ENGINEERING AUTONOMOUS AI AGENTS

Architecture, Tool Use, Security and Operations

- Agent Architectures
- Tool Use
- Multi-Agent Coordination
- Model Routing
- Security
- Reliability Engineering
- Observability
- Testing
- Deployment Patterns
- Production Operations

Build, secure and operate production-ready AI agents with practical architectures, tool use, reliability, observability and working code.

ETHAN WU

Engineering Autonomous AI Agents

Architecture, Tool Use, Security and Operations

Steve Publications

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

This version was published on 2026-09-15

Contents

Architecture, Tool Use, Security and Operations

This book teaches experienced software engineers, AI/ML engineers, platform engineers, and systems architects how to design, build, secure, deploy, and operate autonomous AI-agent systems in production. It covers the complete lifecycle from fundamental agent architectures through tool use, multi-agent coordination, model routing, reliability engineering, security hardening, observability, testing, deployment patterns, and production operations. Every concept is grounded in working code examples that progressively build toward realistic production-grade systems. The material assumes familiarity with software engineering fundamentals and cloud operations, but introduces AI-specific concepts and terminology clearly. This is not a tutorial collection or a framework survey. It is a production reference for engineers who need their agents to work reliably, stay within budget, resist attack, and survive real-world failure modes.

Introduction

A support ticket arrives at 2 AM. The on-call engineer wakes to find an automated agent that was supposed to restart a failed service has instead drained three credit cards and deleted production logs. The agent was not buggy. It was doing exactly what it was told, in exactly the way it was taught, connecting the dots between natural language instructions and real-world APIs with a literalness that human operators never considered. This is not science fiction. Incidents like this are already happening, and they will become routine as more organizations deploy autonomous systems that can act without human oversight.

The transition from chatbot to autonomous agent is not incremental. It is a category shift that changes everything about how you think about software design. A traditional application takes input, runs a deterministic function, and produces output. An autonomous agent takes a goal, plans a sequence of actions involving external systems, reasons about intermediate results, corrects its own mistakes, and executes decisions that have real-world consequences. That autonomy is precisely what makes agents valuable and precisely what makes them dangerous.

Most engineers approaching agent systems bring two assumptions that will get them in trouble. The first assumption is that if you give the model better instructions, it will just work. The second assumption is that frameworks solve the hard problems for you. Both are wrong. Better instructions help until they do not. Frameworks organize your code but they do not make your system reliable, secure, or cost-effective. The real work in agent engineering is the same work as any production system: designing for failure, containing blast radius, enforcing constraints, monitoring behavior, and iterating based on evidence rather than demos.

This book addresses the full engineering lifecycle of autonomous agent systems. We start with foundational concepts and show how modern agent architectures evolved from earlier LLM application patterns. We cover agent control loops, reasoning strategies, planning approaches, memory architectures, and context management. We go deep into tool use, the mechanism that connects model reasoning to real-world action, including how to build tools for

agents, how to design schemas that models can actually use reliably, and how protocols like Model Context Protocol (MCP) are changing interoperability.

We compare single-agent and multi-agent architectures, when each applies, and why multi-agent systems are far more complex than marketing materials suggest. We cover model selection and routing strategies that can reduce costs by 20 to 30 percent without sacrificing quality. We discuss prompt design not as a craft of clever phrasing but as a discipline of reliable specification.

The reliability section applies classic distributed systems engineering to a domain that was built on probability instead of certainty. We cover retries, timeouts, idempotency, checkpointing, circuit breakers, and graceful degradation in systems where the core component is fundamentally non-deterministic. The security chapters address the novel attack surfaces that agents introduce: prompt injection in all its forms, tool poisoning, memory corruption, data exfiltration through tool calls, and privilege escalation through reasoning manipulation. We cover sandboxing, isolation, zero-trust principles, and policy enforcement as they apply specifically to agentic systems.

The operational chapters cover observability, testing, red teaming, deployment architectures, scaling, cost optimization, and governance. We compare tools and frameworks not by feature lists but by production trade-offs: what you gain, what you lose, and when each choice makes sense.

