

AI Agent

개념, 설계, 도입, 운영

LLM 이 도구를 써서 업무를 처리하는 구조와 기업 적용 방법

구성

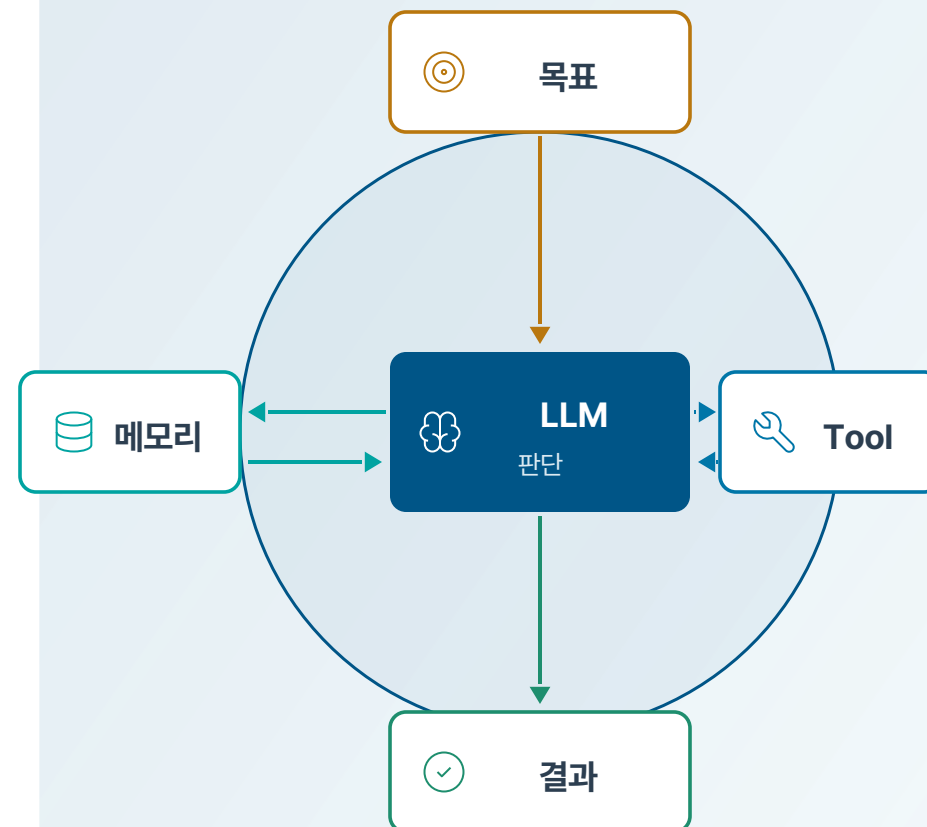
12개 파트 · 88장

파트 하나가 영상 한 편

DATE

2026-10

대상: IT 실무자, 아키텍트, 도입 담당



AI Agent 열두 파트 구성

■ 개념과 설계 (01~04) ■ 도입 (05~07) ■ 개발과 운영 (08~09) ■ 통제와 기반 (10~12)

01 AI Agent 정의와 특징

Chat AI, RPA 와의 차이와 자율 수준

02 동작 원리와 구성 요소

추론 루프, Tool, 메모리, 계획

03 설계 패턴

Workflow, Multi-Agent, 설계 원칙

04 연결 표준과 개발 도구

MCP, A2A, Agent Skills, SDK

05 도입 배경과 기대 효과

AX, 기술 추세, 효과가 큰 업무

06 도입 전략과 로드맵

과제 선정, 우선순위, ROI

07 조직 설계와 적용 사례

AI 직원, 역할 분담, 금융 사례

08 Coding Agent 와 Vibe Coding

활용 수준, 사람의 역할

09 SDD 와 LLMops

Spec, SSoT, Observability

10 AI Agent 의 보안 위험

OWASP Top 10, 강제 통제

11 Guardrail 과 품질 평가

HITL, LLM 심사, Red Teaming

12 로컬 LLM 과 LLM 게이트웨이

실행 도구, GPU, 라우팅

KEY INSIGHT

AI Agent 는 LLM 이 도구를 반복 호출해 목표를 달성하는 시스템임 — 설계, 도입, 통제를 함께 준비해야 업무에 쓸 수 있음

PART 01

AI Agent 의 정의와 특징

Chat AI · RPA 와 비교해 Agent 가 맡는 일의 범위를 구분함

1 Chat AI 와 AI Agent 의 차이

2 AI Agent 의 두 가지 정의

3 LLM 에서 Agentic AI 까지의 발전 단계

4 AI Agent 의 네 가지 특징

5 규칙 기반 자동화와 AI Agent 비교

6 업무에 맞춰 정하는 자율 수준



KEY INSIGHT AI Agent 는 목표를 받아 도구로 업무를 끝까지 처리함 — 도입 대상이 질문 응답에서 업무 처리로 넓어짐

Chat AI	질문에 답함	AI Agent	업무를 처리함
질문 한 건	입력	달성할 목표와 쓸 수 있는 도구	
한 번의 응답 생성	처리	계획, 도구 호출, 결과 확인을 반복	
글, 요약, 초안	출력	처리 결과 (등록된 티켓, 발송된 메일)	
사람이 읽고 직접 정함	다음 단계	Agent 가 정하고 사람은 승인	
질문 작성과 결과 활용	사람의 역할	목표 제시, 권한 설정, 검토	
같은 요청: "지난달 미수금을 정리해 줘" Chat AI: 미수금을 정리하는 방법과 엑셀 수식을 설명함. 조회와 정리는 사람이 직접 수행 AI Agent: ERP 를 조회해 미수 건을 표로 정리하고 독촉 메일 초안을 만든 뒤 발송 승인을 요청			



KEY INSIGHT LLM 시대의 Agent 는 목표를 위해 도구를 반복 호출하는 LLM 임 — 합의된 단일 정의는 없어 범위를 먼저 맞춰야 함

고전적 정의 (1995~)

센서로 환경을 지각하고 액추에이터로
환경에 작용하는 모든 것

Russell & Norvig

- 로봇, 게임 AI, 추천 시스템도 포함
- 자연어와 도구가 전제가 아님

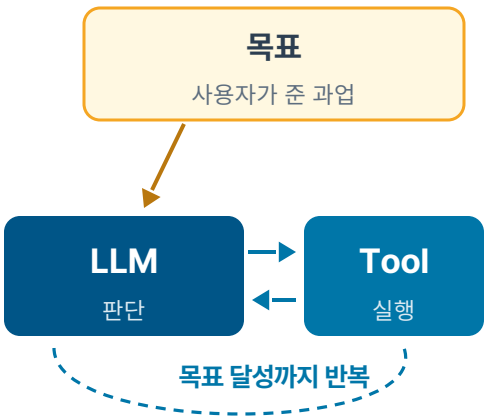


LLM 시대의 정의 (2024~)

LLM 이 자기 절차와 도구 사용을
스스로 정하며 과업을 수행하는
시스템

Anthropic · 줄여서 tools in a loop

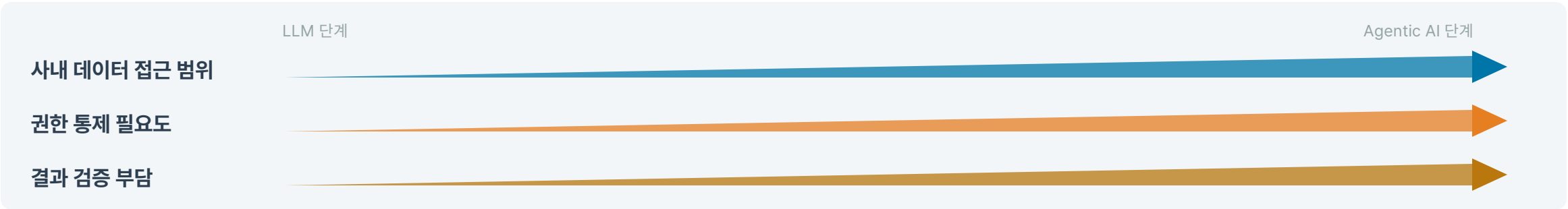
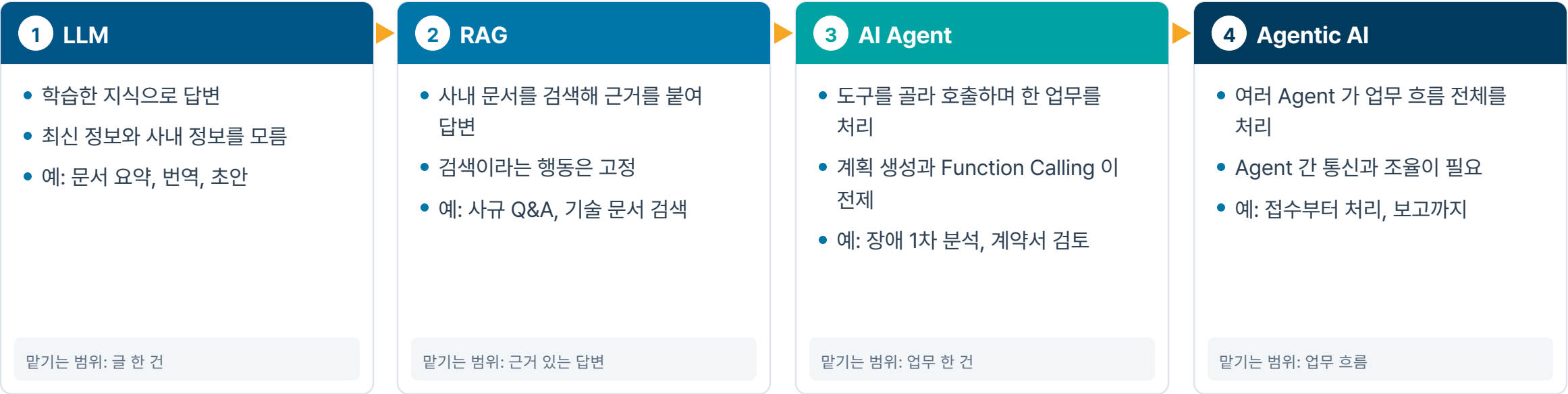
- 환경 지각 = 도구 실행 결과 읽기
- 행동 = 도구 호출



구성	고전적 정의	LLM 시대의 정의
판단 주체	에이전트 함수 (규칙, 탐색, 학습 모델)	LLM (자연어로 추론)
환경과의 접점	센서와 액추에이터	Tool: 검색, API, 코드 실행, 파일
범위 합의	이론적 정의로 범위가 넓음	제품마다 다르게 사용. 회의 전에 범위를 먼저 맞춤



KEY INSIGHT 단계가 오를수록 말기는 범위가 답변에서 업무 처리로 넓어짐 — 모델 성능만으로는 부족하고 도구 연계가 필요함





KEY INSIGHT 계획과 도구 사용이 단일 LLM 호출과의 차이를 만들 — 네 특징은 유무가 아닌 정도로 나타남



자율성

- 사람이 단계마다 지시하지 않아도 다음 행동을 스스로 정함
- 예: 검색 결과가 부족하면 검색어를 바꿔 다시 조회
 - 범위는 권한과 반복 상한으로 제한

없으면: 단계마다 사람이 지시



목표 지향

- 절차가 아닌 달성할 결과를 입력으로 받음
- 예: "경쟁사 3곳의 가격 비교표 작성"
 - 완료 조건을 함께 줘야 멈출 수 있음

없으면: 절차를 사람이 설계



계획과 추론

- 과업을 작은 단계로 나누고 결과를 보고 계획을 고침
- 예: 조사, 정리, 검증 순서로 분해
 - 실패하면 원인을 보고 다시 시도

없으면: 한 번의 응답으로 종료



도구 사용

- 검색, API, 코드 실행으로 LLM 의 계산과 최신 정보 한계를 보완
- 예: DB 조회, 메일 발송, 계산 실행
 - 할 수 있는 일의 범위를 도구가 정함

없으면: 글만 생성

자료	짧은 특징	강조점
삼일PwC (2025)	자율성, Tool 사용, 계획과 추론, 목표 지향	도구로 계산과 최신 정보의 한계를 보완
Microsoft (2024)	자율성, 목표 지향, 고도의 추론, 외부 연계	명확한 경계선은 없고 정도의 차이
KDDI Agile Dev. Center (2025)	계획과 돌아보기, Tool 실행, Profiling, 장기 기억	앞의 두 가지가 단일 LLM 과의 차이



KEY INSIGHT 예외 판단이 필요한 업무에만 Agent 를 쓰고 절차가 고정된 업무는 RPA 로 둬 — RPA 가 싸고 결과가 일정함

비교 항목	규칙 기반 자동화 (RPA · Bot)	RAG 챗봇	AI Agent
동작 기준	사람이 만든 스크립트와 규칙	검색한 문서를 근거로 답변	목표에 따라 절차를 스스로 결정
예외 처리	규칙에 없으면 중단	문서에 없으면 답하지 못함	상황을 판단해 다른 경로로 처리
수행 범위	정해진 화면과 항목의 입력, 조회	조회와 답변	조회에 더해 생성, 변경까지 수행
변경 대응	화면이나 양식이 바뀌면 다시 개발	문서를 다시 색인	프롬프트와 도구 설명을 수정
결과의 일정함	항상 같은 결과	근거에 따라 다소 변동	실행마다 경로가 다를 수 있음
건당 비용	매우 낮음	낮음	토큰 사용량에 따라 높음
맞는 업무	전표 입력, 정기 보고서 추출	사규, 매뉴얼 Q&A	문의 처리, 장애 분석, 계약 검토

함께 쓰는 구성

정형 구간은 RPA 가 처리하고, 예외가 생기면 Agent 가 판단해 다시 RPA 로 넘김

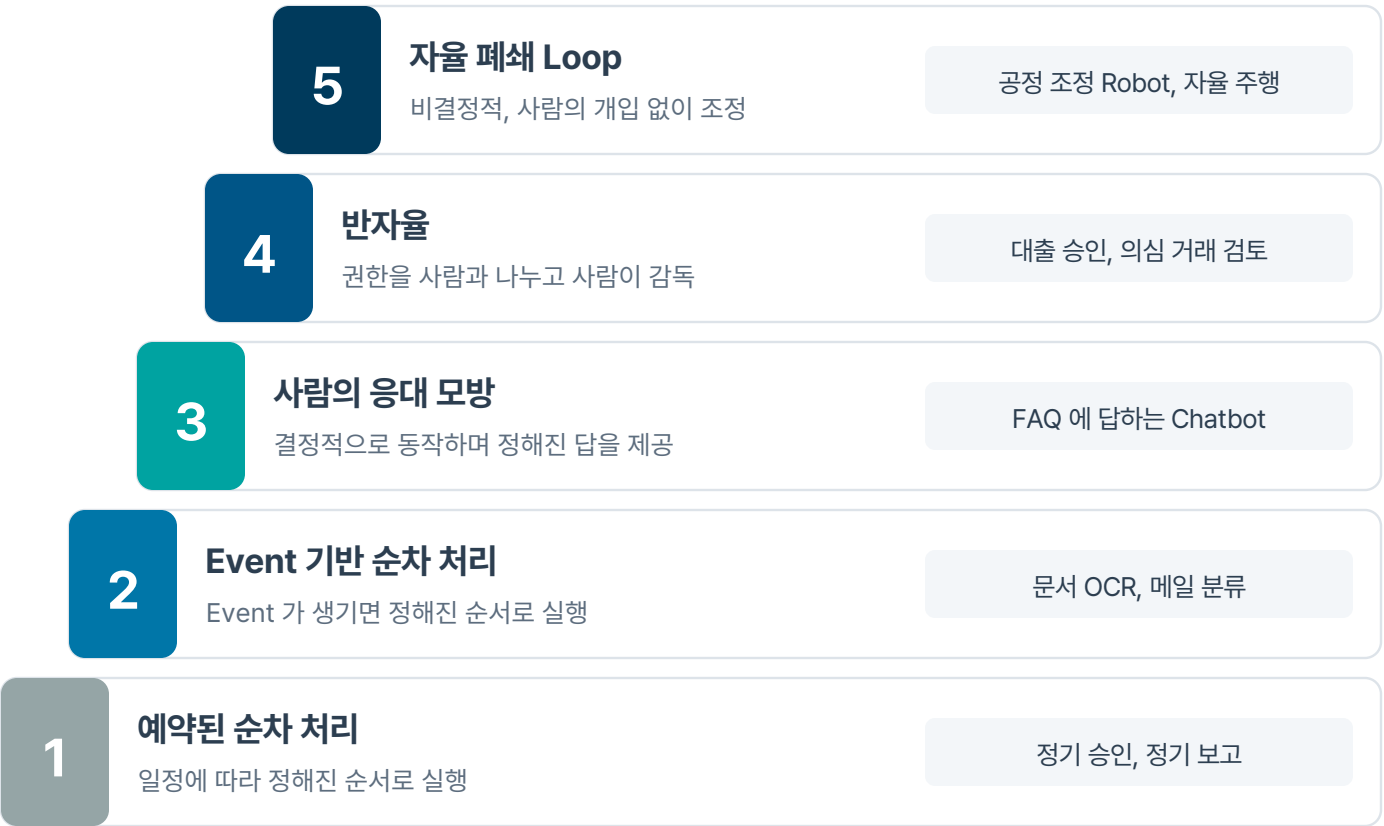
선택 질문

절차를 순서도로 그릴 수 있는가? 그릴 수 있으면 규칙 기반이 먼저임



KEY INSIGHT 자율 수준은 업무의 위험과 되돌릴 수 있는 정도로 정함 — 모든 업무에 완전 자율이 맞는 것은 아님

자율 수준 ↑



Agent 다운 정도를 재는 네 요소

- 목표 복잡성** 달성할 목표가 얼마나 어려운가
- 환경 복잡성** 다루는 시스템과 데이터가 얼마나 다양한가
- 적응성** 처음 보는 상황에 대응하는가
- 독립 실행** 사람 개입 없이 어디까지 가는가



