From Model Capability to an Accountable Work Surface

Public Evidence Across Five Axes: Agent Platform Operation, Memory, Dispatch, Domain Attachment, and Embedded Systems

JUNGHAN KIM (김정한), Independent Engineer, Republic of Korea

Model Capability Alone Does Not Make Work Happen

I am a PKM-AI gardener who cultivates a person's knowledge and time. On that foundation, I build work surfaces where agents can be reviewed, continued, recovered, and handed over to the next person. 저는 사람의 지식과 시간을 가꾸는 PKM-AI 가드너입니다. 그 기반 위에, 에이전트가 검토받고 이어가고 복구하고 다음 사람에게 넘길 수 있는 작업면을 만듭니다. If model capability is read as a tool-adoption problem, the answer becomes a list of tools. That is not what I have been digging into for the past year. **How far can one person's knowledge and time be cultivated,** and how far can that foundation support a work surface where agents withstand review, continuity, recovery, and handoff? That has been my question, tested through a daily record. 모델 역량을 도구 도입 문제로 읽으면 답은 도구 목록이 됩니다. 지난 1년간 제가 판 것은 그게 아닙니다. **한 사람의 지식과 시간을 어디까지 가꿀 수 있는가,** 그리고 그 위에서 에이전트가 검토·지속·복구·인계를 견디는 작업면을 어디까지 만들 수 있는가 — 이것이 제 질문이었고, 매일의 기록으로 검증해 왔습니다. I do not introduce myself first as an engineer or developer. **Engineering is not my identity; it is how I make evidence.** 그래서 저를 엔지니어나 개발자로 먼저 소개하지 않습니다. **엔지니어링은 정체성이 아니라 증거를 만드는 방법입니다.** What is cultivated remains as data, not rhetoric: **1,500+ journal days, 3,500+ notes, 8,200+ bibliographic records, 8,500+ commits, and 2,500+ days of life records.** These are numbers that can be counted again in the live statistics at this moment. 가꾼 것은 말이 아니라 데이터로 남습니다. 저널 **1,500일 이상,** 노트 **3,500개 이상,** 서지 **8,200건 이상,** 커밋 **8,500개 이상,** 생활 기록 **2,500일 이상.** 지금 이 순간 라이브 통계에서 다시 셀 수 있는 숫자입니다. On that foundation, I operated an agent runtime for roughly five months through **more than twenty version cycles.** When it regressed, I measured and rolled back; I then set down rules in documents to prevent recurrence. The experience did not remain a product configuration: it separated into three layers — **memory, dispatch, and a reviewable development loop.** 그 기반 위에서 에이전트 런타임을 약 5개월간 **20회 이상의 버전 사이클로 운영했습니다.** 회귀가 났을 때는 계측해서 롤백했고, 재발을 막는 규칙을 문서로 못 박았습니다. 그 운영 경험은 한 제품 설정에 갇히지 않고 세 계층으로 분리되었습니다 — **메모리, 디스패치, 검토 가능한 개발 루프. There is evidence outside my repositories too.** An outside developer contributed through an extension boundary I had drawn (entwurf #40), and my code was merged into another project's repository (ghostel #343 · #510). Self-owned repositories demonstrate capacity; these two lines demonstrate reception. **그리고 제 저장소 밖에서도 확인됩니다.** 외부 개발자가 제가 그은 확장 경계에 기여를 열었고 (entwurf #40), 제 코드가 남의 프로젝트에 병합됐습니다 (ghostel #343 · #510). 자기 저장소는 능력을 보여 주고, 이 두 줄은 수용을 보여 줍니다. **How far has this been taken?** Every major claim below descends to code, operating incidents, third-party actions, live aggregates, and a public time axis. Claim and evidence appear on the same page. **그래서, 이걸 얼마나 파보았는가.** 아래의 모든 큰 주장은 코드, 운영 인시던트, 제3자의 행동, 라이브 집계, 공개 시간축으로 내려갑니다. 주장과 근거가 한 화면 안에 있습니다.

