2026 EDITION

AI AGENT ENGINEERING · COMPLETE GUIDE

하네스 엔지니어링 완전 정복

AI 에이전트를 제대로 다루는 기술

Constrain Inform Verify Correct

by 바이브코딩 태일러

@shuntailor · shuntailor.net

130+ sources · 23 chapters · Free distribution

하네스 엔지니어링 완전 정복
AI 에이전트 시대의 새로운 상식

초판 발행: 2026년 3월
저자: shuntailor (테일러의 아지트)

Threads: @shuntailor
블로그: shuntailor.net

본문의 정보는 2026년 3월 기준입니다.
AI 분야는 변화가 빠르므로 최신 정보는 각 공식 문서를 참조하세요.

이 책에 인용된 모든 소스는 부록 C에 정리되어 있습니다.

FROM THE AUTHOR

저자의 말

이 책 한 권이면 됩니다. 비용은 0원입니다.

2026년, AI 개발의 키워드가 바뀌었습니다. 하네스 엔지니어링. AI 에이전트가 일하는 환경 전체를 설계하는 기술입니다. Agent = Model + Harness. 모델이 두뇌라면, 하네스는 그 두뇌가 제대로 일하게 만드는 모든 것입니다.

겁먹을 필요 없습니다. 하네스는 여러분이 이미 쓰고 있는 기술들의 조합입니다. AGENTS.md 작성, 테스트 자동화, 도구 연결, 에러 복구. 따로 놓으면 익숙한 것들이지만, 체계적으로 묶어서 설명하는 자료가 없었습니다. 이 책이 그 첫 번째입니다.

200페이지에 130개 이상의 공식 소스를 분석해서 담았습니다. 올해의 트렌드, 하네스의 모든 것.

읽기 전에 세 가지만 말씀드립니다.

첫째, AI는 빠르게 변합니다. 이 책도 빠르게 업데이트합니다. 파일 이름의 V1, V2가 버전 표시입니다.

둘째, 추천 공부법이 있습니다. 읽다가 막히면 AI에게 바로 질문하세요. 뒤로 갈수록 난이도가 올라가기 때문에, 앞에서 막힌 채로 넘어가면 뒷부분이 안 읽힙니다.

셋째, 끝까지 읽으면 비용이 1/5로 줄어듭니다. Opus 4.6 하네스 없이 쓰는 것보다 Sonnet 4.6에 하네스를 얹는 쪽이 결과물이 더 좋습니다. 가격은 1/5. 그런데 Opus 4.6에 하네스까지 완벽하게 다룬다면, 완전히 다른 세상이 열립니다.

팔로우해 주시는 모든 분이 2026년 대박 나시길.

바이브코딩 태일러

@shuntailor

TABLE OF CONTENTS

목차

00

프롤로그: 왜 이 책인가

> 에이전트가 실수할 때마다, 그 실수를 두 번 다시 일으키지 않게 하는 장치를 설계한다.
>
> — Mitchell Hashimoto — 'My AI Adoption Journey', 2026년 2월 5일

같은 AI, 다른 결과

2026년 2월, LangChain이 실험 결과 하나를 공개했다.

GPT-5.2-Codex. 모델은 동일하다. 한 글자도 바꾸지 않았다. 바뀐 건 모델 바깥의 환경뿐이었다. 시스템 프롬프트, 도구 구성, 에러 복구 루프, 컨텍스트 관리 전략. 이걸 통틀어 '하네스'라고 부른다.

TerminalBench 2.0 점수가 52.8%에서 66.5%로 올랐다. 리더보드 순위로 치면 약 30위에서 5위. 같은 모델이다. 코드 한 줄 안 건드렸다. 하네스만 바꿨을 뿐이다.

KEY INSIGHT

같은 모델이 하네스에 따라 33위도 되고 5위도 된다. 이것이 이 책 전체를 관통하는 명제다: 모델 선택보다 하네스 설계가 성과를 더 크게 좌우한다.

비슷한 시기에 OpenAI가 발표한 수치도 흥미롭다. 100만 줄이 넘는 프로덕션 코드. 인간이 직접 작성한 코드는 0줄. 5개월. 엔지니어 3~7명. 그들이 5개월 동안 한 일은 코딩이 아니었다. 하네스를 설계하고 다듬었다. 마이크로VM 격리, 자동 테스트 루프, 에러 피드백 체계. 모델한테 '무엇을 시킬까'가 아니라 '어떤 환경에서 일하게 할까'를 설계한 것이다.

Stripe는 매주 1,000건 이상의 PR을 에이전트가 자동으로 머지하고 있다. 이것도 모델이 뛰어나서가 아니다. 코드 리뷰 하네스, 테스트 게이트, 롤백 장치가 촘촘하게 깔려 있기 때문이다.

핵심 테제

이 책은 하나의 문장 위에 세워졌다.

"모델보다 하네스가 성과를 결정한다."

