AN ANATOMY OF HARNESS ENGINEERING

# 하네스 엔지니어링 해부학

개념부터 실전 활용, 그리고 솔직한 장단점까지

발표자

**이시영**

맞다

# 발표자 소개

## 이시영

| | |
|---|---|
| 소속 | Potens. — B2B AX 교육·컨설팅 매니저 |
| 커뮤니티 | 맞다AI가 오거나이저 |
| GitHub | krsy0411 |
| LinkedIn | 이시영 |

이번 행사는 두 곳에서 후원해 주셨습니다

Microsoft

OceanLightAI

FIG. 04

# 오늘의 여정

I

## 지식 전달

개념 & 장단점

20분

II

## 시연

라이브 시연

15분

III

## recap

정리 및 Q&A

5분

오프닝 5분 포함 — 총 45분

QUESTION

# 아무 설정 없이 AI 코딩 에이전트랑 그냥 대화하는 것도, 하네스 엔지니어링일까요?

FIG. 06

# 세 용어의 탄생

각 층은 앞 층에서 겪은 불편에서 생겨났습니다

01 · 2022

## 프롬프트 엔지니어링

PROMPT

생겨난 이유

같은 말이라도 표현이 다르면 답이 달라짐

→

02 · 2025

## 컨텍스트 엔지니어링

CONTEXT

생겨난 이유

여러 번 주고받을 땐 '무엇을 보여주느냐'가 답을 좌우함

→

03 · 2026

## 하네스 엔지니어링

HARNESS

생겨난 이유

쓸 만큼 똑똑해졌지만, 믿고 맡기기엔 아직 부족함

FIG. 07

# 세 층 한눈 비교

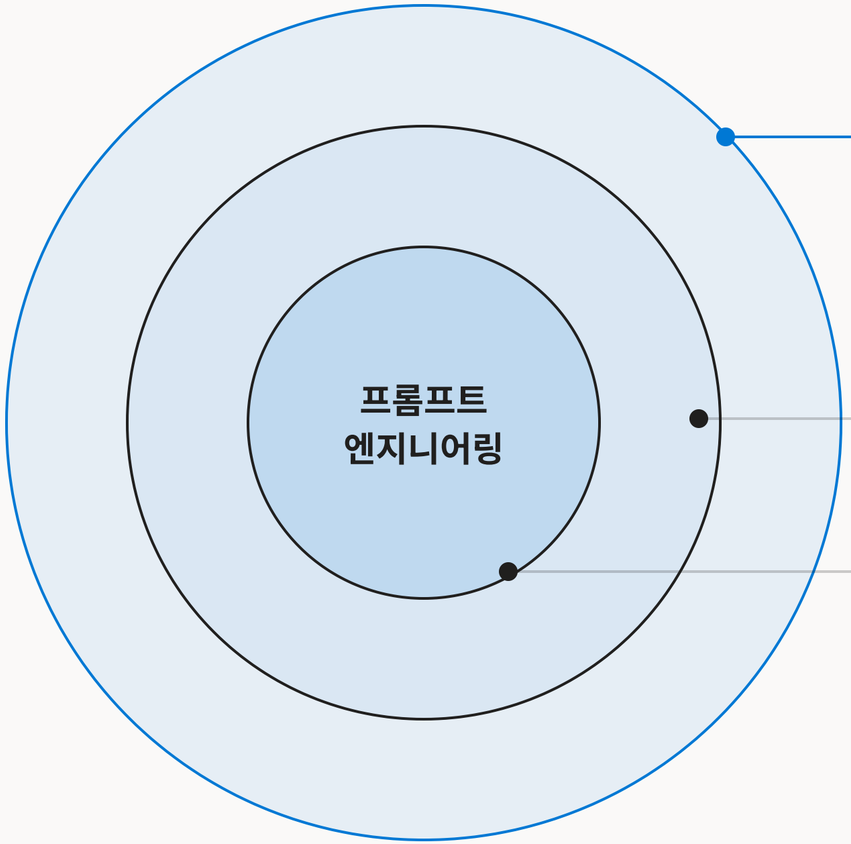
같은 목표라도 손대는 위치가 다릅니다

| | 프롬프트 엔지니어링 PROMPT | 컨텍스트 엔지니어링 CONTEXT | 하네스 엔지니어링 HARNESS |
|---|---|---|---|
| 작동 단위 | 메시지 한 통 | 대화 세션 | **시스템 전체** |
| 다루는 것 | 무엇을 시킬지 | 무엇을 보여줄지 | **어떻게 굴러갈지** |
| 대표 기법 | 역할·출력·제약 지정 | 관련 파일·대화 선별 | **규칙·훅·검증·서브에이전트** |
| 한 문장으로 | 메시지를 다듬는다 | 창에 담을 걸 고른다 | **구조로 강제한다** |

