Desktop

From First Prototype to Production-Ready Systems

Steve Publications

This book is available at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontrolsyourdesktop

This version was published on 2026-07-23

Contents

From First Prototype to Production-Ready Systems

About this book: This book teaches you how to build computer use agents that perceive screens, reason about tasks, and execute actions across desktop applications and web browsers. Starting from a working prototype in under thirty minutes, you will incrementally add perception pipelines, planning systems, memory, tool integrations, safety guardrails, and production infrastructure. Every chapter builds on the previous one with production-quality code, real-world case studies, and honest discussion of trade-offs and failure modes. If you are a software engineer who wants to move beyond chatbots and build agents that actually do work on a computer, this is the book for you.

Introduction: The Agent on Your Screen

Open your browser. Navigate to a project management tool you use at work. Find the task you created last Tuesday. Change its priority to high, assign it to a teammate, and add a comment with today's date.

Now imagine giving those instructions to a program and watching it do all of that itself. Not by calling an API you wrote for this exact scenario, but by seeing the screen, understanding what buttons mean, clicking where needed, and typing in the right fields. That is a computer use agent.

This is not science fiction anymore. In October 2024, Anthropic released Claude Computer Use, allowing their model to navigate desktop environments through screenshots, mouse clicks, and keyboard input. OpenAI followed with its own computer-use tool integrated into the Responses API. Microsoft released Magentic-One, an open-source framework for general-purpose computer agents. Researchers have built benchmarks like OSWorld and WebArena that test agents on hundreds of real tasks across multiple operating systems and web applications.

The progress is dramatic, but the reality is more nuanced than marketing slides suggest. On the OSWorld benchmark, which contains 369 real-world tasks involving desktop applications, file operations, and multi-app workflows, human performers succeed on over 72 percent of tasks. The best model at launch achieved 12.24 percent [1]. That gap tells an important story: agents can do remarkable things, but they are not ready to run unsupervised in mission-critical environments without careful engineering around them.

This book is about closing that gap through deliberate, production-minded design.

What Is a Computer Use Agent?

A computer use agent is a system that executes tasks on a digital device by perceiving the interface, reasoning about what to do next, and performing

actions such as mouse clicks, keyboard input, or programmatic commands. It operates at the same level of abstraction as a human user rather than through tightly coupled APIs.

This definition matters because it distinguishes computer use agents from other kinds of automation:

- Scripted automation tools like Selenium tests or cron jobs follow predetermined steps. They cannot adapt when the interface changes or when an unexpected condition arises.
- Chatbots and assistants that answer questions operate within their context window and do not directly control external software.
- Code-generation agents produce code that a human reviews and runs. They do not execute actions themselves in the target environment.

A computer use agent sits between these categories. It receives a high-level instruction in natural language, perceives the current state of the interface, decides what action to take based on that perception and its goal, executes the action, observes the result, and repeats until the task is complete. This loop of perception, reasoning, and action is the core architecture of every agent we will build in this book.

The term has several aliases in the literature. You may see "GUI agent," "desktop agent," "OS agent," or "agents for computer use" (ACU). They all refer to roughly the same idea: AI systems that interact with graphical user interfaces to accomplish tasks [2].

Why Now: The Convergence That Made This Possible

Computer use agents became feasible only recently because three independent advances converged at roughly the same time.

The first is multimodal large language models capable of understanding screenshots. Early vision-language models could describe images but lacked the fine-grained spatial reasoning needed to identify where a specific button sits on a cluttered screen. Models like GPT-4V, Claude 3, and Gemini demonstrated that sufficiently trained vision-language systems can locate UI elements, read text at various scales, and understand visual hierarchy well enough to guide actions.

The second is the ReAct paradigm and its descendants, which showed that interleaving reasoning with action produces more reliable behavior than either pure chain-of-thought or pure tool use [3]. An agent that thinks about what it wants to do, takes an action, observes the result, and then revises its plan handles uncertainty and error in ways that single-shot prompting cannot.

The third is the maturation of browser and desktop automation infrastructure. Playwright, Puppeteer, Selenium, pyautogui, and accessibility APIs provide reliable mechanisms for translating decisions into actual mouse movements, clicks, keystrokes, and window management. These tools are fast enough that the bottleneck is usually the model's reasoning time rather than the mechanics of interaction.

