

BUILDING AI AGENTS THAT ACTUALLY WORK WITH DEEPSEEK HARNESS

A Production Engineering Guide to Reliable,
Secure, Observable Agents on DeepSeek Models



SECURE



OBSERVABLE



RELIABLE



BUDGET-AWARE

RILEY DONOVAN

Building AI Agents That Actually Work with DeepSeek Harness

A Production Engineering Guide to Reliable, Secure, Observable Agents on DeepSeek Models

Steve Publications

This book is available at <https://leanpub.com/buildingaiagentsthatactuallyworkwithdeepseekharness>

This version was published on 2026-10-06



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2026 Steve Publications

Contents

- A Production Engineering Guide to Reliable, Secure, Observable Agents on DeepSeek Models 1**
- Introduction: The Reliability Gap 3**
 - What a harness is 3
 - What DeepSeek Harness actually is 4
 - The thesis 5
 - The reference implementation 6
 - How to read this book 8
- Part I: Foundations and Architecture 9**
- Chapter 1: What an Agent Is and When You Should Not Build One 10**
 - The loop, in twelve lines of pseudocode 10
 - Workflows versus agents: who holds the control flow 12
 - The four-way trade-off: capability, cost, latency, variance 13
 - A decision procedure and four escape hatches 14
 - Anti-patterns, part one 15
 - Pulling the threads together 16
- Chapter 2: DeepSeek Models as an Engineering Substrate 17**
 - Why the serving cost is low and what that buys you 17
 - Two models, two control planes 18
 - Reasoning modes and the invariant that breaks naive loops 19
 - Tool calling on the wire 21
 - One million tokens is not an excuse 22
 - Hosted-endpoint failure modes 23
 - Pulling the threads together 24
- Chapter 3: Inside the Harness. Architecture of DeepSeek Harness . . . 26**
 - Cordis in one page 26

CONTENTS

Component map	26
The life of a turn	26
Three event planes	26
Extension points, ranked by how much you will use them	26
Where the harness stops	26
Reading the source without drowning	27
Pulling the threads together	27
Chapter 4: First Agent. Scaffold, Config and the Loop	28
Repository layout and dependency pinning	28
Typed settings and environment configuration	28
The client wrapper	28
Running one turn and reading what happened	28
Structured output is a reliability primitive, not a convenience	28
A Makefile you will actually use	28
A debugging session, because it will be your first one	29
Pulling the threads together	29
Chapter 5: Tools. Contracts, Registries and Consequences	30
The split that defines your architecture	30
The tool contract in the harness	30
The result envelope	30
Registry, scoping and least capability	30
Parallel tool calls and why ordering is a trap	30
Idempotency, or how to survive a retry you did not choose	30
Error taxonomy, the highest-leverage page in the book	31
A code review, abridged	31
Pulling the threads together	31
Chapter 6: Context Engineering and Token Budgets	32
Anatomy of a prompt	32
Token accounting you can trust	32
The compaction ladder	32
Prompt caching is an architecture, not a flag	32
What long context actually costs you	32
Diagnosing a context problem from traces	32
Pulling the threads together	33
Chapter 7: Memory and State	34
Three kinds of state and why conflating them is the bug	34

CONTENTS

Conversation state: what the harness already does	34
Task state: the part nobody designs	34
Checkpointing and resumability	34
Goals, todos and the difference between them	34
Durable memory that does not poison	34
Retrieval-backed memory versus always-in-context memory	35
Anti-patterns, part two	35
Pulling the threads together	35
Chapter 8: Retrieval and Knowledge Bases	36
Retrieval is a tool, not a pipeline	36
Chunking that respects structure	36
Hybrid search, in the database you already have	36
Embeddings, reranking and their costs	36
Permissions are a retrieval problem	36
Evaluating retrieval on its own	36
When retrieval is the wrong answer	37
Pulling the threads together	37
Chapter 9: Planning and Reasoning	38
Thinking mode is a cost dial, not a quality switch	38
The effort ladder and where it surprises you	38
The parameters that silently do nothing	38
The echo invariant and why a harness is the place to fix it	38
Planning: three mechanisms and which one to use	38
Plan-then-execute is usually the wrong shape	38
Structured output while thinking	39
Reading reasoning without believing it	39
Pulling the threads together	39
Chapter 10: Orchestration, Queues and Long-Running Agents	40
Two loops and the seam between them	40
The queue and why not the harness scheduler	40
Long-running agents	40
Background jobs and subprocesses the harness already manages	40
Event-driven wakeups	40
Cancellation that actually works	40
Pulling the threads together	41
Chapter 11: Multi-Agent Systems and When Not to Have Them	42

CONTENTS

The tax, stated precisely	42
What the harness actually gives you	42
Permissions are the real reason to delegate	42
The workflow engine and its one sharp edge	42
Three patterns that earn their cost	42
Patterns that do not	42
Debugging a delegation tree	43
Pulling the threads together	43
Chapter 12: Coding and Computer-Use Agents	44
The filesystem contract is the interesting part	44
Search strategy: grep is not a fallback	44
Shell discipline	44
Postmortem 0003: the verification gap	44
Postmortem 0004: attribution requires independent evidence	44
Computer use and browser use	44
Pulling the threads together	45
Part II: Reliability and Security	46
Chapter 13: Containment. Budgets, Loops and Runaway Agents	47
The reliability contract	47
Deterministic boundaries	47
Five independent ceilings	47
Infinite loops: four signatures	47
Runaway agents and the blast-radius ledger	47
Timeouts at every layer and the gap	47
Idempotency and transactional effects	48
What the harness gives you and what it does not	48
Pulling the threads together	48
Chapter 14: Recovery. Retries, Fallbacks and Degraded Modes	49
A closed vocabulary of failure	49
Retries and the retry budget	49
The 429 arithmetic	49
Circuit breakers for tools	49
Model fallback and what it costs you	49
Degraded modes, defined in advance	49
Dead letters and the human queue	50
Concurrency control and the write-conflict problem	50

Pulling the threads together	50
Chapter 15: Security I. Injection, Taint and Exfiltration	51
The threat model, in one page	51
Prompt injection is an authority problem	51
How good are the defences, honestly?	51
Taint tracking: the mechanism nobody ships	51
Exfiltration: enumerate the channels	51
SSRF, the agent-shaped version	51
Malicious tool results, by name	52
Pulling the threads together	52
Chapter 16: Security II. Isolation, Credentials and Tenancy	53
What the sandbox is and what it is not	53
Permission presets: the unit you should be configuring in	53
The profile trap	53
Credentials: the model must never hold one	53
Network isolation: the container is the boundary	53
Tenant isolation, three ways	53
Audit logging that survives the incident	54
Supply chain: plugins are code with your permissions	54
The checklist and the assumption to make	54
Pulling the threads together	54
Chapter 17: Evaluation. Knowing Whether It Got Better	55
The measurement problem, stated honestly	55
Your harness is a variable, not a constant	55
Three tiers of assertion	55
Building the eval set from production, not from imagination	55
Regression detection: what changed and how do you know	55
Replaying a trajectory and why the harness makes it possible	55
Online evaluation: the part the eval set cannot do	56
Anti-patterns in evaluation	56
Pulling the threads together	56
Chapter 18: Observability. Traces, Trajectories and Cost	57
The session log is the trace	57
Spans and the vocabulary that maps onto them	57
Structured logs, with the right keys	57
The OTel seam and its channel discipline	57

