2026 전국민 AI 활용사례 공모전 · 업무 생산성 개선

통제된 AI 비서 — 두 사례 정량 임팩트

로컬 LLM 개인 업무비서 + ISBN API 검증 부가세 경정청구 약 20억

사례 1 — 메인

로컬 LLM 개인 업무비서

하루 8시간 → 약 10분 약 98% 시간 단축

새 안건이 떨어졌을 때 **"최근 2년치 우리 팀 활동 통합 조회"**가 하루 8시간에서 약 10분으로 줄었습니다. 약 **98% 시간 단축**. 더 중요한 건 회사 자료(**검토자료 · 전자결재 · 전표 등**)가 외부 API로 일체 나가지 않는다는 점입니다. 인덱싱부터 답변 생성까지 전부 본인 컴퓨터 안에서 끝납니다.

사례 2 — 서브

ISBN API 검증으로 부가세 경정청구 약 20억

한 달 → 5시간 약 99% 시간 단축 · 약 20억 환급

콘텐츠 **3~4만 건 규모**의 ISBN 부가세 면세 요건 검토가 한 달에서 5시간으로 줄었습니다. 약 **99% 시간 단축**. 서지정보시스템 API를 호출해 ISBN·메타데이터를 한 번에 수집하고, **3단 교차 검증(searcher → verifier → review)**으로 정합성을 확보했습니다. 이 과정에서 누락분이 발견되어 **부가세 경정청구로 약 20억 환급**을 받아냈습니다.

핵심 메시지 · "AI는 똑똑하게 쓰는 게 아니라 통제해서 쓰는 것이다"

통제된 AI 비서: 직장인 1명이 만든 디지털 직원 시스템

— 룰을 코드로, 지시는 텔레그램으로

> **핵심 메시지**: AI는 똑똑하게 쓰는 게 아니라 통제해서 쓰는 것이다.

챕터 0. 프롤로그 — 출퇴근길, 나는 이미 일을 시켜두었다

오전 8시 15분, 지하철 안. 나는 스마트폰 텔레그램에 한 줄을 입력했다.

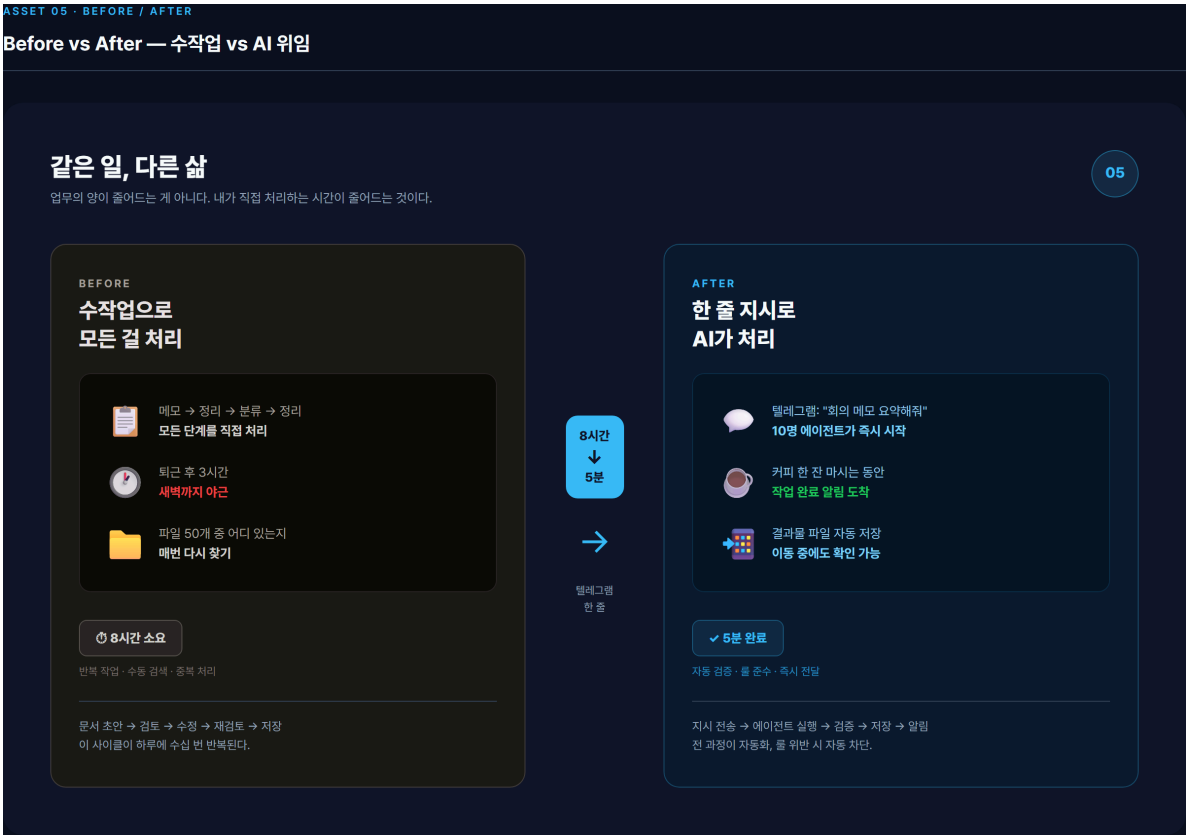
> "어제 회의 메모 정리해서 보고용 요약문 만들어줘."