None of these advances alone would have been sufficient. Excellent vision without a control loop produces commentary, not action. A solid planning framework without perception has nothing to reason about. Reliable automation without intelligence is just scripting with extra steps. Together, they make computer use agents possible.

The Promise and the Peril

The promise is straightforward: computer use agents can automate workflows that were previously impossible to automate because no API existed, the software is legacy and undocumented, or the task requires judgment that exceeds rule-based automation.

Concrete examples include navigating a company's internal dashboard to generate a weekly report, testing a web application by exploring it like a real user would, migrating data between systems with incompatible APIs, or helping non-technical staff complete repetitive tasks across multiple applications. A comprehensive survey of the field covering 87 agents and 33 datasets notes that desktop PC automation remains underexplored relative to web and mobile domains, suggesting significant opportunity for builders who can solve the harder problems [2].

The peril is equally straightforward: an agent that can click and type on your computer can also delete files, send emails you did not intend, transfer money from accounts you control, or expose sensitive data visible on screen. Prompt injection attacks remain a genuine threat. Anthropic's own research on browser-based agents found that while defenses have improved, a 1 percent

attack success rate still represents meaningful risk, and they explicitly state that no current agent is immune to prompt injection [4].

This is not a reason to avoid building agents. It is a reason to build them responsibly. Every chapter of this book addresses safety where relevant: sandboxing in the execution chapters, guardrails in the safety chapter, human-in-the-loop patterns throughout, and threat modeling in the security chapter. A production-ready agent must be safe by design, not safe by accident.

How This Book Is Structured

This book takes you from zero to a production-grade computer use agent through a deliberate progression. Each chapter introduces concepts and techniques that build on everything before it. By the end, you will have the knowledge to design, implement, evaluate, and deploy agents for real-world tasks.

The introduction you are reading now sets up the landscape and shows you a minimal working example in under thirty seconds of code. Chapter 1 establishes the conceptual foundations: what an agent is, the perception-action loop, and why standard prompting fails for computer control.

Chapter 2 is where you write your first real code. We build a minimal but functional agent that captures screenshots, sends them to an LLM, parses the model's response into actions, and executes those actions in a browser. This prototype is intentionally simple so you can see the entire loop in one file. It is also intentionally flawed so you understand what problems each subsequent chapter solves.

Chapters 3 through 8 cover the core technical components of a capable agent: perception (how the agent sees the screen), planning (how it reasons about multi-step tasks), memory (how it maintains context across long interactions), tools (how it accesses structured APIs alongside UI interaction), action execution (how it reliably controls desktop and browser environments), prompt engineering (how to design system prompts that produce reliable behavior), and state management (how the agent tracks progress through complex workflows).

Chapters 9 through 13 address the qualities that separate a demo from a production system: safety and guardrails, evaluation and measurement,

debugging and observability, optimization for speed and cost, and security practices specific to agents that can execute actions.

Chapter 14 covers deployment patterns for running agents at scale, including containerization, orchestration, multi-tenancy, and operational best practices. Chapter 15 presents detailed case studies showing how all these concepts come together in real-world scenarios across different domains. The conclusion synthesizes everything and points toward emerging directions in the field.

The code examples use Python throughout because it is the lingua franca of AI development and has mature libraries for every layer of the stack. Where frameworks like LangGraph or Playwright are used, I explain both what they do and why you might choose them over alternatives so you can make informed decisions for your own projects.