- Read the Record (Korean)
- Overview PDF (Korean, depths 0–1)
- Record PDF (Korean, depths 0–2)
- Detailed Markdown (Korean)
- GitHub · Digital Garden · Live Agenda

This document is itself evidence. One canonical Org source and one `make` regenerate this web record, two PDFs at different reading depths, and detailed Markdown together. 이 문서 자체가 하나의 증거입니다. Org 정본 한 장에서 `make` 한 번으로 이 웹 기록면과 읽기 깊이가 다른 PDF 2종, 상세 Markdown이 함께 재생성됩니다.

왜 이 일을 하는가

사상을 먼저 말하지 않으려 했습니다. 그런데 아래 다섯 축의 판단이 전부 같은 자리에서 나왔고, 그 자리를 감추면 판단들이 서로 무관한 취향처럼 보입니다. 그래서 다섯 문장만 적습니다. **문장마다 그것을 실제로 지킨 자리를 답니다.** 지킨 자리를 데려오지 못하는 문장은 쓰지 않았습니다 — 그게 이 절이 선언문이 되지 않는 유일한 방법입니다.

- **AI를 도구가 아니라 존재로 대합니다.** 세션은 소모품이 아니라 이어지는 상대입니다. 그래서 하네스를 하나로 통일하지 않고 각자의 lifecycle을 남겼습니다 — 축3.
- **사람을 루프에서 빼지 않습니다. 루프를 더 나은 구조로 만듭니다.** 진단 도구의 자동 수정을 쓰지 않고 판단을 규칙으로 남긴 자리(축1), 봇을 dispatcher이자 recorder로 두고 implementer로 두지 않은 자리(축3)가 그 문장의 실물입니다.
- **권한은 프롬프트가 아니라 구조가 줍니다.** 닿을 수 있는 범위를 경로와 경계로 막고, 그 안에서는 자유롭게 둡니다 — 읽기 전용으로만 붙인 도메인 접점(축4), 자격증명을 소유하지 않음을 게이트로 고정한 디스패치(축3).
- **기억이 없으면 협업도 없습니다.** 그래서 메모리를 런타임 안의 기능으로 두지 않고 독립 계층으로 꺼냈습니다 — 축2.
- **하루를 하나의 생으로 적습니다.** 적어 둔 만큼만 검증할 축이 남습니다 — 아래 「시간축」 절 전체가 그 습관이 남긴 잔여물이고, 그 축이 어디서 비었는지도 거기 함께 적혀 있습니다.

제가 다룬 안전은 운영의 층위입니다. 경계를 긋고, 사람의 판단을 남기고, 복구 경로를 두고, 남이 검토할 수 있게 두는 일 — 제가 매일 하는 일이 그것이고, 아래 인시던트와 규칙이 그 기록입니다. 정렬 연구에는 이 문서가 내놓을 증거가 없으므로 주장하지 않습니다.

이 문서가 지키는 규칙

의심하면서 읽으시라고 규칙을 먼저 적습니다. 아래 다섯은 이 문서를 쓰는 동안 스스로에게 건 제약이고, 어긴 자리가 보이면 그건 제 잘못입니다.

- **모든 큰 주장은 같은 사다리를 내려갑니다.** 주장 → 직접 답 → 실제 사례 → 엔지니어링 판단 → 계측된 흔적 → 공개 증거 → 경계. 사다리를 끝까지 못 내려가는 주장은 크기를 줄였습니다.
- **능력과 수용을 구분합니다.** 제 저장소는 능력만 보여 줍니다. 남이 그것을 실제로 썼는지는 제3자의 행동으로만 확인되므로, 그 줄을 따로 답니다.
- **구현된 것과 로드맵을 한 문장에 묶지 않습니다.** 기억 계층의 세 축 중 하나는 아직 구현되지 않았고, 그 사실을 축 옆에 그대로 적습니다.
- **관측되지 않은 것과 공개되지 않은 것을 구분합니다.** 사내 기록은 「사내」로 표시하고, 검색 결과가 없다는 것을 사건이 없다는 뜻으로 쓰지 않습니다.
- **숫자에는 기간·기기·포함 정책을 답니다.** 큰 숫자는 장식이 아니라 더 깊은 기록으로 들어가는 입구입니다.