Throughout the book, you will find complete, runnable code examples in Python. These are not toy demonstrations. They are production-adjacent: they handle errors, validate inputs, implement retry logic, log decisions, and respect security boundaries. The examples progressively build toward realistic agent systems so you can see how individual concepts connect into working architectures.

Some warnings before we begin. This field moves fast. Specific API behaviors, pricing, feature availability, and framework capabilities will change. We flag areas that are rapidly evolving and distinguish stable patterns from experimental ones. Security guidance will always lag behind the latest attacks; we present what we know now with the understanding that the threat landscape shifts continuously. Model capabilities improve over time, which changes what is feasible but does not change the fundamental engineering challenges of building systems that are reliable, safe, and cost-effective at scale.

The central thesis of this book is that autonomous agent systems are pro-

duction software systems that happen to use foundation models as a control plane. Everything you already know about building reliable, secure, observable, scalable software still applies. The new elements are the non-determinism, the natural language interface, the ability to reason and plan, and the novel attack surfaces that emerge when a stochastic process controls access to your systems. Engineer for those realities and you will build systems that actually work in production.

Chapter 1: The Evolution from LLM Applications to Agentic Systems

Understanding how we got to autonomous agents is essential for understanding what they are and what problems they solve. The transition was not sudden. It emerged from practical experience with LLM applications over roughly three years of rapid iteration. Each generation solved the limitations of the previous one until systems evolved from passive responders into active executors.

Single-Shot Prompting: The Baseline

The earliest LLM applications used a single round of interaction: the user submitted a prompt, the model generated a response, and the exchange ended. This pattern worked well for tasks like translation, text summarization, classification, and creative writing where the model could produce the entire answer from its training knowledge without needing external information or multiple steps.

The architecture was simple. Your application constructed a prompt string, called the model API, parsed the response, and returned it to the user. Error handling meant retrying if the API failed or regenerating if the output was unsatisfactory. There was no state between calls, no conversation history, no ability to correct mistakes mid-generation.

This simplicity was a strength. The system was easy to understand, easy to test, and deterministic in the sense that given the same prompt and temperature, you could reproduce results. Latency was bounded by a single API call. Cost was predictable: one response per user request.

The limitation was equally clear. The model could only work with what it knew from training. It could not look up current information, access your databases, call your services, or verify its own answers. If you asked it to generate a SQL query for your schema, it would hallucinate table names. If you asked it to summarize today's news, it would fabricate stories. The model was a powerful reasoning engine trapped in a sealed room.

Conversational Interfaces and Multi-Turn Design

The first major evolution added conversation history to the context. Instead of sending a single prompt, your application maintained a list of messages representing the dialogue and sent the entire history with each new turn. This enabled follow-up questions, clarification, and progressive refinement without repeating yourself.

Conversational interfaces were transformative for user experience but architecturally straightforward. Your application stored the message list in a database or cache, appended each new user message and model response, and sent the growing history with every turn. The model could now reference earlier parts of the conversation, maintain topic continuity, and build on previous reasoning.

Multi-turn conversations introduced new engineering considerations. Context windows were finite, so long conversations eventually exceeded the limit. You had to implement strategies for truncating older messages, summarizing conversation history, or moving context to secondary storage. Cost grew with every turn because you sent the entire history each time. Latency could increase as the context grew longer.

More fundamentally, conversations exposed the difference between appearance and capability. A model that could chat fluently about debugging your code was not actually running or testing that code. It could simulate expertise without access to reality. Users began to expect more: not just to talk about a task but to have the model actually perform it.

Tool-Augmented Models and Function Calling

The next evolution gave models access to external tools. This is the pattern where your application presents the model with a set of available functions, each with a name, description, and parameter schema. The model decides whether to call a tool, which tool to use, and what arguments to pass. Your application executes the tool, captures the result, feeds it back to the model, and the model incorporates the result into its response.

OpenAI introduced this pattern with function calling in their API. Anthropic called it tool use. Google Gemini implemented it as function declarations. The

underlying concept is the same: the model outputs structured data describing a tool invocation instead of natural language text, your code runs the actual tool, and the model continues with the result.