Your First CUA in 30 Seconds

Before we dive deep, let you see the core idea in action. The following minimal example demonstrates the essential loop of a computer use agent: capture a screenshot, send it to an LLM with instructions, parse the action, and execute it. This is not production-ready code. It is a proof of concept that shows you the shape of what we are building.

```
import base64
from io import BytesIO
import pyautogui
from openai import OpenAI

client = OpenAI()

def screenshot_to_base64():
    """Capture screen and return as base64-encoded PNG."""
    img = pyautogui.screenshot()
    buffer = BytesIO()
    img.save(buffer, format="PNG")
    return base64.b64encode(buffer.getvalue()).decode("utf-8")

def ask_model_for_action(screenshot_b64, task):
    """Ask the LLM what action to take next."""
    response = client.chat.completions.create(
        model="gpt-4o",
        messages=[
```

```
            {
                "role": "system",
                "content": (
                    "You control a computer. Look at the screenshot and decide the next action. "
                    "Respond with exactly one JSON object: "
                    '{"action": "click", "x": 100, "y": 200}' or '...'
                ),
            },
            {
                "role": "user",
                "content": [
                    {"type": "text", "text": f"Task: {task}\nWhat should I do next?"},
                    {
                        "type": "image_url",
                        "image_url": {"url": f"{screenshot_b64}"},
                    },
                ],
            },
        ],
    )
    return response.choices[0].message.content

def execute_action(action_json):
    """Parse and execute the action from the model's response."""
    import json
    try:
        action = json.loads(action_json)
    except json.JSONDecodeError:
        print("Could not parse action, trying to extract JSON...")
        # Simple extraction of first JSON-like object
        start = action_json.find("{")
        end = action_json.rfind("}") + 1
        if start != -1 and end > start:
            action = json.loads(action_json[start:end])
        else:
            raise ValueError(f"Could not parse action: {action_json}")

    if action["action"] == "click":
        pyautogui.click(action["x"], action["y"])
        print(f"Clicked at ({action['x']}, {action['y']})")
    elif action["action"] == "type":
        pyautogui.write(action["text"], interval=0.05)
        print(f"Typed: {action['text']}")
    elif action["action"] == "done":
        print("Task complete.")
        return False
    return True
```

```
67
68  def run_agent(task, max_steps=10):
69      """Main agent loop."""
70      for step in range(max_steps):
71          print(f"\n--- Step {step + 1} ---")
72          screenshot = screenshot_to_base64()
73          response = ask_model_for_action(screenshot, task)
74          print(f"Model: {response[:200]}...")
75          if not execute_action(response):
76              break
77
78  # Usage: run_agent("Open Chrome and go to example.com")
```

This code does four things that every computer use agent does, regardless of complexity:

1. It captures the current state of the environment (a screenshot).
2. It asks an LLM to interpret that state and decide on an action, given a task.
3. It parses the model's response into a structured command.
4. It executes that command in the real environment and repeats.

Everything else in this book is about making each of these four steps more capable, more reliable, and safer. We will replace pyautogui with more robust automation frameworks. We will add planning so the agent can handle multi-step tasks without losing track. We will add memory so it remembers what it has already done. We will add guardrails so it cannot accidentally delete your home directory. We will add observability so you can understand what went wrong when it inevitably makes a mistake.

But the core loop remains the same. Understanding that loop is the foundation for everything that follows.

Chapter 1: Foundations: LLMs, Agents, and the Control Loop

Before writing more code, we need to establish a shared vocabulary and mental model for what a computer use agent is and how it works. This chapter covers the conceptual foundations that will inform every design decision you make throughout the book. By the end, you will understand why agents are architecturally different from chatbots, what the control loop looks like in practice, and when simple prompting is sufficient versus when you need a full agentic system.

The Agent Abstraction: Perception, Thought, Action

The word "agent" gets used loosely in AI marketing. To be precise, we should use the definition established by Franklin and Graesser in 1997, which remains the standard reference: an agent is an entity that perceives its environment through sensors and acts upon that environment through actuators to achieve goals [5].

This definition is not new. It comes from artificial intelligence research dating back decades. What has changed is that large language models have become capable enough to serve as the "brain" of such agents, handling perception interpretation, reasoning, and action selection in a unified model.

For computer use agents specifically, the mapping looks like this:

- **Sensors (perception):** Screenshots, accessibility trees, DOM snapshots, OCR output, clipboard contents, file system listings.
- **Actuators (action):** Mouse movements, clicks, keystrokes, window management commands, API calls, file operations.
- **Brain (reasoning):** A large language model that takes perceptions and a goal as input and produces an action decision.

A comprehensive survey of LLM-based autonomous agents proposes a unified architecture with four modules: profiling (defining the agent's role),

memory (storing and retrieving context), planning (deciding what to do), and action (executing decisions) [5]. We will build each of these components incrementally throughout this book.

The key insight is that an agent is not a single model call. It is a system that repeatedly calls a model, executes actions in the real world, observes the consequences, and continues. This loop structure is what distinguishes agents from standard chatbots.

Why LLMs Are Different From Previous AI

Understanding why large language models changed the agent landscape requires knowing what came before them. For decades, researchers built specialized agents using reinforcement learning, rule-based systems, or hand-crafted planners. These approaches had clear strengths and equally clear limitations.