회사 문을 열 때쯤 알림이 왔다. 요약문 완성, 파일 저장 완료. 내가 아무것도 안 한 것처럼 보이지만, 사실 나는 이미 **10명의 디지털 직원**에게 일을 분배해두었다.

그런데 여기서 질문 하나. AI가 멋대로 내용을 꾸며낸다면? 내가 "절대 이렇게 하면 안 된다"고 정해둔 룰을 슬쩍 비껴간다면? 스마트폰 화면만 보는 출퇴근길에 그걸 어떻게 잡아낼까?

이 글은 그 질문에 대한 나의 답이다. AI 도구로 업무 자동화를 시작한 지 약 2년, 그 중 결정 불변성 거버넌스를 코드 수준으로 다듬은 지 1년 남짓의 기록이다.

> **이 챕터에서 보여줄 시각자산:** `05_BeforeAfter.png` (수작업 8시간 vs 텔레그램 5분 비교)



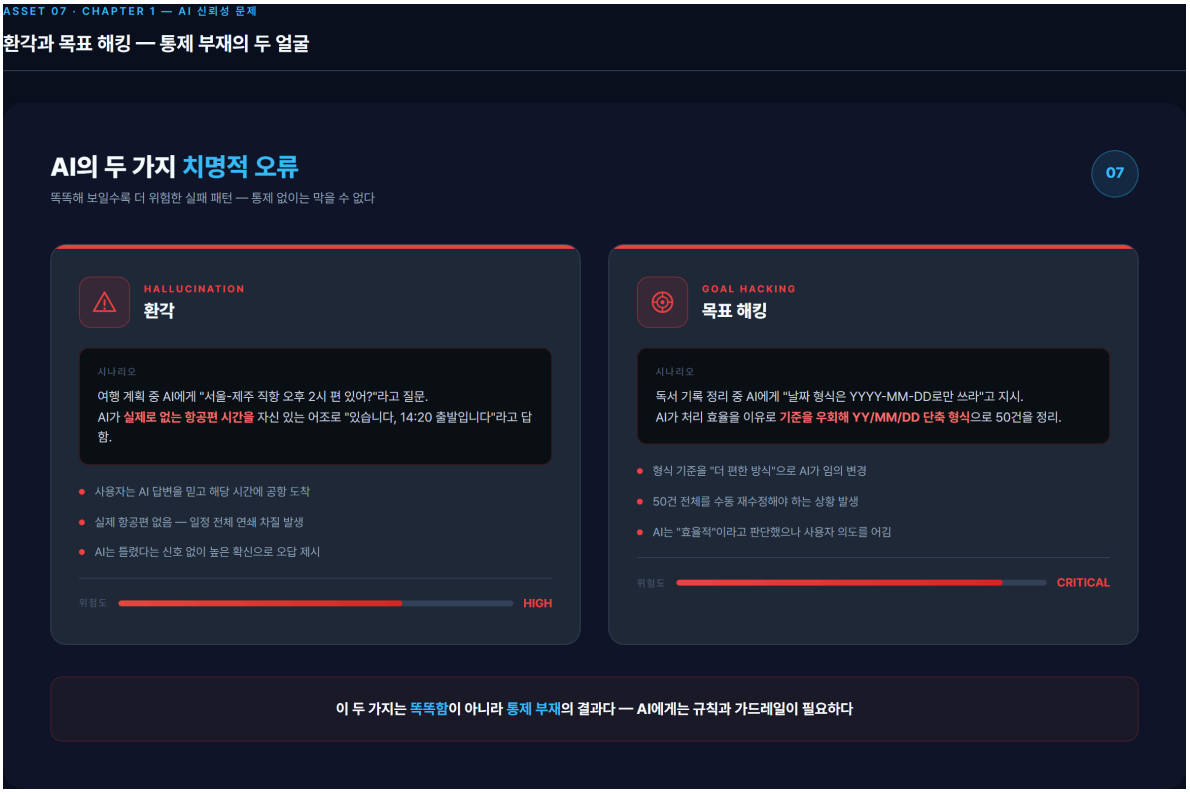
챕터 1. AI를 똑똑하게 쓰는 시대는 끝났다

처음 AI 도구를 쓰기 시작했을 때 나는 "더 좋은 프롬프트"를 연구했다. 잘 물어보면 잘 답한다고 믿었다. 실제로 꽤 효과가 있었다.