Here is what a tool call cycle looks like at the API level. Your request includes the conversation history plus tool definitions:

```
import openai
import json

client = openai.OpenAI(api_key="sk-...")

tools = [
    {
        "type": "function",
        "function": {
            "name": "get_weather",
            "description": "Get the current weather for a city",
            "parameters": {
                "type": "object",
                "properties": {
                    "city": {
                        "type": "string",
                        "description": "The city name, e.g. San Francisco"
                    }
                },
                "required": ["city"],
                "additionalProperties": False
            }
        }
    }
]

messages = [
    {"role": "user", "content": "What is the weather in Paris?"}
]

response = client.chat.completions.create(
    model="gpt-4o",
    messages=messages,
    tools=tools,
    tool_choice="auto"
)

assistant_msg = response.choices[0].message

if assistant_msg.tool_calls:
    # Model wants to call a tool
    for tool_call in assistant_msg.tool_calls:
```

```
        function_name = tool_call.function.name
        arguments = json.loads(tool_call.function.arguments)

        # Execute the actual tool
        if function_name == "get_weather":
            result = fetch_weather_from_api(arguments["city"])

        # Feed result back to the model
        messages.append(assistant_msg)
        messages.append({
            "role": "tool",
            "tool_call_id": tool_call.id,
            "content": str(result)
        })

        # Continue the conversation with the result
        response = client.chat.completions.create(
            model="gpt-4o",
            messages=messages,
            tools=tools
        )
```

This pattern is the foundation of everything that follows. Tool use lets models ground their responses in reality. They can look up data, call APIs, run queries, and integrate with your systems. The model becomes a reasoning layer that orchestrates access to your actual capabilities.

Tool use also introduced new complexity. You had to handle the multi-step cycle of tool invocation and result feedback. You had to validate arguments, handle tool failures, and decide what to do when the model requested an unavailable tool or passed invalid parameters. Cost and latency multiplied with each tool call cycle.

Most critically, tool use created a security boundary that did not exist before. Your model was now making decisions about what external actions to take. If the model could call a delete function on your database, it would, if instructed or manipulated into doing so. The trust model shifted from "the model produces text" to "the model controls actions in my systems."

The Leap to Autonomy: When Tools Become Decisions

The transition from tool-augmented to autonomous is subtle but fundamental. In a tool-augmented application, the user initiates every interaction. The user

asks a question, the model may call tools to answer it, and the conversation continues or ends based on user input. The model is reactive.

An autonomous agent operates differently. It receives a high-level goal and pursues it independently. It decides what tools to call, in what order, when to verify intermediate results, when to correct mistakes, and when the task is complete. It may run for minutes, hours, or days without direct user interaction. It makes its own decisions about how to proceed.

This autonomy emerges from combining tool use with a control loop. The agent loop typically follows this pattern:

1. **Observe:** Check the current state of the environment, review past actions and results.
2. **Think:** Analyze the situation, decide what to do next.
3. **Act:** Execute a tool call or other action.
4. **Reflect:** Evaluate the result, adjust the plan if needed.
5. **Repeat** until the goal is achieved or a termination condition is met.

This is often called the ReAct pattern, standing for Reason and Act, though the specific terminology varies. The key insight is that the model's thinking becomes part of a control loop rather than a one-shot response. Each iteration uses the results of previous actions to inform the next decision.

Anthropic's research on building effective agents emphasizes five patterns that enable autonomy without unnecessary complexity: prompt chaining for fixed sequential tasks, routing for specialized inputs, parallelization for speed and confidence, orchestrator-worker for dynamic delegation, and evaluator-optimizer loops for iterative refinement [1]. These are not novel concepts in software engineering. They are classic control patterns applied to LLM orchestration.

Why Agents Are Fundamentally Different Systems

Autonomous agents are not just fancier chatbots. They are a different class of system with different properties and different engineering challenges.