Reinforcement learning agents like those trained on Atari games could learn impressive behaviors through trial and error, but they required millions of interactions with a specific environment and did not generalize beyond it. An agent trained to play Breakout could not suddenly play Pong without retraining. Transferring this approach to desktop automation would require training separate agents for every application, which is impractical at scale.

Rule-based systems and traditional RPA (robotic process automation) tools like UiPath or Automation Anywhere excel at repetitive, well-defined workflows. They are fast, reliable, and deterministic. But they break when the interface changes even slightly: a button moves two pixels, a label changes wording, a new confirmation dialog appears. Maintaining these systems across evolving software requires constant human intervention.

LLMs change the calculus because they bring three capabilities that previous approaches lacked:

First, generalization from natural language instructions. You can tell an LLM to "find the settings page and turn on two-factor authentication" without writing a custom script for that specific website. The model uses its pretrained knowledge of how interfaces typically work to figure out the steps.

Second, robustness to variation. If a button's label changes from "Save" to "Save Changes," an LLM can still understand it serves the same function. A rule-based system matching on exact text would fail.

Third, compositional reasoning. LLMs can break down complex instructions into subtasks, handle conditional logic ("if there is an error message, try again"), and adapt their strategy based on what they observe. This flexibility is essential for open-ended tasks where the exact path to completion is not known in advance.

None of this means LLMs solve all the problems. They hallucinate. They make mistakes. They are slow compared to deterministic code. But for tasks that require adapting to novel situations within a familiar domain, they offer capabilities that previous approaches could not match. A survey covering the field from 2021 to 2023 notes that LLM-based agents represent a paradigm shift precisely because they combine language understanding with the ability to acquire and use tools [5].

The Control Loop: Observe, Decide, Act, Repeat

Every computer use agent implements a control loop. The exact structure varies, but the core pattern is universal. Let us walk through one iteration of the loop in detail.

Step 1: Observe. The agent captures the current state of the environment. In the simplest case, this is a screenshot. More sophisticated agents may also extract the accessibility tree from a browser, read clipboard contents, or list files in a directory. The observation is everything the agent can perceive at this moment.

Step 2: Contextualize. The agent combines the observation with relevant context: the original task instruction, what it has already done (memory), any constraints or rules it must follow. This combined information forms the input to the reasoning step.

Step 3: Decide. The LLM processes the contextualized input and produces a decision. This decision typically includes both reasoning (why this action makes sense) and the specific action to take (click at coordinates X,Y, type this text, navigate to this URL).

Step 4: Act. The agent executes the decided action in the real environment. This might mean sending a mouse click event, typing characters, calling an API endpoint, or writing a file.

Step 5: Update. The environment changes as a result of the action. The

agent records what happened (for memory and debugging) and loops back to Step 1 with the new state.

This loop continues until one of several termination conditions is met: the task is complete, the agent determines it cannot make further progress, a maximum step limit is reached, or a human intervenes.

The ReAct framework formalized this loop for LLM agents by explicitly interleaving "Thought" steps (internal reasoning) with "Action" steps (environment interaction) and "Observation" steps (results of actions) [3]. The format looks like this:

```
Thought: I need to open the browser first.
Action: click(x=500, y=800)
Observation: Chrome opened. New tab page is visible.
Thought: Now I should type the URL in the address bar.
Action: type("example.com")
Observation: The address bar now contains "example.com".
...
```

This structure is not just a formatting choice. Research shows that interleaving reasoning with action reduces hallucination and error propagation compared to generating an entire plan upfront without checking intermediate results [3]. The agent can correct itself when observations contradict expectations.

When Prompting Is Enough and When It Is Not

Not every task requires a full agentic loop. Understanding where the line lies will save you from over-engineering simple problems.

Single-step tasks with predictable outcomes are often best handled with direct prompting combined with a tool call. If you need to summarize the content of a web page, you can fetch the HTML, send it to an LLM with a summarization prompt, and get the result in one round trip. There is no need for the agent to "navigate" to the page itself if you can provide the content directly.

Similarly, tasks that map cleanly to structured APIs do not benefit from UI automation. If a system provides a REST API for creating users, querying data, or generating reports, calling that API is faster, more reliable, and cheaper than

having an agent click through a web interface. The rule of thumb: use direct integration whenever it is available and practical.

Agentic control becomes necessary when:

- No API exists for the task (legacy software, third-party services with closed interfaces).
- The task requires visual judgment or interaction with elements that are not exposed programmatically.
- The workflow spans multiple applications that do not share a unified API.
- The exact steps depend on dynamic conditions that cannot be fully anticipated in advance.