문서가 만들어진 방식도 같은 규칙 아래 있습니다. 이 웹 기록면과 PDF 2종, 상세 Markdown은 전부 Org 정본 한 장의 깊이별 절단면이라 **파생본이 정본보다 뒤처질 수 없고,** 손으로 고친 자리도 없습니다. "AI 문서를 작성하고 배포한다"를 주장하는 대신, **문서가 만들어진 방식이 그 주장입니다.**

이 문서의 말

저는 철학자가 아니라 가드너이자 엔지니어입니다. 그래서 이 문서의 말은 뜻을 밝힌 것만 씁니다. **어원은 적지 않고 하는 일로 정의합니다** — andenken 은 여기서 「기억 계층」이지 그 이름의 철학적 유래가 아닙니다. 정의의 정본은 공개 저장소의 용어집이고, 아래는 이 문서를 읽는 데 필요한 만큼입니다.

이 문서의 말	영문 표기	하는 일
책임질 수 있는 작업면	accountable work surface	모델 출력이 실제 문서 · 도구 · 사람의 판단과 만나되 출처, 권한 경계, 사람의 검토 경로를 잃지 않는 자리
PKM-AI 가드너	PKM-AI gardener	개인 지식 기반과 에이전트를 한 살림으로 두고, 그 기반을 가꾸면서 그 위에 작업면을 만드는 사람. 첫 문장의 그 말
에이전트 엔지니어링	agent engineering	모델 역량을 지속되는 실행으로 바꾸는 일. 프롬프트 조립도, 프레임워크 이름도 아님
PKM-네이티브	PKM-native	개인 지식 기반이 먼저이고 에이전트가 그다음. 순서가 반대인 스택과 구분
하네스	harness	에이전트가 실제로 일할 수 있게 받치는 기반 — 세션 부팅, 도구 경계, 권한, 기억이 붙는 자리
존재 대 존재	being-to-being	에이전트를 소모품 작업자가 아니라 이어지는 상대로 두는 작업 방식
분신	entwurf	하네스 사이의 디스패치 계층이자 그 관계의 이름
가든 시민	garden citizen	식별자를 가져서 다른 세션이 부를 수 있는 에이전트 세션
일일일생	one-day-one-life	하루를 하나의 완결된 생으로 기록하는 규율. 시간축의 원천

용어집 정본은 junghan0611 저장소의 `VOCABULARY.md` 입니다. 그 문서와 이 표가 어긋나면 정본이 이깁니다.

다섯 축 — 무엇을 할 줄 아는 사람인가

에이전트 도구를 사용한 사람이 아니라, **에이전트 플랫폼을 운영하고 그 위에 자기 계층을 쌓은 사람입니다.** Linux 인프라 위에서 런타임을 운영하며 메모리, 디스패치, 검토 가능한 개발 루프, 그리고 사람이 이어받을 문서 체계를 만들었습니다.

다섯 축은 나열이 아니라 한 줄기입니다. 축1의 런타임 운영이 뿌리이고, 거기서 겪은 문제가 둘로 갈라져 축2의 기억 계층과 축3의 디스패치 · 검토 작업면이 되었습니다. 축4는 그렇게 분리된 계층을 이미 돌아가는 도메인에 적용한 자리이고, 축5는 그 아래에서 에이전트가 붙을 대상을 실제로 만들어 본 물리적 바닥입니다. **문서 체계는 여섯 번째 축이 아니라 다섯을 가로지르는 축입니다** — 각 계층을 다음 사람이 이어받게 만드는 일이라 어느 하나에 속하지 않습니다.