First, agents are non-deterministic in a way that breaks traditional testing assumptions. You cannot write a unit test that expects the agent to produce a specific sequence of tool calls for a given input. Two runs with identical inputs

may take different paths to the same goal. This is not a bug. It is a consequence of probabilistic generation. Testing must focus on outcomes and constraints rather than exact behavior.

Second, agents introduce long-tail latency and cost distributions. A simple question might be answered with one model call. A complex task might require dozens of tool calls, multiple planning iterations, and hours of execution. You cannot provision or budget for agents using average-case analysis. You need strategies for bounding execution: step limits, time budgets, token budgets, and cost thresholds.

Third, agents operate at the intersection of symbolic and statistical computation. The model reasons probabilistically about what to do next. The tools execute deterministically in the real world. Bugs can emerge from the interaction between these layers in ways that are hard to trace. A malformed tool response might cause the model to make an invalid decision that cascades through subsequent steps.

Fourth, agents create novel security surfaces. Traditional web applications parse structured input. Agents process natural language from users, external systems, retrieved documents, and their own prior reasoning. Every source of natural language input is a potential vector for prompt injection. Every tool the agent can call is a potential target for privilege escalation. Every document retrieved into context is a potential carrier of hidden instructions.

The shift to agents is the shift from using LLMs as features to using them as control planes. The engineering discipline required is correspondingly deeper. The rest of this book shows how to meet that discipline.

Chapter 2: Agent Architectures and Control Loops

The architecture of an agent system determines how control flows through it, how decisions are made, how tools are accessed, and how failures are handled. There is no single correct architecture. Different patterns apply to different use cases, and the best choice depends on your requirements for latency, reliability, cost, complexity, and control.

The Basic Agent Loop: Observe, Think, Act, Reflect

At the foundation of every autonomous agent is a control loop. The simplest form cycles through observation, reasoning, action, and feedback:

1. The agent observes the current state: what has happened so far, what tools are available, what the goal is.
2. The agent reasons about what to do next: it analyzes the situation, considers options, and selects an action.
3. The agent executes the chosen action: typically a tool call.
4. The environment responds: the tool returns a result, succeeds or fails.
5. The agent reflects on the outcome and continues the loop or terminates.

This loop can be implemented directly with a model that supports tool calling. You give the model access to tools and instructions about its goal. The model calls tools as needed. Your application runs each tool, feeds back the result, and the model continues until it has completed the task or reached a step limit.

Here is a minimal implementation of this loop:

```python
import openai
import json
import sys
from typing import Dict, Any, Callable, List

class Agent:
    def __init__(self, model: str = "gpt-4o", max_steps: int = 20):
        self.client = openai.OpenAI()
        self.model = model
        self.max_steps = max_steps
        self.tools: Dict[str, Callable] = {}
        self.tool_schemas: List[Dict[str, Any]] = []

    def register_tool(self, name: str, description: str,
                      parameters: Dict[str, Any], func: Callable):
        """Register a tool with name, description, schema, and
        ↪  implementation."""
        self.tools[name] = func
        self.tool_schemas.append({
            "type": "function",
            "function": {
                "name": name,
                "description": description,
                "parameters": parameters
            }
        })

    def run(self, goal: str, system_prompt: str = None) -> str:
        """Run the agent loop to achieve the goal."""
        if not system_prompt:
            system_prompt = (
                "You are an autonomous agent. Use available tools to complete
                ↪  your task. "
                "Think step by step. Call tools when you need information or
                ↪  need to take action. "
                "Return a final answer when done."
            )

        messages = [
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": goal}
        ]

        for step in range(self.max_steps):
            response = self.client.chat.completions.create(
                model=self.model,
                messages=messages,
                tools=self.tool_schemas,
                tool_choice="auto",
                temperature=0.2
```

```
            )

            message = response.choices[0].message

            if message.tool_calls:
                # Execute each tool call
                for tool_call in message.tool_calls:
                    function_name = tool_call.function.name
                    arguments = json.loads(tool_call.function.arguments)

                    messages.append(message)

                    try:
                        if function_name not in self.tools:
                           result = f"Error: Tool '{function_name}' not found."
                        else:
                            result = self.tools[function_name](**arguments)
                    except Exception as e:
                        result = f"Error executing {function_name}: {str(e)}"

                    messages.append({
                        "role": "tool",
                        "tool_call_id": tool_call.id,
                        "content": str(result)
                    })
            else:
                # No tool calls means the agent is done
                return message.content

        raise RuntimeError(f"Agent exceeded maximum steps ({self.max_steps})")

# Usage example
agent = Agent()

agent.register_tool(
    name="search_web",
    description="Search the web for current information",
    parameters={
        "type": "object",
        "properties": {
            "query": {"type": "string", "description": "The search query"}
        },
        "required": ["query"],
        "additionalProperties": False
    },
    func=lambda query: f"Search results for '{query}' would appear here"
)

agent.register_tool(
    name="read_file",
```

```
    description="Read the contents of a file",
    parameters={
        "type": "object",
        "properties": {
            "path": {"type": "string", "description": "File path to read"}
        },
        "required": ["path"],
        "additionalProperties": False
    },
    func=lambda path: f"Contents of {path} would appear here"
)

result = agent.run("Find the latest Python version release notes and summarize
↪  the key changes.")
print(result)
```