A concrete example: automating software testing for a complex web application. You could write Selenium tests for known user journeys, but those tests will miss edge cases that emerge from exploratory testing. An agent that can "explore" the application like a human tester, noticing unexpected behaviors and adapting its approach based on what it sees, complements scripted tests by covering scenarios you did not explicitly program.

Knowing when to use agents versus simpler approaches is a skill that develops with experience. The pattern to watch for: if your solution involves increasingly complex conditionals, brittle selectors, or constant maintenance as interfaces change, an agentic approach may be worth exploring.

Designing the Minimal CUA Architecture

With the foundations established, let us design the minimal architecture that we will incrementally build throughout this book. "Minimal" here means the simplest system that still demonstrates all the essential components of a computer use agent. We will start with this architecture in Chapter 2 and enhance it chapter by chapter.

The minimal architecture has five components:

1. **Environment:** The target system the agent controls. For our prototype, this is a web browser running in a controlled environment (headless mode or within a container). Later chapters will cover desktop environments and multi-application scenarios.

2. **Perception module:** Captures the current state of the environment. Initially, this is screenshot capture. We will later add accessibility tree extraction, OCR, and hybrid perception strategies.
3. **Reasoning engine:** An LLM that takes observations and context as input and produces action decisions. We will start with basic prompting and evolve to sophisticated system prompts, few-shot examples, and structured output formats.
4. **Action executor:** Translates the model's decisions into actual interactions with the environment. Initially using pyautogui for mouse and keyboard events, we will later integrate Playwright for browser-specific control and accessibility APIs for direct element interaction.
5. **Control loop orchestrator:** The glue that ties everything together: captures observations, formats them for the model, parses the response, executes actions, and repeats. This is where we implement the ReAct-style loop, step limits, error handling, and eventually planning and memory systems.

Visually, the data flow looks like this:

```
[Environment] --> (screenshot) --> [Perception]
                                                    \
                                                     v
[Memory + Task] --> [Reasoning Engine] --> (action decision)
     ^                              /                 |
     |                             v                  v
     +------ [Control Loop] <-- [Action Executor] --> [Environment]
```

Each arrow represents data flowing between components. The loop continues until the task completes or a termination condition triggers.

This architecture is intentionally generic. It does not commit to any specific framework, model provider, or automation library. That flexibility is important because the CUA landscape changes rapidly, and you want your design decisions based on principles rather than vendor-specific features. Throughout the book, I will recommend specific tools where they represent strong current choices, but the underlying architecture applies regardless of which tools you select.

The next chapter turns this architecture into working code. We will implement each component in a single Python file, producing an agent that can actually control a browser and complete simple tasks. The code will

be straightforward enough to understand completely and flawed enough to motivate every improvement we make in subsequent chapters.

Chapter 2: Your First Agent: A Working Prototype in Python

Environment Setup: Tools, Libraries, and Dependencies

Capturing the Screen: Your First Perception Pipeline

Sending Screenshots to an LLM

Parsing Actions From Model Output

Executing Clicks, Keystrokes, and Navigation

Running the Prototype End to End

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

What This Prototype Gets Right

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

What This Prototype Gets Wrong

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Chapter 3: Seeing the Interface: Multimodal Perception for UI Understanding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Screenshots Alone Are Not Enough

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Optical Character Recognition: Tesseract, PaddleOCR, and Beyond

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Accessibility Trees and DOM Extraction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Hybrid Perception: Combining Vision, Text, and Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Object Detection for UI Elements

Choosing Your Perception Stack

Chapter 4: Planning: From Single Steps to Multi-Horizon Reasoning

The One-Step Problem: Why Naive Agents Fail on Complex Tasks

Chain of Thought for Task Decomposition

Planning Ahead: Tree Search and MCTS for Agents

Hierarchical Planning: High-Level Goals and Low-Level Actions

Integrating Planning Into the Control Loop

Chapter 5: Memory: Giving Your Agent Context and Continuity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Why Stateless Agents Cannot Do Real Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Short-Term Memory: Conversation History and Context Windows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Episodic Memory: Recording Past Trajectories

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Long-Term Memory: Vector Stores and Retrieval-Augmented Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Procedural Memory: Learning Skills and Routines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Integrating Memory Into the Agent Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Chapter 6: Tools and APIs: Extending Beyond Mouse and Keyboard