Redaction: the seam that ships empty	57
Trajectory inspection: the tool that pays for itself	57
Failure classification in production	58
Debugging sessions: four real shapes	58
Pulling the threads together	58
Part III: Performance, Deployment and Operations	59
Chapter 19: Performance and Cost Engineering	60
The cost model, worked end to end	60
Prompt caching as an architectural constraint	60
Reasoning tokens: the biggest single lever	60
Latency and where it actually goes	60
Throughput and the concurrency ceiling	60
Self-hosting: do the arithmetic before the architecture	60
Quantisation and where it stops being free	61
Total cost of ownership, honestly	61
Pulling the threads together	61
Chapter 20: Deployment. Containers, Configuration and Graceful Shutdown	62
What you are actually deploying	62
The image	62
Configuration: the layering trap, revisited operationally	62
Secrets	62
Health checks that mean something	62
Graceful shutdown, the actual problem	62
Rolling deployments and version skew	63
Local deployment, because some of this runs on a laptop	63
Backups and the recovery drill	63
Pulling the threads together	63
Chapter 21: Scaling and Production Operations	64
The four axes	64
Autoscaling on a metric that means something	64
Worker architecture and queue partitioning	64
Postgres, the real bottleneck	64
Multi-tenancy at scale	64
Five runbooks	64
What is different about on-calling an agent	65

CONTENTS

Cost governance	65
Pulling the threads together	65
Chapter 22: Advanced Architectures and Choosing Your Level of Ab-	
straction	66
What a harness is actually for	66
The comparison, done as architecture rather than as a league table . .	66
The deterministic-first architecture, restated properly	66
Harnesses as research, not as scaffolding	66
The attribution ledger	66
What I would change about Sentinel	66
Migration and the volatility caveat	67
Pulling the threads together	67
Part IV: The Audit	68
Chapter 23: From Prototype to Production. The Audit	69
The prototype, described honestly	69
Remove	69
Constrain	69
Redesign	69
Replace	69
A worked failure analysis: the duplicate pull request	69
The migration order and why that order	70
Before and after	70
The go/no-go checklist	70
Pulling the things together	70
Conclusion: The Discipline	71
What DeepSeek Harness actually is	71
The ledger, one more time	71
What is genuinely hard, still	71
What to do on Monday	71
The last argument	71
References	72
DeepSeek Harness: repository, architecture and subsystems	73
DeepSeek Harness: orchestration, telemetry and versioning	74

DeepSeek models, the API and the Cordis paper 75

Evaluation, security, context and adjacent frameworks 76

A Production Engineering Guide to Reliable, Secure, Observable Agents on DeepSeek Models

About this book. DeepSeek Harness (dsh) is an open-source agent harness that DeepSeek released in 2026 under the MIT license, built on the Cordis plugin framework, where the model adapter, the tool registry, the session log, the sandbox and the agent loop itself are all swappable plugins rather than fixed internals. This book treats it as an engineering substrate: what its components do, how a turn actually executes, where its abstractions help, where they leak and what you must build around it to run agents that a business can depend on. You will follow one reference implementation from a thirty-line script to a budgeted, traced, eval-gated, multi-tenant service and you will see the incidents that forced each change. The audience is experienced software, platform and site-reliability engineers who want implementation-level knowledge rather than a survey.

Introduction: The Reliability Gap

Two teams, same model, same weekend, different outcomes.

Team A builds a support triage agent. They pick a DeepSeek model, wire up a loop, give it a handful of tools and demo it on Thursday. By the following Tuesday it is answering 4,000 internal tickets a day and nobody thinks about it as an AI project any more. Team B builds something smaller: a tool that reads a service's error logs and drafts an incident summary. Six weeks later they have rewritten the prompt eleven times, disabled the shell tool twice, added a second agent to "double-check" the first one and still get a call when the thing decides to re-run a migration it already ran.

Both teams used the same model. Both teams had roughly the same tools available. The difference was not the model and it was not even the prompt. The difference was the **harness**: the code around the model that decides what it gets to see, what it is allowed to do, what happens when it is wrong and whether anyone can tell afterwards what happened.

This book is about that layer and specifically about DeepSeek Harness, the agent harness DeepSeek open-sourced in 2026 under the name dsh. It is worth being precise about what that word means, because it has been diluted.

What a harness is

A harness is the program that owns the loop. It assembles the request, calls the model, parses what comes back, executes the tools the model asked for, appends the results, decides whether to go around again and stops when it should stop. It is the difference between an API endpoint and an agent.

The term predates the current wave of frameworks. Evaluation suites have used it for years to mean the runner that wraps a model under test: the thing that gives an agent a repository checkout, a shell, a task description and a fixed number of steps and then scores the result. DeepSeek uses the word the same way in its own model cards. The DeepSeek-V4.1-Flash card states that its code-agent numbers, including Terminal-Bench 2.1, DeepSWE

v1.1, NL2Repo-Bench and ProgramBench, were produced “with the Minimal mode of DeepSeek Harness and a 1M-token context window”, with Terminal-Bench 2.1 evaluated without network access. The harness is not a wrapper you add for production polish. It is already part of how the model’s capability gets measured.

That fact has a practical consequence that runs through every chapter of this book. A model benchmark is a measurement of a *pair*: model plus harness. Change the harness and you change the score. A 2026 study evaluated 66 configurations across four harnesses (OpenHands, DeepSeek Harness, PI and openJiuwen) and five models and found that model rankings reverse depending on which harness they run in. On Terminal-Bench 4, Claude led GPT by 7.94 points inside OpenHands but trailed it by 30.16 points inside PI. For four of the five models, the best harness changed from benchmark to benchmark. The authors’ conclusion is the one you should carry into your own architecture: model, harness and task have to be evaluated together.

So when someone tells you “the model is good enough now, the rest is plumbing”, they are describing a measurement they have not made.

What DeepSeek Harness actually is

Strip the marketing and dsh is a TypeScript monorepo whose organizing idea is that there is no core. Its tagline is “Everything is a Plugin”. The claim is literal. The model adapter is a plugin. The tool registry is a plugin. The session log, the compaction strategy, the sandbox backend, the approval policy and the agent loop itself are plugins that register themselves into a dependency-injection container called **Cordis** and can be replaced or removed. The repository’s own architecture notes list the core packages as `core/session`, `core/system-prompt`, `core/tools`, `core/agent`, `core/agent-loop`, `core/scope`, `llm/llm` and `webhook/webhook` and describe the running system as “a plugin tree composed of ordered layers”.