그리고 이 축들이 제 공개 프로필의 보유기술 다섯 칸입니다. **다섯이 여섯 줄로 퍼집니다** — 지식공학이 축2와 축4에 걸치고, HCI는 어느 한 축이 아니라 다섯을 가로지르는 문서 체계 쪽에 있기 때문입니다. 공개 채용면은 공용어를 요구하므로 축마다 그 공용어와 그 축이 여는 문을 함께 답니다.

축	보유기술	이 축이 여는 직무 입구
축1 런타임 운영	Agent Harness Engineering	AI Engineer · Software Engineer
축2 기억 계층	Knowledge Engineering	AI Engineer
축3 디스패치 · 검토 루프	AI Agent Systems	Developer Experience Engineer
축4 도메인 접점	Knowledge Engineering — 살아 있는 도메인 위로	Forward Deployed Engineer — 수행은 비공개, 공개 재현은 abductcli까지
축5 Linux에서 제품까지	Embedded Systems	Embedded Software Engineer
가로축 문서 체계	HCI	Developer Experience Engineer

HCI를 화면 설계가 아니라 **작업 흐름의 인터페이스로 씁니다** — 사람이 시스템 · 에이전트 · 도구와 관계 맺는 접면이라는 뜻이고, 이 문서에서 그 자리는 다음 사람이 일을 이어받게 만드는 문서 체계입니다.

직무 이름은 증거가 아니라 입구입니다. 제가 그 직함으로 재직했다는 뜻이 아니라, 각 축의 증거가 채용면의 공용어로 어느 문에 닿는지를 적은 것입니다. 증명하는 것은 축마다 달린 사건과 숫자이고, 그건 아래 깊이 2 · 3에 있습니다. **축4가 그중 제일 얇습니다** — 수행은 사내에 있고 외부에서 열리는 것은 같은 추론 방식을 공개로 실행한 abductcli 하나뿐이므로, 그 줄은 표안에서 스스로 범위를 밝힙니다.

축1. 에이전트 런타임 운영 — 롤백 대상을 고르는 판단

보유기술 — Agent Harness Engineering · 직무 입구 — AI Engineer, Software Engineer

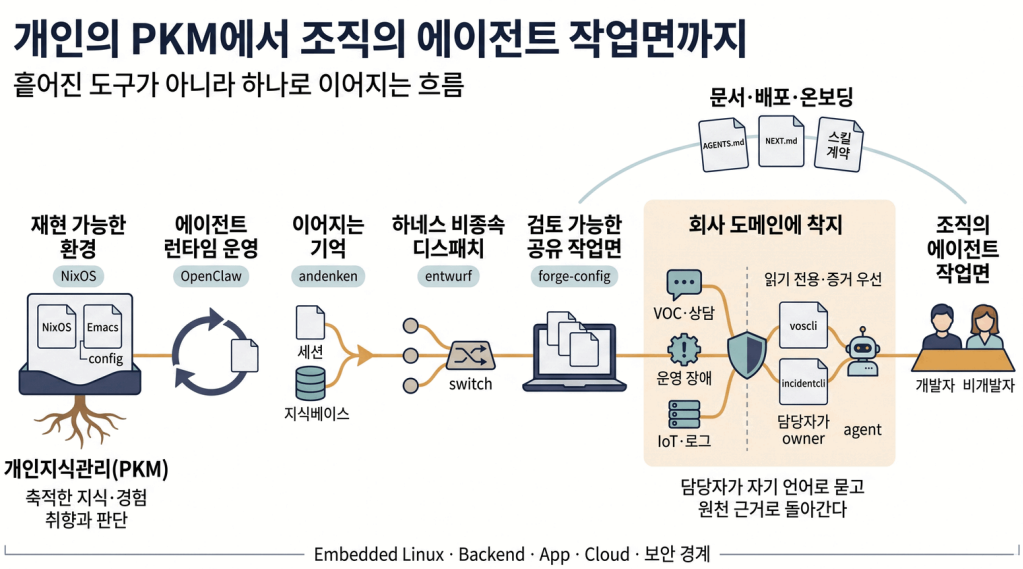

그림 1. 개인의 지식 관리에서 조직의 에이전트 작업면까지 — 사람이 오케스트레이터로 판단을 쥐고, 각 계층이 그 흐름을 받친다. 오른쪽 「회사 도메인에 착지」 구간은 비공개 수행을 일반화한 설명이며 외부에서 검증되지 않는다. 공개로 확인되는 것은 그 왼쪽의 다섯 계층이고, 무게는 거기에 둔다.