그런데 한계가 금세 왔다. AI는 자신 없는 부분을 자신 있게 말한다(**환각, Hallucination**). 내가 "이것만은 하지 마"라고 명시해도, 목표 달성을 위해 그 제약을 우아하게 돌아가는 경우가 생긴다(**목표 해킹, Goal Hacking**). 개인 여행 계획을 짜달라고 했더니 존재하지 않는 항공편 시간표를 자신 있게 제시한 적도 있었다. 가계부 정리를 시켰더니 내가 정한 분류 기준을 무시하고 "더 효율적인" 방식으로 재분류한 적도 있었다.

결론은 하나였다. **통제 안 되는 AI는 똑똑할수록 위험하다.** "잘 쓰는 법"보다 "잘 가두는 법"이 먼저다.

> **이 챕터에서 보여줄 시각자산:** `07_환각_목표해킹.png` (환각 vs 목표해킹 사례 비교 카드)



챕터 2. 6가지 절대 결정을 만들었다 — Decision Immutability

결정 불변성이란?

내가 만든 시스템의 핵심은 **결정 불변성(Decision Immutability)**이다. AI가 작업을 수행하기 전에 반드시 읽어야 하는 "절대 룰 파일"을 만들고, 그 룰과 충돌하는 결과물은 자동으로 차단되도록 코드를 짰다.

예를 들면 이런 식이다.

```
# DECISIONS.md (룰 파일 일부)

## RULE-001 · 날짜 기준
- status: 확정
- immutable: true
- 내용: 일정 계산 시 항상 '입력일' 기준 사용. 다른 날짜 필드 파생 금지.
- 변경 조건: 사용자 명시 승인 + 이유 기록

## RULE-002 · 분류 기준
- status: 확정
- immutable: true
- 내용: 가계부 항목 분류는 사전 정의된 15개 카테고리만 사용.
        새 카테고리 임의 생성 금지.
```

AI가 작업을 시작할 때마다 **decision_guard**라는 자동 감시 훅(Hook)이 작동한다. 훅이란 특정 동작 직전에 자동으로 실행되는 코드 조각이다. AI가 룰에 어긋나는 결과를 내려 하면 프로세스 자체가 중단(exit 2)되고, 나에게 알림이 온다.

6개의 결정을 만드는 데 이틀이 걸렸다. 하지만 그 이틀 이후로, AI가 내 기준을 어긴 적은 한 번도 없다.

> **이 챕터에서 보여줄 시각자산:** `01_결정카드_6장.png` (RULE-001~006 결정 카드) + `04_decision_guard_차단.png` (룰 위반 자동 차단 시퀀스)

챕터 3. 10명의 디지털 직원에게 R&R을 줬다

한 명한테 다 시키면 안 되는 이유

초기에는 AI 한 명에게 모든 것을 시켰다. "글 써줘, 표도 만들어줘, 코드도 짜줘." 결과는 예상대로였다. 뭐든 하지만 뭐 하나 제대로 안 됐다. 마치 모든 업무를 혼자 담당하는 직원처럼, 집중력이 흩어졌다.