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

The Limits of Pure GUI Automation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Structured Tool Calling: OpenAI Functions, Claude Tools, and MCP

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Designing a Tool Schema Your Agent Can Use Reliably

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Browser APIs: Direct DOM Manipulation vs Clicking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

File System, Clipboard, and System-Level Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

The Model Context Protocol (MCP)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

When to Use Tools versus UI Automation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Chapter 7: Action Execution: Reliable Control of Desktop and Browser

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

The Action Space: What Can Your Agent Do?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Mouse and Keyboard: pyautogui, Input Events, and Pitfalls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Browser Automation: Playwright, Selenium, and Puppeteer Compared

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Accessibility APIs: Direct Element Control

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Handling Timing, Loading States, and Race Conditions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Error Recovery When Actions Fail

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Chapter 8: Prompt Engineering for Agents: Designing Robust System Prompts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Why Agent Prompts Are Different From Chat Prompts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

The Anatomy of a Production System Prompt

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Defining the Action Schema Clearly

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Few-Shot Examples That Actually Help

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Constraining Behavior: Guardrails in the Prompt

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Testing and Iterating on Prompts

Chapter 9: State Management: Tracking Progress Through Complex Tasks

The State Problem: How Does the Agent Know Where It Is?

Task Graphs and State Machines

Progress Tracking: Checklists, Milestones, and Completion Criteria

Self-Correction When the Agent Loses Track

Persistence: Saving and Restoring Long-Running Tasks

Chapter 10: Safety and Guardrails: Preventing Catastrophic Mistakes

What Can Go Wrong: A Taxonomy of Agent Failures

Confirmation Flows and Human Oversight

Sandboxing: VMs, Containers, and Isolated Environments

Action Restrictions: Whitelists, Blacklists, and Risk Scoring

Rate Limiting, Budget Caps, and Emergency Stops

Adversarial Considerations and Prompt Injection

Chapter 11: Evaluation: Measuring Whether Your Agent Actually Works

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Why "It Looks Cool" Is Not an Evaluation Metric

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Task Success Rate: Defining What "Done" Means

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Automated Evaluation Harnesses and Test Suites

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Human Evaluation: When You Need a Person in the Loop

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Efficiency Metrics: Steps, Time, and Cost

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Existing Benchmarks: WebArena, OSWorld, and Others

Chapter 12: Debugging and Observability: Understanding Agent Behavior

Why Agents Are Hard to Debug

Structured Logging for Every Step

Tracing Decisions: From Observation to Action

Replay and Recording: Watching the Agent Run Backwards

Visualization Dashboards for Agent Behavior

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Common Failure Patterns and How to Diagnose Them

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Chapter 13: Optimization: Speed, Cost, and Reliability at Scale

The Three Constraints: Latency, Cost, Reliability

Model Selection: When to Use Small Models vs Large Ones

Caching Strategies for Perception and Planning

Parallelizing Independent Subtasks

Infrastructure: GPUs, Edge Devices, and Cloud Costs

Chapter 14: Security: Protecting Your Agent and Your Users

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

The Attack Surface of an Agent System

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

API Key Management and Secrets Handling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Data Privacy: Screenshots, Logs, and User Content

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Secure Execution: Sandboxing, Least Privilege, and Isolation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Transport Security and Integrity Verification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Threat Modeling Your CUA Deployment

Chapter 15: Deployment: From Local Script to Production Service

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Packaging Your Agent for Deployment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Containerization with Docker

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Orchestration: Kubernetes, Serverless, and Managed Services

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Multi-Tenancy: Supporting Multiple Users Safely

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

Monitoring, Alerting, and Incident Response

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr

CI/CD for Agent Systems

Chapter 16: Case Studies: Real-World CUA Implementations

Automated Software Testing: A QA Agent

Customer Support: Agents That Navigate Legacy Systems

Research Assistant: Cross-Tab Data Collection and Synthesis

Personal Productivity: Email, Calendar, and File Management

Lessons Learned Across Deployments

Conclusion: The Next Frontier: Where CUA Technology Is Heading

What We Have Built: A Recap of the Journey

Emerging Paradigms: World Models and Embodied Agents

Agentic Workflows: When Multiple Agents Collaborate

The Human-Agent Interface: Designing for Partnership

Your Next Steps as a CUA Builder

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/computeruseagentsapracticalguidetobuildingaithatcontr