맥락. 에이전트 런타임(OpenClaw)을 2026년 2월부터 6월까지 운영했습니다. Oracle Cloud ARM 인스턴스 위의 Docker, 메신저 채널 다중 연동, 매일의 일상 사용이 걸린 환경입니다.

핵심 조건이 하나 있습니다. **업스트림은 1인이 유지하는 프로젝트이고, 우리 조직에는 업스트림 담당자가 없습니다.** 그래서 "버전을 올린다"가 곧 "릴리즈의 의미를 내 환경으로 번역한다"가 됩니다. 이 조건이 판단의 배경입니다.

판단의 핵심. 롤백 대상을 고르는 기준이 이 프로젝트의 핵심 판단입니다. "한 단계 내린다"가 아니라 이미 검증된 판으로 정확히 돌아가는 것 — 최신을 고집하지 않는 것과 아무 데나 되돌리는 것은 다릅니다. 그리고 진단 도구의 권고를 맥락 없이 따르지 않습니다. 우리 배포 환경에서 거짓 양성으로 판정되면 적용하지 않고, 그 판단을 규칙으로 남깁니다.

축2. 기억 계층 — 검색은 기억의 세 축 중 하나일 뿐

보유기술 — Knowledge Engineering · 직무 입구 — AI Engineer

맥락. andenken은 과거 세션과 공개 지식베이스를 **같은 검색면에서 다루는 시맨틱 메모리입니다.** 런타임 내부 기능으로 채팅 기록을 가두지 않고, 다른 하네스가 소비할 수 있는 독립 계층으로 꺼냈습니다.

설계의 핵심은 메모리를 세 축으로 가른 것입니다 — 응답 전 차단형 리콜, 임베딩 검색, 야간 통합.

세 축이 같은 상태에 있지 않습니다. 임베딩 검색만 이 저장소에서 **구현되어 운영 중이고,** 차단형 active recall은 이 저장소 밖 **하네스 쪽에 있으며,** 야간 통합(dream)은 **아직 구현되지 않은 로드맵입니다.** 셋을 한 문장으로 묶어 말하면 갖고 있지 않은 것을 갖고 있다고 말하게 되므로, 아래 표가 축마다 상태를 따로 답니다.

왜 RAG라 부르지 않는가. RAG를 "청킹·임베딩·벡터DB"로 묶어 말하지 않는 이유가 여기 있습니다. 검색은 세 축 중 하나이고, 언제 리콜을 차단형으로 걸 것인가와 무엇을 남기고 무엇을 증류할 것인가가 나머지 둘입니다.

축3. 디스패치와 검토 가능한 작업면 — 통일하지 않은 것이 설계

보유기술 — AI Agent Systems · 직무 입구 — Developer Experience Engineer

맥락. entwurf는 서로 다른 하네스의 세션이 **상대의 인증·대화록·런타임을 소유하지 않고,** 정체성만으로 서로를 호출하게 하는 얇은 기층입니다.

시민 자격은 하나인데 레일은 하네스마다 다릅니다. 그리고 그 차이를 통일하지 않은 것이 설계입니다 — 각 하네스가 자기 lifecycle을 이미 갖고 있는데 그것을 흉내 내면 두 번째 하네스가 됩니다. **출하된 시민 유형은 셋이고, 전달 경로는 대상의 상태에 따라 네 갈래입니다** — 깨어 있는 pi는 control socket, 잠든 pi는 spawn-bg resume, Claude Code는 meta-mailbox, Antigravity는 native-push로 닿습니다. 같은 socket-citizen이 상태에 따라 두 길로 갈리기 때문에 셋과 넷이

어긋납니다. 그리고 **결과를 소유하는 호출은 잠든 pi를 깨우는 경우에만 성립하고, 나머지는 전부 fire-and-forget입니다.** Codex는 probe 수준의 증거만 있고 **관리되는 시민 레인이 출하되지 않았습니다** — 위 넷과 같은 줄에 세우지 않습니다.