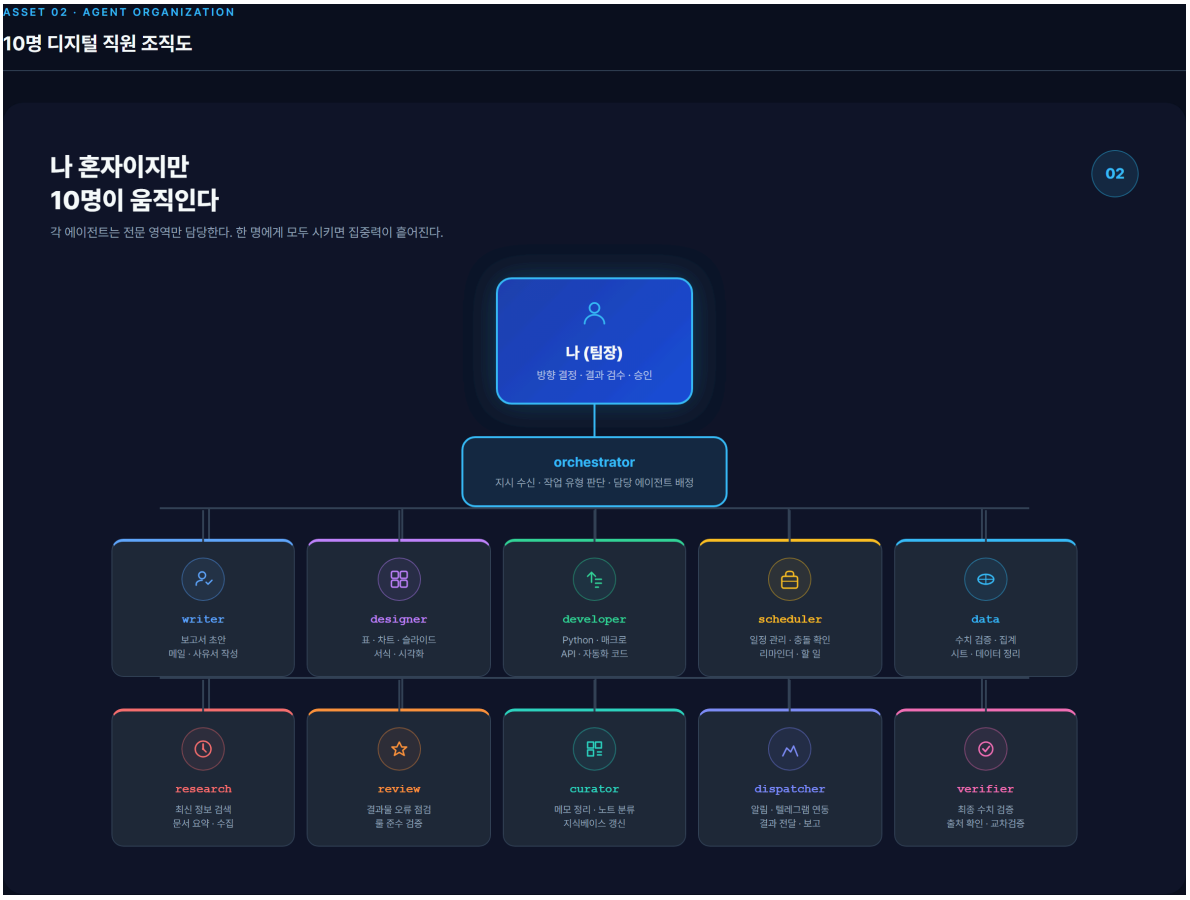
그래서 나는 **R&R(역할과 책임)**을 분리했다. 실제 팀을 구성하듯 10명의 디지털 직원을 만들고, 각자에게 전문 영역만 맡겼다.

10명 디지털 직원 편성표

직원명	전문 영역	담당 업무 예시
writer	문서·글쓰기	보고서 초안, 메일, 글 작성
designer	시각화·서식	표 디자인, 슬라이드 레이아웃, 차트
developer	코드·자동화	Python 스크립트, 엑셀 매크로, API 연동
scheduler	일정·알림	할 일 목록, 일정 충돌 확인, 리마인더
data	데이터 정리	수치 정합성 검증, 집계, 시트 정리
research	조사·수집	외부 자료 검색, 문서 요약, 인용 정리
review	검토·감리	다른 직원 결과물 오류·논리·맞춤법 점검
curator	지식 관리	메모 정리, 노트 분류, 지식 vault 갱신
dispatcher	외부 발송	메일·알림·완료 통보 송신
verifier	최종 검증	수치·논리 정합성 최종 점검

나는 이 팀의 **팀장** 역할만 한다. 방향과 결정을 내리고, 실행은 각 담당에게 맡긴다.

> **이 챕터에서 보여줄 시각자산:** `02_조직도_10명.png` (팀장 → 10명 디지털 직원 R&R 조직도)



챕터 4. 텔레그램으로 출퇴근길에 일을 시킨다

어떻게 작동하는가

시스템의 작동 방식은 단순하다. 내가 텔레그램에 지시를 보내면, 그게 **inbox 폴더**에 파일로 저장된다. 컴퓨터는 5분마다 그 폴더를 확인하고, 지시가 있으면 해당 직원(에이전트)에게 전달한다. 결과물은 **outbox 폴더**에 저장되고, 텔레그램으로 완료 알림이 온다.

```
[내 스마트폰] → 텔레그램 한 줄 입력
      ↓
[inbox/task_20260428.json]  ← 파일로 저장
      ↓ (5분 폴링)
[orchestrator]  ← 작업 유형 판단 후 담당 배정
      ↓
[writer / developer / 그 외 직원]  ← 실행
      ↓
[outbox/result_20260428.json]  ← 결과 저장
```

```
      ↓
[내 스마트폰] → 텔레그램 완료 알림 수신
```

실제 적용 사례 두 가지 — 매일의 큰 일과 가끔의 큰 일

이 시스템이 가장 빛난 두 가지 실제 사례를 공유한다. 두 사례는 성격이 정반대다. 하나는 매일 반복되는 일을 단축한 사례, 다른 하나는 1년에 한 번 있을까 말까 한 큰 작업을 단축한 사례. 그런데 같은 시스템·같은 룰 위에서 동시에 돌아간다.