Cordis comes from a 92-page programming-languages paper, *A Programming Paradigm for Spatiotemporal Composability*, by Yifan Shi, Wei Zhang and Tianyi Cui, submitted in August 2026. The paper formalizes dynamic software composition along two axes. **Temporal composability** means that when you remove a component, its side effects revert. **Spatial composability** means that dependencies between components are managed reactively, so that loading

one plugin can turn another one on or off. The mechanisms are “reversible effects” (the runtime keeps an inverse for every change a plugin makes to the shared context) and “reactive coeffects” (context changes are classified against declarations, which drives activation and deactivation). The paper argues that merging effects and coeffects into a single context paradigm yields observational equivalence, so unrelated components can interleave without interfering with each other.

That is a lot of theory for a plugin system. It matters here for one concrete reason. Because every registration in Cordis must be a reversible effect, hot-swapping a component is not a trick layered on top of the runtime; it is the thing the runtime guarantees. The harness docs state the operational consequence directly: editing a plugin’s configuration triggers hot module replacement, unloading the old instance and loading a new one while cleaning up old registrations. If a required service disappears, dependent plugins dispose automatically and reload when it comes back. You can replace the model adapter under a running agent.

The practical shape of the product is this. You install `@deepseek-ai/dsh` from npm and run `npx @deepseek-ai/dsh web`, which serves a Web UI on `http://127.0.0.1:3080`. You get a coding and general-purpose agent with filesystem, shell, web, subagent, scheduling and browser tools. You can also drive it headlessly, embed it in an editor over ACP, or drive it from Python through `deepseek-harness-sdk`, which spawns the same runtime as a subprocess and speaks newline-delimited JSON-RPC over stdio. The CLI is the single entry point and the docs are emphatic that “SDK and ACP are profiles, not separate public bins”.

The thesis

Here is the argument the rest of the book makes.

An agent is a loop around a stateless probabilistic function and everything that makes it dependable lives outside the model. DeepSeek’s V4-series models are genuinely strong. DeepSeek-V4-Pro reports 80.6 on SWE Verified and 67.9 on Terminal Bench 2.0 at maximum reasoning effort. DeepSeek-V4.1-Flash reports 90.6 on Terminal-Bench 2.1. Those numbers describe what the model can do when somebody else’s harness gives it a fair chance. They say nothing about whether your agent will still be correct on a Tuesday in

November when the retrieval index is stale, one tool is timing out and the conversation has been running for ninety minutes.

Three properties of the underlying system create all the interesting engineering work:

1. **The model is stateless.** It remembers nothing between calls. What looks like memory is your code re-sending text. DeepSeek Harness makes this explicit and auditable: a `Session` is an append-only log of typed events and the LLM message history is *derived* from that log rather than stored separately.
2. **The model is probabilistic.** The same context can produce different outputs. You cannot test an agent the way you test a parser, but you can and must bound the consequences of a bad draw.
3. **The model is only as safe as its tools.** A model with no tools can only say something wrong. A model with `bash`, network access and your cloud credentials can *do* something wrong. OWASP's LLM Top 10 makes the pairing explicit: prompt injection (LLM01) becomes dangerous in proportion to excessive agency (LLM06).

DeepSeek Harness addresses all three, unevenly and honestly. It gives you an event-sourced session log you can replay. It gives you a tool pipeline with guards, approvals and fail-closed outcomes. It gives you a sandbox seam with Landlock, bubblewrap, Seatbelt and Windows ACL backends that reports enforcement as `full` or `partial` and fails closed when no backend exists. What it does not give you and says it does not give you, is a security boundary you can trust for untrusted workloads. `SAFETY.md` is blunt: sandboxing and permission controls “do not guarantee isolation” and the software should not be the “sole security control for untrusted workloads”.

That sentence is the most useful thing in the repository, because it tells you exactly where the book's real subject lives. The harness is the unit of *composition*. Your infrastructure is the unit of *trust*. Learning to build agents that work means learning which responsibilities belong to which layer and this book will keep score on that question in every chapter.

The reference implementation

Chapters build one system, called **sentinel**, from nothing to production. It starts as an internal engineering-assistant agent that can read a repository, run

commands and answer “why did this deploy fail”. As the book proceeds it gains typed tools, retrieval, budgets, checkpoints, an approval workflow, a multi-agent research mode, an evaluation harness, traces, a queue-backed worker, containers and an incident runbook. The same architecture is then reshaped into six other workloads: coding, data analysis, SRE, document processing, customer support and a long-running autonomous workflow.

The stack is deliberately boring and deliberately consistent:

Concern	Choice	Reason
Harness	DeepSeek Harness 0.2.x-rc, sdk and sdk-minimal profiles	the subject of the book
Model	deepseek-flash, escalating to deepseek-v4-pro	1M context, tool calling, price
Application language	Python 3.12+	the SDK boundary is Python; most teams’ platform code is too
Contracts	Pydantic v2	one schema generates validation, tool specs and eval fixtures
State	PostgreSQL 16 + pgvector	run state, checkpoints and vectors in one auditable store
Queue and cache	Redis 7 with arq	long-running runs, leases, scheduling, rate limits
API	FastAPI with SSE	a run is a resource, not a request
Observability	OpenTelemetry to OTLP	vendor-neutral, self-hostable, diffable

Concern	Choice	Reason
Isolation	containerized workers, egress allowlist	the harness says it is not a boundary, so bring one

Alternatives appear only where they represent a real fork in the road and the book says which way it went and why.

How to read this book

Chapters are cumulative. Chapters 1 through 3 are conceptual and short; if you have shipped an agent before you can skim them, but Chapter 3 defines vocabulary the rest of the book uses without re-explaining. Chapters 4 through 12 build the agent. Chapters 13 through 16 are the reliability and security core and are the chapters most likely to save you a bad quarter. Chapters 17 and 18 cover evaluation and observability. Chapters 19 through 22 cover cost, deployment, scaling and operations. Chapter 23 is the audit you run when an experiment becomes a system and the Conclusion is short.

Part I: Foundations and Architecture

Chapter 1: What an Agent Is and When You Should Not Build One

A staffing platform I consulted for in 2025 had a feature that took a candidate's résumé and a job description and produced a screening summary. It ran for eleven months as a deterministic pipeline: parse, chunk, extract, score, render. It cost about \$0.004 per document and failed in ways engineers could reproduce on demand.

In month twelve, a capable engineer replaced it with an agent. The agent could browse the candidate's public profiles, cross-check dates, notice that a "VP of Engineering" title at a four-person company meant something different than it did at a bank and generally do the judgement calls the pipeline could not. It also cost nine cents per document, took forty seconds instead of two and produced a different summary on 8 percent of reruns. The team spent the next four months rebuilding the determinism they had thrown away, this time on the far side of the model call.