This is the baseline pattern. Everything else in agent architecture is a variation on how to structure, constrain, and extend this loop.

Event-Driven vs Request-Response vs Streaming Architectures

How your agent system receives input and produces output shapes its architecture significantly.

Request-response is the simplest pattern. A client sends a task, the agent processes it through its control loop, and returns a result. This works well for batch tasks: process this document, generate this report, answer this question. The architecture is straightforward but does not handle interactive or ongoing tasks well.

Event-driven architectures treat agent tasks as events in a larger system. An event (a new support ticket, a deployment failure, a user request) triggers agent execution. The agent may run asynchronously, emit events as it progresses, and complete without blocking the caller. This pattern scales better and integrates naturally with existing event-driven infrastructure: message queues, event buses, workflow systems. The trade-off is added complexity in tracking agent state and coordinating with other system components.

Streaming architectures expose agent activity in real time. Instead of waiting for the agent to complete, the system streams intermediate outputs: reasoning steps, tool calls, partial results. This is valuable for interactive

interfaces where users want to see the agent working, for debugging and observability, and for long-running tasks where users need feedback. Streaming requires additional infrastructure to capture and deliver intermediate events without blocking the main execution loop.

In production, most systems combine these patterns. An agent receives an event from a queue, executes its task asynchronously, streams progress updates through websockets or SSE, and emits completion events back into the system.

Sequential Chain vs Parallel Fan-Out vs Recursive Decomposition

Within the agent loop, decisions about how to structure work flow into three primary patterns.

Sequential chains execute steps one after another, with each step's output feeding the next step's input. This is the simplest pattern and works when the task naturally decomposes into a fixed order of operations. Example: extract data from a document, transform it, load it into a database, generate a summary report. Each step depends on the previous one, so parallelization is not possible.

Sequential chains are easy to reason about and debug. They have predictable latency (sum of individual step latencies) and predictable cost. They work best when the sequence is known in advance rather than determined dynamically by the model.

Parallel fan-out executes multiple operations simultaneously and aggregates the results. This pattern is useful when you can break a task into independent subtasks: search multiple sources, query multiple databases, generate multiple drafts for comparison. The latency is determined by the slowest parallel branch rather than the sum of all branches. Cost may be higher due to concurrent model calls but total wall-clock time is reduced.

Parallelization is particularly valuable for improving confidence. If you ask a model to generate multiple independent analyses of the same data and then compare results, you can identify disagreements and surface them for review. Anthropic's research shows that parallelized methods are simpler and more maintainable than complex multi-agent setups for many use cases [1].

Recursive decomposition has the agent break a complex task into subtasks, solve each subtask (possibly recursively), and combine the results. This pattern handles tasks whose structure is not known in advance and requires dynamic planning. Example: a software engineering task where the agent identifies files to modify, generates changes for each file, tests the changes, fixes failures, and repeats.