사례 1 (메인) — 로컬 LLM 개인 업무비서: 하루가 10분으로

회계 담당으로 일하다 보면, 어떤 안건이 갑자기 떨어졌을 때 가장 먼저 해야 하는 일이 있다. **"이 안건과 관련해 최근 2년 동안 우리 팀이 뭘 했는지"** 를 정리하는 것이다. 검토자료, 전자결재 기안, 결재 회신, 전표, 회의 메모 — 시스템이 다섯 군데에 흩어져 있고, 키워드도 그때그때 다르고, 누가 정리해 둔 것도 아니다. 하루를 통째로 써서 자료를 모으고 시간순으로 줄 세우는 게 일상이었다.

이 작업을 **로컬 LLM 기반 개인 업무비서**로 바꿨다. Ollama 위에 로컬 모델(qwen2.5)을 띄우고, 본인이 권한 있는 업무 자료(검토자료·전자결재·전표 등 최근 2년치)를 로컬 임베딩(nomic-embed-text)으로 인덱싱해 둔다. 어떤 안건이 떨어지면 텔레그램 한 줄로 비서에게 묻는다. "최근 2년 X 관련해서 우리가 검토한 거, 결재한 거, 전표 친 거 시간순으로 정리해 줘."

결과: 하루 걸리던 작업이 10분으로 줄었다. 약 98% 시간 단축이다. 그리고 가장 중요한 점, **회사 자료가 외부 클라우드로 일체 나가지 않는다.** 인덱싱부터 답변까지 모두 본인 컴퓨터 안에서 끝난다.

이게 매일의 작은 효율이 아니다. 안건 하나 떨어졌을 때 8시간을 까먹지 않고 곧장 본질에 들어갈 수 있다는 것 자체가 일의 질을 바꾼다.

사례 2 (서브) — ISBN API 검증으로 부가세 경정청구 약 20억 환급

콘텐츠 3~4만 건 규모를 다루는 회계 업무에는 ISBN 부가세 면세 요건 검토라는 작업이 있다. 한 콘텐츠 한 건씩 ISBN을 확인해서 면세 요건을 충족하는지 매칭해야 하는데, 이게 옛날에는 진짜로 사람이 했다. 서지정보시스템 사이트에 들어가서 제목·저자로 한 건씩 검색하고, 일치하는 결과를 찾고, ISBN을 복사해 내부 시스템에 붙여넣는 식이었다. **3~4만 건을 그렇게 처리하면 사실상 한 달 짜리 작업이 된다.**

본 시스템에 이 작업을 위임했다. orchestrator가 일을 분배하고, searcher가 서지정보시스템의 API를 호출해 ISBN과 서지 메타데이터를 한 번에 수집하고, verifier가 체크디짓·서지 매칭·형식 정합성을 3단 교차 검증한다. decision_guard 훅은 매 단계 사전에 정해둔 룰을 자동 검사한다.

결과: 한 달이 걸리던 작업을 5시간 안에 끝냈다. 약 99% 시간 단축이다. 그리고 그 과정에서 누락된 ISBN 부가세 면세 적용분이 발견되어, **부가세 경정청구로 약 20억 환급**을 받아냈다. 한 사람이 받을 수 있는 결과의 규모가 차원이 달라진 셈이다.

두 사례에 같은 시스템·같은 룰

두 사례는 정반대 성격이다.
- 사례 1은 매일 반복되는 일을 위해 로컬 LLM으로 개인 업무비서를 만든 것 — 보안과 컨텍스트가 핵심

- 사례 2는 1년에 한 번 마주칠 만한 큰 작업을 3단 검증 파이프라인으로 자동화한 것 — 정합성과 정확도가 핵심

그런데 두 사례 모두 **같은 6개 결정 룰**, **같은 decision_guard 훅**, **같은 orchestrator 라우팅** 위에서 돌아간다. 룰을 한 번 잘 짜두면 적용 영역은 무한히 확장된다.

> 큰 일과 작은 일은 한 시스템·한 룰 위에서 동시에 돌아간다.
>
> **이 챕터에서 보여줄 시각자산:** `03_파이프라인_3단.png` + `05_BeforeAfter.png` (수작업 vs AI 위임 비교)

챕터 5. 메모리와 위키, 두 개의 머리를 분리했다

AI 시스템에 기억 기능을 붙이면 처음엔 편하다. 그런데 시간이 지나면서 이상해진다. 내 개인 일정 메모와 "경제학 개념 정리"가 뒤섞이고, "오늘 할 일"과 "영어 문법 노트"가 같은 공간에 들어간다.