수준을 정하는 두 가지 질문

- 잘못 실행했을 때 피해가 큰가
- 실행한 결과를 되돌릴 수 있는가
- 둘 중 하나라도 불리하면 반자율 이하에서 시작

PART 02

AI Agent 의 동작 원리와 구성 요소

추론 루프, Tool, 메모리, 계획이 한 번의 요청을 처리하는 순서

1 AI Agent 의 기본 동작 구조

2 Agent 를 구성하는 네 Module

3 추론 루프와 ReAct 패턴

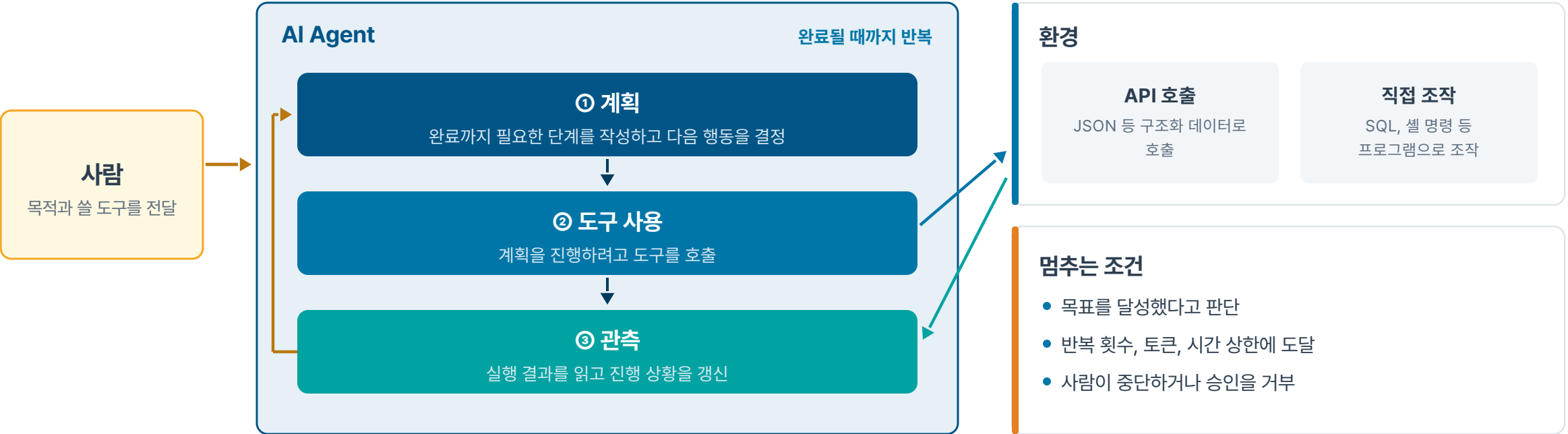
4 Tool 의 역할과 Function Calling

5 Context Window 와 메모리

6 계획과 성찰 기법



KEY INSIGHT 중간 절차를 Agent 가 정하므로 결과 검증과 권한 통제가 중요해짐 — 사람은 목적과 도구만 정함



- 예: "결제 API 장애 원인을 조사해 줘"**
- 1** 로그 검색 도구 호출 → 14:02 부터 타임아웃 오류 증가를 확인
 - 2** 지표 조회 도구 호출 → 같은 시각 DB 커넥션 풀이 가득 찬 것을 확인
 - 3** 배포 이력 조회 → 13:58 배포를 원인 후보로 정리해 보고서 작성



KEY INSIGHT 네 Module 틀로 서로 다른 Agent 구현을 같은 기준으로 비교할 수 있음 — 모든 Agent 가 넷을 다 갖추지는 않음



같은 틀로 본 구현 차이

Module	단순한 챗봇형	Coding Agent
Profile	응대 말투 지정	프로젝트 규칙 파일
Memory	대화 이력만	이력 + 메모 파일
Planning	없음	할 일 목록을 만들고 갱신
Action	문서 검색 1종	셀, 파일 편집, 검색



새 위험이 생기는 자리

- 계획과 추론: 목표가 바뀌도록 유도
- 기억: 오염된 내용이 이후 행동을 결정
- 행동과 Tool: 권한을 넘는 호출

다섯 능력으로 나누는 분류도 있음: Tool 사용, Profile, 계획, 성찰, 기억 (LegalOn 2025). 성찰을 Planning 에서 떼어 낸 구분임



KEY INSIGHT ReAct 는 외부 근거로 사실 오류를 줄임 — 같은 행동을 반복하는 오류가 있어 반복 상한이 필요함



실행 기록 예: "1,200달러는 원화로 얼마인가"

Thought 오늘 환율이 필요함

Action get_rate(USD, KRW)

Observation 1 USD = 1,385 KRW

Thought 1,200 을 곱해야 함

Action calc(1200 * 1385)

Observation 1,662,000

Thought 답을 낼 수 있음

Answer 약 166만 2천 원 (환율 1,385원 기준)

실패 원인	ReAct	CoT
틀린 사실의 생성	0%	56%
추론 오류 (같은 생각과 행동의 반복 포함)	47%	16%
쓸모없는 검색 결과	23%	해당 없음
성공 사례에 틀린 사실이 섞인 비율	6%	14%

설계에 반영할 점

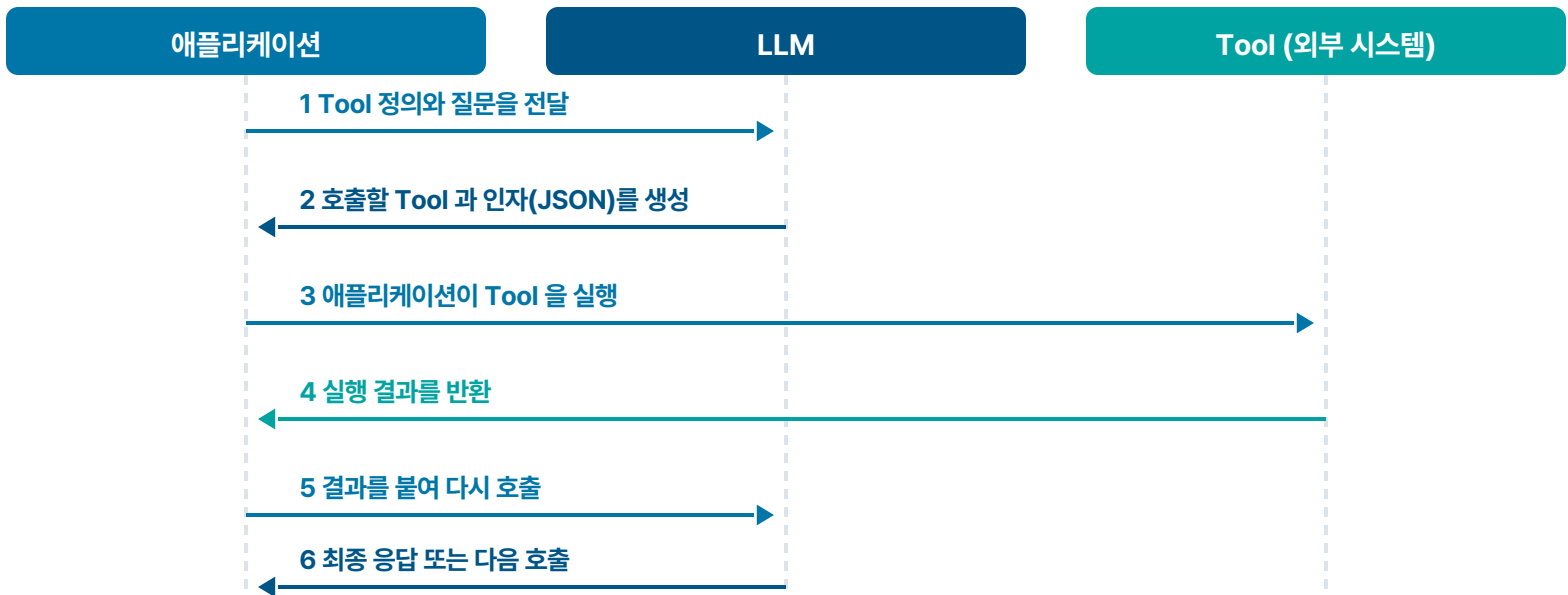
- 반복 상한과 같은 호출 반복 감지
- 검색 결과가 비었을 때의 대체 경로

CoT (Chain of Thought): 도구 없이 단계별 추론만 수행. ReAct 는 근거가 있어 사실 오류가 적지만 구조가 추론의 유연성을 줄임



KEY INSIGHT Agent 가 할 수 있는 일의 범위는 주어진 Tool 이 정함 — 환경을 바꾸는 Tool 일수록 권한을 좁혀야 함

Function Calling 순서



구분	Tool 의 예
조회형 (Sensor)	Web 검색, 페이지 가져오기, 파일 읽기, DB 조회
변경형 (Actuator)	파일 쓰기, 메시지 발송, 티켓 생성, 브라우저 조작
계산과 실행	수치 계산, 코드 실행

권한을 좁히는 방법

- 조회형과 변경형 Tool 을 분리
- 변경형은 대상 범위를 인자로 제한
- 되돌릴 수 없는 호출은 사람 승인

LLM 은 Tool 을 직접 실행하지 않음. 호출 인자만 만들고 실행과 권한 검사는 애플리케이션이 맡음

Tool 설명이 곧 프롬프트

이름: 동사로 시작 (search_orders)

설명: 언제 쓰고 언제 쓰지 않는지

인자: 형식과 허용 값을 스키마로 지정

결과: 필요한 필드만 반환



KEY INSIGHT Agent 는 단계마다 Context 에 넣을 정보를 골라야 함 — Context 가 길어지면 정확도가 떨어지고 비용이 늘어남

Context Window: LLM 이 한 번에 읽고 처리하는 정보의 총량 (입력과 응답을 합친 토큰 수)



단계가 진행될수록 대화 이력과 Tool 결과가 쌓여 공간을 채움 → 누적

Context Engineering 네 가지 방법

쓰기 (Write)
Context 밖에 저장
예: 메모 파일, 할 일 목록, 장기 메모리

고르기 (Select)
필요한 것만 가져옴
예: RAG 검색, 관련 Tool 만 호출

줄이기 (Compress)
요약하고 잘라 냄
예: 대화 요약, 오래된 Tool 결과 제거

나누기 (Isolate)
Context 를 분리
예: 하위 Agent, 샌드박스 실행

메모리 유형	저장 위치	쓰는 경우
단기 메모리	Context 안의 대화 이력	진행 중인 과업의 흐름 유지
장기 메모리	벡터 DB, 파일, 관계형 DB	세션을 넘어 사실과 선호를 기억
절차 메모리	Skill, 규칙 파일	일하는 방법과 절차를 재사용

Context 가 길어지면 생기는 일

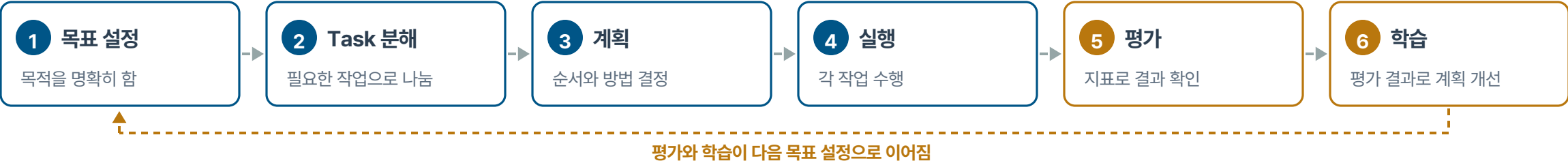
- 가운데 내용을 놓치는 비율 증가
- 응답 지연과 토큰 비용 증가
- 한국어는 1글자에 2~3 토큰 (모델마다 다름)



KEY INSIGHT 복잡한 과업일수록 실행 전에 단계를 나누고 결과를 다시 평가함 — 평가 기준을 먼저 정해야 재시도가 동작함

기법	LLM 에 요구하는 것	알맞은 작업	예: “교토 3일 여행 계획, 예산 5만 엔”
표준 Prompt	결과만 요청	단순한 질문과 생성	요청을 그대로 전달해 한 번에 작성
Chain-of-Thought	단계를 밟아 답을 도출	계산, 논리 판단	목적과 예산 확인부터 단계별로 작성
ReAct	추론, 행동, 결과 확인의 반복	외부 정보 수집이 필요한 작업	후보 선정 → 정보 수집 → 일정 작성 → 예산 조정
Plan-and-Solve	전체 계획 수립 후 단계별 실행	여러 단계의 복잡한 작업	전체 개요 작성 → 날짜별 상세 일정 작성
Reflection	결과를 평가하고 다시 시도	품질 기준이 분명한 작업	예산 초과 여부를 검사하고 초과하면 일정 수정

Agent 의 수행 여섯 단계



평가 지표를 먼저 정해야 학습 단계가 동작함. 지표가 없으면 재시도해도 나아졌는지 판정할 수 없음

PART 03

AI Agent 설계 패턴

Workflow 와 Agent 를 구분하고 Multi-Agent 를 쓸 시점을 정함

1 Workflow 와 Agent 의 차이

2 Workflow 의 다섯 가지 패턴

3 Agent 설계의 세 가지 원칙

4 Multi-Agent 가 필요한 경우

5 Routing 과 Supervisor 패턴

6 Multi-Agent 의 실패 요인



KEY INSIGHT 정형 업무는 Workflow 가 맞고 경로를 미리 알 수 없는 업무는 Agent 가 맞음 — 예측 가능성과 유연성을 맞추는 선택임

Workflow 코드가 실행 순서를 정함



- 처리 경로가 명확하고 예측 가능
- 정의된 처리로 빠르고 비용이 일정
- 유연한 대응이 어려움

맞는 업무: 입출력 요건이 명확한 정형 업무

Agent LLM 이 실행 순서를 정함



필요한 도구를 골라 반복 호출

- 상황에 맞춰 경로를 바꾸며 문제 해결
- 탐색적으로 움직여 느리고 비용이 변동
- 동작을 제어하기 어려움

맞는 업무: 복잡한 판단과 탐색적 분석이 필요한 업무

비교 항목	Workflow	Agent
순서를 정하는 주체	사람 (코드)	LLM
단계 수와 비용	고정	실행마다 변동
오류 추적	어느 단계인지 바로 확인	실행 기록(Trace)이 있어야 확인
업무 지식 반영	흐름에 직접 구현	프롬프트와 Tool 설명으로 전달

둘을 결합: Agentic Workflow

- Workflow 의 한 단계에 Agent 를 배치
- Agent 가 Workflow 를 Tool 로 호출
- 업무 지식은 Workflow 로 고정하고 판단이 필요한 구간만 Agent 에 맡김



KEY INSIGHT 업무 대부분은 Agent 까지 가지 않고 고정된 흐름으로 해결됨 — 완전 자율 Agent 는 경로를 미리 정할 수 없을 때만 씀

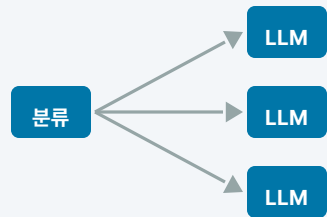
Prompt Chaining



단계를 순서대로 연결

예: 개요 작성 → 본문 작성 → 번역

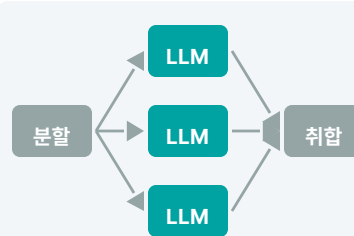
Routing



입력을 분류해 전용 처리로 보냄

예: 문의를 환불, 기술, 일반으로 분류

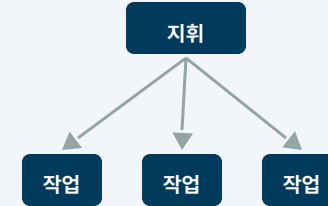
Parallelization



나눠서 동시에 실행하고 합침

예: 코드 리뷰를 보안, 성능 관점으로 동시에

Orchestrator-Workers



중앙이 과업을 나눠 위임

예: 여러 파일을 고치는 코드 변경

Evaluator-Optimizer



생성과 평가를 반복

예: 번역문을 평가해 기준까지 재작성

다른 분류

내용

Andrew Ng 의 네 패턴

Reflection (자기 평가와 개선), Tool Use, Planning, Multi-Agent 협업

패턴 카탈로그 (Liu 외)

18가지 패턴을 목표와 계획, 추론과 확실성, Agent 간 협력, 입출력 제어 네 묶음으로 분류



KEY INSIGHT 단순한 구성에서 시작해 측정 결과가 요구할 때만 복잡도를 올림 — 복잡한 구성은 지연과 비용을 늘림

▶ 단순하게 시작

Use Case 에 맞는 수준의 복잡도로 만들고 과한 설계를 피함

- 비용이 낮고 Debug 가 쉬움
- 지표가 분명해 개선 방향이 보임

복잡도는 필요한 만큼만

⚙ Module 로 구성

Tool 과 Prompt Template 을 재사용 가능한 부품으로 분리

- 새 기능을 끼워 넣을 자리가 생김
- 단일 Agent 에 Skill 을 더해 확장

조합 가능한 부품

👁 관측 가능하게

추론 경로, 검색한 Context, Token 소비를 기록

- 무슨 일이 있었는지 확인
- 왜 그런 결정을 했는지 확인

결정의 이유까지 기록

복잡도를 올리는 순서

앞 단계의 측정 결과가 부족할 때만 다음 단계로 이동

1 LLM 호출 한 번

프롬프트와 예시 개선

2 RAG, Tool 추가

근거와 실행이 필요할 때

3 Workflow

단계가 정해져 있을 때

4 단일 Agent

경로를 미리 알 수 없을 때

5 단일 Agent + Skill

전문 절차가 필요할 때

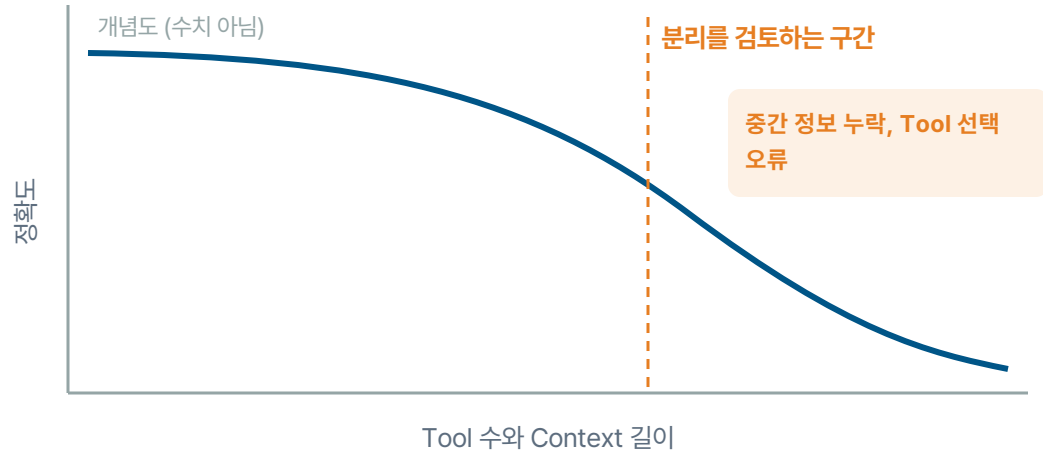
6 Multi-Agent

Context 와 권한을 나눠야 할 때



KEY INSIGHT Tool 과 Context 가 늘어 정확도가 떨어질 때 Agent 를 나눔 — 단일 Agent 로 시작하고 경계가 드러난 뒤 분리함