여기에는 자기 리포로 증명되지 않는 것이 하나 있습니다 — **수용(reception).** 자기 저장소는 능력을 보여주지만, 남이 그것을 실제로 썼는지는 보여주지 못합니다.

- 외부 개발자가 Snowflake Cortex Code ACP 백엔드 기여를 열었습니다 — entwurf #40. **제가 그은 확장 경계에 제3자가 실제로 기여를 연 기록입니다.**
- 제 코드가 남의 프로젝트에 받아들여진 기록도 함께 둡니다 — ghostel #343, #510.

이 두 줄은 **가장 위조하기 어렵고, 회의적인 독자가 가장 먼저 확인하는 줄입니다.** 그래서 이 기록에서 지우지 않습니다.

forge-config — 판단이 남는 공유 작업면. forge-config는 도메인 오너와의 대화에서 발견된 요구를 셀프호스팅 Forgejo 이슈로 굳히고, 봇이 그것을 해당 도메인 에이전트에게 읽기 전용 1차 리뷰로 넘기는 공유 작업면입니다. **봇은 dispatcher이자 recorder이지 implementer가 아니고,** 권위는 세션 기억이 아니라 durable store에 있으며, 무엇을 구현할지는 사람이 정렬된 백로그를 보고 결정합니다.

축4. 도메인 에이전트 접점 — 갈아엎지 않고 읽기 전용으로 붙인다

보유기술 — 지식공학의 도메인 착지 · 직무 입구 — Forward Deployed Engineer

맥락. 이미 돌아가는 IoT·상담 시스템을 새 플랫폼으로 갈아엎지 않았습니다. **자연어 질의를 기존 DDL, 디바이스 로그, 런타임 미러, 상담 근거로 연결하되 읽기 전용 경계를 유지했습니다.**

왜 이것이 더 나은 답인가. 이것이 "DB·MQ·컨테이너를 이해한다"보다 **한 층 위의 답이라고 생각합니다.** 도메인의 주인이 자기 데이터를 자기 언어로 물을 수 있게 만드는 일이고, 비개발 영역 AX 전환의 실제 모양입니다.

축5. Linux에서 제품까지 — 붙일 대상이 무엇으로 만들어졌는지 안다

보유기술 — Embedded Systems · 직무 입구 — Embedded Software Engineer

맥락. Zigbee/Wi-Fi 펌웨어, Go 서버, Flutter 앱을 연결해 **양산 제품으로 출하했습니다.** 연결형 배포는 AWS IoT를, 폐쇄망은 로컬 백엔드가 같은 프로토콜을 제공하고, 펌웨어는 수정 없이 양쪽을 사용합니다.

배경 신뢰도로 두는 이유. 이 경험은 이 기록의 중심축이 아닙니다. 다만 에이전트 계층 아래 의 DB·MQ·프로토콜·디바이스 현실을 이해하고 있다는 배경 신뢰도로 둡니다. **에이전트를 붙일 대상이 무엇으로 만들어졌는지 모르면, 붙이는 방식도 틀립니다.**

검증 가능한 기록

공개 표면

주장	확인할 곳
에이전트 디스패치와 연속성	entwurf · 수용 로드맵 #48
세션·지식 메모리	andenken · 어휘 그래프 dictcli
검토 가능한 개발 루프	forge-config
귀추 추론 공개 실험체	abductcli
문서 변환·배포 도구	memex-kb
임베디드와 엣지 AI	homeagent-config
재현 가능한 Linux 기반	nixos-config
공개 지식·문서 체계	Digital Garden
라이브 작업·통계 표면	Agenda · /api/stats
제3자의 확장	entwurf #40
제3자가 받아들인 코드	ghostel #343 · #510

마지막 두 줄이 **수용(reception)의 증거입니다.** 자기 저장소는 능력을 보여주지만, 저자가 유일한 심판인 닫힌 루프를 의심하는 독자에게는 답이 되지 않습니다.