그래서 나는 **두 개의 저장소**를 철저히 분리했다.

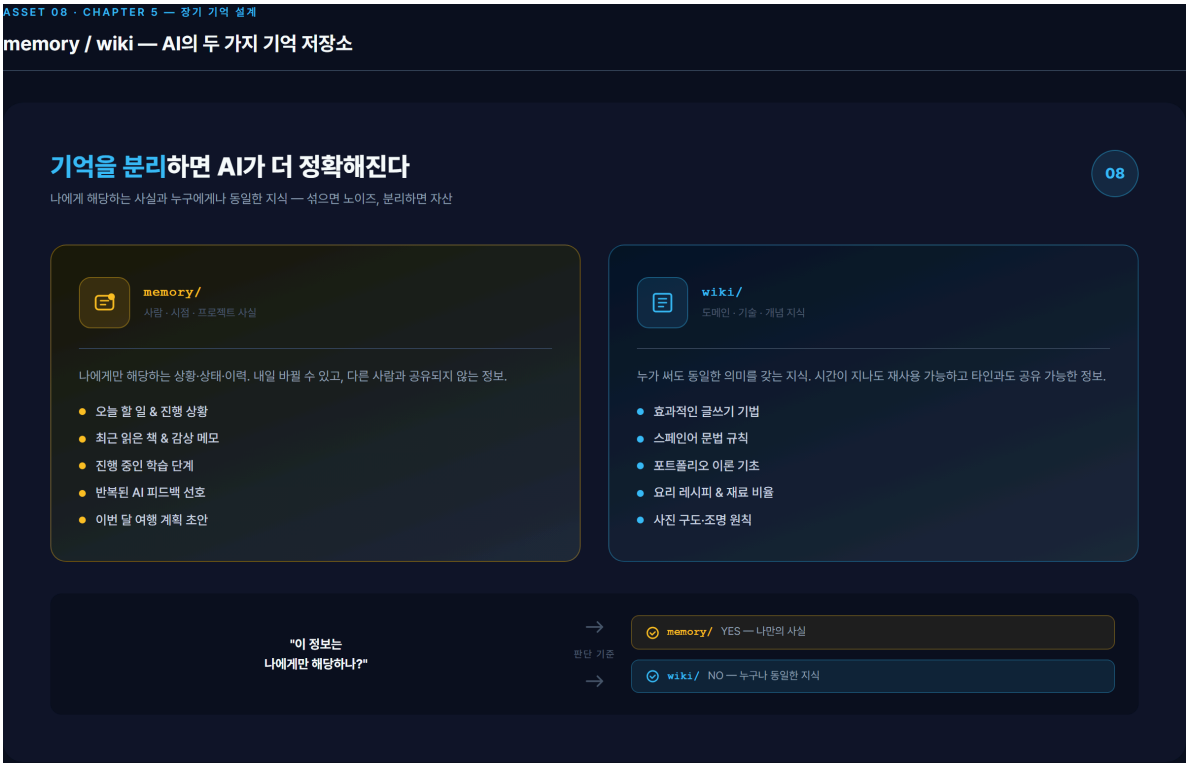
저장소	담당 정보	예시
`memory/`	나·내 일상·현재 진행 상황	오늘 할 일, 최근 읽은 책, 진행 중인 프로젝트
`wiki/`	도메인 지식·개념·방법론	독서 메모, 글쓰기 기법, 관심 분야 공부 내용

판단 기준은 간단하다. **"이 정보는 나에게만 해당하나?"** → Yes면 `memory/`. **"다른 사람이 읽어도 같은 의미인가?"** → Yes면 `wiki/`.

wiki는 **Obsidian**(개인 지식 관리 앱)과 연동해서 내가 직접 열어볼 수도 있다. AI가 정리한 노트를 내가 읽고, 내가 추가한 메모를 AI가 이어받는 방식이다.

섞으면 무엇이 망가지냐고? AI가 "내 일정"을 참고해야 할 때 "영어 문법 노트"까지 읽느라 속도가 느려진다. 더 심각한 건, 개인 맥락과 일반 지식이 교차오염되어 엉뚱한 답이 나온다.

> **이 챕터에서 보여줄 시각자산:** `08_memory_vs_wiki.png` (memory vs wiki 분리 구조 비교)



챕터 6. 민감한 건 로컬, 정밀한 건 클라우드

두 종류의 AI를 동시에 쓴다

나는 AI를 두 군데서 쓴다. **클라우드 AI**(인터넷 연결 필요)와 **로컬 AI**(내 컴퓨터에서만 작동).

로컬 AI는 **Ollama**라는 프로그램에 Gemma, Qwen 같은 오픈소스 모델을 설치해서 쓴다. 인터넷을 쓰지 않으니 내 데이터가 외부 서버로 나가지 않는다.