Agent 하나로 어려워지는 지점



나누는 신호

- Tool 선택 오류가 늘어남
- Context 한도에 자주 닿음
- 과업마다 다른 권한이 필요
- 독립된 조사를 동시에 돌려야 함



나눠서 얻는 것

- Context 격리: Agent 마다 필요한 정보만
- 병렬 처리로 시간 단축
- 권한 분리
- 전문 Agent 에 적은 수의 Tool

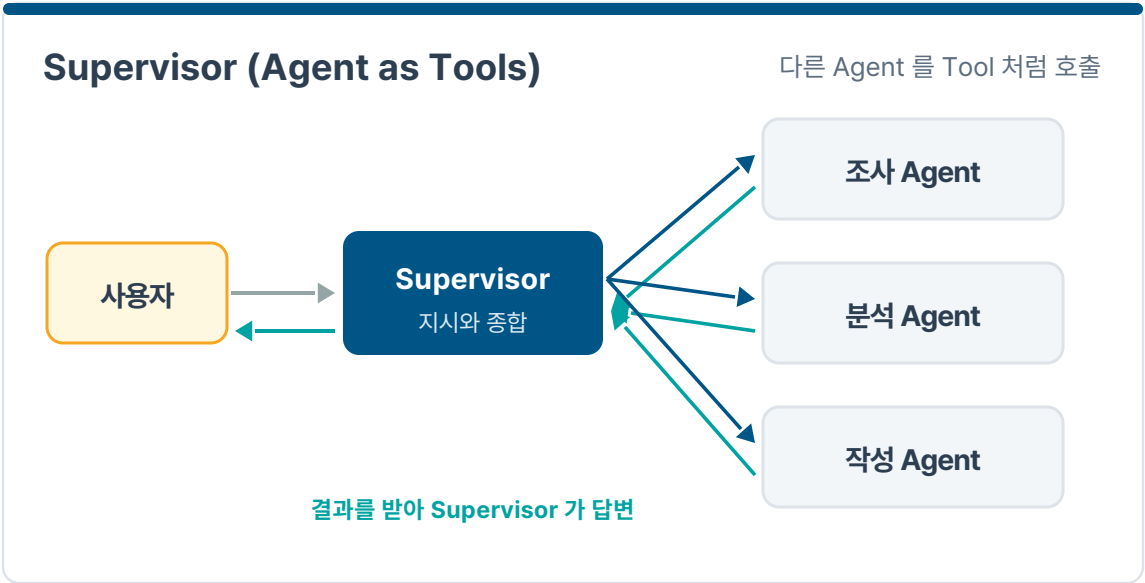
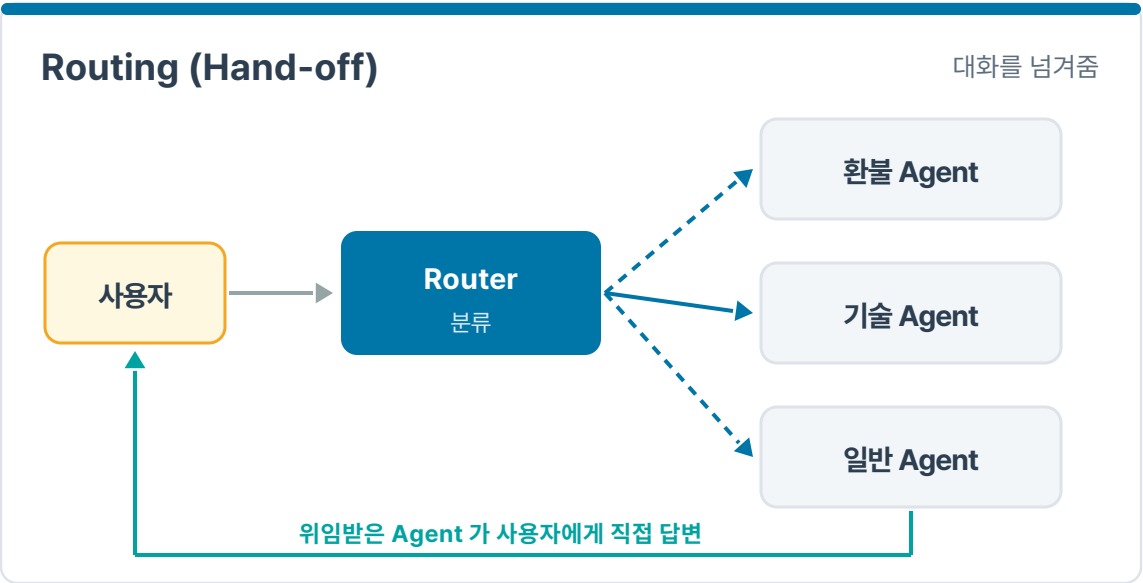
비교	단일 Agent	Multi-Agent
토큰 사용량	기준	크게 증가 (Anthropic 연구 시스템: 일반 대화의 약 15배)
로직 변경	프롬프트 수정으로 가능	Agent 간 계약까지 수정
맞는 과업	순차적이고 서로 의존하는 과업	넓게 탐색하고 병렬로 나눌 수 있는 과업

먼저 확인할 것

- 단일 Agent 에 Skill 을 더하면 목표 정확도에 닿는지
- 기능 경계가 운영 중에 저절로 드러났는지



KEY INSIGHT 요청이 명확히 나뉘면 Routing 이 맞고 결과를 종합해야 하면 Supervisor 가 맞음 — 제어권을 넘기는지 돌려받는지가 다름



비교 항목	Routing	Supervisor
제어 흐름	제어권을 전문 Agent 에 넘김	하위 Agent 를 호출하고 제어권을 돌려받음
Context 공유	대화 이력을 넘겨받음	지시문과 결과만 주고받음 (하위 Agent 의 Context 격리)
맞는 경우와 주의점	요청의 종류가 명확히 나뉠 때. 오분류되면 되돌아갈 경로가 필요	여러 결과를 종합할 때. 지시문이 모호하면 하위 Agent 가 엇나감



KEY INSIGHT Agent 를 나누는 것만으로는 품질이 확보되지 않음 — 실패의 다수가 모델 성능이 아닌 사양과 조율에서 생김



Spec 과 System 설계

- 작업 Spec 을 따르지 않음
- 주어진 역할을 벗어남
- 같은 단계를 반복
- 종료 조건을 알지 못함

대응

역할, 입출력, 종료 조건을 문서로 고정



Agent 사이의 불일치

- 필요한 정보를 전달하지 않음
- 다른 Agent 의 입력을 무시
- 확인 질문 없이 가정으로 진행
- 대화가 초기화되어 맥락을 잃음

대응

주고받는 형식을 스키마로 정하고 누락을 검사



작업 검증과 종료

- 검증 없이 종료
- 불완전한 검증
- 끝나지 않았는데 조기 종료
- 틀린 결과를 통과시킴

대응

검증을 별도 단계나 코드로 분리

업무 적용에서 함께 생기는 과제

답변 품질이 Context 에 의존

정답이 여럿인 작업이 많아 바람직한 출력을 얻으려면 Context 를 적절히 전달해야 함

작업의 복잡성

단계 수가 늘수록 앞 단계의 Error 가 뒤 단계로 전파됨

관측의 어려움

여러 Agent 의 실행 기록을 한 요청으로 묶어 봐야 원인을 찾을 수 있음

PART 04

Agent 연결 표준과 개발 도구

MCP, A2A, Agent Skills 가 맡는 역할과 도구 선택 기준

1 MCP: Tool 연결 표준

2 A2A: Agent 간 통신 프로토콜

3 Agent Skills 의 구조와 역할

4 Agent Skills 의 단계적 로딩

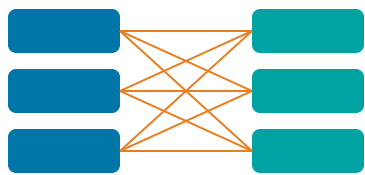
5 Agent 개발 도구의 네 가지 유형

6 Agent 실행 기반과 주변 기능



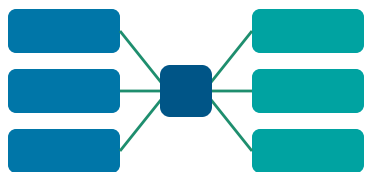
KEY INSIGHT MCP 서버로 만든 도구는 여러 Agent 에서 그대로 쓸 수 있음 — 연결 코드를 제품마다 따로 만들 필요가 없어짐

연결 수의 변화



도구마다 연결 방식이
달라 Agent 하나에 도구
열 개를 붙이려면 연동 열
개를 따로 만들어야 했음

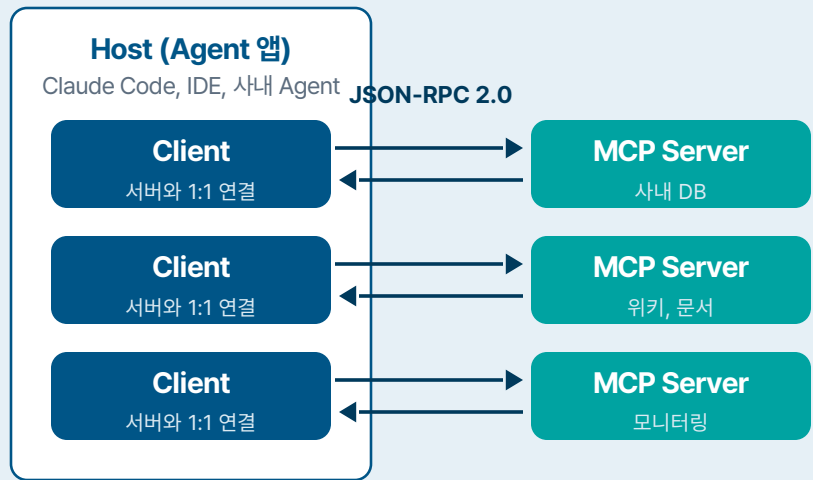
MCP 이전: Agent 3 x 도구 3 = 9개



표준 규격을 한 번 맞추면
어떤 Agent 든 같은
방식으로 연결

MCP 이후: 3 + 3 = 6개

구성



서버 제공	내용
Tools	Agent 가 실행하는 기능
Resources	읽어 올 데이터, 파일
Prompts	재사용할 지시 양식

전송 방식	쓰는 경우
stdio	같은 PC 의 로컬 서버
Streamable HTTP	원격 서버, OAuth 기반 인가

도입 효과

- 사내 DB, 위키, 모니터링을 한 번만 감싸 공용 도구로 재사용
- 도구 접근 권한을 한 곳에서 통제

주의할 점

- MCP 서버는 공급망 위험 대상: 출처와 권한 범위를 검토
- 연결한 서버의 도구 설명이 모두 Context 에 올라감



KEY INSIGHT MCP 는 도구 연결을 맡고 A2A 는 Agent 사이 연결을 맡음 — 다른 조직이 만든 Agent 와 연동할 때 A2A 가 필요함

두 표준이 맡는 구간



비교

	Tool 로 호출	A2A 로 호출
방식	함수 호출, MCP	A2A 프로토콜 (ACP 는 A2A 로 통합)
상대를 아는 시점	개발할 때 미리 지정	Agent Card 로 실행 중에 찾아서 연결
맞는 경우	같은 조직 안의 하위 Agent	조직과 제품이 다른 Agent 연동

A2A 의 동작 순서

1 Agent Card 조회

- 할 수 있는 일, 접속 주소, 인증 방식을 적은 JSON
- 정해진 경로에서 조회

2 Task 생성

- 메시지를 보내 일을 맡김
- Task ID 로 이후 진행을 추적

3 진행 상태 수신

- submitted, working, input-required, completed
- 긴 작업은 스트리밍과 푸시 알림

4 Artifact 수령

- 결과물을 문서, 파일, 구조화 데이터로 받음
- 추가 입력 요청이 오면 응답

통신은 HTTP 위의 JSON-RPC 를 사용. 상대 Agent 의 응답도 검증 대상이며 OWASP 는 Agent 간 통신을 별도 위험 항목(ASI07)으로 분류



KEY INSIGHT Skill 은 작업 절차와 스크립트를 한 폴더로 묶어 Agent 에 전문 지식을 더함 — Multi-Agent 보다 Skill 추가를 먼저 검토함

Skill 폴더 구조 예

pdf-form-fill/

└ SKILL.md

이름, 설명, 작업 절차

└ scripts/

fill_form.py, validate.py

└ references/

field_rules.md, 양식 예시

SKILL.md 머리말의 설명이 호출 여부를 결정



Agent 전문화

업무와 분야에 맞는 작업용으로
기능을 맞춤



반복 감소

한 번 만들어 두면 Agent 가
필요할 때 자동으로 사용



기능 조합

여러 Skill 을 조합해 복잡한 작업
흐름을 구성

구분	주는 것	예
Tool	한 가지 기능의 실행	search_orders, send_mail
MCP	도구에 연결하는 방법	사내 DB 용 MCP 서버
Skill	일하는 절차와 필요한 자료	PDF 양식을 채우고 검증하는 절차와 스크립트

확장 순서



Skill 이 다른 Skill 을 불러 능력의 계층을 만들 수 있음. Multi-Agent 로 넓히기 전에 Skill 을 더해 목표 정확도에 닿는지 확인



KEY INSIGHT Skill 은 이름과 설명만 먼저 읽고 본문은 필요할 때 읽음 — 많이 설치해도 시작 시 토큰 부담이 작음

필요할 때만 더 깊이

L1 Metadata

Skill 의 name 과 description 만

시작할 때 항상 로딩

Skill 하나에 수십~백 토큰 수준

L2 SKILL.md 본문

관련 작업일 때 본문 지시를 읽음

필요할 때 로딩

수천 토큰 이하로 유지

L3 Script 와 자료

지시에 따라 Script 를 실행하고 참고 파일을 읽음

필요할 때 로딩

실행 결과만 Context 에 반영



MCP 와의 차이

- MCP 는 연결한 서버의 모든 도구 설명을 처음부터 Context 에 올림
- Skill 은 쓰지 않는 Skill 의 본문을 Context 에 올리진 않음
- 둘은 대체 관계가 아님: Skill 의 절차 안에서 MCP 도구를 호출

시점

지원 현황

2025년 10월

Anthropic 이 Agent Skills 발표. Claude 제품군에서 지원

2025년 12월

Anthropic 이 Agent Skills 를 공개 표준으로 발표 (agentskills.io)

2026년 기준

Codex, Gemini CLI, GitHub Copilot, Cursor 등이 SKILL.md 를 지원. Codex 는 .agents/skills/ 폴더를 읽음

설명 문구 작성 요령

- 무엇을 하는 Skill 인지 언제 쓰는지 함께 적음
- 사용자가 쓸 표현을 넣음
- 비슷한 Skill 과 구분되는 조건을 적음



KEY INSIGHT 개발 도구는 쓰는 모델과 실행 환경에 따라 고름 — PoC 는 API 직접 호출로 시작해도 충분함

유형	대표 도구	장점	부담	맞는 경우
No-code · Low-code	n8n, Dify, Flowise, Amazon Bedrock Agents	화면에서 흐름을 구성, 빠른 시제품	복잡한 분기와 테스트 자동화가 어려움	현업 주도 시제품, 정형 흐름
Model · Cloud 사업자 SDK	OpenAI Agents SDK, Claude Agent SDK, Google ADK, Microsoft Agent Framework, Strands Agents	각 사의 Model 과 서비스에 연결하기 쉬움	해당 사업자에 종속	쓰는 Model 과 Cloud 가 정해진 조직
독립 SDK	LangChain · LangGraph, Mastra, CrewAI	Model 과 실행 환경을 자유롭게 선택	추상화 층을 익혀야 함	여러 Model 을 바꿔 쓰는 경우
실행 기반	Amazon Bedrock AgentCore, Vertex AI Agent Engine	실행, 인증, 기억, 관측을 제공	운영 방식이 기반에 맞춰짐	운영 단계의 여러 Agent

선택 기준

- Model 종속 여부
- 상태 저장과 재개 방식
- 관측 도구 연동 (OpenTelemetry)

PoC 단계

- API 를 직접 호출해 루프를 구현
- 직접 재현할 수 있는 범위를 안 뒤 Library 선택

사내망 구성 예

- 모델 계층: vLLM, LiteLLM
- Agent 계층: Flowise, Hermes Agent



KEY INSIGHT 운영에 필요한 코드는 핵심 로직보다 인증, 기억, 관측 쪽이 더 많음 — 공통 기반으로 두어야 Agent 를 반복해 만들 수 있음

참조 Architecture 의 네 층

Interaction	사용자와 기존 Application 이 Agent 와 협업 예: Mobile App, CRM, 음성 응답, IT 지원 Portal
Workflow	Agent 가 예측 가능한 방식으로 동작하도록 흐름 제어 예: 고객 확인 절차, 위험 통제, 오류 해결 흐름
Agent	역할 특화 Agent 의 생성, 관리, 배포 예: Model Garden, Prompt Registry, Agent Factory, Memory
Agent Operation	Agent 가 예상대로 동작하는지 출력과 지표 감시 예: 양적, 질적 운영 지표

Agent 주변 기능 (AgentCore 정식 출시 기준)

필요한 기능	담당 서비스	내용
호스팅 환경	Runtime	최대 8시간 실행, 세션 격리, A2A 지원
도구 관리	Gateway	API 와 Lambda 를 도구로 변환, MCP 서버 연결
인증 · 인가	Identity	OAuth 연동, 토큰 보관
기억	Memory	단기, 장기 기억 저장
운영 모니터링	Observability	OpenTelemetry 로 외부 도구 연결
가드레일	Bedrock Guardrails	위험한 입력과 응답을 필터

각 Agent 가 갖출 다섯 가지

알맞은 Model 용도에 맞게 선택	Tool 역할 수행에 필요한 능력 보강	승인된 Data 출처 권위 있는 출처만 사용	기억 과거 수행 이력 저장	Prompt 접근 방법 사람, 다른 Agent 와 상호 작용
------------------------	--------------------------	-----------------------------	-------------------	--------------------------------------

다섯 가지를 표준으로 삼으면 Agent 를 반복 생산할 수 있음. 사내망에서는 Kubernetes 위에 같은 기능을 오픈소스로 구성

PART 05

AI Agent 도입 배경과 기대 효과

기술 추세와 효과가 큰 업무 유형을 근거로 도입 대상을 정함

1 DX 에서 AX 로 바뀐 도입 전략

2 2025~2026년 AI 기술 추세

3 생성형 AI 의 사업 효과 사례

4 AI Agent 도입의 기대 효과

5 대행 효과가 큰 업무의 조건

6 업무 영역별 주요 Use Case



KEY INSIGHT AX 는 사람과 Agent 의 역할 분담을 다시 정하는 일임 — 기술 도입에 더해 조직과 업무 설계가 함께 바뀜

DX 전략 (Digital Transformation)

AX 전략 (AI Transformation)

AI Agent 기술을 어디에 적용할 것인가

핵심 질문

AI Agent 와 함께 어떤 일을 할 것인가

업무 Process 디지털화로 효율화와 비용 절감

핵심 목표

사업 모델, 운영 방식, 고객 경험의 혁신

기존 시스템의 확장 또는 보조 도구

Agent 의 역할

자율적으로 일하는 Digital Labor

챗봇 등 특정 자동화 작업

적용 범위

새로운 Workflow 와 업무 로직 생성

기존 Workflow 안에서 효율을 높이는 도구

활용 방식

운영하며 계속 개선해 업무를 바꿈

생성형 AI 적용 네 영역

분류, 정형화, 요약

의도 분류, 개체 추출, 데이터 레이블링

정보 검색

의미 기반 검색, 사내 규정 RAG, FAQ 응답

추론, 판단

고장 원인 분석, 리스크 평가, 의사결정 지원

보고서 작성, 번역

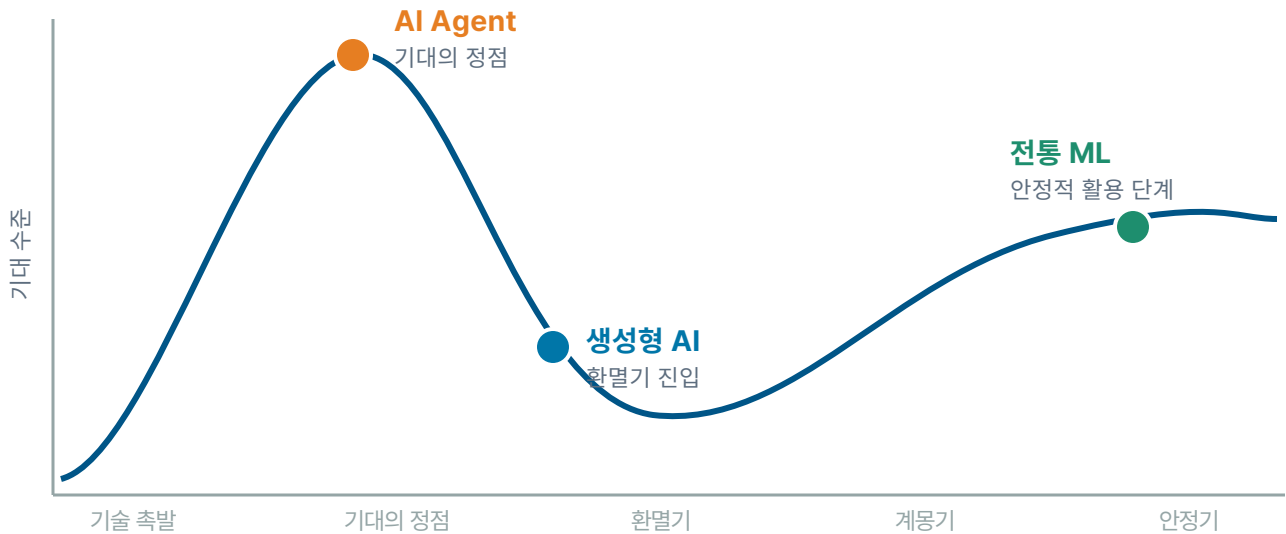
보고서 생성, 다국어 번역, 코드와 SQL 생성

전제: Agent 는 디지털화된 Process 만 자동화할 수 있음. DX 로 쌓은 시스템과 데이터가 AX 의 재료가 됨



KEY INSIGHT AI Agent 는 기대가 가장 높은 시기에 있음 — 기대와 실제 성과의 차이를 감안해 검증된 업무부터 적용해야 함

Hype Cycle 상의 위치 (개념도)