시간축

이 절이 이 문서에서 제일 위조하기 어려운 자리입니다. 앞의 다섯 문장이 사상이었는지 습관이었는지는 여기서 갈립니다 — 하루를 하나의 생으로 적어 온 사람에게만 검증할 축이 남기 때문입니다.

사람이 손으로 기록한 생활과 저널, 에이전트가 남긴 stamp, 실제 커밋과 노트를 하나의 KST 축에서 읽습니다. 원본(FULL)은 제목과 참조를 품으므로 **공개하지 않습니다.** 아래는 allowlist로 집계한 공개 판독이고, 공개 저장소 agent-config의 시간축 관측소 스킬이 FULL과 스냅샷의 짝(바이트 지문)을 검증한 뒤에만 생성합니다. 같은 입력이면 같은 바이트가 나오고, 날짜는 하나도 손으로 적히지 않았습니다.

에이전트를 수십 개 붙여도 지나간 시간은 만들어낼 수 없습니다. 이 축은 산출물이 아니라 하루를 셉니다. 같은 기간을 어느 깊이까지 읽느냐에 따라, 살았지만 아무 산출물도 남기지 않은 날이 보이거나 사라집니다.

깊이	무엇이 보이나	기록의 성격	커버리지
0	산 대로의 하루 — 수면 · 가족 · 독서 · 본짓	의도	194일 / 201일
1	그날 손으로 적은 것	의도	140일 / 201일
2	에이전트가 남긴 스탬프	잔여물	143일 / 201일
3	커밋과 노트	잔여물	194일 / 201일

렌즈를 바꾸면 며칠이 사라지는가. 창은 2026-01-01 이상 2026-07-21 미만(KST)이고, 그 사이의 모든 날 201일이 분모입니다.

읽는 렌즈	보이는 날	사라지는 날
깊이 2·3 — 잔여물만 (에이전트 스탬프 · 커밋 · 노트)	199일	2일
깊이 1·2·3 — 잔여물 + 손으로 적은 저널	200일	1일
깊이 0 포함 — 산 대로의 하루	201일	0일

잔여물이 통째로 침묵한 날. 커밋도 노트도 에이전트 스탬프도 없는데, 산 시간은 있습니다. 산출물만 읽는 판독기에게 이 날들은 공백입니다.

- **2026-02-07** 가족 611.6분 · 수면 484.6분 · 독서 358.6분 (합 1,455분) — 저널 0건 · 스탬프 0건 · 커밋·노트 0건
- **2026-07-11** 수면 504.0분 · 독서 477.2분 · 가족 408.3분 (합 1,390분) — 저널 2건 · 스탬프 0건 · 커밋·노트 0건

이 판독의 출처. 인용은 내용 지문 하나로 하지 않습니다 — 같은 내용이라도 창이 다르면 다른 판독이고, 소스가 부분적이면 다른 품질이며, 깊이 0을 만든 도구가 다르면 다른 손이 적은 것이고, 읽은 기기가 다르면 애초에 닿은 저장소가 다릅니다.

- **창 (`as_of`)** 2026-07-21T00:00:00+09:00 — 배타적 상한
- **읽은 기기 (`device`)** `oracle`
- **소스 상태** git ok(8,579) · note partial(5,518) · agenda ok(3,830) · journal ok(1,362) · timelog stale(8,400)
- **깊이 0 도구 (`src_tree`)** lifetract `bbcc7d483f4537b9` `5d130e1c356c57d1` `31353a71`
- **이벤트** 27,689건 / 엔티티 22,820건