2025년까지 업계의 관심은 '어떤 모델이 더 나은가'에 쏠려 있었다. GPT-4 vs. Claude 3.5 vs. Gemini. 벤치마크 1~2% 차이에 희비가 갈렸다. 하지만 2026년, 게임의 규칙이 바뀌었다.

같은 모델이라도 하네스가 좋으면 상위 5위 안에 든다. 반대로, 최고의 모델이라도 하네스가 부실하면 30위 밖으로 밀려난다. 모델은 상향 평준화됐다. 차이를 만드는 건 모델 바깥이다.

2026년 2월 5일, Terraform과 Vagrant를 만든 Mitchell Hashimoto가 이 개념에 이름을 붙였다. '하네스 엔지니어링(Harness Engineering)'. 말의 에너지를 방향으로 바꾸는 마구(harness)처럼, AI 모델의 지능을 안정적인 결과로 바꾸는 환경을 설계하는 기술이다.

이 이름이 나온 지 2주 만에 OpenAI가 공식 채택했다. Anthropic, LangChain, Martin Fowler, Philipp Schmid가 잇달아 관련 글을 발표했다. 2026년 3월 현재, 하네스 엔지니어링은 AI 개발의 가장 뜨거운 키워드다.

이 책이 당신에게 주는 것

이 책은 하네스 엔지니어링을 처음부터 끝까지 다룬다. 130개 이상의 공식 소스를 분석했다. Hashimoto 원문, OpenAI 공식 발표, Anthropic 기술 블로그, LangChain 실험 보고서, Martin Fowler의 분석, arXiv 논문까지. 흩어진 지식을 하나의 체계로 정리한 것이 이 책이다.

- 23개 챕터 + 3개 부록, 약 255페이지
- 130+ 공식 소스 기반 — 블로그 감상문이 아니라 1차 소스 분석
- 4대 플랫폼(Claude Code, Codex, Cursor, Devin) 하네스 완전 비교
- 복사해서 바로 쓰는 AGENTS.md, Eval 템플릿 포함
- 1장부터 21장까지 하나의 프로젝트(TaskFlow)가 진화하는 러닝 케이스

읽고 나면 이렇게 된다. 어떤 AI 에이전트를 쓰든, '모델을 바꿀까' 대신 '환경을 어떻게 설계할까'를 먼저 묻게 된다. 그리고 그 질문에 대한 답을 스스로 설계할 수 있게 된다.

이 책의 구성

6개 파트로 나뉜다. '왜'에서 시작해서 '무엇을', '어떻게', '비교', '운영', '실습'으로 이어진다. 각 파트는 독립적으로도 읽을 수 있지만, 순서대로 읽으면 하나의 이야기가 된다.

파트	주제	핵심 질문
파트 1	왜 지금인가	바이브코딩의 한계는 무엇이고, 왜 하네스가 필요한가?
파트 2	4개의 기둥	Constrain · Inform · Verify · Correct란 무엇인가?
파트 3	실전 하네스 설계	AGENTS.md, 도구, 컨텍스트를 어떻게 설계하나?
파트 4	플랫폼별 비교	Claude Code, Codex, Cursor, Devin의 차이는?
파트 5	운영과 최적화	Eval, 멀티에이전트, 보안, 비용은 어떻게 관리하나?
파트 6	실전 프로젝트	실제로 나만의 하네스를 어떻게 구축하나?

읽는 방법

세 가지 경로가 있다. 자신에게 맞는 걸 고르면 된다.

순서대로 읽기

하네스 엔지니어링을 처음 접하는 사람. 파트 1부터 차근차근. 이론 → 실전 → 심화의 자연스러운 흐름을 따라간다.

필요한 파트만

이미 에이전트를 운영 중인 사람. 파트 4(플랫폼 비교)나 파트 5(Eval, 보안)부터 직접 필요한 부분을 골라 읽는다.

실습부터 시작

손으로 배우는 타입. 파트 6의 실전 프로젝트(19~21장)를 먼저 따라한 뒤, 궁금한 이론은 해당 챕터로 돌아가서 채운다.

레퍼런스로 활용

이미 개념은 아는 사람. 부록의 소스 목록과 템플릿을 레퍼런스 문서처럼 사용한다. 필요할 때 꺼내 본다.

러닝 케이스: TaskFlow

이 책에는 하나의 프로젝트가 처음부터 끝까지 함께한다. 'TaskFlow'라는 할 일 관리 웹앱이다.

1장에서 바이브코딩으로 MVP를 뚝딱 만든다. 동작은 하지만 테스트도 없고, 보안도 신경 쓰지 않은 상태다. 챕터가 진행될수록 하네스를 하나씩 추가한다. 4장에서 제약 조건을, 5장에서 컨텍스트를, 6장에서 검증 체계를, 7장에서 에러 복구를 붙인다. 21장쯤 되면, TaskFlow는 프로덕션에 내놔도 되는 프로젝트가 되어 있다.