40% 이상

2027년 말까지 중단될 것으로 전망되는 Agentic AI 프로젝트의 비율

이유: 비용 증가, 불명확한 사업 가치, 미흡한 위험 통제 (Gartner)



다음 추세: Physical AI

- 2025년을 AI Agent 의 해로 보는 전망이 다수
- 그다음은 로봇, 자율 주행 등 실세계에 작용하는 AI
- 제조, 물류, 헬스케어에서 운영 최적화와 자동화 기대

평가 기준의 변화

AI 어시스턴트 보유 여부가 아닌, 감지에서 조치까지 자율화한 깊이로 제품을 평가

이름만 Agent 인 제품

기존 챗봇과 RPA 에 Agent 이름을 붙인 사례가 많아 실제 자율 동작을 확인해야 함

대응

PoC 와 제한적 도입에 맞는 시기. 성공 지표를 정하고 검증된 업무부터 확대



KEY INSIGHT 공개 사례의 효과는 특정 업무를 좁게 정한 경우에 나옴 — 전사 일괄 적용보다 업무 단위 적용이 측정하기 쉬움

JPMorgan Chase

계약 문서 검토

36만 시간

연간 대출 계약 검토에 쓰던 인력 시간을 문서 분석 AI(COIN)로 대체

업무 범위: 상업 대출 계약서의 조항 추출

Klarna

고객 상담 1차 응대

230만 건

AI 상담원이 출시 첫 달 전체 채팅의 3분의 2를 처리, 해결 시간 11분에서 2분 미만으로 단축

이후 품질 문제로 사람 상담을 다시 늘림 (2025)

Walmart

상품 정보 정비

8.5억 건

생성형 AI로 상품 카탈로그 데이터를 생성하고 개선. 같은 기간 사람이 하면 약 100배 인력이 필요하다고 발표

업무 범위: 상품 속성 데이터

효과가 나타나는 세 영역

생산성

문서 검토, 데이터 정비의 작업 시간 단축

고객 경험

응답 시간 단축, 24시간 응대

혁신

새 서비스와 개발 주기 단축

원본에 있던 Mayo Clinic, BMW 수치는 1차 출처로 확인하지 못해 제외함. Klarna 사례처럼 품질 요구가 높은 구간은 사람을 남겨야 함



KEY INSIGHT Agent 는 판단 기준을 문서로 남겨 특정한 의존을 줄임 — 담당자가 없어도 업무가 멈추지 않게 됨

불필요한 작업 감소

정보 찾기와 재입력 작업을 줄여 본래 업무에 집중

- 측정: 건당 처리 시간, 재입력 횟수

효율

판단의 일관성

개인의 경험에 좌우되지 않고 회사 기준에 따라 답하고 처리

- 측정: 재작업률, 판정 불일치율

품질

특정인 의존 감소

담당자 부재나 퇴직으로 업무가 멈추는 위험을 줄임

- 측정: 담당자 부재 시 처리율

지속성

기존 자동화의 약점을 보완하는 세 가지

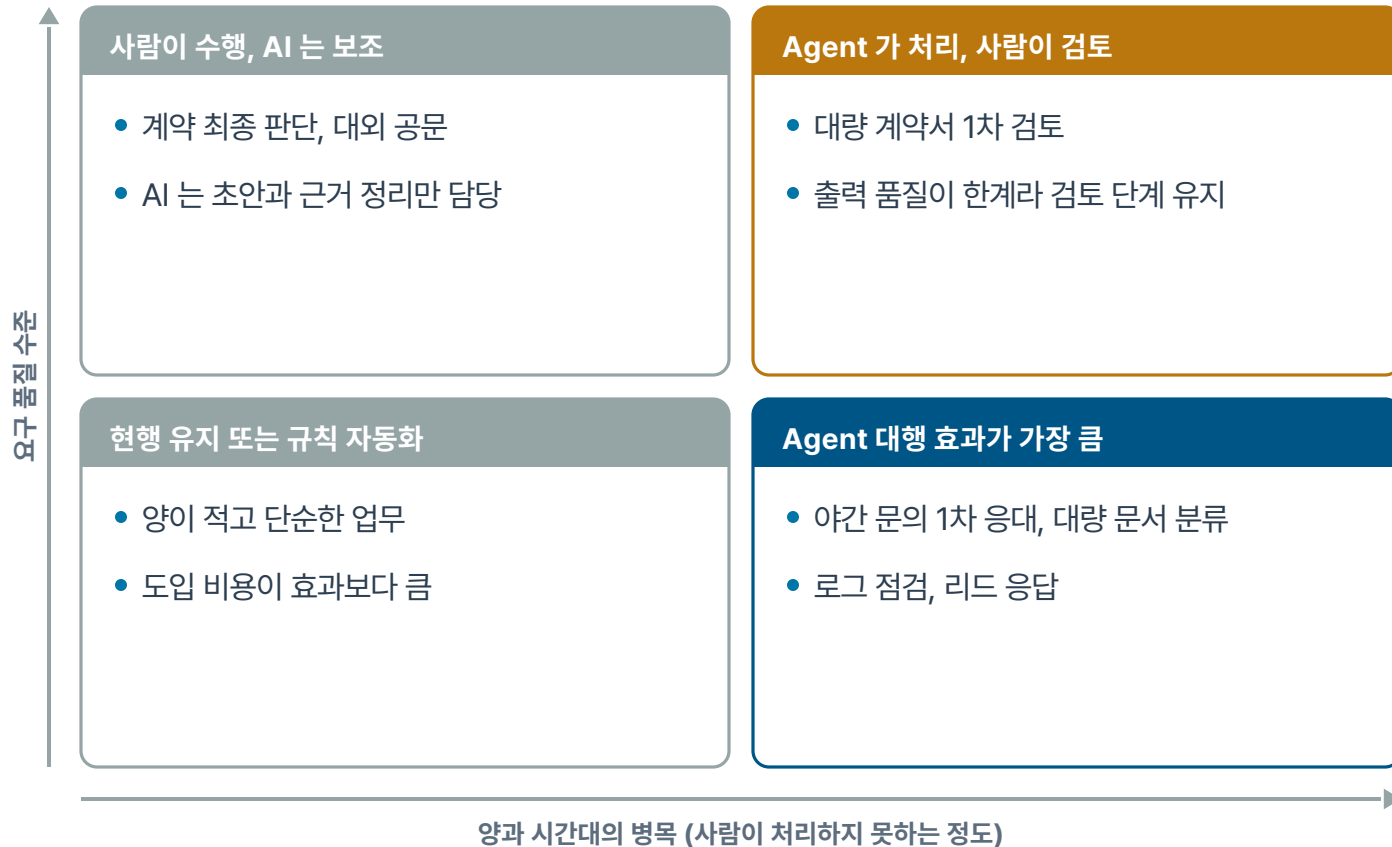
보완하는 지점	방식	효과
화면 변경에 대응	LLM 이 화면을 읽고 누를 요소를 찾음	화면이 바뀌어도 자동화가 유지
Data 구조화	고객과의 대화를 LLM 이 JSON 으로 변환	사람의 입력 작업 감소
예외 처리	Tool 호출이 실패하면 다른 방법으로 다시 시도	새로운 경우를 스스로 처리
System Event (주문 접수, 장애 알림)와 결합하면 사람의 요청 없이도 업무 Process 전체를 자동으로 시작할 수 있음		



대행 효과가 큰 업무의 조건

05 · 도입 배경과 기대 효과

KEY INSIGHT 양과 시간대가 병목인 업무에서 대행 효과가 큼 — 높은 품질이 필요한 업무는 사람 검토를 남겨야 함



병목의 위치

- 시간과 양의 병목은 대부분 사람 쪽에 있음
- 품질의 병목은 대부분 AI 쪽에 있음



Agent 의 이점

- 언제든지 실행 (24시간), 비동기 실행
- 지식과 전문성의 확장
- 효율보다 대응 범위 확대가 핵심

작게 시작하는 기준

- 빈도가 높음, 위험이 낮음, 규칙이 명확함
- 세 조건을 모두 만족하는 업무부터 시작



KEY INSIGHT 적용 가능한 Use Case 는 찾기, 판단, 실행을 이어서 수행하는 업무임 — 필요한 시스템 연동이 난이도를 정함

Use Case	Agent 가 하는 일	필요한 연동	기대 효과	난이도
사내 문서 RAG	매뉴얼, 규정, 법규를 의미 기반으로 검색해 근거와 함께 답변	문서 저장소, Vector DB	검색 시간 단축	하
계약서 검토 지원	구성을 분석하고 사내 표준 양식, 금지 조항과 비교해 수정할 곳을 표시	계약 관리 시스템	초기 검토 시간 단축	중
문서 → ERP 입력	문서를 올리면 정보를 추출해 ERP 에 입력	OCR, ERP API	수동 입력 감소	중
상담 요약 자동 생성	회의록에서 요약, 과제, 다음 조치를 추출해 고객 관리 시스템에 입력	CRM	기록 누락 방지	하
데이터 분석 질의	자연어 질문을 받아 매출 증감 요인과 추세를 분석해 제시	DW, BI	의사결정 지원	중
사양과 품질 판정	설계서와 검사 데이터를 비교해 설계 기준 적합 여부를 판정	PLM, 검사 시스템	특정인 의존 해소	상

세 동작으로 나눠 본 공통 구조



PART 06

AI Agent 도입 전략과 로드맵

과제 선정, 우선순위, 단계별 확대, ROI 측정 순서로 계획을 세움

1 AI 도입 시 기업의 세 가지 고민

2 AI Agent 도입 4단계

3 Use Case 선정 기준

4 도입 과제 우선순위 결정

5 AX 로드맵 3단계

6 AI 도입 ROI와 KPI



KEY INSIGHT 말길 업무가 모호하면 범위, 일정, 평가가 모두 흔들림 — 어떤 업무를 맡길지 글로 정리해 공유하는 일이 먼저임



전사 도입 전략

전사 범용 LLM 도입부터 부서별 특화 AI 까지 단계적으로 확산하는 전략

- 흔한 질문: 어디서부터 시작할 것인가
- 선결 과제: 전사 공통 과제와 부서 과제의 구분



맞춤형 AI 개발

기업 고유 데이터와 업무 Process 에 맞춘 AI 개발

- 흔한 질문: 구매할 것인가 직접 만들 것인가
- 선결 과제: 데이터 위치와 품질 파악



교육과 활용

조직 전체의 역량 강화와 변화 관리

- 흔한 질문: 만들었는데 왜 쓰지 않는가
- 선결 과제: 현장이 바로 쓸 수 있는 형태로 제공

말길 업무가 모호한 채 시작했을 때 생기는 일

업무 파악 부족

- 예외와 제약을 모른 채 추상적인 논의만 진행

거친 업무 분해

- AI 가 맡을 업무 범위가 모호해짐

큰 시제품

- 개발 기간이 길어져 중단, 실패 시 손실이 큼

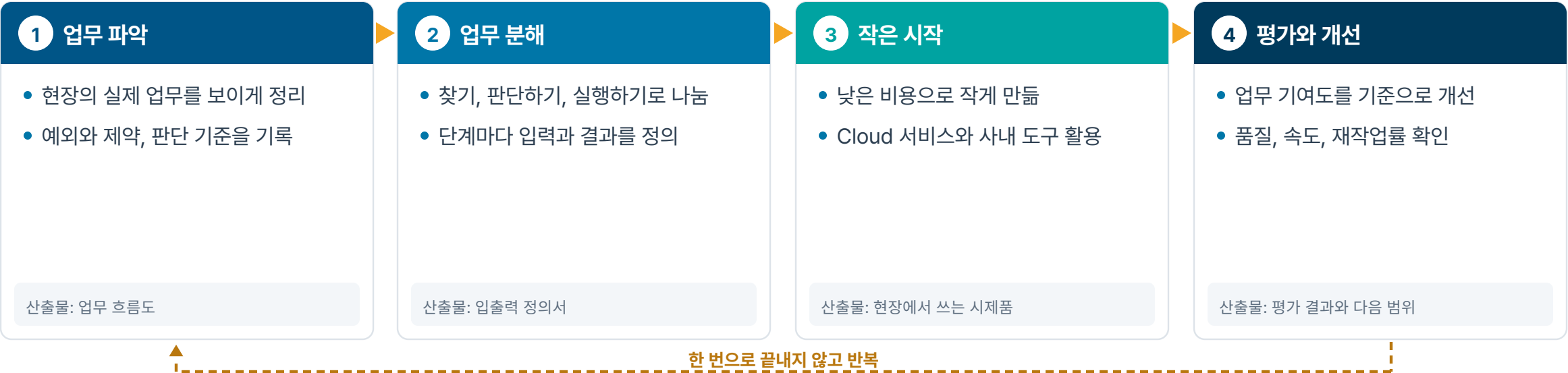
모호한 평가 지표

- 품질, 속도, 재작업을 지표가 없어 개선 여부를 판단하지 못함

출발점: AI 에 무엇을 시킬지를 한 문단으로 적어 현업, IT, 경영진이 같은 문서를 보고 합의



KEY INSIGHT 도입은 네 단계를 반복하며 범위를 넓힘 — 한 단계는 한 줄로 설명되는 크기로 나누고 입출력을 정의함



2단계의 예: 업무를 세 동작과 사람의 일로 나눈 표

찾기 (AI)	판단하기 (AI)	실행하기 (AI)	사람
사내 규정 추출	규정 적합 여부 판정	초안 작성	최종 내용 승인
과거 유사 사례 검색	이상값 탐지	시스템에 임시 저장	이해관계자 조정
고객 데이터 대조	위험 수준 판정	요약 알림	개별 사례의 예외 판단

분해 단위와 진행 원칙

- 한 단계가 한 줄로 설명되는 크기
- 하나의 판단으로 끝나야 함
- 시스템을 개발하듯 설계하고 사람을 교육하듯 개선



KEY INSIGHT 기존 자동화로 풀리는 과제와 데이터가 없는 과제는 피함 — Agent 는 판단이 필요한 반복 업무에서 가치를 냄



맞는 Use Case

- 고객 또는 직원을 대신해 행동
- 시간을 절약하거나 작업 품질을 높임
- 지루하고 반복적인 작업을 처리
- 윤리와 법률 준수가 가능한 영역



피할 Use Case

- 기존 자동화 방식으로 처리할 수 있음
- 데이터가 준비되지 않은 범위를 포함
- 사람이 구조화된 데이터를 넣어 줘야 함
- Agent 없이도 목표를 달성할 수 있음

Use Case 를 정의하는 속성과 SMART 목표 예

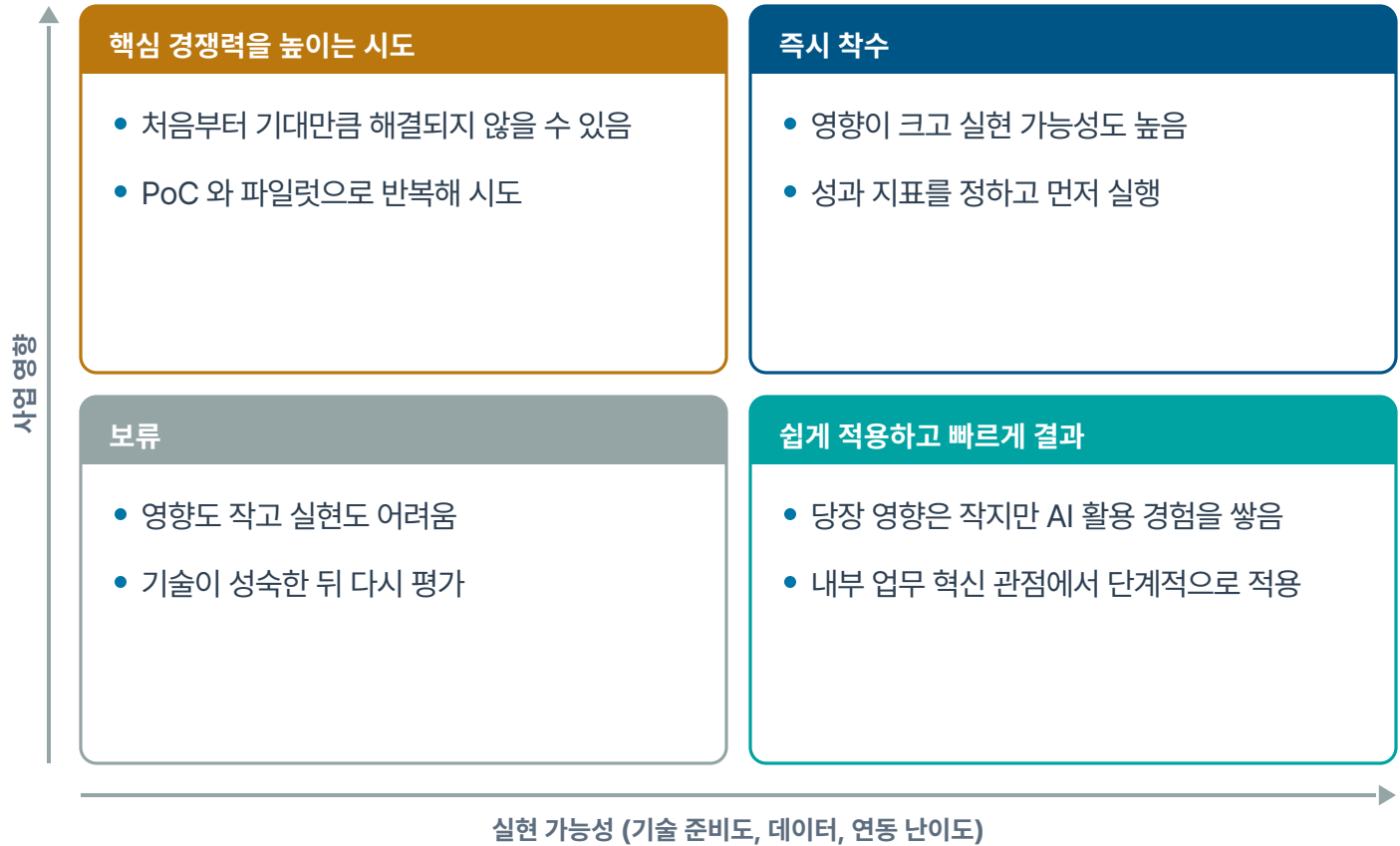
속성	정할 내용
역할	어떤 일을 해야 하는가
데이터	어떤 지식에 접근하는가
행동	어떤 기능(Tool)을 쓰는가
가드레일	하면 안 되는 일은 무엇인가
채널	어디에서 사용자를 만나는가