Recursive decomposition is powerful but risky. Without careful constraints, the agent can decompose tasks too deeply, creating exponentially growing work. Implementing depth limits, step budgets, and convergence checks is essential. This pattern also makes debugging harder because the execution tree can become large and complex.

Many production agents use a hybrid approach: a planned structure with sequential and parallel branches, but with the flexibility for the agent to dynamically adjust within that structure based on intermediate results.

Supervisor-Subordinate Patterns

In more complex systems, a single model handles too much responsibility. The supervisor-subordinate pattern divides labor: a supervisor agent coordinates and delegates, while subordinate agents specialize in specific tasks or domains.

The supervisor receives the overall task, decomposes it, assigns subtasks to appropriate specialists, collects results, and produces the final output. Subordinates focus on their domain: one might handle data retrieval, another code generation, another quality review.

This pattern is valuable when:

- The task requires expertise across multiple domains
- Different subtasks benefit from different models or prompts
- You want to contain errors within a specific component

However, supervisor-subordinate architectures add overhead. The supervisor must correctly assess what each subordinate can do. Communication between agents consumes tokens and adds latency. Debugging becomes harder because a failure might originate in the supervisor's planning, a subordinate's execution, or the integration between them.

The key question is whether specialization actually improves outcomes or just adds complexity. Anthropic's guidance emphasizes starting with a single well-tooled agent and only adding layers when evaluation shows measurable benefit [1].

Reactive Agents vs Deliberative Agents

Agent design also involves a trade-off between reactivity and deliberation.

Reactive agents respond directly to their current observation with minimal planning. They maintain a simple policy: if X happens, do Y. This pattern is fast and robust because there is no planning overhead and no risk of plan failure. Reactive agents work well for well-defined domains with predictable events.

Deliberative agents plan before acting. They analyze the goal, consider possible action sequences, select a plan, and execute it while monitoring for changes. This pattern handles complex, uncertain domains better because the agent can reason about trade-offs and alternatives. The trade-off is latency and computational cost. Deliberation requires tokens, time, and careful design to avoid runaway reasoning loops.

Most production agents use a middle ground: deliberate planning for high-level strategy combined with reactive patterns for routine operations. A customer support agent might plan its approach for a complex escalation but follow standard procedures for routine queries.

Trade-Offs Summary

Every architectural choice involves trade-offs. Here is a practical summary:

Single-agent loop – General tasks; medium latency/cost; low complexity; medium reliability. Sequential chain – Fixed-order workflows; additive latency; predictable cost; low complexity; high reliability. Parallel fan-out – Independent subtasks; latency set by slowest branch; higher cost; medium complexity; high reliability. Recursive decomposition – Unknown structure; unpredictable latency/cost; high complexity; low reliability without limits. Supervisor-subordinate – Multi-domain tasks; higher latency/cost; high complexity; medium reliability. Event-driven – Integration and scale; asynchronous; efficient cost; medium complexity; high reliability.

The engineering principle here is the same as in traditional systems: choose the simplest architecture that meets your requirements, then add complexity only when evidence shows it is necessary.