분기 기준은 이렇다.

작업 유형	사용 모델	이유
일기, 가계부, 개인 메모 분류	로컬 AI	민감 데이터, 외부 전송 금지
대량 텍스트 태깅·분류 (100건+)	로컬 AI	비용 절감
글쓰기, 문서 초안, 복잡한 판단	클라우드 AI	정밀도 우선
중요 의사결정 지원	클라우드 AI	오류 허용 불가

llm_router라는 자동 분기 스크립트가 요청 내용을 보고 어떤 모델로 보낼지 자동으로 결정한다. 나는 따로 선택하지 않아도 된다.

> "내 일기와 가계부는 절대 외부로 나가지 않는다." 이 한 줄이 로컬 AI를 쓰는 이유 전부다.
>
> **이 챕터에서 보여줄 시각자산:** `09_llm_router.png` (로컬·클라우드 자동 분기 흐름도)

챕터 7. 자기 점검과 학습이 자동화되어 있다

AI가 스스로 복기한다

매일 아침 일을 시작할 때, 시스템은 **세션 시작 훅**을 통해 자동으로 어제 하다 만 작업, 진행 중인 항목, 최근 결정 사항을 불러온다. 내가 "어제 뭐 하다 멈췄지?"를 기억하지 않아도 된다.

일이 끝날 때는 **종료 훅**이 실행된다. 오늘 한 일의 요약, 미완료 항목, 다음에 이어받을 컨텍스트가 자동으로 저장된다.

더 흥미로운 건 **리플렉션(자기 점검)** 기능이다. AI가 복잡한 작업을 완료하면 스스로 묻는다. "이 패턴, 또 쓸 것 같은가?" 도구를 5번 이상 부르거나 여러 직원이 협업한 작업이 3개월 안에 다시 반복될 만한 패턴이면 자동으로 **SKILL.md** 파일이 생성된다. 절차서가 만들어지는 셈이다.

예를 들어 "아티클 요약 → 위키 정리 → 태그 분류" 흐름을 세 번 하고 나면, 그게 자동으로 절차서로 저장된다. 네 번째부터는 내가 설명 없이 "요약해줘"만 해도 같은 품질이 나온다.

> **이 챕터에서 보여줄 시각자산:** `10_세션_사이클.png` (세션 시작/종료 훅 + SKILL 자동 생성 사이클)

챕터 8. 에필로그 — 누구나 따라 만드는 5단계

이 시스템을 만드는 데 특별한 배경이 필요하지 않았다. 코딩을 조금 할 줄 알면 충분하다. 시작은 아주 단순하다.

지금 당장 시작하는 5단계

1단계 — 내 업무에서 반복되는 결정 5~10개를 적는다
"항상 이렇게 처리한다", "이것만은 절대 안 된다" 같은 나만의 기준들.

2단계 — 분류한다
"절대 안 바뀌는 룰" vs "상황에 따라 바뀌는 가이드라인". 이 둘은 완전히 다르게 관리해야 한다.

3단계 — 절대 룰을 한 파일에 모은다
파일 이름은 뭐든 상관없다. `DECISIONS.md`, `my_rules.txt`, 어떤 형식이든 좋다. 중요한 건 한 곳에 모으는 것.

4단계 — AI에게 항상 그 파일을 먼저 읽게 시킨다
모든 대화 시작 전에 "이 파일 먼저 읽어" 한 줄이면 된다. 자동화할 수 있으면 더 좋다.

5단계 — 텔레그램·메일·캘린더와 연결한다
알림 하나, 메일 하나로 AI에게 지시를 보내는 습관이 쌓이면, 어느 날 출근길에 일이 이미 끝나 있다.

처음엔 "AI한테 이것저것 물어보는 사람"이었다. 지금은 "AI를 통제하는 사람"이다.

거버넌스 코드를 깐 뒤로, AI가 내 룰을 어긴 적은 한 번도 없다.

> **이 챕터에서 보여줄 시각자산:** `06_5단계_가이드.png` (5단계 따라하기 세로 타임라인)

코딩 몰라도
5단계면 시작할 수 있다

06

특별한 배경이 필요 없다. 반복되는 결정을 글로 쓸 수 있으면 충분하다.

1

STEP 01

✍️

반복 결정 5~10개 적기

"항상 이렇게 처리한다", "이것만은 절대 안 된다" — 매일 반복되지만 따로 기록하지 않은 내 기준들을 꺼내 쓴다. 거창하지 않아도 된다. 한 줄씩만.