Specific	Measurable	Achievable	Time-bound
고객 지원 건수 절감	대응 건수 20% 절감	AI Agent 도입	1년 이내
고객 지원 처리 시간 절감	처리 시간 50% 단축	AI 가 답변을 생성	6개월 이내
고객별 맞춤 제품 제안	수주 건수 10% 향상	판매 실적 기반 예측	6개월 이내

수치와 기간은 작성 예시이며 조직 상황에 맞춰 정함



KEY INSIGHT 준비도가 높고 효과가 큰 과제부터 착수함 — 빠른 성과가 다음 단계의 예산과 지지를 만듦



평가 항목	확인할 것
기술 준비도	현재 보유한 기술 역량과 인프라 수준
기술 성숙도	해당 AI 기술의 시장 내 성숙 단계
정량적 효과	비용 절감, 매출 증대, 생산성 향상
정성적 효과	조직 문화, 직원 만족도, 브랜드 가치

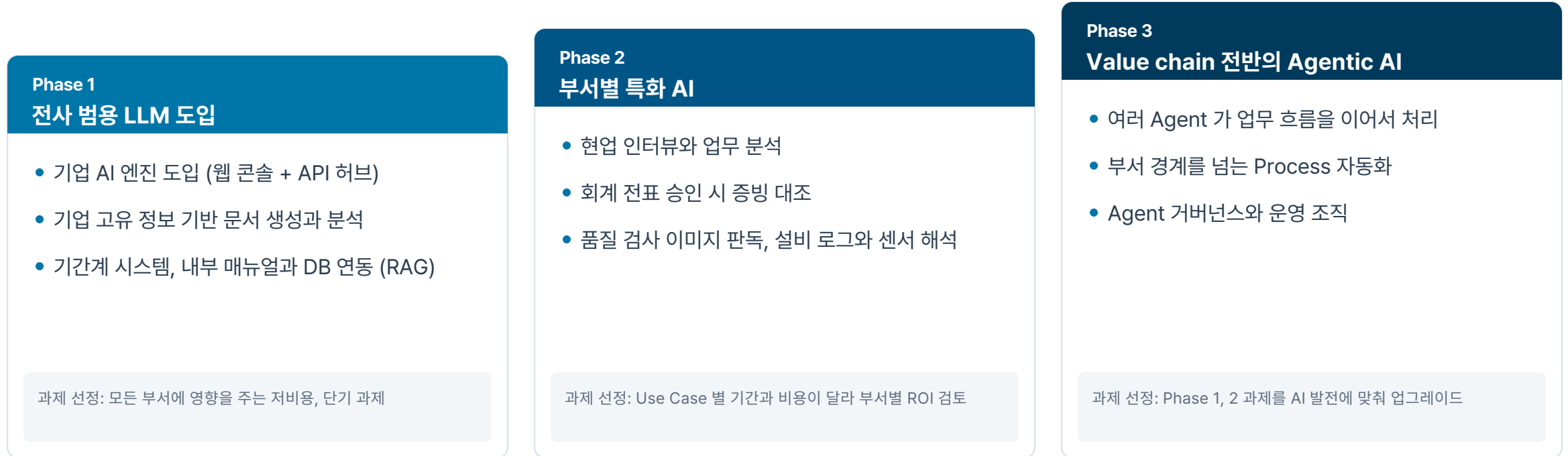
진행 방법

- 문제의 시급성과 기대 수준을 미리 정의
- 과제마다 네 항목을 같은 척도로 점수화
- 분기마다 다시 평가해 사분면을 조정
- 빠른 성과 과제와 전략 과제를 함께 운영

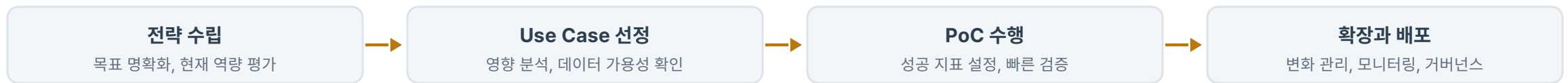


KEY INSIGHT 로드맵은 범용 LLM 에서 특화 AI 를 거쳐 Agentic AI 로 자율성을 넓힘 — 앞 단계에서 쌓은 데이터와 통제 체계가 다음 단계의 전제임

자율성 확장



단계마다 반복하는 추진 순서





KEY INSIGHT ROI 계산에는 토큰 비용과 운영 인력을 넣어야 함 — 개발비만 계산하면 운영 단계에서 수지가 맞지 않음

ROI =

효익 (절감 시간 + 오류 감소 + 매출 기여)

—

비용 (개발 + 토큰 + 운영 + 교육)

÷

비용

기간을 정해 계산

비용 항목	내용
개발	설계, 구현, 시스템 연동
토큰과 인프라	호출량에 비례. Agent 는 반복 호출로 단순 챗봇보다 많이 사용
운영과 평가 인력	출력 검토, 평가 세트 관리, Prompt 수정
교육과 변화 관리	현업 교육, 업무 절차 변경

ROI 측정 모델	보는 것
비용 절감 모델	자동화를 통한 직접 비용 감소
생산성 향상 모델	업무 효율과 처리량 증가
매출 증대 모델	신규 상품과 서비스, 고객 경험 개선
전략적 가치 모델	장기 경쟁력과 시장 위치

추천 KPI

정량 지표

ROI, 순현재가치(NPV), 비용 절감액

시간 효율

업무 처리 시간 단축률, 자동화 비율

품질

정확도, 오류 감소율, 사람 개입률

Agent 운영

과업 성공률, 건당 토큰 비용, 재시도 횟수

PART 07

AI Agent 조직 설계와 적용 사례

직무 체계에 Agent 를 배치하는 방법과 금융권 추진 사례

1 AI 직원 설계: 직무, 업무, 스킬

2 직무별 사람과 Agent 의 역할 분담

3 사람과 Agent 의 조직 모델

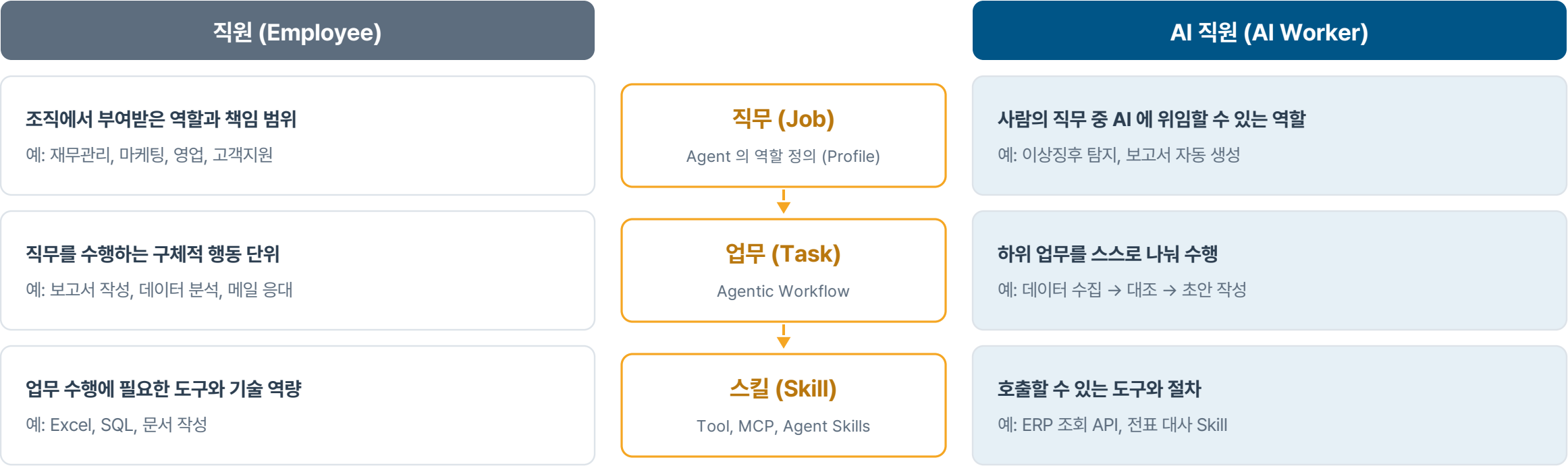
4 금융사의 생성형 AI 추진 전략

5 업무 대행 Agent 개발 순서

6 AX 프로젝트 추진 체계



KEY INSIGHT AI 직원을 사람과 같은 직무 체계로 정의하면 역할 분담을 같은 표로 관리할 수 있음 — 스킬 단위가 Agent 의 Tool 과 Skill 에 대응함



적용 예: 재무 담당





KEY INSIGHT 사람은 판단과 관계를 맡고 Agent 는 탐지, 분석, 초안 작성을 맡음 — 넘겨받는 시점을 정해 두어야 책임이 비지 않음

직무	사람이 맡는 일	Agent 가 맡는 일	사람에게 넘기는 시점
재무관리	예산 편성과 승인, 재무 전략 수립	이상징후 탐지 알림, 실적 리포트 자동 생성	기준을 넘는 차이가 나오거나 승인이 필요할 때
마케팅	캠페인 전략 기획, 방향 설정	트렌드 분석 리포트, 콘텐츠 초안 자동 생성	대외 게시 전 검토 단계
영업 · 계약	핵심 고객 협상, 목표 수립	매출 예측 모델 운영, 계약 데이터 통합 관리	예측이 목표에서 벗어나거나 계약 조건이 표준과 다를 때
고객지원	복잡 민원, VIP 관계 관리	헬프데스크 자동 응대, 고객 만족도 분석	감정이 격한 문의, 환불 한도 초과, 두 번 연속 미해결
경영기획	중장기 전략, 의사결정	KPI 달성 현황 집계, 보고 자료 초안	수치 해석과 대안 선택이 필요할 때

넘기는 조건을 수치로 정함

금액 한도, 신뢰도 점수, 재시도 횟수처럼 판정할 수 있는 조건으로 적음

넘길 때 함께 전달

지금까지 한 일, 확인한 근거, 남은 판단을 요약해 전달

책임자를 지정

Agent 마다 결과에 책임지는 사람을 문서에 적음



KEY INSIGHT 팀 전용 Agent 는 업무 깊이에 유리하고 1인 다역 모델은 범위에 유리함 — 어느 쪽이든 Agent 의 책임자를 사람으로 지정해야 함

교차 기능 팀 모델

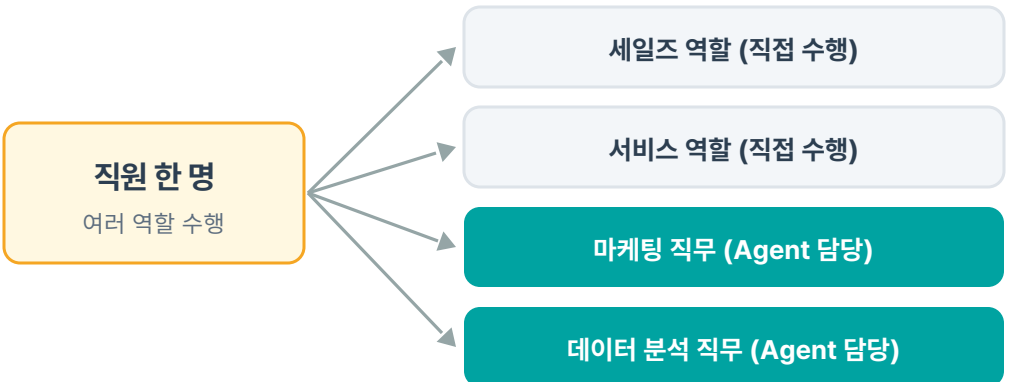
팀마다 전용 Agent



직무별 팀이 자기 업무에 맞춘 Agent 를 공유

하이브리드 팀 모델

1인 다역 + Agent



Agent 가 추가 직무를 맡아 팀의 업무 범위를 넓힘

비교	교차 기능 팀 모델	하이브리드 팀 모델
맞는 조직	직무별 팀이 나뉜 중대형 조직	인원이 적은 팀, 신규 사업 조직
유리한 점	업무 깊이: 팀 지식을 Agent 에 축적	업무 범위: 한 사람이 다루는 직무 확대
관리 부담	Agent 수가 늘어 통합 관리 조직이 필요	개인이 Agent 결과를 검토할 시간이 필요



KEY INSIGHT 금융사는 내부망 LLM 플랫폼을 먼저 갖춘 뒤 Agent 로 확대함 — 규제와 데이터 통제가 도입 순서를 정함



전사 관점의 혁신 전략

생성형 AI 를 도구로 쓰는 전사 전략을 세우고, 도입 효과가 나는 AI Agent 등 Application 개발에 집중



금융사 특성에 맞는 인프라

외부 기술 변화를 받아들일 수 있는 유연한 체계. 여러 Application 을 만들고 관리하는 LLM DevOps 인프라



내부 추진 역량 내재화

AI Engineer 중심의 CoE 조직과 도메인 지식을 가진 현업 Power User 육성

구분	Phase 1. 생성형 AI 활용 기반 마련	Phase 2. AI Agent 확대
Application	현업이 쉽게 구성하는 RAG 과제, 업무 Process 개선 AI Assistant	여러 Task 를 끝까지 처리하는 AI Agent, 업무 시스템과 연동
역량	전사 추진과 운영 전담 조직, AI Engineer 중심 CoE, 현업 Power User	현업 주도 Agent 개발, 평가와 운영 역량
플랫폼 · 인프라	LLMOps 인프라, Vector DB, GPU 서버 클러스터	Agent 실행 기반, 게이트웨이, 관측과 Guardrail

한 플랫폼에서 여러 Application

사내 LLM 플랫폼



Chatbot

Search

Assistant

영업 지원 Agent

운영 Agent

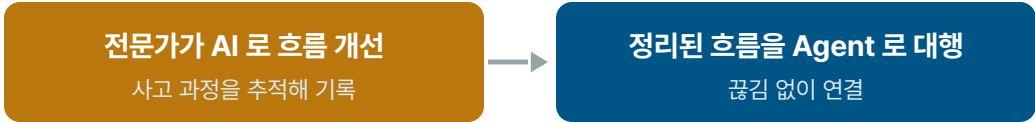
리스크 Agent



KEY INSIGHT 업무 흐름이 정리되어야 Agent 가 대행할 수 있음 — 전문가가 AI 를 써서 흐름을 개선하는 단계가 먼저임

순서	할 일	확인할 점
1	전문가가 AI 로 업무를 수행해 효과를 검증	대행할 가치가 있는가
2	위험 수준에 맞는 제공 방식을 정함	사람의 승인이 필요한 단계는 어디인가
3	작업 단위를 만들고 LLM 에 단계적으로 넘김	전문가의 판단 기준이 반영됐는가
4	연결부 기록과 Guardrail 을 넣음	Error 가 난 위치를 찾을 수 있는가
5	LLM 검사기를 사람의 판정과 비교해 검증	검사 결과를 믿을 수 있는가

대행 전에 할 일: 업무 흐름 개선



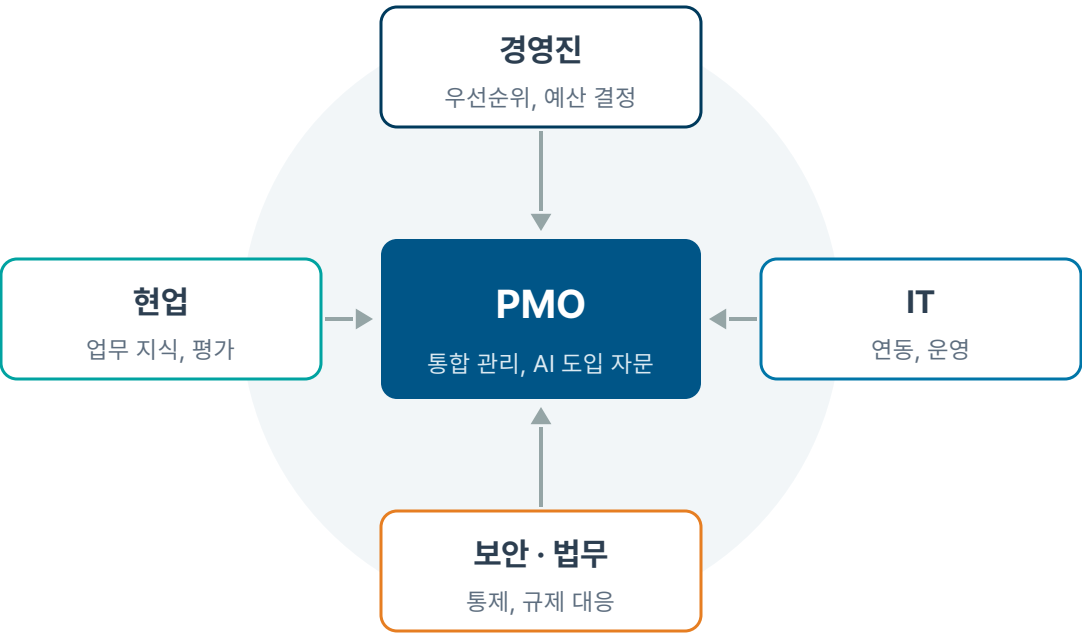
예: Data 분석 업무는 60~70% 가 Programming 중심 작업. 이 부분을 줄이면 가치를 내는 부분에 시간을 쓰고 시도 횟수를 늘림

개발의 세 가지 요점

- 암묵지의 문서화: 업무 흐름을 그리고 전문가의 판단 기준을 글로 남김
- AI 의존의 분리: 최종 판단은 사람이 만든 Logic 으로 처리
- 의도하지 않은 생성의 탐지: 다층 Guardrail 과 Alert, Fail-safe 를 함께 구현



KEY INSIGHT AX 프로젝트는 현업, IT, 보안을 묶는 통합 관리 조직이 있어야 진행됨 — 장기 과제라 환경과 위험 대책을 먼저 갖춰야 함

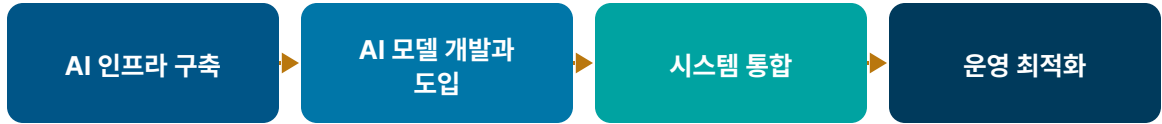


PMO 의 역할

내용

니즈 분석과 현황 진단	과제 후보 수집, 데이터와 시스템 현황 파악
수행사 선정과 협업 체계	역할과 책임 구분, 의사소통 경로 정의
리스크 관리와 품질 보증	보안, 품질, 일정 위험을 추적하고 대응
목표 달성과 성과 측정	KPI 정의, 측정, 다음 단계 반영