FIG. 08

# 대체가 아니라 중첩

세 층은 하나가 다른 걸 밀어내는 게 아니라 동심원 — 프롬프트는 지금도 가장 안쪽에 남아 있습니다




## 하네스 엔지니어링

시스템이 어떻게 굴러가는지

작동 레벨 – 시스템

## 컨텍스트 엔지니어링

무엇을 보여주는지

작동 레벨 – 세션

## 프롬프트 엔지니어링

무엇을 시키는지

작동 레벨 – 메시지

# 프롬프트 엔지니어링

같은 요청도, 무엇을·어떻게를 잡아 주면 답이 달라집니다

## 이전 그냥 던지기

```
# 이전 — 그냥 던지기
"로그인 버그 고쳐줘"

# 모델이 알아서 추측
어떤 버그? · 어떤 형식?
설명은 얼마나? · 테스트는?
```

✗ 답이 매번 달라지고 산으로 감

## 이후 구조를 잡기

```
# 이후 — 구조를 잡기
역할 : 시니어 백엔드 개발자
입력 : 첨부한 스택트레이스
출력 : 수정 diff 만 · 설명 3줄 이내
제약 : 기존 테스트를 깨지 말 것
```

✓ 정확하고 일관된 결과

# 컨텍스트 엔지니어링

무엇을 보여줄지 골라 주면, 같은 모델도 답이 안정됩니다

## 이전

다 밀어넣기

```
// 이전 – 다 밀어넣기
const ctx = [
  ...allMessages,     // 전체 대화
  ...allFiles,        // 저장소 전체
]
// 창이 넘치고 핵심이 묻힘
```

✗ 토큰 낭비 · 정확도 하락

## 이후

필요한 것만

```
// 이후 – 필요한 것만
const ctx = [
  systemPrompt,          // 역할·규칙
  search(query, 5),      // 관련 파일만
  history.slice(-10),    // 최근 대화만
  testResults,           // 실행 결과
]
```

✓ 핵심만 남아 답이 안정적

# 하네스의 정의

"

## Agent = Model + Harness

에이전트는 모델과 하네스를 합친 것 — 모델을 감싸서 에이전트로 만들어 주는 모든 소프트웨어가 하네스입니다

출처 Mitchell Hashimoto — HashiCorp 공동창업자, 2026년 2월 하네스 엔지니어링 최초 정의

FIG. 12

# 사용과 엔지니어링의 스펙트럼

하네스 없는 에이전트는 없습니다 — 중요한 건 '그 하네스를 누가 만들었느냐'