The agent was the right call eventually. It was the wrong call in month twelve, because the problem had not yet earned it. That distinction and how to make it on purpose, is where this book begins.

The loop, in twelve lines of pseudocode

Every agent I have ever read or written reduces to this:

```

1  def agent_loop(task, tools, budget):
2      state = new_state(task)
3      while budget.allows(state):
4          request = assemble_context(state, tools)    # 1. decide what the model
              ↪ sees
5          response = model.generate(request)          # 2. one probabilistic call
6          state.append(response)                     # 3. record it durably
7          if response.is_final:
8              return response.content
9          for call in response.tool_calls:            # 4. the model acts on the
              ↪ world
10             result = tools.execute(call, budget)    # 5. with policy applied
11             state.append(result)
12     return state.degrade_to_partial()              # 6. exhausted, not failed

```

Six numbered steps and every framework worth its name is an argument about one or more of them.

Step 1 is **context assembly**. What goes into the window, in what order, with what budget. DeepSeek Harness puts this behind `ctx.systemPrompt`, which collects registered `PromptSection` objects sorted by an `order` field and then by name, plus a separate `PromptContext` for dynamic material that is logged only when it changes or after compaction. That last detail is a cost optimization you get for free: stable prefixes are what DeepSeek’s on-disk context cache keys on.

Step 2 is the only step the model performs. Everything else is your code.

Step 3 is **durable recording**. In dsh this is `Session.append(type, data, opts)`, which validates that the payload is JSON-serializable and that surface-event contracts hold, then returns the logged event. Note what that ordering buys you: because the log is the source of truth and the message list is derived, a crash between steps 2 and 4 leaves an artifact you can reason about.

Step 4 is where capability and risk both live. Step 5 is where you decide how much of each to allow. Step 6 is the step that most tutorials omit and most production incidents consist of.

DeepSeek Harness names the same structure with a three-level vocabulary worth memorizing, because the rest of the book uses it. A **step** is one model request plus the tools it triggers. A **turn** is the drain of pending input across zero or more steps. A **round** is an iteration of an outer policy loop, such as the fresh-agent workflow the harness calls Ralph. The lifecycle events map one to one onto the pseudocode: `turn/start` claims pending inbox messages and

assembles prompts, agent/pre-step waterfalls run, step/start fires, the model streams, tool/call and tool/result bracket execution and turn/end closes.

Twelve lines. The reason it takes a book to do well is that each line is a seam where reality disagrees with the tutorial.

Workflows versus agents: who holds the control flow

The distinction is not “does it use a model”. Deterministic pipelines call models constantly and are not agents. The distinction is **who decides what happens next**.

In a workflow, the control graph is written by an engineer before the program runs. The model fills in slots. It classifies, extracts, summarizes, translates, generates. Every branch is a branch you wrote. If the model returns something unexpected, your code takes a defined path, which might be an error and that is fine.

In an agent, the model decides what happens next. It chooses which tool to call, with what arguments and whether it is finished. Your code executes its choices and stops it when a policy says so.

That is the whole trade. The agent gains the ability to handle the long tail you did not enumerate. It loses three things you did not price in: reproducibility, bounded cost and the ability to say at review time what the system will do.

A useful test I have not found contradicted: **if you can write the branch list, write the branch list**. An agent earns its place when the branch list is genuinely open. “The user might ask anything” is an open branch list. “The document might be a PDF, a DOCX, or a scan” is not; that is three branches.

There is a hybrid shape that carries most real systems and it is worth naming before it gets fuzzy: a **workflow with agent steps**. The pipeline is deterministic. At two or three points where judgement is genuinely required, the pipeline calls an agent with a narrow tool set, a hard token budget and a typed output contract. The agent’s answer becomes a validated row in a table. Everything downstream is ordinary software again.

DeepSeek Harness supports this shape cleanly and it is worth seeing how, because the mechanism recurs all book. The tool registry exposes Tool-

Restriction, which applies per-scope allow and deny name lists over the inherited global tool set, while tools registered at that scope are exempt. Scope identity is opaque object identity, typically the live Agent object. So an agent step inside a document-processing pipeline can be given exactly three tools and no others and that restriction is enforced by the registry rather than requested in prose. The difference between those two sentences is the difference between a demo and a system.

The four-way trade-off: capability, cost, latency, variance

Every increment of autonomy moves four dials at once.

Capability goes up, at least on tasks in the tail. **Cost** goes up superlinearly, because autonomy means more steps and each step re-sends a longer context. **Latency** goes up linearly in step count, which is brutal for interactive use. **Variance** goes up multiplicatively, because per-step error compounds.

The compounding is the part teams miss. Suppose your agent is 95 percent reliable per step, which is a good number. A five-step task succeeds with probability 0.95^5 , about 77 percent. A fifteen-step task succeeds about 46 percent of the time. This is a crude model and real errors correlate, sometimes helpfully, but the shape is right: **step count is the enemy and autonomy buys steps.**

The economics are not subtle once you write down the prices. As of 6 October 2026 the DeepSeek API charges, at peak rates, \$0.30 per million cache-miss input tokens and \$1.20 per million output tokens for deepseek-flash and \$1.32 in / \$3.96 out for deepseek-v4-pro. Off-peak rates are exactly half, with peak defined as 01:00 to 04:00 and 06:00 to 10:00 UTC on weekdays.

Now price a fifteen-step agent that accumulates context. If the average step sends 24,000 input tokens and produces 600 output tokens, fifteen steps send roughly 270,000 input tokens and produce 9,000 output tokens. At Flash peak rates with no cache hits that is about \$0.081 plus \$0.011, call it nine cents. With DeepSeek's prefix caching working well and the cache is enabled by default and hits when a request prefix fully matches a persisted cache prefix unit, the same run can land near a cent. The spread between those two numbers is not a rounding error. It is the difference between a feature and a budget line and it is determined almost entirely by how your harness builds prompts.

Chapter 19 does this arithmetic properly. The point here is that the autonomy decision is a financial decision and it should be made with a spreadsheet open.

A decision procedure and four escape hatches

Here is the procedure I run before agreeing to build an agent. It takes an afternoon and has saved more than one.

Step one: write the deterministic version first, badly. Not a prototype of the agent. The workflow. Enumerate the cases you actually observed in production traffic, in frequency order. Ship the version that handles 80 percent of them with if-statements and one or two narrow model calls. You now have a baseline, a cost floor and a set of residual failures that is small enough to read individually.

Step two: measure the residual. If the cases your deterministic version cannot handle are 2 percent of traffic and cost \$40 a month in human time, an agent is a research project, not a roadmap item. If they are 40 percent and cost \$40,000, build the agent.