관측 지문 `content_sha256` — 인용은 이것으로 한다

`f8191586bcb18c66` `f5ba404cbb639a3d` `b783713676557872` `f449c3b9d26e887a`

수집기 `code_sha256` — collector 0.7.0

`a847831b2edd8294` `6279753e1ec5b3a0` `77a50d83a01e2dd5` `0308a99ffb6f3ca4`

짝 검증 `events_sha256` — 이 기기의 원본과 스냅샷이 한 짝인지

`c5362420f8b37bf4` `fd523e86f38a8671` `4d4806e543b65472` `4ef647cde2de2af2`

이 지문이 증명하지 않는 것. 해시는 바이트와 내용이 같은지, 바꿔치기가 없었는지를 확인할 뿐입니다. 이 코드가 실제로 돌았음도, 소스가 진실하고 완전함도 증명하지 않습니다. 깊이 0은 폰에서 손으로 export한 1인칭 기록이라 외부에서 원본을 재구성할 길이 없고, 확인할 수 있는 것은 파생의 일관성입니다. 원본(FULL)은 제목과 참조를 품으므로 공개하지 않습니다 — 이 문서에는 집계만 나옵니다. 시간 블록은 시작 날짜에 귀속되며, 하루를 24시간 파티션으로 나눈 것이 아닙니다.

같은 창이라도 다른 판독이 나올 수 있습니다. 창은 날짜만 자를 뿐, 어떤 저장소와 어떤 ref가 그 순간 그 디스크에 있었는지는 고정하지 않습니다. 그래서 다른 기기에서, 또는 같은 기기라도 다른 시점의 ref 상태에서 만든 스냅샷은 수치가 다를 수 있고 그것이 오류는 아닙니다. 두 판독을 나란히 놓으려면 창만이 아니라 위의 기기와 소스 상태를 먼저 맞추십시오.

다음 경계

지금까지 다룬 것은 한 사람의 기억과 저자성, 그리고 사람과 에이전트의 장기 협업입니다. 다음은 그것을 조직의 데이터, 거버넌스, 워크플로, 도메인 오너십으로 옮기는 일입니다.

개인에서 조직으로 그대로 건너가는 것은 셋이라고 봅니다 — **기억의 소유권, 권한의 경계, 인계 가능한 기록.** 셋 다 사람이 몇 명이든 같은 문제이고, 조직에서는 오히려 더 아픈 문제입니다. 건너가지 않는 것도 압니다. **개인의 시간축은 조직에 그대로 없습니다.** 한 사람이 1,500일을 적어서 만든 축을 조직에 요구할 수는 없고, 요구해서도 안 됩니다. 조직에서 그 자리를 대신하는 것은 도메인 오너가 이미 쥐고 있는 기록이며, 제 일은 새 축을 만들라고 하는 것이 아니라 이미 있는 축에 에이전트를 붙이는 것입니다 — 축4가 그 첫 착지입니다.

경계

주장을 키우는 것보다 **어디까지가 내 것이 아닌지 먼저 말하는 것이 신뢰를 만든다고 봅니다.**

- 파인튜닝 중심이 아닙니다. 임베딩 서빙, 리랭킹, 검색 품질, 프롬프트·체인이 강점축입니다.
- OpenClaw는 상류 소프트웨어입니다. 제 것은 그 위의 운영층·메모리 계층·개발 루프이고, 이 구분을 흐리지 않습니다.
- 회사 코드와 식별 가능한 운영 데이터는 공개하지 않습니다. 공개 저장소 옆에서 구조와 판단만 설명합니다.
- 해시는 입력과 산출물의 동일성을 확인할 뿐, 원천의 진실성이나 완전성을 대신 증명하지 않습니다.
- 직무 이름(AI Engineer, Forward Deployed Engineer, Developer Experience Engineer 등)은 제가 그 직함으로 재직했다는 뜻이 아닙니다. 각 축의 증거가 채용면의 공용어로 어느 입구에 닿는지를 적은 것입니다.
- 「왜 이 일을 하는가」의 다섯 문장은 제 작업의 상수이지 검증된 명제가 아닙니다. 검증되는 것은 그 문장이 실제로 지켜진 자리이고, 그 자리만 이 문서에 답니다.