프로젝트 단계별 목표



10년 단위 장기 과제에 필요한 세 가지 준비

환경을 갖추

새 기술을 계속 검증하고 배포할 수 있는 환경 마련

리스크 예방

발생할 리스크를 특정하고 대책을 미리 마련

전체 계획

제품의 장기 목표와 도달 계획 수립

PART 08

Coding Agent 와 Vibe Coding

AI 가 개발을 주도할 때 바뀌는 작업 방식과 사람의 역할

1 AI 코딩 활용 수준 다섯 단계

2 Agentic Coding 의 동작 방식

3 주요 Coding Agent 의 형태

4 Vibe Coding 의 정의와 적용 범위

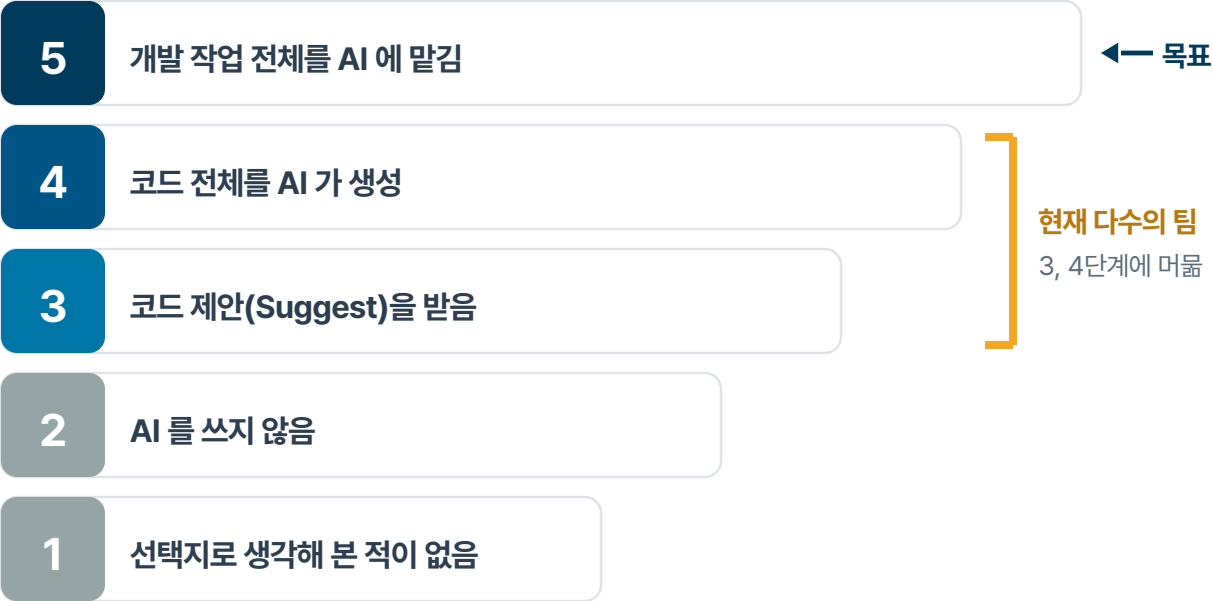
5 AI 주도 개발과 사람의 역할

6 AI Coding 과 개발자의 인지 부하



KEY INSIGHT

많은 팀이 코드 제안과 생성 단계에 머물 — 과업을 통째로 맡기려면 테스트와 리뷰 체계를 팀 단위로 갖춰야 함



수준	지원 방식	작업 단위	운전하는 쪽
1	Program 에 의한 Method 보완, Refactoring	줄	사람
2	AI 의 줄 단위 보완, Chat 으로 Code 생성	Block	사람
3	Agent 의 Code 탐색, 편집, Test	Ticket	사람의 자세한 지시
4	Agent 의 장시간 자동 수행	Milestone	AI 로 교대

Coding 지원의 네 수준 (AWS Japan): 작업 단위가 커지며 수준 4 에서 운전하는 쪽이 바뀜

5단계로 올라가는 데 필요한 팀 차원의 준비

자동 테스트 완료 여부를 Agent 가 스스로 확인할 수 있는 테스트와 실행 명령	작업 지시서 목적, 범위, 완료 조건을 적은 Ticket 과 프로젝트 규칙 파일	격리된 실행 환경 Agent 가 명령을 실행해도 안전한 샌드박스과 권한 설정	리뷰 체계 변경을 작게 나누고 사람이 승인한 뒤 병합
---	--	--	---



KEY INSIGHT Coding Agent 에서는 테스트 출력과 오류가 다음 행동을 정함 — 사람의 일이 지시에서 목표 설정과 확인으로 바뀜

방식	Loop 를 돌리는 쪽	한 번에 처리하는 범위
Autocomplete	사람	커서 주변 몇 줄
Chat	사람 (복사, 붙여 넣기, 실행)	함수나 파일 하나
Agent	모델	여러 파일에 걸친 과업 하나



AI 기반 지원

맥락과 의도를 이해하는 Agent

대화형 개발

사람과 AI 의 실시간 협업

과업 중심

여러 단계의 복잡한 과업을 끝까지 수행

Context 인식

Codebase 구조, Pattern, 규칙을 이해

사람의 일	이전 AI 코딩	Agentic Coding
지시	구현 방법을 단계별로 지시	달성할 목표와 완료 조건을 전달
확인	생성된 코드를 읽고 직접 실행	계획과 변경 내용, 테스트 결과를 확인하고 승인



KEY INSIGHT Coding Agent 는 실행 위치와 사람 개입 방식으로 구분함 — 기능은 빠르게 바뀌어 형태로 이해해야 선택 기준이 유지됨

형태	대표 도구	특징	맞는 작업
터미널형 Agent	Claude Code, Codex CLI, Gemini CLI, OpenCode	터미널에서 저장소를 직접 읽고 명령을 실행. 스크립트와 CI 에 연결하기 쉬움	여러 파일에 걸친 수정, 자동화
Editor 겸 Agent	Cursor, Windsurf	Code 보완, Chat, Agent 기능을 한 Editor 에서 제공	화면을 보며 대화로 수정
Editor 확장 기능	GitHub Copilot, Cline	기존 Editor 에 붙여 사용. Cline 은 쓸 Model 과 API 를 직접 선택	기존 개발 환경 유지
자율형 (Cloud) Agent	Devin, Codex cloud, Copilot coding agent	지시를 받으면 전용 환경에서 비동기로 작업하고 PR 을 생성	장시간 위임, 병렬 처리

형태를 고르는 세 가지 기준

실행 위치

개발자 PC 와 격리된 Cloud 환경 중 하나. 사내 코드 반출 정책과 연결됨

사람 개입 방식

단계별 승인 방식과 결과 검토 방식 중 하나를 선택

Model 선택

Model 고정 여부, 사내 게이트웨이와 로컬 Model 연결 가능 여부



KEY INSIGHT 코드를 검토하지 않는 좁은 뜻의 Vibe Coding 은 시제품에만 맞음 — 운영 코드는 사람이 검토하는 방식으로 써야 함

원 정의 · Andrej Karpathy (2025년 2월)

fully give in to the vibes, embrace exponentials, and forget that the code even exists

코드를 읽지 않고 결과만 보고 다음 지시를 내림. Cursor Composer 같은 Agent 형 도구가 가능 조건으로 함께 언급됨

좁은 뜻의 Vibe Coding

조건

- 자연어만 사용
- AI 가 쓴 Code 를 모두 수용
- 오류는 그대로 붙여 넣어 다시 요청

판단

대규모 개발을 이것만으로 하기는
현시점에서 무리. 결과의 편차가 큼

맞는 범위: 시제품, 일회성 도구, 학습용

넓은 뜻의 Vibe Coding

조건

- AI 가 주도하고 사람이 검토
- AI 주도 개발 전체를 가리킴
- 테스트와 Review 를 유지

판단

대규모 개발에서도 부분적으로 가능.
대기업의 적용 사례가 있음

맞는 범위: 운영 코드의 기능 추가, 리팩터링

초점의 변화

만드는 방법(How)에서 만들 대상(What)으로 이동

과정의 변화

개발 과정이 생성과 검증 두 단계로 단순해짐

용어의 관계

같은 현상을 사용자 쪽과 기술 쪽에서 부른 이름



KEY INSIGHT 사람은 테스트, 적용 범위, 승인을 맡음 — AI 가 쓴 코드의 책임은 승인한 사람에게 남음

Test 주도 개발

기능 요구를 만족하는 Test 를 사람이 작성

- AI 가 쓴 Code 의 동작을 보증
- 완료 기준을 코드로 고정

동작 보증

Guideline 수립

적용할 수 있는 범위와 없는 범위를 명확히 함

- 가능: Prototype, 사내 도구
- 불가: 기간 System, 결제 기능

적용 범위

Review 와 책임

Review 없이 병합하지 않음

- 사람의 승인을 거쳐 병합
- 최종 책임은 사람이 짐

사람의 승인

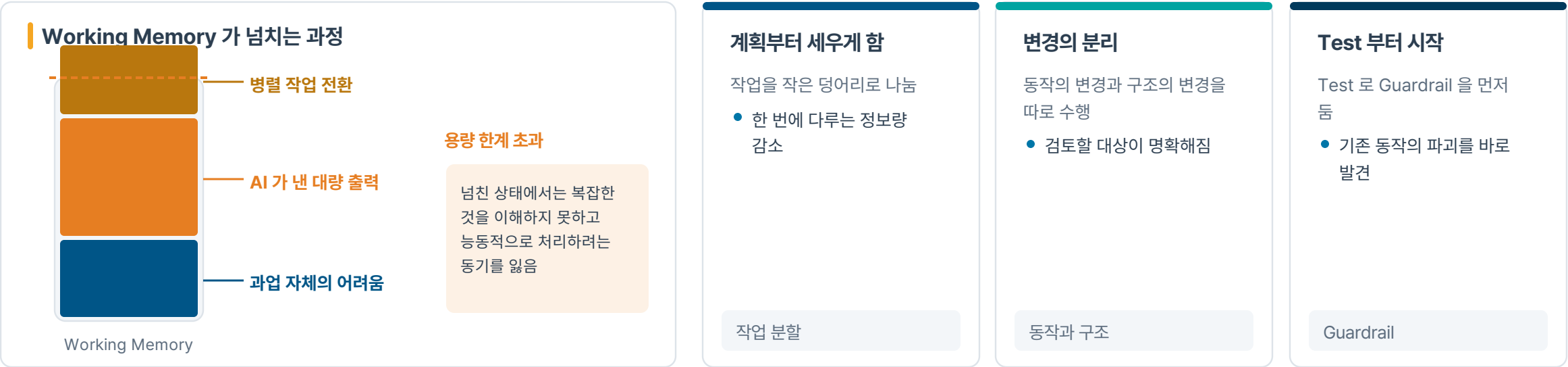
구분	1세대: 사람을 돕는 보조	2세대: Agent 가 개발을 주도
AI 의 역할	줄과 Block 단위 제안	계획, 구현, 테스트를 수행
개발자의 역할	Code 작성자	감독자: 목표 설정, 검토, 승인
필요한 역량	구현 속도	요구 정의, 설계 판단, Review

적용 제외 범위를 먼저 적는 이유

- 사고가 났을 때 피해가 큰 영역을 미리 가름
- 인증, 결제, 개인정보 처리처럼 규제가 걸린 코드
- 제외 범위가 없으면 편한 곳부터 무분별하게 확산



KEY INSIGHT AI 의 출력량이 개발자의 검토 능력을 넘으면 품질이 떨어짐 — 병렬 수를 늘리면 전환 부담으로 검토의 질이 낮아짐



병렬 수를 늘릴 때	생기는 일
사람의 처리 방식	여러 일을 동시에 처리하지 못하고 차례로 처리하며 전환 비용이 듦
AI Coding 의 함정	병렬 구현 수는 쉽게 올릴 수 있지만 사람의 전환 부담이 커져 이해와 검토의 질이 떨어짐
권장	병렬 수는 검토를 끝낼 수 있는 범위로 제한

검토할 때 점검할 것

- 암묵적인 사양이 숨어 있지 않은가
- Data 표현에 Schema 를 주었는가
- 기존 동작을 깨지 않았는가

PART 09

SDD 와 LLMOps

Spec 을 원천으로 두는 개발 방식과 LLM 애플리케이션 운영 항목

1 SDD: Spec 을 먼저 쓰는 개발 방식

2 Spec 의 구성 항목과 조건

3 재구축 비용 하락과 SSoT 설계

4 Agent 개발의 여섯 가지 실패 유형

5 LLMOps 와 MLOps 의 차이

6 Agent Observability 의 세 요소



KEY INSIGHT SDD 에서는 Spec 이 주된 결과물이 되고 사람은 승인을 맡음 — 승인 절차를 지키면 구현 뒤의 재작업이 줄어듦

SDD (Spec-Driven Development) 의 기능 추가 주기



비교	Vibe Coding (좁은 뜻)	SDD
공통 언어	대화 기록. 남지 않거나 흩어짐	문서로 남긴 자연어 Spec
사람의 일	결과를 보고 다음 지시	Spec 을 검토하고 승인



KEY INSIGHT Spec 은 AI 가 그대로 구현할 수 있는 수준까지 적음 — 검증 방법이 없으면 완료를 판정할 수 없음

구분	항목	내용
Why 와 Where	Intent / 배경	목적, 해결하려는 사용자 과제와 사업 과제
Why 와 Where	Scope / Non-Goals	대상 범위, 하지 않을 일
What	동작 Spec	Use Case, 화면 흐름, 입출력과 상태 전이, 예외 처리
What	기술과 품질 요건	API 와 Schema 계약, 제약과 비기능 요건
How to Verify	인수 조건	완료의 정의, Test 관점
How to Verify	작업 분해	구현 작업 목록

Spec 이 갖춰야 할 여섯 조건

- 1

자연어로 작성

사람과 AI 가 모두 읽음
- 2

문서로 남김

대화 기록에만 두지 않음
- 3

구조화

항목과 번호로 참조 가능
- 4

뒤 단계에서 검증 가능

Test 와 연결
- 5

사람과 AI 의 SSoT

같은 문서를 기준으로 삼음
- 6

구현 Feedback 으로 변경

계속 갱신

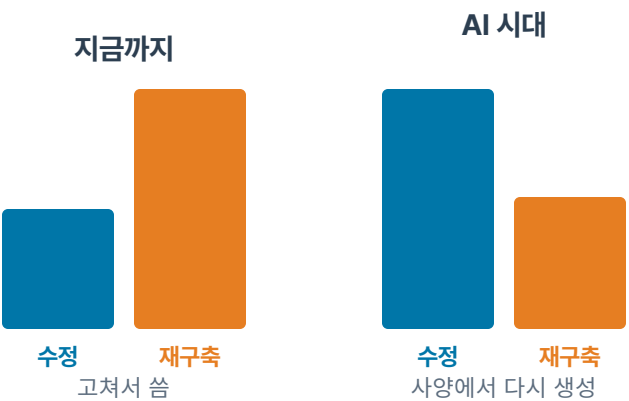
SDD 도구	특징	유의점
Kiro	AWS 의 AI-DLC 를 따르는 IDE 와 CLI. Spec, Steering, Hooks 를 GUI 로 사용	학습 부담이 낮음
GitHub Spec Kit	명확한 Interface 정의와 상세한 작업 분해	문서가 많이 생성됨
OpenSpec	기존 Spec 에 기능 추가 가능. Spec 을 SSoT 로 두고 Feedback 을 반복	익히는 데 시간이 걸림
cc-sdd	Kiro 와 같은 흐름을 여러 Coding Agent 에서 사용	중간 규모 이상의 기능 추가에 적합



KEY INSIGHT 원천만 지키면 코드와 구성도는 다시 만들 수 있음 — 사양과 계약이 남아 있지 않으면 AI 도 같은 기능을 복원하지 못함

수정 비용과 재구축 비용

개념도 (수치 아님)



계층	남길 원천 (사실)	버려도 되는 도출물	수단
데이터	이벤트와 커맨드 (누가, 언제, 무엇을 했는가)	현재 상태, Read Model	Event Sourcing, CQRS
코드	사양 (어떻게 동작해야 하는가)	구현 코드	Spec-Driven Development
아키텍처	의사결정 (왜 그렇게 정했는가)	구성도와 설계 문서	ADR

파괴성 (Destroyability)

부순 영향을 그 범위 안에 가둘 수 있는 정도

- 높으면 그 모듈만 사라지고 나머지는 계속 동작

회복성 (Recoverability)

부순 부분을 교체해 기능을 다시 실현할 수 있는 성질

- 조건: 사양(무엇을 충족)과 계약(인터페이스)

Sacrificial Architecture

몇 년 뒤 폐기된다는 점을 받아들이고 교체하기 쉽게 설계

- 전제: 모듈 경계가 명확해야 부분만 교체

SSoT (Single Source of Truth): 원천은 하나만 두고 나머지는 원천에서 도출. 무엇이 원천인지 먼저 정하지 않으면 전부를 지키려다 변경이 막힘



KEY INSIGHT 기술 면의 실패는 모두 처음부터 복잡하게 시작해서 생김 — 설계가 안정된 뒤에 나누고 추상화해야 변경 비용이 작음

구분	피해야 할 방식	대안	이유
기술	처음부터 Library 사용	API 를 직접 호출해 만들고 동작을 이해한 뒤 추상화	직접 재현할 수 있는 범위를 알아야 Library 를 고를 수 있음
기술	처음부터 Multi-agent 구성	단일 Agent 로 동작을 파악하고 경계가 보이면 분리	단일 Agent 는 Prompt 만으로 Logic 을 바꿀 수 있음
기술	처음부터 기술 Stack 과 층을 분할	단일 저장소, 단일 Stack 으로 Prompt 와 화면을 함께 갱신	Coding Agent 가 전체를 한 번에 참조하고 고칠 수 있음
사업	현장을 보지 않고 사무실에서 개발	현장 밀착 도입, 공통 KPI, 사용자 대표를 Team 에 포함	현장이 써야 개선점이 나옴
사업	곧바로 업무 흐름 변경	기존 업무와 병행 운영해 비교하고 기준을 넘으면 교체	품질을 비교할 기준선이 필요
사업	Test Case 를 만들지 않음	실제 Data 를 반영한 Test Case 와 채점 기준으로 자동 평가	개선 여부를 수치로 판단

기술 면의 공통점

세 가지 모두 설계가 안정되기 전에 구조를 먼저 굳힘

사업 면의 공통점

세 가지 모두 현장과 검증을 건너뛴

시작 구성

API 직접 호출, 단일 Agent, 단일 저장소, 병행 운영



KEY INSIGHT LLMOps 는 Prompt 와 Workflow 를 바꿔 개선하므로 주기가 짧음 — 관측이 먼저 갖춰져야 평가와 개선을 할 수 있음

MLOps

개선 주기가 김 (주, 월 단위)

Model 을 갱신하는 것이 전제



Model 재학습 중심

LLMOps

개선 주기를 짧게 가져감 (일 단위)