STAGE 1

## 모델 하나만

API 한 번 호출

에이전트 아님

STAGE 2

## 모델 + 기본 하네스

도구를 기본 설정 그대로 사용

남이 만든 하네스를 그대로 씀

STAGE 3

## 모델 + 커스텀 하네스

규칙 파일 · 스킬 · 훅 · 서브에이전트

여기서부터 엔지니어링

손 안 댐 → 많이 손봄

# 룰북과 하네스

룰북은 안 지켜도 그만인 부탁, 하네스는 아예 못 지나가게 막는 것


## 룰북 RULEBOOK

모델한테 하는 부탁 — 안 지켜도 그만

예: 규칙 파일(CLAUDE.md / AGENTS.md 등) 속 한 줄

## 하네스 HARNESS

구조로 막음 — 못 지나감

예: 테스트 통과 못 하면 완료를 막는 훅

FIG. 14

# 판별 기준 한 줄

## 에이전트가 그 규칙을 어기고도
## 계속 진행할 수 있을까요?

진행 가능 →

**룰북**

프롬프트 · 컨텍스트 층

진행 불가 →

**하네스**

구조 층

FIG. 15

# 하네스 해부도

실제 서비스에 쓰는 하네스의 6가지 기능 묶음

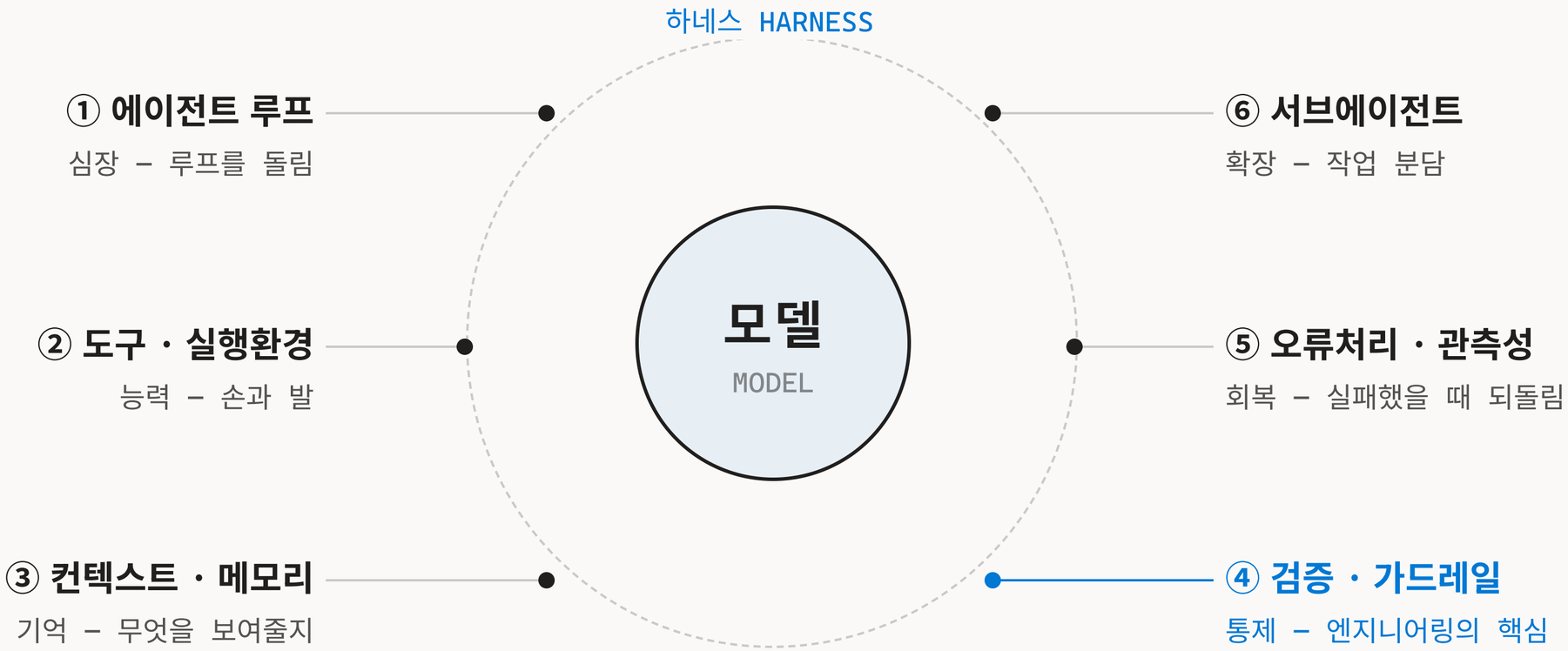


FIG. 16

# 책임의 경계 — 내가 직접 챙기는 부분

하네스는 '모델을 뺀 나머지 전부' — 셋의 차이는 하네스가 있고 없고가 아니라, 미리 만들어진(빌더) 하네스와 내가 직접 쌓는(유저) 하네스의 경계입니다


출처 — M. Fowler · B. Böckeler, "Harness engineering for coding agent users" (martinfowler.com, 2026) · Agent = Model + Harness

FIG. 17

# 층별 기술 스택

각 층을 실제로 구현할 때 자주 쓰는 도구들 — 이름은 외우지 않아도 됩니다

### ① 에이전트 루프 · 오케스트레이션

- LangGraph
- Microsoft Agent Framework
- OpenAI Agents SDK
- CrewAI

### ② 도구 · 실행환경

- MCP
- Function Calling
- Docker 샌드박스
- E2B

### ③ 컨텍스트 · 메모리

- pgvector
- Pinecone
- 임베딩 검색
- 대화 요약

### ④ 검증 · 가드레일

- 린터 · 타입체커
- pytest
- Pydantic / zod
- 훅(hooks)

### ⑤ 오류처리 · 관측성

- LangSmith
- Langfuse
- OpenTelemetry
- 재시도 · 백오프

### ⑥ 서브에이전트

- 멀티에이전트
- 역할 분담
- 커스텀 서브에이전트

# 하네스 엔지니어링 — 실제 예시

'테스트 꼭 돌려라'는 부탁을, 어길 수 없는 구조로 바꾸는 것

## 룰북

안 지켜도 그만

```
# AGENTS.md
커밋하기 전에는 항상
테스트를 실행하세요.
```

✗ 모델이 그냥 지나칠 수 있음

## 하네스

못 지나감

```
// settings.json — 검증 훅
"hooks": {
  "PostToolUse": [{
    "matcher": "Edit|Write",
    "command": "npm test"
  }]
}
// 실패하면 다음 단계가 막힘
```