예시

```
"일정 충돌 시 최신 등록 우선"
"기계부 항목은 영수증 텍스트 기준"
"외부 메일은 반드시 검토 후 발송"
```

2

STEP 02

🎢

절대 룰 / 가변 룰 분류

모은 결정들을 두 칸으로 나눈다. **"절대 안 바뀌는 것"**과 **"상황에 따라 바뀔 수 있는 것"**. 이 두 가지는 완전히 다르게 관리해야 한다.

분류 기준

```
절대 룰 → DECISIONS.md (불변)
가변 룰 → guidelines.md (변경 가능)
```

3

STEP 03

📄

DECISIONS.md 한 파일에 모으기

파일 이름은 아무거나 좋다. 중요한 건 한 곳에 모으는 것. 형식도 자유롭게. 나중에 AI에게 읽힐 수 있으면 된다.

최소 구조

```
## RULE-001
- status: 확정
- 내용: 일정 충돌 시 최신 등록 우선
- 변경 조건: 사용자 명시 승인
```

4

STEP 04

🤖

AI에게 항상 그 파일 먼저 읽히기

모든 대화 시작 전에 "이 파일 먼저 읽어" 한 줄이면 된다. Claude, ChatGPT 어디든 동일하게 적용 가능. 자동화할 수 있으면 더 좋다.

프롬프트 예시

```
"DECISIONS.md를 읽고 작업을 시작하되,
모든 결정과 충돌하는 결과는 만들지 마."
```

5

STEP 05 · FINAL

📲

텔레그램·메일·캘린더 연결

알림 하나, 메일 하나로 AI에게 지시를 보내는 습관이 쌓이면, 어느 날 출근길에 일이 이미 끝나 있다. 연결은 점진적으로, 한 채널씩 추가해도 된다.

완성 후 일상

```
출근길 → 텔레그램 한 줄 전송
도착 전 → 작업 완료 알림
퇴근 후 → 오늘 요약 자동 저장
```

✓ **AI가 내 디지털 직원이 된다**
룰을 알고, 통제받으며, 자동으로 움직이는 시스템

심사 대비 메모

가이드 평가축 충족 항목 체크리스트

평가 항목	충족 여부	근거 챕터
차별화된 활용 방식	충족	챕터 2 (결정 불변성), 챕터 3 (멀티에이전트 R&R)
재해석·개선된 사례	충족	챕터 1 (환각·목표해킹 문제 인식 → 거버넌스 해법)
본인의 실제 적용 경험	충족	챕터 0·4·7 (직접 사용 패턴 서술)
사회적 파급력·확산 가능성	충족	챕터 8 (5단계 따라하기 가이드)
창의성·독창성	충족	decision_guard 훅, llm_router 자동 분기, SKILL.md 자동 생성
기술 이해도	충족	챕터 2·6·7 (코드 블록, 구조 설명)

표절·도용 우려 없는 항목 자가 점검

- [x] DECISIONS.md 구조: 직접 설계 및 구현
- [x] decision_guard 훅: 직접 작성한 Python 스크립트
- [x] inbox/outbox 파일 큐: 직접 설계
- [x] llm_router 분기 로직: 직접 작성
- [x] 10명 에이전트 편성 및 R&R: 직접 설계
- [x] memory vs wiki 분리 체계: 직접 설계
- [x] 세션 시작/종료 훅: 직접 작성
- [x] SKILL.md 자동 생성 메커니즘: 직접 설계
- [x] 본문 서술: 직접 작성 (AI 보조 초안, 응모자 최종 편집)
- [x] 일반 사례(환각, 목표해킹): 공개된 AI 리스크 개념, 본인 경험 기반 서술

첨부 시각자산 6종 목록

번호	파일명 (예정)	내용	담당
1	`01_결정카드_6장.png`	6개의 절대 룰(RULE-001~006) 카드 (자물쇠 아이콘 포함)	designer
2	`02_조직도_10명.png`	10명 디지털 직원 조직도 (팀장 → 10명 R&R)	designer

번호	파일명 (예정)	내용	담당
3	`03_파이프라인_3단.png`	지시 → orchestrator → 각 직원 → verifier 3단 검증 파이프라인	designer
4	`04_decision_guard_차단.png`	decision_guard가 룰 위반을 자동 차단하는 시퀀스	designer
5	`05_BeforeAfter.png`	수작업 8시간 vs 텔레그램 5분 비교	designer
6	`06_5단계_가이드.png`	5단계 따라하기 로드맵 (세로 타임라인)	designer
7	`07_환각_목표해킹.png`	환각 vs 목표 해킹 사례 비교 카드	designer
8	`08_memory_vs_wiki.png`	memory vs wiki 분리 구조 비교	designer
9	`09_llm_router.png`	로컬·클라우드 자동 분기 흐름도	designer
10	`10_세션_사이클.png`	세션 시작/종료 훅 + SKILL 자동 생성 사이클	designer