RUNNING CASE

TaskFlow 코드는 각 챕터 끝의 'TaskFlow 진행 상황' 섹션에서 확인할 수 있다. 바이브코딩 MVP에서 출발해 완전한 하네스 프레임워크로 진화하는 과정을 21개 챕터에 걸쳐 직접 추적한다.

저자 shuntailor는 AI 에이전트와 하네스 설계를 연구하는 엔지니어이자 콘텐츠 크리에이터다. Threads(@shuntailor)에서 AI 개발 트렌드를 공유하고 있으며, 이 책은 130개 이상의 공식 소스를 분석·정리한 결과물이다.

자, 시작하자. 왜 바이브코딩만으로는 안 되는지, 1장에서 이야기한다.

PART 1

왜 지금인가

바이브코딩의 한계와 하네스 엔지니어링의 등장

01

바이브코딩만으로는 안 되는 이유

바이브코딩이 열어준 세계

금요일 밤, TaskFlow라는 할 일 관리 앱을 만들기로 했다. Claude에게 "할 일 목록을 관리할 수 있는 웹앱을 만들어줘"라고 입력했다. 3분 뒤, 화면에 깔끔한 To-Do 앱이 돌아가고 있었다. 드래그 앤 드롭도 되고, 다크 모드도 적용돼 있었다. 코드를 한 줄도 쓰지 않았다.

이런 경험을 한 사람이 2025년 한 해에만 수십만 명 생겼다. 2025년 2월, Andrej Karpathy가 트윗 하나를 올렸다.

> There's a new kind of coding I call 'vibe coding', where you fully give in to the vibes, embrace expon entials, and forget that the code even exists.
>
> — Andrej Karpathy, 2025년 2월

코드를 읽지도 않는다. 코드가 존재한다는 사실 자체를 잊어도 된다. AI에게 느낌만 전달하면 된다. 이 선언은 개발 문화의 판을 바꿨다.

바이브코딩이 가져온 변화는 단순한 생산성 향상이 아니었다. 코딩 경험이 없는 디자이너, 마케터, 기획자도 자기 아이디어를 직접 프로토타입으로 구현할 수 있게 됐다. Lovable, Bolt, Replit Agent 같은 도구가 등장하면서 "코딩의 민주화"가 현실이 됐다. TaskFlow 같은 앱을 만드는 데 필요한 시간이 수 주에서 수 시간으로 줄었다.

프로토타입과 프로덕션 사이의 골짜기

문제는 그 다음이었다. TaskFlow에 사용자 인증을 붙이려고 했다. 이메일 로그인, 비밀번호 리셋, 세션 관리. AI가 만들어준 코드는 동작했지만, 비밀번호를 평문으로 저장하고 있었다. 세션 토큰에 만료 시간이 없었다.

"동작하는 것"과 "신뢰할 수 있는 것" 사이에 깊은 골짜기가 있었다.

바이브코딩은 0에서 1로 가는 데는 압도적이다. 개인 프로젝트, 해커톤 프로토타입, 주말 사이드 프로젝트에서는 이보다 효율적인 방법을 찾기 어렵다. 하지만 그 1을 10으로, 100으로 키우려는 순간 벽이 나타난다.

벽에 부딪히는 순간들

'눈으로 확인'의 한계

바이브코딩의 검증 방식은 "눈으로 확인(human eyeballing)"이다. AI가 코드를 생성하면, 사람이 눈으로 결과를 확인하고 "괜찮아 보이면" 넘어간다. 10줄짜리 함수라면 괜찮다. 하지만 TaskFlow가 500줄, 1,000줄, 5,000줄로 커지면?

에이전트가 한 번에 200줄의 코드를 출력할 때, 그 안에 숨어 있는 SQL 인젝션 취약점이나 경합 조건(race condition)을 눈으로 발견할 수 있을까. 2025~2026년 보안 분석 결과, AI 생성 코드의 약 45%에서 보안 취약점이 발견됐다. 동작 여부가

아니라 안전 여부가 문제였다.

> **WARNING**
> AI가 작성한 코드가 '동작한다'는 것은 '안전하다'는 뜻이 아니다. 바이브코딩에서 가장 위험한 착각은 "돌아가니까 괜찮겠지"다.

재현 불가능한 결과

TaskFlow에 태그 기능을 추가해달라고 요청했다. 월요일에는 태그를 배열로 구현했고, 수요일에 같은 요청을 하니 별도의 Tag 모델을 만들어서 다대다 관계로 구현했다. 금요일에는 또 다른 접근법이 나왔다.

같은 프롬프트를 3번 입력하면 3개의 다른 코드가 나온다. LLM은 확률적으로 다음 토큰을 생성하기 때문에, 동일한 입력에 대해 동일한 출력을 보장하지 않는다. 아키텍처 결정이 세션마다 바뀌고, 네이밍 컨벤션이 들쭉날쭉하며, 에러 핸들링 전략이 파일마다 다르다.