Chapter 3: Reasoning, Planning, and Reflection

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

Chain-of-Thought and Its Production Variants

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

Tree of Thoughts, Graph of Thoughts, and Planning Trees

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

Backtracking and Self-Correction Loops

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

Reflection Patterns: Self-Critique, Outcome Analysis, Iterative Improvement

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

Decomposition Strategies: Top-Down, Bottom-Up, Mixed-Initiative

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

When Reasoning Helps vs When It Adds Cost Without Value

Chapter 4: Memory, State Management, and Context Engineering

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

Short-Term vs Long-Term Memory Architectures

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

Vector Databases for Semantic Memory

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

Working Memory: Conversation State, Intermediate Results, Scratchpad

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

State Persistence: Databases, Queues, Event Sourcing

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

Context Window Management: Compression, Summarization, Selective Retrieval

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

Context Engineering: What to Include, What to Exclude, Ordering Effects

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

Memory Poisoning Attacks and Data Integrity

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

Chapter 5: Tool Use and Function Calling

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

The Function Calling Protocol Across Major Providers

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

Schema Design: Types, Descriptions, Constraints, Examples

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

Tool Discovery and Dynamic Registration

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

Error Handling and Retry Semantics for Tool Calls

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

Parsing Structured Outputs: Strict Mode vs Lenient Mode

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

Tool Composition: Pipelines, Parallel Execution, Conditional Branching

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

Latency and Cost of Tool Calls in Multi-Step Workflows

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

Chapter 6: Building Tools for Agents: APIs, Shells, Files, Browsers

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

Designing APIs for Agents vs Humans

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

File System Operations: Listing, Reading, Writing, Diffing

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

Shell and Command Execution: Parsing, Streaming Output, Handling Failures

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

Database Interactions: SQL Generation, Result Formatting, Safety Constraints

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

Browser Automation for Agents

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

Rate Limiting, Authentication, and Error Semantics for External Services

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

Chapter 7: Model Context Protocol and Interoperability

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

What MCP Is and Why It Was Created

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

MCP Architecture: Servers, Clients, Transport Layers

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

MCP Resource and Tool Primitives

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

Building MCP Servers for Custom Tools and Data Sources

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

Comparing MCP to Alternative Approaches

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

Multi-Vendor and Multi-Model Interoperability Challenges

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

Limitations and What MCP Does Not Solve

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

Chapter 8: Single-Agent vs Multi-Agent Architectures

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

Single Agent: Monolithic Reasoning with Tool Use

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

Multi-Agent Patterns: Specialization, Debate, Hierarchy, Marketplace

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

Routing Tasks to Specialized Agents

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

Agent Communication Protocols and Message Formats

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

Coordination: Shared State, Consensus, Deadlock Avoidance

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

When Multi-Agent Adds Value vs When It Adds Pointless Complexity

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

Cost, Latency, and Debugging Implications

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

Chapter 9: Model Selection, Routing, and Inference

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

Model Capabilities: Reasoning, Coding, Vision, Multilingual, Tool Use

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

Model Routing Strategies: Cost-Based, Capability-Based, Latency-Based

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

Fallback and Escalation Patterns

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

Structured Outputs: JSON Mode, Strict Schemas, Validation

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

Token Budgets and Cost Estimation

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

Caching Strategies for Inference

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

Running Models Locally vs Remotely

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

Chapter 10: Prompt and Context Design

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

System Prompts: Structure, Specificity, Guardrails

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

Few-Shot Examples: Selection, Formatting, Maintenance

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

Instruction Hierarchy: What Overrides What

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

Output Formatting Requirements That Actually Work

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

Prompt Templates vs Dynamic Prompt Generation

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

Testing Prompt Changes: A/B, Evaluation Harnesses

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

Prompt Drift, Regression, and Version Control

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

Chapter 11: Reliability Engineering for Agent Systems

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

Non-Determinism and Reproducibility Challenges

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

Retries with Exponential Backoff and Jitter

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

Timeouts: Per-Step, Per-Operation, Per-Session

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

Idempotency: Making Actions Safe to Repeat

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

Checkpointing and State Recovery

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

Circuit Breakers for Failing Tools and Models

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

Deadlocks, Livelocks, and Infinite Loops

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

Graceful Degradation Strategies

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

Chapter 12: Security Foundations for Agent Systems

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

Why Agents Are Security Nightmares by Default

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

Identity and Access Management for Agents

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

Secrets Management and Key Rotation

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

Zero-Trust Principles Applied to Agent Architectures

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

