00 — Introduction

🕒 state of the art 2026-07 · revised 2026-08-01 · 📖 ~7 min read

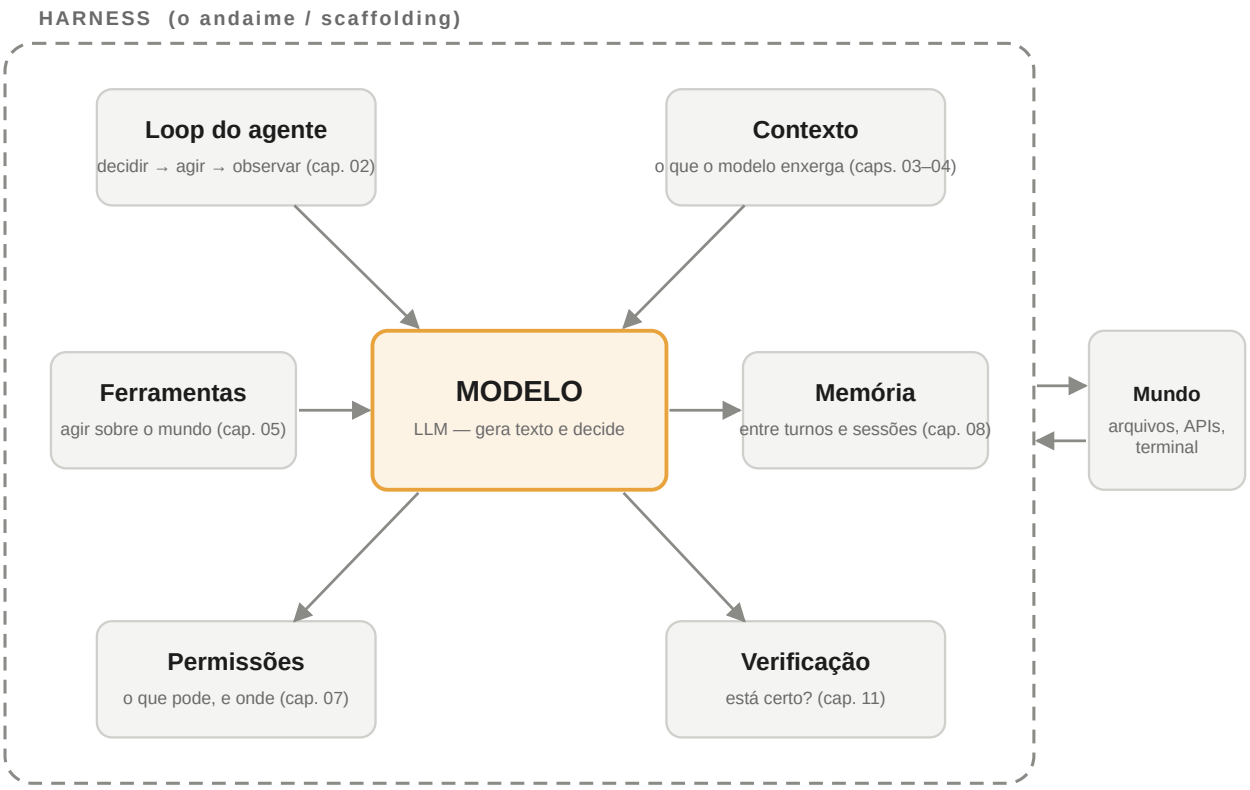
Agent = model + harness

Start with a question anyone who has used an AI chat can ask: why does ChatGPT *answer questions about* your problem, but not *solve* your problem? It explains how to fix the bug — but it does not open the file, run the test, or check that it worked. The short answer: a chat is just the **model**. For the model to *act* — touch files, run commands, verify its own work, and stop at the right moment — an entire structure must be built around it. That structure is the subject of this book.

When an AI agent solves a real task — fixing a bug, migrating a module, answering questions grounded in dozens of files — two distinct things are at work. The first is the **model**: the network that reads context and decides the next step. The second is everything around it: whoever assembles the context it reads, whoever executes the tools it invokes, whoever decides what it may or may not do, whoever remembers what happened yesterday, whoever checks whether the result is correct. That "everything around it" is the **harness** — the rigging, the scaffold, the *scaffolding*.

The formula that organizes this book is simple:

> **agent = model + harness**



The model at the center; the harness — the scaffolding — around it. Each block is a chapter of this book.

The model is interchangeable and improves with every generation. The harness is classic software engineering — and it is where most agents fail or succeed. Two products using exactly the same model deliver radically different results depending on the quality of the harness: how context reaches the model, which tools it has, how errors come back, what happens when the **context window** (the limit of text the model can "see" at once) runs out.

Harness engineering is the discipline of designing that scaffolding: context delivery, tool interfaces, planning artifacts, verification loops, memory systems and sandboxes.