프로토타입에서는 이게 문제가 되지 않는다. 코드가 10,000줄을 넘어가는 순간, 일관성 없는 코드베이스는 유지보수 불가능한 상태가 된다. 바이브코딩에는 '테스트'가 없다는 것이 여기서 치명적으로 작용한다.

컨텍스트 유실과 기술 부채

TaskFlow 개발을 3주째 이어가고 있었다. 어느 날 에이전트가 기존 API 구조를 완전히 무시한 코드를 생성했다. 첫 주에 설계한 RESTful 엔드포인트 규칙을 3주차의 에이전트는 전혀 기억하지 못했다.

컨텍스트 윈도우에는 한계가 있다. 세션이 길어지거나 새 세션을 시작하면, 에이전트는 이전의 설계 결정을 잊는다. 문서화되지 않은 결정들이 쌓이면서, 아무도 "왜 이렇게 구현했는지" 모르는 코드가 늘어난다. 이것이 바이브코딩이 만드는 기술 부채(technical debt)다.

한 파일을 고치면 다른 파일이 깨진다. 그걸 고치면 또 다른 곳이 망가진다. 이 끝없는 수정 루프에 한 번이라도 빠져 본 사람이라면, 어딘가에서 근본적으로 잘못되고 있다는 걸 알 것이다.

차원	바이브코딩	하네스 엔지니어링
접근법	비공식적, 대화형 프롬프팅	체계적 인프라 설계
검증 방식	사람이 눈으로 확인	자동 테스트 + Eval 파이프라인
적합한 범위	프로토타입, 개인 프로젝트	프로덕션, 팀 개발, 고위험 시스템
스케일	단일 개발자, 단기 세션	팀, 다개월 코드베이스
품질 보증	"돌아가면 OK"	자동화된 에이전트 상호 리뷰
컨텍스트 관리	세션에 의존	AGENTS.md, 메모리 아키텍처

직감에서 시스템으로

여기까지 읽으면 "바이브코딩은 쓸모없다"는 결론으로 가는 것 같지만, 그렇지 않다.

바이브코딩은 입문이지, 목적지가 아니다.

바이브코딩을 통해 우리는 AI 에이전트가 어떻게 작동하는지에 대한 직관을 키웠다. 어떤 프롬프트가 좋은 결과를 만들고, 어떤 상황에서 AI가 혼란에 빠지는지를 경험으로 알게 됐다. 이 직관이 없으면 하네스를 설계할 수도 없다.

Martin Fowler의 블로그에 기고한 Birgitta Bockeler(Thoughtworks)가 이런 관점을 명쾌하게 정리했다.

> 바이브코딩은 '코딩의 민주화'였다. 누구나 만들 수 있게 됐다. 하지만 '제대로 돌아가는 것을 만드는 것'은 여전히 엔지니어링이 필요하다. 그 엔지니어링의 대상이 '코드'에서 '환경(하네스)'으로 바뀐 것이다.
>
> — Birgitta Bockeler, Martin Fowler's blog, 2026년 2월

대상이 바뀌었을 뿐, 엔지니어링의 필요성은 사라지지 않았다. 코드를 직접 쓰는 대신, AI가 코드를 잘 쓸 수 있는 환경을 설계한다. 그것이 하네스 엔지니어링이고, 이 책에서 다루려는 핵심 주제다.

바이브코더에서 하네스 엔지니어로

전환의 핵심은 관점의 변화다. "AI에게 뭘 시킬까"에서 "AI가 잘 동작하려면 어떤 환경이 필요할까"로 질문이 바뀐다.

이전 질문 (바이브코딩)	이후 질문 (하네스 엔지니어링)
어떤 모델이 가장 좋을까?	어떤 환경을 설계해야 할까?
프롬프트를 어떻게 쓸까?	컨텍스트를 어떻게 관리할까?
AI가 잘 만들어줬으면...	AI가 실수해도 시스템이 잡아준다
결과물 = 모델의 능력	결과물 = 하네스의 품질

TaskFlow 이야기로 돌아가자. 바이브코딩으로 만든 TaskFlow MVP는 데모용으로는 충분했다. 하지만 실제 사용자에게 배포하려면, 인증 보안, 데이터 일관성, 에러 복구, 테스트 커버리지 — 이 모든 것이 필요했다. 이것들을 수작업으로 해결하는 게 아니라, 에이전트가 스스로 지키도록 하는 시스템을 만드는 것. 그것이 앞으로 이 책에서 배울 내용이다.

> **POINT**
> 당신이 바이브코더라면, 가장 먼저 해야 할 일은 바이브코딩을 버리는 게 아니라, 그 위에 검증 레이어를 쌓는 것이다. AGENTS.md 한 장을 작성하는 것부터 시작할 수 있다.

바이브코딩의 한계를 느꼈다면, 해결책을 정의한 사람의 이야기를 들어보자. Mitchell Hashimoto가 'harness engineering'이라는 이름을 붙이기까지의 여정이다.