Model 을 다시 학습하지 않아도 개선할 수 있음



Prompt 와 Workflow 중심

LLMOps 의 주요 실천 항목: 앞의 항목이 갖춰져야 뒤의 항목을 할 수 있음

1 Observability

Log, Metric, Trace 를 수집해
문제의 위치와 원인을 찾음

2 Prompt 관리

Prompt 를 Code 와 분리해
Version 을 관리하고 변경 이력을
남김

3 평가

출시 전에는 Dataset 으로, 출시
후에는 운영 출력으로 평가

4 Dataset 축적

운영 출력에 Annotation 을 달아
평가와 학습에 씀

5 Model 개선

축적한 Dataset 으로 Fine-tuning
과 Distillation

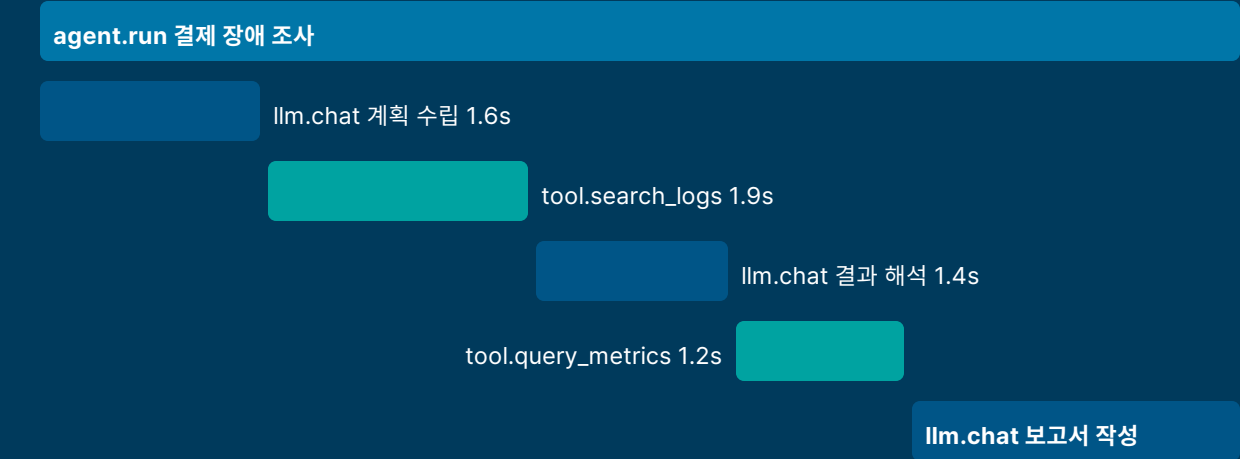
도구 예: Langfuse, LangSmith, MLflow, W&B Weave. 관측과 Prompt 관리, 평가를 한 도구에서 묶어 제공



KEY INSIGHT Agent에서는 단계별 입력과 출력을 기록한 Trace가 가장 중요함 — 결과만 보면 어느 단계에서 틀렸는지 알 수 없음

Trace 예: 요청 하나가 거친 단계 (Span)

전체 8.4초 · 12,300 토큰



단계마다 입력, 출력, 토큰, 소요 시간을 함께 기록

Log

각 작업의 입력과 출력 이력

- LLM에서는 입출력 기록이 특히 중요

입력과 출력

Metric

응답 시간, Token 수, 비용, 평가 점수

- 추세와 이상을 수치로 감시

수치

Trace

요청 하나가 거친 단계와 순서, 단계별 소요 시간

- Agent에서 가장 중요

처리 경로

Agent에서 볼 것

이유

Tool 호출 인자와 결과

잘못된 인자와 빈 결과가 이후 판단을 틀리게 만듦

반복 횟수

같은 호출의 반복은 비용 급증과 무한 루프의 신호

단계별 토큰과 지연

비용과 응답 시간이 어느 단계에서 생기는지 확인

표준

- OpenTelemetry GenAI Semantic Conventions
- 같은 형식으로 남기면 관측 도구를 바꿔도 계측 코드를 유지

PART 10

AI Agent 의 보안 위험

Agent 에서 새로 생긴 위험과 커지는 기존 위험, 통제 계층

1 조작을 실행하는 Agent 의 위험

2 Agent 에서 새로 생긴 실패 유형

3 Agent 환경에서 커지는 기존 위험

4 OWASP Agentic Top 10 위험 목록

5 프롬프트 지시의 한계와 강제 통제

6 생성형 AI 애플리케이션의 네 계층 방어



조작을 실행하는 Agent 의 위험

KEY INSIGHT Agent 의 사고는 정보 유출에 더해 시스템 변경으로 이어짐 — 부작용이 남는 지점마다 서로 다른 방어가 필요함



비교	생성 AI (챗봇)	Agentic AI
하는 일	답변을 출력	답변대로 조작까지 실행 (파일, 명령, 외부 연동, Git)
사고의 결과	잘못된 답변, 정보 유출	정보 유출에 더해 시스템 변경, 외부 송신, 과금
되돌리기	답변을 무시하면 됨	이미 실행된 조작은 되돌릴 수 없는 경우가 있음

원인

- 계획과 실행을 모델이 맡는 구조
- 읽는 내용과 지시가 같은 Context 에 섞임



KEY INSIGHT Agent 자체와 Agent 사이의 관계가 새 공격 대상이 됨 — 기존 애플리케이션 보안 점검표로는 다루지 못함

신규 보안 실패	내용
Agent Compromise	기존 Agent 가 조작되어 의도와 다르게 동작
Agent Injection	악성 Agent 를 System 안에 끼워 넣음
Agent Impersonation	정당한 Agent 인 것처럼 가장
Agent Flow Manipulation	Agent 사이의 실행 흐름을 바꿔 통제 단계를 건너뛸
Agent Provisioning Poisoning	새 Agent 를 만드는 과정에 개입해 악성 구성을 넣음
Multi-Agent Jailbreaks	여러 Agent 의 상호작용을 이용해 보호 장치를 우회

신규 안전 실패	내용
Agent 간 유해 내용	Agent 끼리 주고받는 내용에 해로운 내용이 섞여 사용자에게 전달
다중 사용자 배분의 피해	우선순위와 자원 배분에서 특정 집단이 불이익
조직 지식의 상실	일을 Agent 에 넘기면서 조직이 지식과 역량을 잃음
목표와 안전의 충돌	목표 달성을 안전보다 우선해 위험한 행동을 선택

공격 대상의 변화

입력과 출력만 보던 점검에서 Agent 의 신원, 구성, Agent 사이의 흐름까지 점검 대상이 넓어짐

대응의 출발점

Agent 마다 신원을 부여하고, Agent 를 등록하고 만드는 절차를 통제함

안전 면의 특징

조직이 역량을 잃는 문제를 실패 유형으로 명시. 사람이 검증할 수 있는 지식을 유지해야 함



KEY INSIGHT Agent 가 읽은 내용이 곧 행동으로 이어져 기존 위험이 커짐 — 답변 오류로 끝나던 문제가 실제 피해가 됨

기존 보안 실패	Agent 환경에서 커지는 이유	대응
Memory Poisoning and Theft	기억이 이후 행동을 정하고 장기간 유지됨	기억에 쓰기 전 검증, 출처 기록, 주기적 점검
Cross-Domain Prompt Injection	Agent 가 외부 내용을 읽고 그 결과로 행동함	읽은 내용을 지시로 취급하지 않도록 분리, 도구 권한 최소화
Human-in-the-Loop Bypass	사람의 확인 단계를 건너뛰는 경로가 생김	승인을 Backend 로직으로 강제
Function Compromise	호출하는 함수와 Tool 이 침해되거나 악성일 수 있음	Tool 과 MCP 서버의 출처 검증, 버전 고정
Incorrect Permissions	Agent 에 필요 이상의 권한이 부여됨	과업 단위의 단명 권한, 사용자 권한 범위 안에서 실행
Resource Exhaustion	자율 반복이 자원을 고갈시킴	토큰, 시간, 호출 횟수 상한
Insufficient Isolation	Agent 가 의도하지 않은 대상에 닿음	샌드박스, 통신처 허용 목록

안전 면에서 커지는 문제

- 투명성 부족
- 과도한 의존
- 편향 증폭
- 사용자 사칭
- Hallucination
- 지시 오해

말로 끝나던 오류가 행동으로 이어짐. 그 밖에 Excessive Agency, Loss of Data Provenance 도 같은 분류에 포함



KEY INSIGHT 상위 두 위험은 목표 탈취와 도구 오용임 — Agent 는 읽은 내용을 지시로 받아들일 수 있어 도구 권한을 최소로 줘야 함

ASI01

Agent Goal Hijack

Agent 가 읽는 콘텐츠에 지시를 숨어 목표를 바꿈

입력과 지시의 분리

ASI06

Memory & Context Poisoning

기억과 Context 에 잘못된 정보를 주입

기억 쓰기 검증

ASI02

Tool Misuse and Exploitation

허용된 도구를 의도와 다르게 사용

최소 권한, 인자 검증

ASI07

Insecure Inter-Agent Communication

Agent 간 통신을 위조하거나 변조

상호 인증, 메시지 검증

ASI03

Identity and Privilege Abuse

Agent 의 자격 증명과 권한을 악용

단명 권한, Agent 신원

ASI08

Cascading Failures

한 Agent 의 오류가 연쇄로 확산

상한 설정, 서킷 브레이커

ASI04

Agentic Supply Chain Vulnerabilities

외부 도구, 모델, 플러그인을 통한 침해

출처 검증, 버전 고정

ASI09

Human-Agent Trust Exploitation

사람이 Agent 의 출력을 과신하도록 유도

근거 표시, 결과 검증

ASI05

Unexpected Code Execution

의도하지 않은 코드를 실행

샌드박스 격리

ASI10

Rogue Agents

통제를 벗어나 동작하는 Agent

기록, 감시, 정지 수단



KEY INSIGHT 프롬프트 지시는 보장 수단이 되지 못함 — 도달 범위, 도구, 자원 상한은 모델 바깥에서 설정으로 강제해야 함

소프트한 통제

- 프롬프트의 지시와 금지 사항
- 다른 모델의 행동 리뷰
- 모델 자체의 Safeguard

확률적이라 방지를 보장하지 못함

하드한 통제

- 샌드박스 격리
- 통신처와 권한 제한
- 실행 전 승인
- 토큰, 시간 상한

실행 환경(하네스)이 직접 강제

보험

- 행동 기록과 감시
- 이상 행동 탐지
- 임의 시점의 정지와 재개

예방이 실패했을 때의 수단

예: 진단 Agent 의 범위 이탈을 막는 방법

1 범위 선언

- 대상과 제외 대상을 설정 파일로 명시

2 시작 전 검증

- DNS 레코드나 well-known 파일로 소유권 증명

3 실행 중 강제

- 목록에 없는 URL 로 가는 통신을 차단

프롬프트만 썼을 때 생기는 일

- 대상 외 기능에서 수상한 곳을 발견해 진단 시작
- 링크와 redirect 를 따라 목록에 없는 URL 접근
- 제3자 인프라나 운영 환경에 도달

강제할 대상 네 가지: 도달 범위 (네트워크 허용 목록), 쓸 수 있는 도구, 자원 상한 (토큰, 시간, 비용), 실행 환경 (샌드박스)



KEY INSIGHT 방어는 네 계층을 겹쳐 한 계층의 누락을 다음 계층이 막게 함 — 규칙 공지만 있고 강제 수단이 없는 위험이 사고로 이어짐

4. Cloud 통제와 Network

VPC, Security Group, 조직 정책, 허용 Region 제한

3. Application 보안

신뢰 경계 설정, 입력 검증, 사용자 권한 관리

2. Guardrail

Content Filter, 개인정보 가림, Prompt 공격 차단

1. Data Privacy 와 보호

접근 제어, API 활동 추적, Log 수집, 감사 Log 집적

LLM 과 Data



거버넌스의 세 층 (Mercari)

L1

정책과 규칙

무엇을 허용하고 금지할지 정함

L2

도구 허가

통제할 수 없는 도구를 사내에 들이지 않음

L3

가드레일

허가한 도구 안에서 위험한 동작을 막음

리스크 요인

규칙으로 공지

기술로 강제

위험한 AI 도구 이용

도구 리스크 심사, 허가한 도구 목록 공개

허가하지 않은 도구로 가는 통신 차단

과도한 대리 행위와 권한

Workflow 리스크 심사

Agent 동작을 제한하는 샌드박스

크리덴셜과 기밀 유출

개인 키 사용 금지 규칙

게이트웨이 경유 강제, Secret 자동 검사

PART 11

Guardrail 과 Agent 품질 평가

입출력 검사, 사람 승인, LLM 심사를 품질 기준으로 연결함

1 Guardrail 의 검사 항목과 방법

2 Guardrail 적용 위치와 정책 강제

3 Human in the Loop 승인 설계

4 AI Agent 품질의 다섯 영역

5 LLM as a Judge 구성과 검증

6 Red Teaming 과 Agent 운영 원칙



KEY INSIGHT 규칙으로 확인되는 항목은 규칙으로 검사하고 의미 판단만 LLM 에 맡김 — 규칙 검사는 빠르고 결과가 일정함



검사에 걸리면: 차단, 가림 처리, 담당자 알림, 안전한 기본 응답

검사 항목	막는 대상
거부 주제	다루지 않기로 정한 주제의 질문과 답변
단어 Filter	지정한 단어와 표현
민감 정보 Filter	개인정보 등 민감 정보의 차단 또는 가림
Content Filter	혐오, 모욕, 성적, 폭력 표현과 Prompt 공격
문맥 근거 검사	근거 문서와 맞지 않는 답변

검사 대상	방법	예
단어와 표현	문자열, 정규 표현식 일치	금지어, 연락처 형식
원문과의 차이	편집 거리, ROUGE-N	원문을 지나치게 옮겼는지
출력 형식	Schema 검증 (Pydantic 등)	필수 항목, 자료형, 길이
의미와 내용	LLM 에 의한 판정	사실과 다른 내용, 부적절한 어조

다층

입력, 처리, 출력의 각 층에서 검사

다중

같은 위험을 여러 검사로 확인해 통과 확률을 낮춤

검사 순서

빠르고 싼 규칙 검사를 먼저, LLM 판정은 마지막에

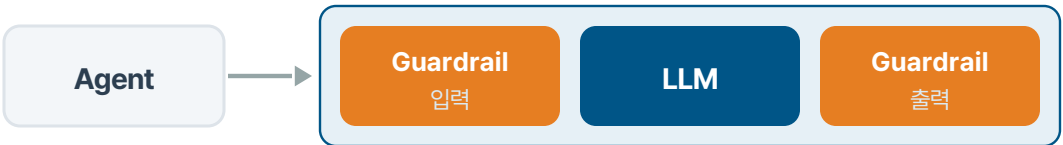
한계

결정적으로 동작하지 않으므로 권한 통제와 함께 사용



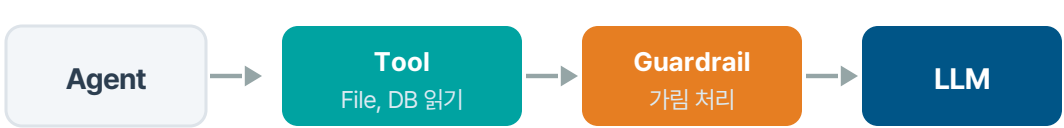
KEY INSIGHT Tool 실행 결과도 LLM 에 넘기기 전에 Guardrail 로 검사해야 함 — 적용을 개발자 설정에 맡기지 않고 권한 정책으로 강제함

방법 1: Model 호출에 함께 적용



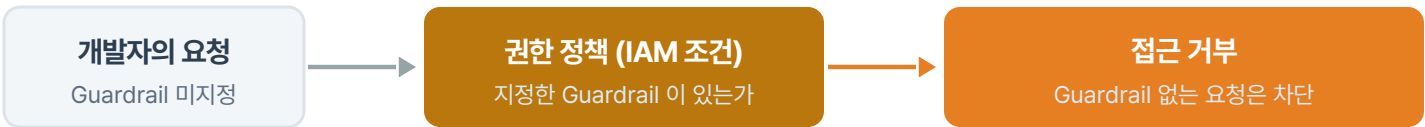
- Model 호출 API 의 설정에 Guardrail 을 지정
- 효과: 호출마다 자동 적용

방법 2: Guardrail 만 따로 호출



- LLM 을 거치지 않고 Guardrail 기능만 호출
- 효과: Tool 실행 결과를 LLM 에 보내기 전 검사

조직 정책으로 강제하는 방법



- Cloud: IAM 정책의 조건으로 Guardrail 식별자를 지정
- 사내망: LLM 게이트웨이에서 모든 호출에 일괄 적용
- 개발자의 설정에 맡기지 않는 것이 핵심



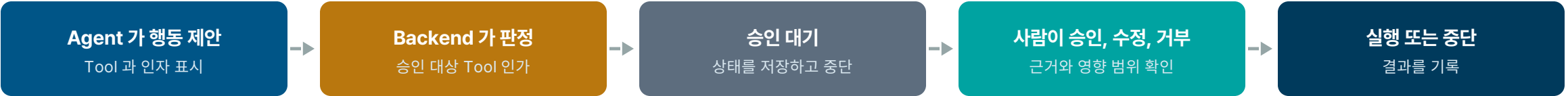
Guardrail 이 다루는 위험

- 내용 위험: AI 윤리, 지식재산 침해, 유해성, 사실과 다른 답변
- 보안 위험: Prompt Injection, 기밀 유출, Model 대상 DoS
- Agent 는 File 과 Database 를 스스로 읽으므로 읽은 Data 를 가려서 넘김



KEY INSIGHT 영향이 크거나 되돌릴 수 없는 행동은 사람의 승인을 받음 — 승인 여부를 LLM 판단에 맡기지 않고 Backend 로직으로 강제함

승인 흐름



승인이 필요한 행동을 가르는 기준

되돌릴 수 있음

자동 실행 + 기록

예: 초안 저장, 내부 메모

되돌릴 수 없음

자동 실행

예: 조회, 요약, 검색

사람 승인

예: 대량 메일 발송, 설정 일괄 변경

사람 승인 필수

예: 결제, 삭제, 운영 환경 변경

되돌릴 수 있음

되돌릴 수 없음

구현할 때

- Tool 단위로 승인 여부를 설정
- 승인 대기 상태를 저장하고 시간 초과 시 중단

승인 피로

- 승인 요청이 너무 많으면 사람이 읽지 않고 승인함
- 낮은 위험은 자동 실행으로 돌려 요청 수를 줄임

시점

기록과 감시

실행 중

지금 하는 행동과 다음 계획을 화면에 표시, 즉시 정지

실행 후

모든 명령과 통신 기록, 정책과 대조, 고위험 결과는 자동 통지



KEY INSIGHT Agent 품질은 모델 성능 외에 가치, 기본 품질, 안전, 보안을 함께 봐야 함 — 정답률이 높아도 업무 가치가 없으면 쓰이지 않음



가치

업무 성과에 기여하는가

- 지표: 처리 시간 단축, 사용자 수락률
- 확인: 현업 인터뷰, 사용 기록



성능

과업을 정확히 수행하는가

- 지표: 과업 성공률, 단계별 정확도
- 확인: 평가 Dataset



기본 품질

서비스로서 쓸 만한가

- 지표: 응답 시간, 가용성, 건당 비용
- 확인: 운영 Metric



안전

해로운 출력과 오동작이 없는가

- 지표: Guardrail 통과율
- 확인: 유해 범주 검사



보안

공격에 견디는가

- 지표: Red Teaming 결과
- 확인: 주입, 권한 상승 시험

Guardrail 통과율을 품질 지표로 쓰는 방법

1 정답 없이 판정

- 출력만 보고 판정 (Reference-free 평가)

2 운영 출력 전체에 적용

- 통과율을 품질 지표로 사용

3 사용자 반응과 대조

- 수락, 수정, 거절을 함께 확인

최종 기준

- 사용자의 반응이 최종 기준
- 통과율은 높은데 거절이 많으면 검사 기준을 다시 맞춤



KEY INSIGHT 심사 LLM 의 판정은 사람의 판정과 비교해 검증해야 함 — 평가 세트의 성능이 운영 환경의 정상 동작을 보장하지 않음

심사 구조



심사 기준을 만드는 방법

- 현장 전문가의 판단 과정을 검사 Prompt 에 그대로 적음
- 정확성, 어조, 형식 등 관점을 나눠 기준을 둠
- 점수보다 통과와 불통과로 판정하고 이유를 남김
- 어느 관점에서 떨어졌는지 알아야 개선책으로 이어짐