Why a book — and why now

Between 2024 and 2026, coding-agent harnesses stopped being experiments and became a product category: Claude Code, Codex CLI, Gemini CLI, opencode, Aider, Cline, Goose, OpenHands and dozens of others. The most remarkable thing is not the quantity but the **convergence**: independent projects, in different languages, arrived at the same solutions — hierarchical context files, layered compaction, plan mode as a permission mode, lifecycle hooks, MCP (Model Context Protocol) as the integration standard.

When independent implementations converge, there is a discipline behind them. This book documents that discipline.

The method: read code, not marketing

This book is empirical. Each chapter covers one harness capability (the loop, context, compaction, permissions...) and is written from reading the source code of real open source harnesses. The project's most important editorial rule:

> Claims about a harness require **evidence**: the file path in the source code where the capability is implemented.

READMEs promise; code delivers. Several projects advertise dimensions their code does not have — the evidence requirement is what separates evaluation from marketing.

A note on authorship and method

For transparency — and consistency with the evidence rule above — this book is **co-written with an AI agent** (Claude Code, by Anthropic) operating under **human authorship, curation and responsibility**. The agent carries out the research, the writing and the production cycle; the human author defines the scope, decides, **verifies every source** and answers for the content. Following editorial authorship policies (ICMJE, COPE, *Nature*, *Science*), the AI is **not** listed as an author — it cannot be held responsible — and its use is disclosed here, at the opening.

This is not a detail: a book about the discipline of properly instrumenting AI agents uses that very discipline to write itself, and exposes it. The full method — dual research verified by cross-search, spec-driven cycle, review and dating — is documented in the Editorial Guide §6, with a survey of the traditional and AI-era writing methodologies that ground it.

How to read this book — three doors in

This book was written to be dense; this section exists so the density is not a wall. Pick your door:

- **If you are just arriving** (you have used AI chats but never built an agent): read 00 → 01 → 02 in order, unhurried, with the Glossary as support — every acronym in the book is there, spelled out and explained (in the online version, just hover over the acronym). After 02, chapters 03–13 can be read in any order: each is self-contained and opens by defining its own problem.
- **If you already operate an agent** (you use Claude Code, Codex, Cursor or similar and want to understand what is inside): the **Executive summary** at the end of each chapter is your shortcut — the state of the art of the dimension in one paragraph, with the "what to steal" section. Go straight to the chapters you care about and descend into the body when you want the evidence.
- **If you build harnesses**: the whole book is yours, including the Appendices A (per-repository evidence, with file paths), the Benchmark and the two hands-on tracks — **harness-zero** (didactic build, one feature per step) and **harness-um** (the complete reference implementation, own appendix).

Structure of the book

- **Foundations** (chapter 01): the formal definitions, the canonical papers and the problem taxonomy that organizes everything that follows.
- **Chapters 02–13**: one capability per chapter. Each defines the problem, presents the known implementation patterns and shows, with evidence, how each studied harness implements it.
- **Convergences and trends** (chapter 14): what the industry has already standardized, where real divergence remains, and the "expiration clause" — the thesis that every harness component exists because the model cannot yet do that on its own, and must be designed knowing that one day it will be unnecessary.
- **Chapters 15–17**: the frontiers — the harness embedded in a product (15), the harness that learns from usage (16) and the protocol layer that binds the ecosystem together (17).
- **Benchmark** (`benchmark/`): the empirical section — standardized, per-dimension evaluations, with 0–3 scores and evidence, of every harness studied, plus the consolidated comparison.

The harnesses in the study

As of this edition, the study covers **twenty-one open source systems**, evaluated through systematic code reading across five archetypes (the method is in chapter 01, §6):

- **Coding harnesses** — opencode, gemini-cli, OpenHarness, Codex CLI, Goose, Aider, OpenHands, Grok Build, Pi, Kimi Code and Prime Agent;
- **Self-hosted personal agents** — OpenClaw, Hermes Agent, IronClaw, ohmo;
- **Organizational agents** — QM;
- **Embedded harnesses** — n8n (AI Agent node);
- **Frameworks** — LangGraph, CrewAI, OpenAI Agents SDK (Software Development Kit), Software Agent SDK.

Each was chosen for representing a different *archetype* (replication logic, not sampling): mature provider-agnostic product (opencode), big-tech control regime (gemini-cli), readable didactic port (OpenHarness), sandbox-first (Codex CLI), MCP-native (Goose), context-first (Aider), academic eval culture (OpenHands), whole-organization agent with a swappable loop (QM), and so on.

The full list — with the **exact origin, version, fork and commit read** in each evaluation, and the link to each one's analysis and diagnosis — is in the **Appendix — The study**. The consolidated per-dimension scoreboard is in the Comparative.

As a theoretical reference and to explore the ecosystem beyond the corpus, there is also the living collection **Awesome Harness Engineering** (~426 resources organized by problem, in the same organization as this book) — the source of the harness definition used in chapter 01 and of the taxonomy that structures the chapters.