02

Agent = Model + Harness

명명자 Mitchell Hashimoto

Terraform, Vagrant, Consul. 인프라 자동화 분야에서 일하는 사람이라면 이 이름들을 모를 수가 없다. 전부 Mitchell Hashimoto가 만들었다. HashiCorp를 창업해 수십억 달러 기업으로 키운 뒤 퇴사하고, 지금은 Ghostty라는 터미널 에뮬레이터를 만들고 있다.

2026년 2월 5일, 그가 블로그에 'My AI Adoption Journey'라는 글을 올렸다. Ghostty 개발에 AI 에이전트를 도입하면서 겪은 시행착오를 기록한 글이었다. 이 포스트에서 하나의 용어가 탄생했다.

> I don't know if there is a broad industry-accepted term for this yet, but I've grown to calling this 'harness engineering.' It is the idea that anytime you find an agent makes a mistake, you take the time to engineer a solution such that the agent never makes that mistake again.
>
> — Mitchell Hashimoto, 'My AI Adoption Journey', 2026년 2월 5일

"업계에서 통용되는 용어가 있는지는 모르겠지만, 나는 이걸 '하네스 엔지니어링'이라고 부르기로 했다." 그의 표현은 겸손했지만, 이 단어는 폭발적으로 퍼졌다.

Ghostty AGENTS.md의 실체

Hashimoto는 Ghostty 프로젝트에 AGENTS.md라는 파일을 만들어뒀다. 이 파일의 구조가 독특하다. 과거에 에이전트가 저지른 실수 1건마다, 그 실수를 방지하는 규칙 1줄이 추가된다.

```markdown
# AGENTS.md — Ghostty Project
## Rules
- Never modify files in src/core/ without explicit permission
- Always run `zig build test` before committing
- Use snake_case for all function names

## Past Failures → Prevention
- Do not use `std.mem.alloc` directly → use arena allocator
- Config parsing: always validate UTF-8 before processing
- Terminal escape sequences: test with all 6 major terminals
```

에이전트가 직접 메모리 할당을 사용해서 메모리 누수를 일으킨 적이 있다. → arena allocator를 사용하라는 규칙이 추가됐다. UTF-8 검증 없이 설정 파일을 파싱해서 크래시가 발생했다. → 반드시 UTF-8을 검증하라는 규칙이 추가됐다. 실패에서 학습한 방지책이 한 줄 한 줄 쌓여서, 에이전트의 작업 환경 자체가 진화한다.

Hashimoto는 이렇게 정리했다.

> 에이전트가 실수할 때마다, 그 실수를 두 번 다시 일으키지 않게 하는 장치를 설계한다. 그게 하네스 엔지니어링이다.
>
> — Mitchell Hashimoto

같은 달, OpenAI가 "Harness Engineering: Leveraging Codex in an Agent-First World"라는 글을 공식 발표하면서 이 용어를 채택했다. OpenAI의 발표는 단순한 용어 사용이 아니었다. "100만 줄 이상의 프로덕션 코드를, 인간이 한 줄도 쓰지 않고 생성했다. 핵심은 모델이 아니라 하네스였다." 하네스 엔지니어링은 순식간에 업계의 공통 언어가 됐다.

Agent = Model + Harness 공식

Agent = Model + Harness

날것의 LLM은 에이전트가 아니다. GPT-5든 Claude Opus든, 모델 그 자체는 "지능"만 갖고 있다. 상태를 기억하지 못하고, 도구를 사용할 줄 모르며, 실수를 감지할 수 없고, 자기 자신을 제어할 수 없다.

하네스가 이 모든 것을 부여한다. 상태 관리, 도구 실행, 피드백 루프, 제약 조건. 하네스가 모델을 감싸야 비로소 에이전트가 된다.

각 요소가 없으면 어떻게 되나

이 공식의 각 요소가 빠졌을 때 어떤 일이 벌어지는지 보면, 왜 하네스가 필수인지 분명해진다.

빠진 요소	현상	결과
Model만 있고 Harness 없음	API를 직접 호출, 채팅만 가능	바이브코딩 수준에서 정체
Harness만 있고 Model 없음	도구·규칙은 있지만 추론 능력 없음	규칙 기반 자동화(기존 스크립트)
Model + 부분적 Harness	도구는 있지만 검증·수정 없음	동작하지만 신뢰 불가
Model + 완전한 Harness	추론 + 도구 + 검증 + 수정	프로덕션 수준의 에이전트

컴퓨터 아키텍처 비유

Hugging Face의 Philipp Schmid가 이 관계를 컴퓨터에 빗대어 설명했다. 업계에서 널리 인용되는 비유다.