Least Privilege for Tool Access

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

Authentication Flows: API Keys, OAuth, mTLS

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

Data Classification and Handling Requirements

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

Security as a Design Constraint, Not a Bolt-On

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

Chapter 13: Sandboxing and Isolation

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

The Need for Isolation: Agents Can and Will Escape Constraints

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

Container-Based Isolation: Docker, gVisor, Firecracker

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

Virtual Machine Isolation for High-Risk Operations

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

Filesystem Isolation: Read-Only Roots, Ephemeral Storage, Volume Mounts

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

Network Isolation: Egress Filtering, DNS Control, Proxy Enforcement

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

CPU and Memory Limits: Preventing Resource Exhaustion

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

Process Namespaces and Capability Dropping

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

Choosing Isolation Level Based on Risk

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

Chapter 14: Agent-Specific Attack Vectors

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

Prompt Injection: Direct, Indirect, Nested, Image-Based

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

Tool Poisoning and Malicious Tool Registration

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

Data Exfiltration Through Tool Calls and Outputs

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

Privilege Escalation Through Reasoning Manipulation

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

Supply-Chain Attacks: Malicious Packages, Dependencies, Datasets

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

Jailbreaking and Guardrail Evasion

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

Indirect Injection Through Retrieved Context

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

Mitigations Summary

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

Chapter 15: Policy Enforcement and Guardrails

Input Filtering and Sanitization

Output Validation and Filtering

Runtime Policy Enforcement: OPA, Custom Policy Engines

Content Safety: Toxicity, PII, Copyright, Regulated Data

Action Authorization: What Tools Can Be Called Under What Conditions

Rate Limiting and Quota Enforcement

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

Human Approval Workflows for High-Risk Actions

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

Guardrail Performance: Latency Impact, False Positives

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

Chapter 16: Observability, Tracing, and Monitoring

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

The Observability Challenge: Non-Deterministic, Multi-Step, Multi-Component

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

Logging: What to Log, When, at What Verbosity, Structured Formats

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

Tracing: Distributed Traces Across Model Calls and Tool Invocations

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

Metrics: Latency, Cost, Success Rates, Error Patterns, Token Usage

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

Dashboards and Alerting

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

Debugging Specific Agent Failures: Replay, State Inspection

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

Audit Trails for Compliance and Forensics

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

Privacy and Data Retention in Logs

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

Chapter 17: Testing, Evaluation, and Red Teaming

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

Unit Testing Individual Components

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

Integration Testing: End-to-End Agent Workflows

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

Evaluation Harnesses: Accuracy, Completeness, Safety

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

Automated Evaluation with LLM-as-Judge

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

Human Evaluation Processes

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

Red Teaming Methodologies for Agents

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

Regression Testing for Prompt and Model Changes

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

Chaos Engineering for Agent Systems

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

Continuous Evaluation in Production

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

Chapter 18: Deployment Architectures and CI/CD

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

Deployment Models: Serverless, Containerized, Bare Metal

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

Infrastructure as Code for Agent Platforms

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

CI/CD Pipelines for Agent Systems: Testing Prompts, Tools, Models

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

Environment Separation and Promotion Gates

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

Blue/Green and Canary Deployments for Agent Features

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

Configuration Management: Environment-Specific Settings

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

Rollback Strategies for Failing Agent Deployments

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

Chapter 19: Scaling, Performance, and Cost Optimization

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

Scaling Patterns: Horizontal Scaling of Agent Workers, Tool Servers

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

Queue-Based Task Processing

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

Batch Processing vs Real-Time Processing

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

Latency Optimization: Streaming, Early Exit, Parallel Tool Calls

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

Token Optimization: Context Reduction, Efficient Prompting

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

Cost Monitoring and Attribution

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

Auto-Scaling Based on Queue Depth and SLAs

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

Caching Layers for Repeated Operations

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

Chapter 20: Governance, Compliance, and Human-in-the-Loop

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

Regulatory Landscape: EU AI Act, Sector-Specific Requirements

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

Documentation and System Cards

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

Incident Response for Agent Failures

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

Human Oversight: Escalation, Approval, Correction Workflows

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

Change Management for Agent Systems

Ethical Considerations and Algorithmic Accountability

Vendor Risk Management for Model Providers

Building a Culture of Responsible Agent Engineering

Conclusion: Principles for Building Agent Systems

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

Ten Principles for Production Agent Engineering

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

What We Know Now vs What Remains Uncertain

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

The Trajectory: Where Agent Systems Are Headed

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

Skills and Mindset Shifts for Engineers

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

Final Recommendations

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

References

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