Step three: bound the blast radius before you widen it. Decide, in writing, the worst thing the agent can do with the tools you are considering. If the answer includes “delete customer data”, “spend money”, “send email”, or “modify production infrastructure”, the agent does not get that tool. It gets a tool that proposes the action and a human or a policy engine that approves it. This is not paranoia; it is the same reasoning that keeps a junior engineer off production for their first month.

Step four: define “correct” before you write a line. Not “good output”. A checkable predicate. If you cannot write one, you do not have a product requirement, you have a demo idea.

Four escape hatches let you relax the procedure honestly and you should know them because sometimes the deterministic version really is the wrong answer.

The input space is genuinely unenumerable. Open-domain research over an internal corpus, or a coding agent that must handle arbitrary repositories. Here the branch list has no closed form and the agent is the only option.

The task is short-horizon and verifiable. A three-step task with a mechanical check at the end can tolerate a lot of variance, because you can retry. Verification is the great enabler of autonomy. Where you can check cheaply, you can be sloppy confidently.

The cost of a wrong answer is near zero and the cost of no answer is high. Drafting an internal summary nobody will act on without reading it. Here variance is a feature.

You are explicitly exploring. Fine. Timebox it, keep it off the production path and read Chapter 23 before it tries to become one.

Anti-patterns, part one

Two of the failure modes in this book's catalogue are so common they deserve naming now, before we have the vocabulary to fix them.

Excessive autonomy. Giving the model more authority than the task requires, usually because it is easier than writing the narrow tool. The OWASP taxonomy calls the resulting condition Excessive Agency, LLM06 and describes it as AI components operating with more privilege or autonomy than strictly necessary; the guidance is least-privilege access and human approval for high-risk actions.

The reason this is the top of the list is compositional. Prompt injection by itself gets an agent to say something you did not want. Prompt injection plus a shell tool gets an agent to run something you did not want. Anthropic's own measurement of browser-agent injection is instructive about how much residual risk survives serious engineering: after reinforcement training on simulated injection content, classifiers scanning untrusted content and scaled expert red-teaming, their best-of-N adaptive attacker at 100 attempts per environment achieved roughly a 1 percent attack success rate, down from much higher rates in the original preview. One percent is excellent. One percent of an agent with your credentials is still an incident.

DeepSeek Harness gives you the right seams here and, to its credit, does not pretend the seams are a wall. The default permission presets are `workspace-write` (workspace-write sandbox plus ask approval) and `danger-full-access` (no confinement plus never approval). Read the second one out loud. The name is honest in a way most software is not. Note also that the shipped `sdk-minimal` profile pins `danger-full-access`, which grants the

shell unrestricted path visibility. That is a sensible default for a benchmark harness running on a disposable machine and a terrible one for your service. Chapter 16 changes it.

The agent-shaped hammer. Using an agent where a workflow is better, usually justified by a roadmap slide. The tell is a system whose steps are always the same but whose implementation asks the model what to do next. If your agent calls `search`, then `read`, then `summarize`, then `write` on every run, you have a four-function workflow with a stochastic control plane and a per-step bill.

A specific sub-case deserves its own warning because it is fashionable: **unnecessary multi-agent architecture**. Adding a second agent to review the first is the single most reliable way to make a system feel more robust while making it slower, more expensive and harder to debug, because now you have two stochastic components and an interface between them that nobody specified. Chapter 11 covers the four situations where decomposition genuinely pays and one of them is never “the output was bad, so add another agent”.

Pulling the threads together

An agent is a loop where the model holds the control flow. That is the entire definition and it is a trade rather than a feature. You buy coverage of the long tail with reproducibility, money, latency and variance. The engineering discipline consists of buying only as much of the tail as you can pay for and only with tools whose worst case you can live with.

DeepSeek Harness is an unusually good substrate for this discipline because its abstractions put the four decision points in the open. Context assembly is `ctx.systemPrompt`. Tool authority is `ctx.tools` with restrictions, guards and an approval waterfall. Durable state is the append-only session log. Termination is a set of explicit turn-end reasons: `completed`, `aborted`, `blocked`, `error`, `max-tokens`, `interrupted`, `forked`. Nothing about that list is hidden behind a `.run()` call that does whatever it wants.

Chapter 2: DeepSeek Models as an Engineering Substrate

The most expensive mistake in agent engineering is treating the model as a black box with a quality rating. Not because the model is simple, but because its specific properties leak through your architecture whether or not you acknowledge them. A model that bills you for re-sent context every step pushes your design toward prefix stability. A model that ignores temperature in one mode and honours it in another pushes your design toward mode-aware configuration. A model that returns HTTP 400 when you omit a field you thought was optional pushes your design toward a canonical message serializer, or it pushes your agent loop into a crash you cannot reproduce.

This chapter is the inventory of those properties for the DeepSeek V4 line as it stands in October 2026 and the design consequences each one carries. Read it once carefully. You will return to it whenever behaviour surprises you.

Why the serving cost is low and what that buys you

DeepSeek-V4 comes in two sizes. **V4-Pro** is a 1.6 trillion parameter mixture-of-experts model activating 49 billion parameters per token. **V4-Flash** is 284 billion total with 13 billion activated. Both ship with a one-million-token context window and both are released under the MIT license.

The technical report, *DeepSeek-V4: Towards Highly Efficient Million-Token Context Intelligence*, attributes the efficiency to three changes: hybrid attention combining Compressed Sparse Attention with Heavily Compressed Attention, Manifold-Constrained Hyper-Connections (branded mHC) and the Muon optimizer. The headline number is the one that matters to your invoice: at 1M-token context, V4-Pro uses 27 percent of the single-token inference FLOPs and 10 percent of the KV cache compared with DeepSeek-V3.2.

The current hosted Flash model goes further in the same direction. The DeepSeek-V4.1-Flash card describes a causal encoder-decoder backbone of 40 layers split 20/20, a global KV cache of **890 bytes per token**, roughly

a quarter of V4-Flash's, achieved through a second-generation compressed sparse attention with full, reindex and reuse modes, FP4 KV caching and sliding window attention with bounded replay that leaves about an eighth of the persistent KV footprint. It adds 196 billion parameters of “Engram” conditional memory and speculative decoding under the name DSpark.

You do not need to care about any of this for the API. You need it for two reasons.

First, it explains the price. KV cache is the dominant memory cost of long-context serving. If your provider can hold a million tokens of context in a quarter of the memory, they can sell you long context cheaply and they do. That is why a 1M-token window is not a marketing number on this endpoint.

Second, it tells you what to expect from self-hosting, which Chapter 19 works through. The architecture is aggressively optimized for the memory-bound regime, which means a self-hosted deployment's throughput is dominated by KV cache capacity rather than raw FLOPs. Teams that size DeepSeek clusters by counting parameter bytes get this wrong in both directions: they overbuy compute and underbuy memory.