컴퓨터	AI 에이전트	역할
CPU	기반 모델 (LLM)	추론 능력, 사고력
RAM	컨텍스트 윈도우	휘발성 작업 메모리
OS	에이전트 하네스	컨텍스트 관리, 부트 시퀀스, 라이프사이클 관리

Application	에이전트	사용자 고유의 로직과 워크플로우

이 비유에서 핵심적인 통찰이 하나 있다. 같은 CPU를 장착해도, OS가 다르면 퍼포먼스가 달라진다. 동일한 하드웨어에서 Linux와 Windows의 성능이 다르듯, 동일한 LLM이라도 하네스가 다르면 결과물의 품질이 완전히 달라진다. TerminalBench 2.0에서 같은 Opus 4.6 모델이 하네스에 따라 33위도, 5위도 된 것이 바로 이 원리다.

TaskFlow로 돌아와서 생각해보자. 1장에서 바이브코딩으로 만든 TaskFlow MVP가 있다. Model은 이미 있다(Claude). 하지만 Harness가 없다. 테스트 자동화도, 코드 스타일 규칙도, 에러 복구 로직도, 컨텍스트 관리 체계도 없다. 지금의 TaskFlow는 "Model만 있고 Harness 없음" 상태다. 바이브코딩 수준에 머물러 있는 이유가 여기에 있다.

4개의 기능적 기둥

그렇다면 하네스는 구체적으로 뭘 하는가? OpenAI, Martin Fowler, Anthropic의 분류를 종합하면, 하네스의 기능은 4가지 기둥으로 집약된다.

Constrain (제약)
에이전트가 '할 수 있는 것'을 제한한다. 샌드박스, 도구 허용 리스트, 파일 경로 제한, 비용 상한선. 해공간을 좁힐수록 에이전트의 정확도는 높아진다.

Inform (정보 제공)
에이전트에게 '무엇을 해야 하는지' 알려준다. AGENTS.md, CLAUDE.md, 프로젝트 문서 정비. 적시에 필요한 최소한의 고품질 컨텍스트를 제공한다.

Verify (검증)
에이전트가 '올바르게 실행했는지' 확인한다. Eval 파이프라인, 자동 테스트, 셀프 검증 루프. 확률적 출력에 대한 허용 범위 기반 검증.

Correct (수정)
에이전트가 '실수했을 때' 복구한다. 에러 리커버리, 재시도 로직, Human-in-the-loop. 실패를 학습으로 전환하는 피드백 루프.

이 4개 기둥은 파트 2에서 각각 한 챕터씩 깊이 다룬다. 지금은 전체 구조를 머릿속에 잡아두면 된다. 중요한 점은 이 4가지가 독립적이 아니라 서로 연결되어 있다는 것이다. Constrain이 해공간을 좁히면 Verify의 부담이 줄고, Inform이 정확할수록 Correct가 작동할 일이 적어진다.

말의 하네스에서 AI의 하네스로

"하네스"라는 단어의 어원은 승마다. 말에게 채우는 고삐와 안장 — 강력한 동물의 에너지를 생산적인 방향으로 유도하는 장치. 말의 힘 자체를 바꾸는 게 아니라, 그 힘이 올바른 방향으로 발휘되도록 제어한다.

AI 하네스도 같은 원리다. 모델의 추론 능력 자체는 건드리지 않는다. 모델의 "바깥쪽"을 설계해서, 그 지능이 안정적이고 예측 가능한 결과물로 변환되도록 만든다. 파인튜닝이나 모델 교체가 아니라 환경 설계로 성과를 끌어올리는 것. 이 접근법이 2026년 AI 개발의 패러다임이 됐다.

용어 정리: 하네스 엔지니어링의 위치

3개의 시대: 프롬프트 → 컨텍스트 → 하네스

AI 개발은 3개의 시대를 거쳐왔다. 각 시대는 "인간이 무엇을 설계하는가"로 구분된다. 이 프레임워크를 이해하면, 현재 자신이 어디에 있고 어디로 가야 하는지가 보인다.

1기: 프롬프트 엔지니어링
핵심 질문: 어떤 말을 쓸 것인가?
작업 단위: 단일 API 호출
인간의 역할: 프롬프트 작성자
문제 해결: 프롬프트를 다시 쓴다

2기: 컨텍스트 엔지니어링
핵심 질문: 어떤 정보가 필요한가?
작업 단위: 멀티턴 세션
인간의 역할: 정보 아키텍트
문제 해결: 컨텍스트 데이터를 조정한다

3기: 하네스 엔지니어링
핵심 질문: 어떤 환경이 필요한가?
작업 단위: 완성된 기능
인간의 역할: 환경 디자이너
문제 해결: 도구·가드레일을 추가한다

결정적으로, 이 3개 시대는 교체가 아니라 누적이다. 하네스 엔지니어링은 프롬프트 엔지니어링과 컨텍스트 엔지니어링을 포함한다. 이전 시대의 기술이 쓸모없어지는 게 아니라, 더 큰 프레임워크 안에 흡수되는 것이다.

하네스 vs. 에이전틱 vs. 프롬프트

비슷한 용어가 많아서 혼란스러울 수 있다. 관계를 정리하자.