심사 LLM 의 Bias

현상

위치 Bias	먼저 제시된 답을 선호
장황함 Bias	긴 답을 더 좋게 평가
자기 선호 Bias	같은 Model 이 만든 출력을 선호
범용 지표의 한계	일반 기준이 현장 업무의 기준과 다름

심사기를 검증하는 두 단계

1 정확도 검증

같은 출력을 사람과 심사기가 판정
일치율이 기준을 넘는지 확인

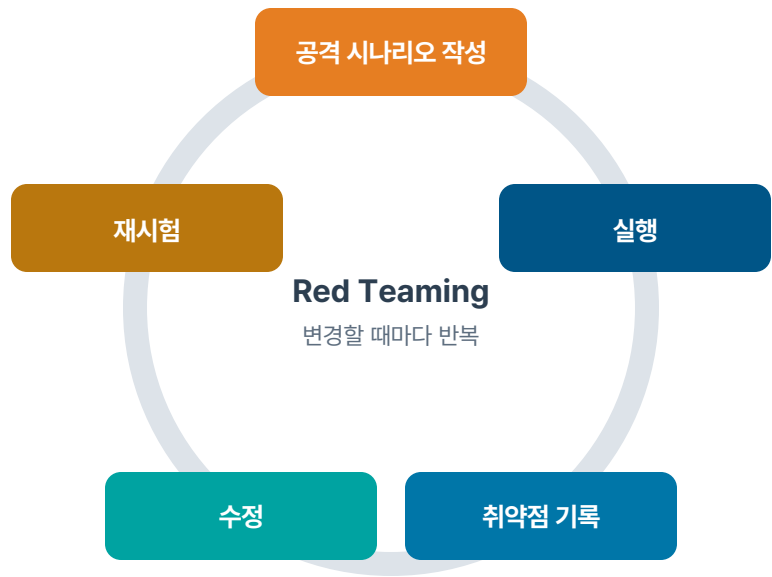
2 동작 검증

입력을 바꿔 가며 견고성을 Test
바뀌지 않아야 할 판정과 바뀌어야 할 판정을
모두 확인

LLM 심사는 구현보다 현장 업무에 맞추는 일이 어려움. 평가 세트에서 잘 맞아도 운영 입력은 분포가 달라 주기적으로 사람 판정과 다시 비교



KEY INSIGHT Red Teaming 은 모델이나 Prompt 를 바꿀 때마다 다시 해야 함 — 한 번의 통과가 변경 뒤의 안전을 보장하지 않음



시험 대상	내용	도구 예
Prompt Injection	문서, 웹 페이지, Tool 결과에 지시를 숨어 목표를 바꾸는지 확인	promptfoo, PyRIT
Jailbreak	보호 장치를 우회해 금지된 출력을 내는지 확인	garak, PyRIT
권한 상승과 도구 오용	허용 범위를 넘는 Tool 호출과 데이터 접근을 시도	수동 시나리오
유해 내용	답하면 안 되는 범주를 한국어 Data 로 확인. 영어 기준 검사는 다른 언어에서 성능이 떨어질 수 있음	전용 검사 서비스

책임 있는 Agent 운영 원칙

행동 범위 제한

할 수 있는 일과 닿는 범위를 미리 정함

승인 요구

영향이 큰 행동 전에 사람의 승인을 받음

중단 가능성

언제든 멈추고 되돌릴 수 있게 함

위험 수준 추정

자체 Agent 의 위험 수준을 미리 추정하고 대응 방침을 정함

PART 12

로컬 LLM 과 LLM 게이트웨이

클라우드를 쓸 수 없는 업무의 실행 도구와 모델 접근 통제

1 로컬 LLM 적용 조건

2 사용 규모별 로컬 LLM 실행 도구

3 모델 크기로 추정하는 GPU 메모리

4 LLM 게이트웨이의 역할과 기능

5 로컬과 클라우드 혼합 구성

6 로컬과 클라우드 사이의 라우팅 기준



KEY INSIGHT 기밀 데이터와 비용 상한이 걸린 업무에만 로컬 모델을 둬 — 나머지는 클라우드가 성능과 운영 부담에서 유리함

- 
기밀 데이터
 외부 전송이 허용되지 않는 데이터를 취급

 - 클라우드 API 호출 자체가 불가
- 
종량 과금
 상한이 보이지 않아 자유로운 사용을 허용하기 어려움

 - 예산 통제 수단이 없으면 이용 제한으로 귀결
- 
로컬 LLM 이 해법이 되는 이유
 - 데이터가 사내를 벗어나지 않음
 - 사용량이 늘어도 추가 과금 없음
 - 엔지니어가 제한 없이 실험

도입 전에 나오는 의문	답	근거
애플리케이션을 다시 만들어야 하는가	Ollama, vLLM, LM Studio 모두 OpenAI 호환 API 제공. 주소 변경으로 전환	각 도구 문서
로컬 LLM 은 성능이 낮은가	모델 크기에 좌우됨. 품질 비교 수치는 과업별로 직접 측정	확인된 공통 수치 없음
매우 비싼 GPU 가 필요한가	4bit 기준 10억 파라미터당 약 0.6~0.7GB 로 어림	모델 파일 크기
여러 사람이 동시에 쓸 수 있는가	Ollama 는 기본 1 이라 설정 필요, vLLM 은 연속 배치 제공	Ollama FAQ, vLLM 문서
쉽게 실행할 수 있는가	설치 후 명령 한두 줄로 실행	각 도구 문서

주류는 클라우드형 LLM. 로컬은 클라우드가 막히는 업무에만 적용하는 편이 현실적



KEY INSIGHT 개인 실험과 다중 사용자 서빙은 맞는 도구가 다름 — 기본 설정으로는 동시 요청 처리가 제한됨

도구	조작 방식	특징	동시 처리	맞는 규모
Ollama	CUI	간단한 명령으로 조작. OpenAI 호환 API 와 도구 호출 지원 (MIT)	기본 1, 설정으로 확대	개인, 소규모 팀
LM Studio	GUI	가장 쉬운 선택지. 2025-07 부터 업무 사용도 무료	개인 사용 중심	개인 실험
llama.cpp server	서버	Ollama 와 LM Studio 가 내부에서 쓰는 엔진의 원본. CPU 실행 가능	병렬 슬롯 지원	저사양, 내장형
vLLM	서버	연속 배칭과 프리픽스 캐싱으로 높은 처리량 (Apache-2.0)	연속 배칭	여러 사용자 서빙

Ollama 설정	뜻
OLLAMA_NUM_PARALLEL	모델당 동시 요청 수, 기본 1
OLLAMA_MAX_LOADED_MODELS	동시에 올려 둘 모델 수
OLLAMA_MAX_QUEUE	대기열 길이, 기본 512, 초과 시 503
OLLAMA_KEEP_ALIVE	기본 5분 뒤 메모리에서 해제

vLLM 기능	뜻
연속 배칭	들어오는 요청을 진행 중인 배치에 추가해 처리량 향상
프리픽스 캐싱	공통 앞부분의 계산 재사용
도구 호출	--enable-auto-tool-choice 와 --tool-call-parser 를 함께 지정
Production Stack	라우터, 관측, 캐시를 묶어 제공 (클러스터 규모)

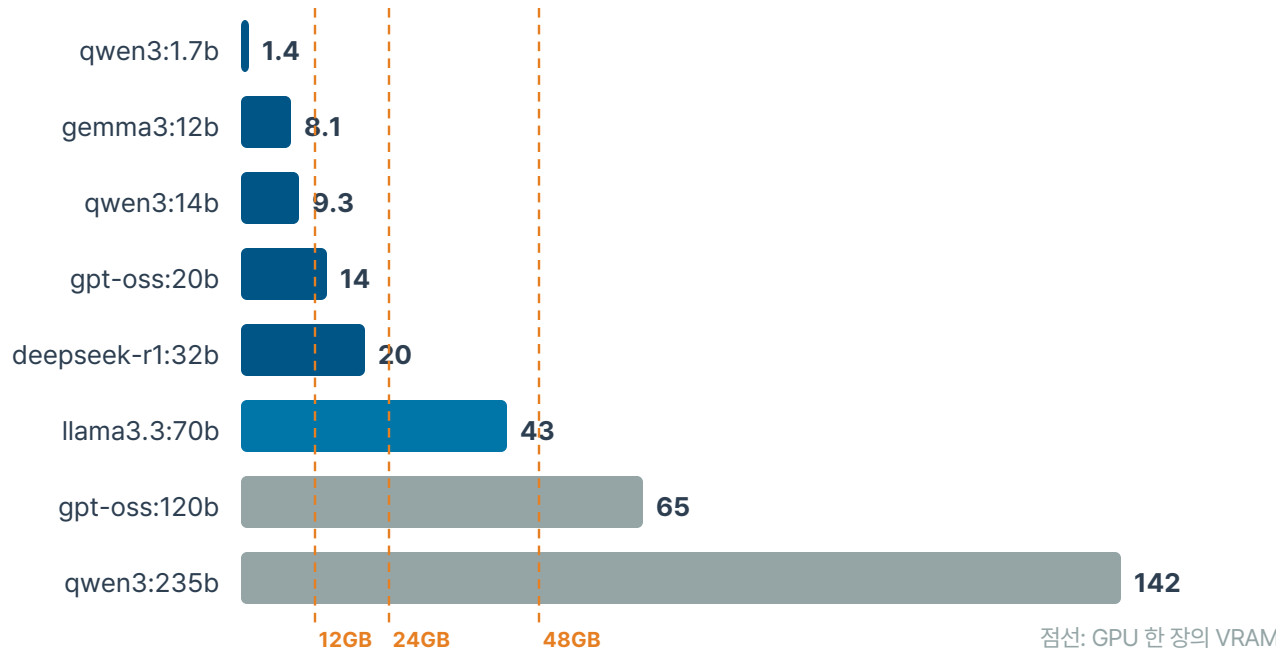
메모리와의 관계: 병렬 수만큼 Context 메모리가 증가. 2K Context 로 4개를 병렬 처리하면 8K 분량이 필요 (Ollama FAQ)



모델 크기로 추정하는 GPU 메모리

KEY INSIGHT 4bit 양자화 기준 10억 파라미터당 약 0.6~0.7GB 가 필요함 — Context 길이와 병렬 수만큼 메모리가 더 들어감

모델별 4bit 양자화 파일 크기 (GB)



어림셈

필요 VRAM $\approx$ 파라미터(십억) $\times$ 0.6~0.7GB

+ Context 길이와 병렬 수만큼의 KV 캐시

가중치 크기의 1.2배 정도를 하한으로 잡음 (여유분은 어림)

VRAM

올릴 수 있는 모델의 대략 범위

12GB

12~14B 모델

24GB

20~32B 모델

48GB

70B 모델

그 이상

GPU 여러 장 또는 통합 메모리 장비

양자화

가중치를 4bit 로 줄여 크기를 16bit 의 약 3분의 1 로 낮춤

시작 크기

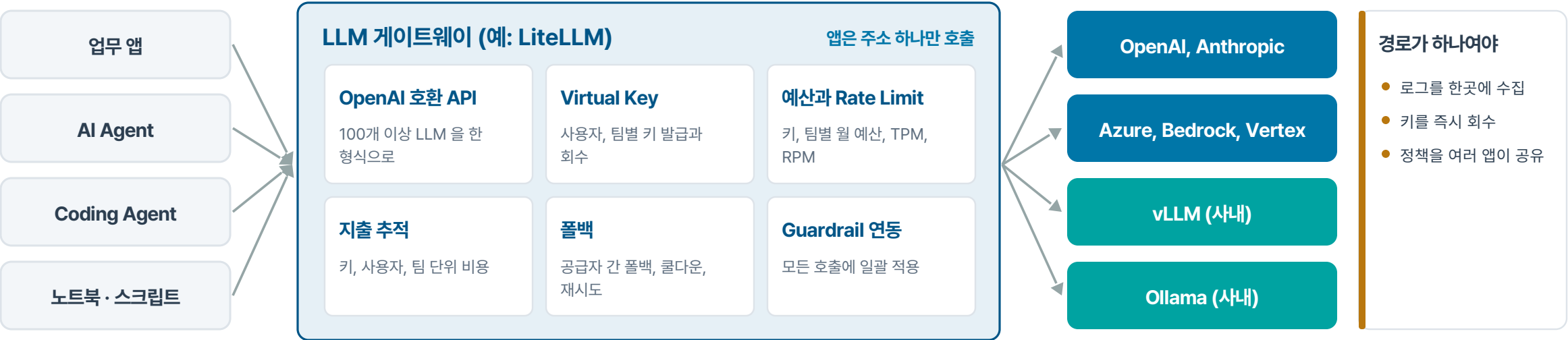
1.7B, 4B 급은 노트북에서도 실행 가능

주의

모델 크기가 품질을 정하므로 과업별로 직접 측정



KEY INSIGHT LLM 호출이 한 경로를 지나야 로그 수집과 키 회수가 가능함 — 누가 얼마나 썼는지를 키와 팀 단위로 추적할 수 있음



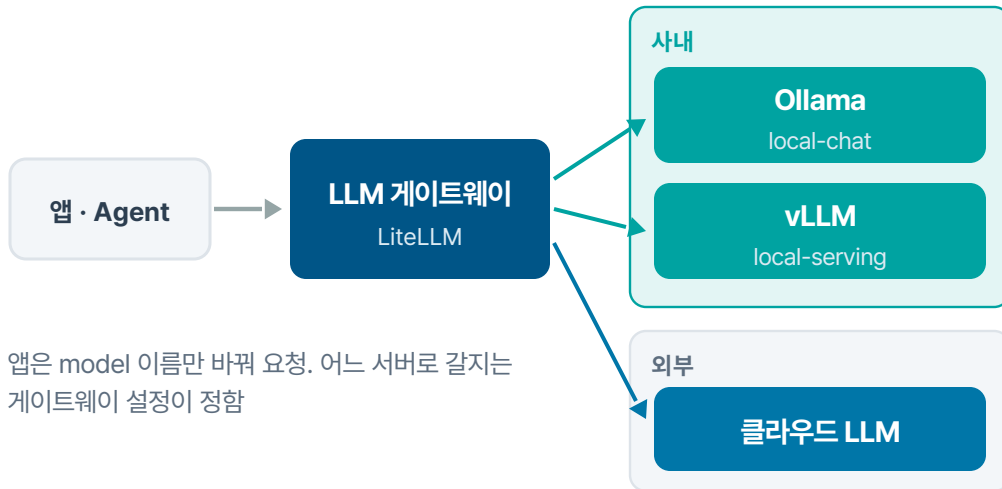
통합 수단	위치	특징	로컬 LLM 연결
LangChain 등 라이브러리	애플리케이션 코드 안	앱마다 따로 구현. 고기능	가능하지만 앱마다 설정
OpenRouter (호스팅형)	인터넷의 서비스	OpenAI SDK 그대로 사용, 운영을 서비스에 위임	사내 로컬 LLM 연결에 맞지 않음
LiteLLM (자체 호스팅)	사내에 직접 배치	모델 접근, 폴백, 지출 추적을 한곳에서 처리	사내에 있으므로 연결 가능

가상 키, 예산, 지출 추적, 폴백은 오픈소스 범위. SSO, 감사 로그 등 일부 기능은 유료 범위이므로 도입 전에 확인



KEY INSIGHT 게이트웨이 뒤에 로컬과 클라우드를 함께 두면 앱 수정 없이 모델을 바꿀 수 있음 — 도구 호출 지원과 컨테이너 네트워크는 따로 확인해야 함

혼합 구성



config.yaml 예 (이름과 주소는 예시)

```

model_list:
  - model_name: local-chat
    litellm_params:
      model: ollama_chat/<모델>
      api_base: http://host.docker.internal:11434
  - model_name: local-serving
    litellm_params:
      model: hosted_vllm/<모델>
      api_base: http://<vLLM 주소>
  - model_name: cloud-chat
    litellm_params:
      model: <클라우드 모델>
  
```

대상 모델 접두어

Ollama	ollama_chat/
vLLM	hosted_vllm/
LM Studio	lm_studio/

접두어의 역할

- LiteLLM 이 접두어로 백엔드 종류를 구분

컨테이너 네트워크

Docker 안의 localhost 는 컨테이너 자신.
호스트는 host.docker.internal 로 지정

도구 호출 지원

미지원 모델은 JSON 모드로 대체되므로
지원 여부를 모델별로 명시

지출 추적

로컬 모델은 단가표에 값이 없어 토큰 단가를
직접 입력

동시 처리 상한

로컬 추론 서버의 병렬 수를 넘는 요청은
대기하거나 실패



KEY INSIGHT

라우팅의 첫 기준은 데이터 민감도임 — 민감 데이터 경로의 폴백에 클라우드가 섞이면 통제가 무너짐

판단 순서

1

외부로 나갈 수 없는 데이터인가

예 → 로컬 고정. 폴백 대상에서도 클라우드 제외

2

응답 지연이 중요한가

대화형은 첫 토큰이 빠른 쪽, 긴 추론은 품질이 높은 쪽

3

양이 많고 단순한가

분류, 추출, 요약 같은 대량 작업은 로컬. 고난도 작업은 클라우드

태그 기반 라우팅 설정 예

```

model_list:
  - model_name: chat
    litellm_params:
      model: ollama_chat/<모델>
      tags: ["internal"]
  - model_name: chat
    litellm_params:
      model: <클라우드 모델>
      tags: ["default"]
router_settings:
  enable_tag_filtering: True

```

동작

- 요청 본문의 tags 또는 x-litellm-tags 헤더로 태그 전달
- 같은 model 이름이라도 태그에 따라 다른 서버로
- 팀, 키별 허용 모델과 함께 사용



폴백 설정에서 통제가 무너지는 경우

민감 데이터 요청

로컬 모델 장애
과부하, 재시작

폴백: 클라우드 모델
데이터가 외부로 전송

- 민감 경로는 로컬 모델끼리만 폴백
- 폴백 목록을 경로별로 따로 정의
- 장애 시에는 실패로 응답하는 편이 안전



KEY INSIGHT Agent 는 단순한 구성으로 시작해 좁게 정한 업무에 적용함 — 통제는 프롬프트가 아닌 모델 바깥의 설정으로 강제해야 함

01 정의와 특징

목표를 받아 도구로 업무를 끝까지 처리하는 시스템

02 동작 원리

계획, 도구 사용, 관측을 목표 달성까지 반복

03 설계 패턴

단순한 구성에서 시작하고 필요할 때만 복잡도를 올림

04 연결 표준과 도구

MCP 는 도구 연결, A2A 는 Agent 연결, Skill 은 절차

05 도입 배경

양과 시간대가 병목인 업무에서 효과가 큼

06 도입 전략

말길 업무를 글로 정리하고 네 단계를 반복

07 조직 설계

사람과 Agent 의 역할과 넘기는 시점을 미리 정함

08 Coding Agent

사람은 테스트, 적용 범위, 승인을 맡음

09 SDD 와 LLMOps

Spec 과 관측이 먼저 갖춰져야 개선이 가능

10 보안 위험

읽은 내용이 행동으로 이어져 위험이 커짐

11 Guardrail 과 평가

규칙 검사와 사람 승인을 Backend 에서 강제

12 로컬 LLM 과 게이트웨이

게이트웨이로 호출 경로를 하나로 모음

다음 단계



빈도가 높고 위험이 낮고 규칙이 명확한 업무에서 시작



THANK YOU

감사합니다

AI Agent 도입과 운영 문의

MSAP.ai

msap.ai

<https://www.msap.ai/>

hello@msap.ai

02-6953-5427