Two models, two control planes

The current official API exposes two model IDs: **deepseek-flash**, which resolves to DeepSeek-V4.1-Flash and **deepseek-v4-pro**, which resolves to DeepSeek-V4-Pro-0813. Legacy names `deepseek-v4-flash` and `deepseek-v4-flash-vision-exp` are still accepted but retired; requests to them are served by V4.1-Flash and billed at Flash prices. If your configuration still says `deepseek-v4-flash`, it works and you are one silent routing change away from a surprise. Pin the current IDs.

Both models support JSON output, tool calling, the Responses API and the Anthropic-compatible API. Both support chat prefix completion in beta and fill-in-the-middle completion in beta, the latter only in non-thinking mode. **Vision input is Flash only.** That single row decides your architecture for any workload that reads screenshots, scanned documents, or charts. A document-processing agent that must handle image PDFs cannot run entirely on Pro.

The practical routing rule I use and which sentinel implements from Chapter 19 onward: default to Flash, escalate to Pro on a small, measured fraction of requests where a cheap verifier says the Flash answer is weak. The price gap

justifies it. At peak, Flash costs \$0.30 per million cache-miss input tokens and \$1.20 per million output; Pro costs \$1.32 and \$3.96. Pro is roughly 4.4 times the input cost and 3.3 times the output cost. If escalation fires on 10 percent of requests and doubles your task success rate on those, you have bought a lot of reliability for 35 percent.

Concurrency is the constraint nobody plans for. The account-level limits are **2500 concurrent requests for deepseek-flash and 500 for deepseek-v4-pro** and exceeding them returns HTTP 429. Higher concurrency is available at no extra cost through a capacity-expansion request, which is unusual and worth doing before you need it. The 5-to-1 ratio between the two models is a design hint: Pro is not your high-throughput path.

The `user_id` parameter is the most underused control on this API. Passing it (up to 512 characters, matching `[a-zA-Z0-9\-\_]+`) gives you content-safety isolation, **KV-cache isolation** and scheduling isolation and on expanded accounts each `user_id` gets its own concurrency allowance. For a multi-tenant agent platform this is close to free: per-tenant cache isolation stops one tenant's long documents from evicting another tenant's cached prefix and per-tenant concurrency accounting turns "the platform is slow" into "tenant 44 is at its limit". DeepSeek Harness already sends an anonymous session and user identifier on official DeepSeek requests via `x-deepseek-harness-user-id` and `x-deepseek-harness-session-id` headers, which is a different mechanism from the billing-level `user_id` and does not substitute for it.

Reasoning modes and the invariant that breaks naive loops

The V4 models expose three reasoning modes. Non-think is the fast path. Think High is described as the mode for logical analysis. Think Max is maximum reasoning, with the model card recommending a context window of at least 384K tokens when you use it. V4.1-Flash additionally documents "continuously controllable reasoning effort" on a 1-to-100 scale, though the public API exposes the coarser three-way control.

On the OpenAI-compatible endpoint you control this with two parameters. Thinking is toggled with `{"thinking": {"type": "enabled"}} or "disabled"`, passed through `extra_body` in the Python client. Effort is set with `{"reasoning_effort": "low" | "high" |`

"max"}. On the Anthropic-compatible endpoint the equivalents are {"reasoning": {"effort": "none" | "low" | "high" | "max"}} and {"output_config": {"effort": "...}}, where none disables thinking. DeepSeek documents the mapping from client-side effort labels to its own: minimal and low both map to low; medium, high and xhigh all map to high; max and ultra map to max. That mapping matters because a framework that emits xhigh is not asking for something between high and max. It is asking for high.

Thinking mode is **on by default at high effort**. This surprises people. If you build a cheap extraction pipeline on deepseek-flash and never set thinking, you are paying for reasoning tokens you did not want.

Three consequences follow and the third is the one that breaks code.

Sampling parameters are partly inert. In thinking mode, temperature, presence_penalty and frequency_penalty are silently ignored. top_p is effective only in the range 0.95 to 1.0, with lower values clamped to 0.95 and is fixed at 1.0 in non-thinking mode. The word “silently” is the documentation’s and it is a design criticism worth making: a parameter that does nothing should either be rejected or reported in the response. Your mitigation is to make your configuration layer mode-aware, so that setting temperature alongside thinking is a startup error in your system rather than a mystery. The DeepSeek-V4.1-Flash card’s own recommendation is temperature=1.0 with top_p=0.95, which is consistent with the clamping behaviour.

Reasoning tokens are billed as output. DeepSeek Harness models this exactly and its definition is worth copying into whatever usage accounting you build: TokenUsage uses disjoint counts of inputTokens (uncached), cacheReadTokens, cacheWriteTokens and outputTokens, billed input is the sum of the three input fields and **reasoningTokens is already included in outputTokens**. If you add reasoning tokens to output tokens in your cost dashboard you will double-count and your reasoning-heavy agents will look twice as expensive as they are.

With tools, you must send the reasoning back. This is the invariant. In multi-turn conversations without tools, prior reasoning_content is ignored and is not concatenated into the context. In multi-turn conversations **with tools, reasoning_content from all prior turns must be passed back and omitting it returns an HTTP 400**. The docs spell out the sharp edge: for requests carrying the tools parameter, reasoning_content must be fully

passed back in all subsequent requests, even for turns where the model did not perform a tool call.

Read that again if you have ever written an agent loop by hand, because it is where naive implementations die. The natural thing to do when you serialize a conversation into a message list is to keep content and `tool_calls` and drop the chain-of-thought field, since it is “just” scratch space. On this endpoint, with tools, that is a 400. Worse, it is a 400 that appears only on the second tool-using turn, so your unit test with one tool call passes and your production traffic fails.

DeepSeek Harness handles this for you and it is a good example of why the harness layer earns its existence. Chain-of-thought is a first-class content block type in its message model, alongside text, image, file, tool-call, tool-addition and tool-removal. The session log records the exact timed model stream inside `assistant/message` events, and message derivation is a pure per-node projection from those events. The reasoning content is in the log, so it is in the derived messages, so it goes back to the provider. The invariant holds because the architecture makes the correct path the only path.

Tool calling on the wire

Tool calling works in thinking mode and the model may interleave several reasoning and tool-call turns before producing a final answer. The wire format is the OpenAI-compatible one you already know: you send a `tools` array of function schemas, the model returns `tool_calls` with a `call_id`, a name and arguments as a JSON string, you execute and you append a role: “tool” message keyed by the call id.

Two details at the harness boundary are worth internalizing.

First, **tool-call arguments stay raw JSON strings all the way through the adapter layer** in DeepSeek Harness. Parsing happens above the adapter, in the tool pipeline, where a malformed argument becomes a typed `ToolArgsError` with code `INVALID_ARGS` rather than an exception that unwinds your loop. The reason this matters is that malformed arguments are not exceptional. The 2026 harness comparison study found that Kimi performed best inside the openJiuwen harness specifically because that harness tolerated its tendency to emit malformed tool calls. Tool-call syntax errors are a model property. Your harness either absorbs them or it does not.