용어	범위	초점	비유
프롬프트 엔지니어링	단일 상호작용	입력 텍스트 최적화	질문하는 기술
컨텍스트 엔지니어링	멀티턴 세션	정보 설계·관리	도서관 사서
하네스 엔지니어링	에이전트 라이프사이클	인프라 설계 (도구·제약·검증·수정)	운영체제 설계
에이전틱 엔지니어링	전체 개발 프로세스	에이전트 오케스트레이션 + 하네스 + 배포	시스템 통합

에이전틱 엔지니어링은 가장 넓은 상위 개념이다. AI 에이전트의 계획·구현·테스트·배포 전 프로세스를 다룬다. 하네스 엔지니어링은 그 안에서 "인프라 설계"에 특화된 부분 집합이다. LangChain의 Vivek Trivedy는 이렇게 구분했다. "오케스트레이터는 로직과 제어 흐름을 담당하고, 하네스는 도구와 사이드이펙트를 담당한다. 둘은 함께 동작한다."

POINT

Hashimoto의 AGENTS.md 작성 원칙: 에이전트가 실수를 저지를 때마다, 방지책을 1줄 추가한다. 삭제하지 않는다. 시간이 지나면 이 파일 자체가 프로젝트의 '학습된 지혜'가 된다.

개념은 이해했다. 그런데 이게 진짜 효과가 있는 걸까? "모델보다 하네스가 중요하다"는 주장이 구호가 아니라 사실인지, 숫자로 검증해 보자.

03

숫자가 증명하는 것

벤치마크가 보여준 역전

2026년 2월 17일, LangChain이 블로그에 올린 글 한 편이 업계를 뒤흔들었다. "Improving Deep Agents with Harness Engineering." 제목만 봐서는 평범한 기술 포스트 같았다. 하지만 안에 담긴 숫자는 평범하지 않았다.

LangChain: 모델 고정, 하네스만 변경으로 14%p 향상

LangChain 팀은 deepagents-cli라는 코딩 에이전트의 성능을 개선하는 실험을 했다. 실험 조건은 단순했다. 모델은 GPT-5.2-Codex로 완전히 고정한다. 바꾸는 것은 오직 하네스뿐이다.

구체적으로 뭘 바꿨나. 3가지 미들웨어를 추가했다.

- PreCompletionChecklistMiddleware — 에이전트가 답을 제출하기 전에 체크리스트를 스스로 검증하게 하는 셀프 검증 루프
- LocalContextMiddleware — 작업 환경의 디렉터리 구조, 패키지 매니저, 프레임워크 정보를 자동으로 수집해서 컨텍스트에 주입
- LoopDetectionMiddleware — 에이전트가 같은 파일을 반복 수정하는 '둠 루프'를 탐지해서 중단

여기에 "추론 샌드위치(reasoning sandwich)" 전략을 적용했다. 계획 단계에서 높은 추론 예산을 쓰고, 구현 단계에서는 표준 수준으로 낮추고, 최종 검증 단계에서 다시 높인다. 모든 트레이스는 LangSmith에 기록해서 데이터 기반으로 반복 개선했다.

두 줄의 표가 모든 것을 말해준다.

지표	변경 전	변경 후	변화
TerminalBench 2.0 점수	52.8%	66.5%	+13.7%p
리더보드 순위	약 30위	Top 5	약 25단계 상승
변경된 것	—	하네스만	모델 동일

모델을 한 단계 업그레이드해도 벤치마크 점수가 2~5%p 오르기 어려운 시대에, 하네스 변경만으로 13.7%p가 올랐다. 모델은 완전히 동일했다.

TerminalBench: 같은 모델, 28단계 순위 차이

더 충격적인 데이터가 있다. TerminalBench 2.0 리더보드에서, 완전히 동일한 모델인 Opus 4.6이 사용하는 하네스에 따라 전혀 다른 순위를 기록했다.

모델	하네스	순위
Claude Opus 4.6	Claude Code 기본 하네스	33위
Claude Opus 4.6	최적화된 대체 하네스	5위

같은 두뇌. 다른 환경. 28단계의 순위 차이. 이 한 장의 표가 "모델보다 하네스가 성과를 결정한다"는 이 책의 핵심 테제를 증명한다.

KEY INSIGHT

모델을 교체해서 5%를 개선하는 것보다, 하네스를 설계해서 15%를 개선하는 것이 현실적이다. 이것이 2026년 AI 개발의 새로운 상식이다.

프로덕션에서의 증거

벤치마크는 실험실이다. "실전에서도 그런가?"라는 질문이 자연스럽다. 답은 "그렇다"이고, 규모는 벤치마크보다 훨씬 크다.

OpenAI: 100만 줄, 인간 코드 제로

OpenAI는 자사의 Codex 에이전트를 개발하면서, 5개월간 하네스 설계에 집중했다. 투입 인원은 3~7명의 엔지니어. 이 팀이 생산한 코드는 100만 줄이 넘는다. 인간이 직접 작성한 코드는 한 줄도 없다.