✓ 구조가 대신 막아 줌

같은 규칙을 '부탁'에서 '강제'로 옮기는 것 — 해부도 ④ 검증·가드레일이 코드로는 이렇게 생김

# 솔직한 장점

하네스는 곁가지가 아니라, 성능과 신뢰성 그 자체입니다

| | | |
|---|---|---|
| 01 | **구조적 신뢰성** | 들쭉날쭉한 출력을 믿을 수 있는 시스템으로 바꿈 |
| 02 | **결과의 재현성** | 한 번 설계하면 팀원 모두가 원하는 output 기대값에 쉽게 도달 가능 |
| 03 | **성능 한계 향상** | 같은 모델이라도 하네스에 따라 벤치마크 점수가 달라짐 |

# 장점, 조금 더 깊이

앞의 세 가지를 실제 현장 관점에서 풀어 보면

01

## 구조적 신뢰성

검증·가드레일이 출력 형식을 강제

실패한 결과는 자동으로 차단·재시도

'운이 좋으면 되는' 상태에서 벗어남

02

## 결과의 재현성

규칙·스킬·검증을 한 번 문서화하면 팀 공용 자산

누가 실행하든 비슷한 품질의 output

신입도 시니어 수준의 기대값에 도달

03

## 성능 한계 향상

같은 모델도 하네스에 따라 벤치마크 점수 상승

모델 교체 없이 성능 상단을 끌어올림

Terminal-Bench 등에서 하네스가 점수를 좌우

# 솔직한 단점

무조건 많이 쌓는다고 좋은 건 아님 — 하네스에도 비용과 함정이 있음

| | | |
|---|---|---|
| 01 | **모델별 재설계 필요** | 모델이 좋아지면 그에 맞춰 하네스를 다시 설계해야 함 |
| 02 | **구축·유지 비용** | 하네스도 결국 코드 — 쌓일수록 관리 부담이 늘어남 |
| 03 | **비용 부담** | 검증·서브에이전트가 늘수록 토큰과 실행 비용이 커짐 |

# 단점, 조금 더 깊이

이 비용과 함정을 조금 더 구체적으로 보면

01

## 모델별 재설계 필요

모델이 능력을 내재화하면 기존 하네스는 과잉

새 모델마다 무엇을 남기고 뺄지 재검토

앤트로픽도 새 모델에선 계획 단계를 삭제

02

## 구축·유지 비용

하네스도 결국 코드 — 테스트·문서·리팩터링 대상

층이 쌓일수록 디버깅·추적이 어려워짐

오래된 규칙은 오히려 방해가 됨

03

## 비용 부담

검증·서브에이전트가 늘수록 토큰·API 호출 증가

실행 시간과 인프라 비용도 함께 상승

효과 대비 비용을 늘 저울질해야 함

하네스 엔지니어링의 본질

# 달래는 걸 멈추고,
# 시스템을 짓는 것.

다음 — 실제 프로젝트를 함께 살펴봅니다

FIG. 24

# 시연

평소 쓰던 프로젝트를 오늘은 다른 코딩 에이전트로 — 하네스가 달라져도 원리는 그대로 통함

시연 프로젝트 나만의 LLM Wiki 프로젝트

① 

### 룰북 층 살펴보기

AGENTS.md 등 지시 문서를 열어 규칙 층 확인

→

② 

### LLM Wiki 프로젝트 아키텍처 확인

프로젝트가 어떤 하네스 층으로 구성됐는지 살펴보기

→

③ 

### Output 통제로 원하는 산출물 생성

검증·가드레일로 결과 형식을 강제해 원하는 형태의 산출물 얻기

# 요약과 Q&A

① 세 층은 대체가 아니라 중첩

② 구조로 강제하느냐가 룰북과 하네스를 나눔

③ 신뢰성·성능을 얻지만 재설계와 비용이 따름 — 얼마나 쌓을지는 판단의 몫

NEXT
다음 행사 안내

맞다

# 맞다톤

MATDA-THON · HACKATHON

## 하루 만에 AI 서비스 MVP를 만들고 배포하는 해커톤

| | |
|---|---|
| 일시 | 2026. 8. 22 (토) · 09:00–18:00 |
| 장소 | 대구권 행사장 (최종 장소 추후 공지) |
| 대상 | AI에 관심 있는 누구나 · 비전공/전공 무관 |
| 참가비 | 얼리버드 1만원 · 일반 2만원 |

참가자 전원

**GitHub Copilot Max 1개월 이용권 무료**

수상팀

**공식 GitHub 굿즈 · 총 50만원 상당**

선착순 100명 · 얼리버드 30명 한정 · 준비물 개인 노트북

후원 Microsoft · GitHub · 문의 matdaaiga@outlook.com

THANK YOU

# 감사합니다



이시영 · krsy0411