Second, a **max-tokens finish drops all tool calls** in the harness assembler. This is the correct behaviour and it is also a silent failure mode. A truncated response that looked like it was going to call a tool becomes a turn with no action. The session records the finish reason and the turn-end reason map gives max-tokens precedence over completed if any step hit the ceiling. If your agent ever appears to stop mid-task without explanation, check for max-tokens before you blame the prompt.

The adapter contract itself is small and worth reading as a specification of what “one model call” means: `usage` arrives before `finish` and nothing arrives after; there are exactly two error paths, a throw or an in-band finish reason; one adapter call equals one provider attempt; and an empty completion is a retryable error. Failures surface as a serializable, provider-neutral `LlmFailure` carrying `message`, `code`, `status`, `providerRetryAfterMs`, `requestId` and an `offloadImages` flag. `providerRetryAfterMs` is the field to respect in your backoff logic and `requestId` is the field to include in every support ticket you file.

One million tokens is not an excuse

Both current models take a 1M-token context and produce up to 384K output tokens. The instinct is to use it. The measurements say not to.

Chroma’s *Context Rot* technical report, published July 2025, evaluated 18 models including GPT-4.1, Claude 4, Gemini 2.5 and the Qwen3 family on controlled tasks where complexity was held constant and only input length varied. The findings are worth stating precisely because they are more specific than the “long context is bad” slogan:

- Performance degrades **non-uniformly** as input length grows, even on trivially simple tasks.
- Lower semantic similarity between the needle and the question accelerates the decline.
- **A single distractor reduces accuracy; four compound the effect** and the impact is non-uniform across distractors. Claude-family models tended to abstain under ambiguity while GPT-family models hallucinated confidently.
- Counter-intuitively, **shuffled, incoherent haystacks consistently improved performance** over logically structured ones across all 18 models.

- On LongMemEval at roughly 113K tokens, focused prompts of about 300 tokens produced significantly higher accuracy than full prompts across every model family tested.

The authors' conclusion is the engineering thesis of Chapter 6: "What matters more is how that information is presented."

None of this is measured on V4 specifically and I want to be explicit that extrapolating Chroma's numbers to DeepSeek's models is my inference rather than a cited result. The mechanism, attention dilution over a longer sequence, is architectural rather than model-specific and V4's own design is an explicit attack on the cost side of that trade rather than a claim to have removed the accuracy side. Treat the direction as reliable and the magnitudes as unmeasured until you measure them on your own tasks, which Chapter 17 shows how to do.

The economics point the same way as the accuracy data. DeepSeek's context caching is on-disk, enabled by default and hits when a subsequent request's prefix fully matches a persisted cache prefix unit. Cache hits are dramatically cheaper than misses: \$0.006 per million cache-hit input tokens for Flash at peak against \$0.30 for a miss. A fifty-fold discount. But the cache matches only the input prefix, is best-effort with no hit guarantee and unused caches clear within hours to days.

Put those together and the optimal prompt shape is not "stuff everything in". It is a **long, stable, cache-friendly prefix plus a small volatile suffix**, which is exactly what the harness is built to produce. The system prompt is a surface node that is replaced in place rather than re-appended. Dynamic prompt context is logged only when it changes or after compaction. The DeepSeek-specific wire extensions, including the session-log and package-inventory payloads, are deliberately kept **outside model-visible prefixes and token counts**, precisely so that adding telemetry does not invalidate your cache.

Hosted-endpoint failure modes

Four behaviours of the hosted endpoint will shape your error handling.

Long queue waits are normal, not failures. When the service is saturated, pending requests return empty lines for non-streaming calls or : keep-alive

comments for streaming ones and connections close if inference has not started after ten minutes. Your client timeout must be longer than your patience. A 60-second HTTP timeout on a saturated endpoint converts a queue wait into a client-side failure that you then retry, which lengthens the queue. This is a textbook retry-amplification outage and it is entirely self-inflicted.

429 is a concurrency signal, not a rate signal. The documented limits are concurrent requests, not requests per second. A token bucket tuned for requests-per-second will let you open 501 simultaneous long-running requests to Pro and get all of them rejected. Your limiter must count in-flight requests.

Retries are already happening inside the harness. DeepSeek Harness captures a retry policy per model route at registration: either normal mode with a finite `maxRetries` defaulting to 5, or `always` mode. If you add your own retry loop without reading this, you have a multiplicative retry budget. Five inner attempts times five outer attempts is twenty-five calls to an endpoint that is already telling you to stop. Chapter 13 turns this into a policy.

Model identity is not stable. The API has already renamed models once in the V4 line, with retired names silently routing to a replacement. DeepSeek Harness's own upgrade guides document breaking changes per release. Pin what you can, snapshot what you cannot and treat every model string in configuration as a dependency with a version.

Pulling the threads together

The DeepSeek V4 line gives you strong agentic capability at a price that makes long-horizon work financially possible and it gives you a short list of sharp edges that will each break something if you ignore them.

The load-bearing facts, collected: thinking mode is on by default at high effort; sampling parameters are silently ignored in thinking mode; reasoning tokens are billed inside output tokens; **reasoning_content must be echoed back on every turn once you use tools**; vision is Flash-only; concurrency is capped at 2500 and 500; caching is prefix-based, on by default and fifty times cheaper than a miss; and long context degrades accuracy even when it does not overflow.

DeepSeek Harness absorbs most of these at the adapter and session layers. That is the point of it. What it cannot absorb is a decision about which model to use for which request, how much reasoning effort to buy, or what to do when

the endpoint queues for nine minutes. Those are yours and the next chapter shows where in the harness they belong.

Chapter 3: Inside the Harness.