하루 평균 3.5개의 PR을 엔지니어 1인당 처리했고, 총 약 1,500개의 PR이 머지됐다. 수동 코딩 대비 10배 속도로 추정됐다. 심지어 프로젝트 초기의 AGENTS.md 파일마저 Codex가 작성했다.

OpenAI가 공유한 5가지 핵심 교훈을 보면, 이들이 5개월을 어디에 쏟았는지 알 수 있다.

1. 환경 설계 > 모델 성능 — 진척이 느리면 AI의 한계가 아니라 환경의 결함이다
2. "지도를 줘라" — 리포지토리를 유일한 진실의 원천(Single Source of Truth)으로 만든다
3. 해공간을 제약하라 — 패턴, 경계, 구조를 명시할수록 에이전트의 정확도가 올라간다
4. 실패에서 반복하라 — 에이전트가 막히면, 부족한 도구·가드레일·문서를 찾아 보충한다
5. 가비지 컬렉션 — 정기적으로 에이전트를 돌려 문서의 불일치와 규칙 위반을 탐지한다

5개 교훈 중 단 하나도 "더 좋은 모델을 써라"가 아니다. 전부 환경 설계에 관한 것이다.

Stripe: 주간 1,000건 PR 완전 자동 머지

Stripe는 매주 1,000건 이상의 풀 리퀘스트를 완전히 무인(unattended)으로 자동 머지하고 있다. 이 규모를 가능하게 한 것은 멀티 에이전트 하네스다.

Stripe의 접근법에는 3가지 특징이 있다. 첫째, 작은 에이전트 여러 개가 각자의 전문 영역을 담당한다. 둘째, 매 스텝마다 검증이 이루어진다 — 컴파일, 테스트, 스키마 체크. 셋째, 컨텍스트 관리 시스템이 각 스텝에 필요한 정보만 정확히 제공한다. 필요한 것만, 필요한 시점에, 필요한 에이전트에게.

Shopify, Airbnb, 그리고 Manus

Shopify는 AI 에이전트의 출력을 제품 QA와 동일한 엄격도로 평가한다. 자동화된 평가 파이프라인이 AI 출력을 그라운드 트루스와 비교하고, 프롬프트가 바뀔 때마다 회귀 테스트를 실행한다. 코드 리뷰 시간이 50% 단축됐다.

Airbnb는 대규모 JavaScript → TypeScript 마이그레이션에 LLM 에이전트를 투입했다. 수백 인월(𝌆𝌆)의 엔지니어 공수가 필요했을 작업을, 하네스가 갖춰진 에이전트가 자동으로 처리했다. 대화형 도구로는 불가능한 규모의 일괄 변환이었다.

가장 인상적인 사례는 Manus다. 이 회사는 6개월 동안 에이전트 하네스를 5번 완전히 다시 만들었다. 5번의 재작성. 6개월. 신뢰할 수 있는 하네스를 구축하는 데 이만큼의 투자가 필요하다는 뜻이다.

> You can't download harnesses from Hugging Face.
>
> — Manus 팀 (하네스에 대한 업계 격언)

모델은 API 키 하나로 바꿀 수 있다. 하네스는 수천 시간의 엔지니어링으로만 만들 수 있다. 그래서 하네스가 경쟁 우위(moat)가 된다.

하네스의 경제학

하네스에 투자해야 한다는 건 알겠는데, 비용은 얼마나 들까? Anthropic이 직접 실험한 비용 데이터가 있다.

$9 vs. $200 — 그리고 둘의 차이

Anthropic의 Prithvi Rajasekaran 팀이 레트로 게임 메이커를 만드는 실험을 했다. 동일한 프로젝트를 두 가지 방식으로 접근했다.

접근법	소요 시간	비용	결과
싱글 에이전트 (하네스 없음)	20분	$9	비기능적 결과물
풀 하네스 (3에이전트 시스템)	6시간	$200	완전 기능, 기능 완성
간소화 하네스 (디지털 오디오)	3시간 50분	$124.70	품질 유지, 비용 절감

$9짜리 결과물은 돌아가지 않았다. 20분 만에 코드가 나왔지만, 게임이 실행되지 않았고, 핵심 기능이 구현되지 않았다. "빠르고 싸게"는 "쓸 수 없는 것을 빠르고 싸게" 만든 것일 뿐이었다.

$200짜리 결과물은 완전히 동작했다. 6시간이 걸렸고, 비용이 22배 더 들었지만, 실제로 사용할 수 있는 소프트웨어가 나왔다. Anthropic은 이를 3에이전트 시스템으로 달성했다. Planner가 설계하고, Generator가 구현하고, Evaluator가 Playwright로 자동 검증한다.

여기서 중요한 건 세 번째 행이다. 간소화된 하네스로 비용을 $124.70까지 줄이면서도 품질을 유지했다. 하네스의 비용 효율을 최적화하는 것도 하네스 엔지니어링의 영역이다.