Architecture of DeepSeek Harness

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

Cordis in one page

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

Component map

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

The life of a turn

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

Three event planes

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

Extension points, ranked by how much you will use them

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

Where the harness stops

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

Reading the source without drowning

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

Pulling the threads together

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

Chapter 4: First Agent. Scaffold, Config and the Loop

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

Repository layout and dependency pinning

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

Typed settings and environment configuration

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

The client wrapper

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

Running one turn and reading what happened

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

Structured output is a reliability primitive, not a convenience

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

A Makefile you will actually use

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

A debugging session, because it will be your first one

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

Pulling the threads together

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

Chapter 5: Tools. Contracts, Registries and Consequences

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

The split that defines your architecture

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

The tool contract in the harness

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

The result envelope

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

Registry, scoping and least capability

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

Parallel tool calls and why ordering is a trap

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

Idempotency, or how to survive a retry you did not choose

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

Error taxonomy, the highest-leverage page in the book

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

A code review, abridged

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

Pulling the threads together

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

Chapter 6: Context Engineering and Token Budgets

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

Anatomy of a prompt

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

Token accounting you can trust

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

The compaction ladder

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

Prompt caching is an architecture, not a flag

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

What long context actually costs you

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

Diagnosing a context problem from traces

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

Pulling the threads together

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

Chapter 7: Memory and State

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

Three kinds of state and why conflating them is the bug

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

Conversation state: what the harness already does

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

Task state: the part nobody designs

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

Checkpointing and resumability

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

Goals, todos and the difference between them

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

Durable memory that does not poison

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

Retrieval-backed memory versus always-in-context memory

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

Anti-patterns, part two

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

Pulling the threads together

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

Chapter 8: Retrieval and Knowledge Bases

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

Retrieval is a tool, not a pipeline

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

Chunking that respects structure

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

Hybrid search, in the database you already have

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

Embeddings, reranking and their costs

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

Permissions are a retrieval problem

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

Evaluating retrieval on its own

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

When retrieval is the wrong answer

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

Pulling the threads together

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

Chapter 9: Planning and Reasoning

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

Thinking mode is a cost dial, not a quality switch

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

The effort ladder and where it surprises you

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

The parameters that silently do nothing

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

The echo invariant and why a harness is the place to fix it

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

Planning: three mechanisms and which one to use

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

Plan-then-execute is usually the wrong shape

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

Structured output while thinking

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

Reading reasoning without believing it

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

Pulling the threads together

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

Chapter 10: Orchestration, Queues and Long-Running Agents

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

Two loops and the seam between them

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

The queue and why not the harness scheduler

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

Long-running agents

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

Background jobs and subprocesses the harness already manages

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

Event-driven wakeups

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

Cancellation that actually works

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

Pulling the threads together

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

Chapter 11: Multi-Agent Systems and When Not to Have Them

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

The tax, stated precisely

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

What the harness actually gives you

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

Permissions are the real reason to delegate

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

The workflow engine and its one sharp edge

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

Three patterns that earn their cost

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

Patterns that do not

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

Debugging a delegation tree

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

Pulling the threads together

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

Chapter 12: Coding and Computer-Use Agents

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

The filesystem contract is the interesting part

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

Search strategy: grep is not a fallback

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

Shell discipline

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

Postmortem 0003: the verification gap

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

Postmortem 0004: attribution requires independent evidence

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

Computer use and browser use

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

Pulling the threads together

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

Part II: Reliability and Security

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

Chapter 13: Containment. Budgets, Loops and Runaway Agents

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

The reliability contract

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

Deterministic boundaries

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

Five independent ceilings

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

Infinite loops: four signatures

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

Runaway agents and the blast-radius ledger

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

Timeouts at every layer and the gap

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

Idempotency and transactional effects

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

What the harness gives you and what it does not

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

Pulling the threads together

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

Chapter 14: Recovery. Retries, Fallbacks and Degraded Modes

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

A closed vocabulary of failure

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

Retries and the retry budget

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

The 429 arithmetic

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

Circuit breakers for tools

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

Model fallback and what it costs you

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

Degraded modes, defined in advance

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

Dead letters and the human queue

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

Concurrency control and the write-conflict problem

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

Pulling the threads together

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

Chapter 15: Security I. Injection, Taint and Exfiltration

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

The threat model, in one page

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

Prompt injection is an authority problem

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

How good are the defences, honestly?

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

Taint tracking: the mechanism nobody ships

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

Exfiltration: enumerate the channels

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

SSRF, the agent-shaped version

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

Malicious tool results, by name

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

Pulling the threads together

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

Chapter 16: Security II. Isolation, Credentials and Tenancy

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

What the sandbox is and what it is not

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

Permission presets: the unit you should be configuring in

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

The profile trap

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

Credentials: the model must never hold one

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

Network isolation: the container is the boundary

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

Tenant isolation, three ways

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

Audit logging that survives the incident

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

Supply chain: plugins are code with your permissions

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

The checklist and the assumption to make

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

Pulling the threads together

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

Chapter 17: Evaluation. Knowing Whether It Got Better

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

The measurement problem, stated honestly

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

Your harness is a variable, not a constant

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

Three tiers of assertion

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

Building the eval set from production, not from imagination

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

Regression detection: what changed and how do you know

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

Replaying a trajectory and why the harness makes it possible

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

Online evaluation: the part the eval set cannot do

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

Anti-patterns in evaluation

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

Pulling the threads together

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

Chapter 18: Observability. Traces, Trajectories and Cost

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

The session log is the trace

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

Spans and the vocabulary that maps onto them

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

Structured logs, with the right keys

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

The OTel seam and its channel discipline

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

Redaction: the seam that ships empty

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

Trajectory inspection: the tool that pays for itself

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

Failure classification in production

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

Debugging sessions: four real shapes

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

Pulling the threads together

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

Part III: Performance, Deployment and Operations

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

Chapter 19: Performance and Cost Engineering

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

The cost model, worked end to end

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

Prompt caching as an architectural constraint

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

Reasoning tokens: the biggest single lever

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

Latency and where it actually goes

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

Throughput and the concurrency ceiling

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

Self-hosting: do the arithmetic before the architecture

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

Quantisation and where it stops being free

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

Total cost of ownership, honestly

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

Pulling the threads together

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

Chapter 20: Deployment. Containers, Configuration and Graceful Shutdown

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

What you are actually deploying

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

The image

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

Configuration: the layering trap, revisited operationally

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

Secrets

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

Health checks that mean something

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

Graceful shutdown, the actual problem

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

Rolling deployments and version skew

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

Local deployment, because some of this runs on a laptop

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

Backups and the recovery drill

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

Pulling the threads together

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

Chapter 21: Scaling and Production Operations

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

The four axes

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

Autoscaling on a metric that means something

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

Worker architecture and queue partitioning

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

Postgres, the real bottleneck

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

Multi-tenancy at scale

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

Five runbooks

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

What is different about on-calling an agent

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

Cost governance

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

Pulling the threads together

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

Chapter 22: Advanced Architectures and Choosing Your Level of Abstraction

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

What a harness is actually for

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

The comparison, done as architecture rather than as a league table

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

The deterministic-first architecture, restated properly

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

Harnesses as research, not as scaffolding

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

The attribution ledger

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

What I would change about Sentinel

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

Migration and the volatility caveat

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

Pulling the threads together

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

Part IV: The Audit

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

Chapter 23: From Prototype to Production. The Audit

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

The prototype, described honestly

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

Remove

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

Constrain

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

Redesign

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

Replace

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

A worked failure analysis: the duplicate pull request

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

The migration order and why that order

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

Before and after

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

The go/no-go checklist

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

Pulling the things together

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

Conclusion: The Discipline

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

What DeepSeek Harness actually is

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

The ledger, one more time

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

What is genuinely hard, still

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

What to do on Monday

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

The last argument

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

References

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

DeepSeek Harness: repository, architecture and subsystems

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

DeepSeek Harness: orchestration, telemetry and versioning

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

DeepSeek models, the API and the Cordis paper

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

Evaluation, security, context and adjacent frameworks

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