LangChain / LangGraph, 2026년의 의미

Workflow에서 Harness까지 — 진화의 지도

노토랩 변형호

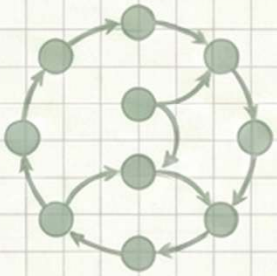
2026년 4월

바이브 코딩 시대에도, Framework의 철학을 알아야 하는 이유

LangChain을 "쓰느냐 마느냐"의 문제가 아니다. 이 framework가 어떤 문제를 풀기 위해 어떻게 진화했는지를 아는 것이, 좋은 agent를 만드는 능력과 직결된다.

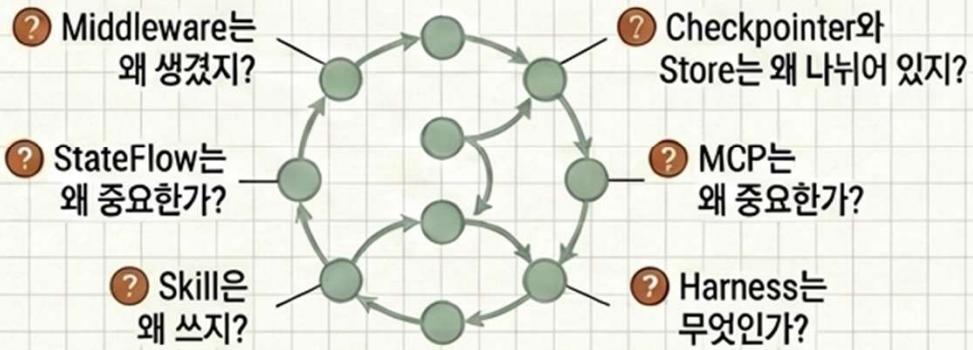
쉬운 선택

실행 결과



짤 수는 있다

진짜 질문



왜 이렇게 생겼는지를 알아야, 잘 만든다

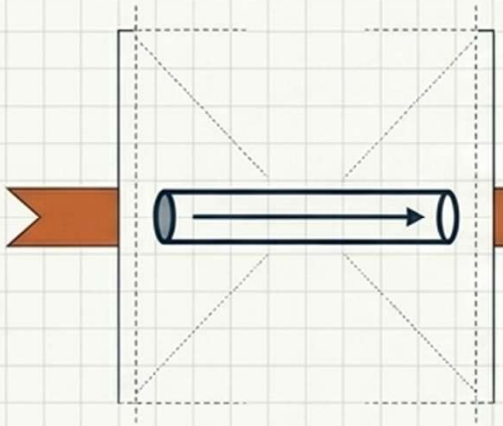
- AI 코딩 에이전트 시대 — 바이브 코딩으로도 agent를 짤 수 있다
- Framework 없이 raw API로 돌리는 것도 충분히 가능한 선택지

- **이 자료의 목적:** LLM 활용이 Workflow → Agent로 이동하면서 LangChain이 어떻게 변했는지, 2026년 시점에서 그 진화의 의미를 정리한다

Framework의 진화를 아는 것은 곧, 좋은 agent를 설계하는 능력이다.

2024 → 2026, 차원이 확장되는 아키텍처 지도

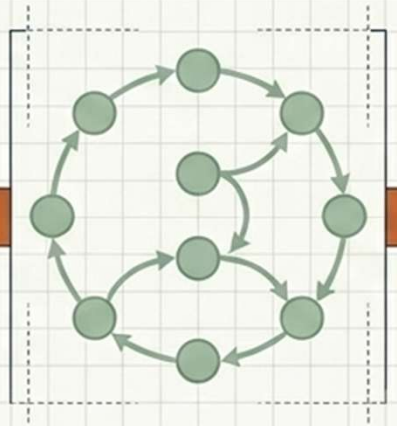
2024



Workflow

- 선형 파이프 (1D)
- LCEL Pipeline 시대
- 결정론적 데이터 흐름

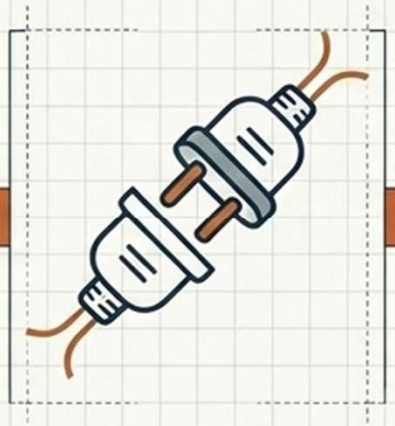
2025 상반기



Agent Loop

- 순환형 평면 그래프 (2D)
- LangGraph StateGraph
- 상태와 제어 흐름

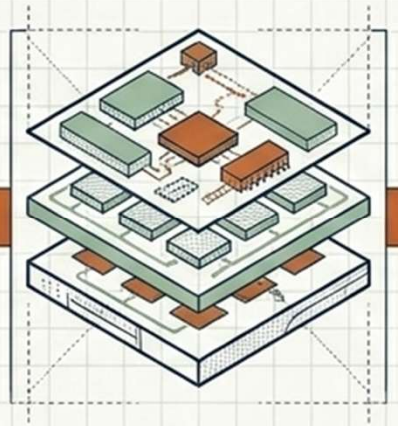
2025 하반기



Pluggable Context

- 확장 포트 결합
- MCP 어댑터
- LangChain 1.0 (Middleware 알파)

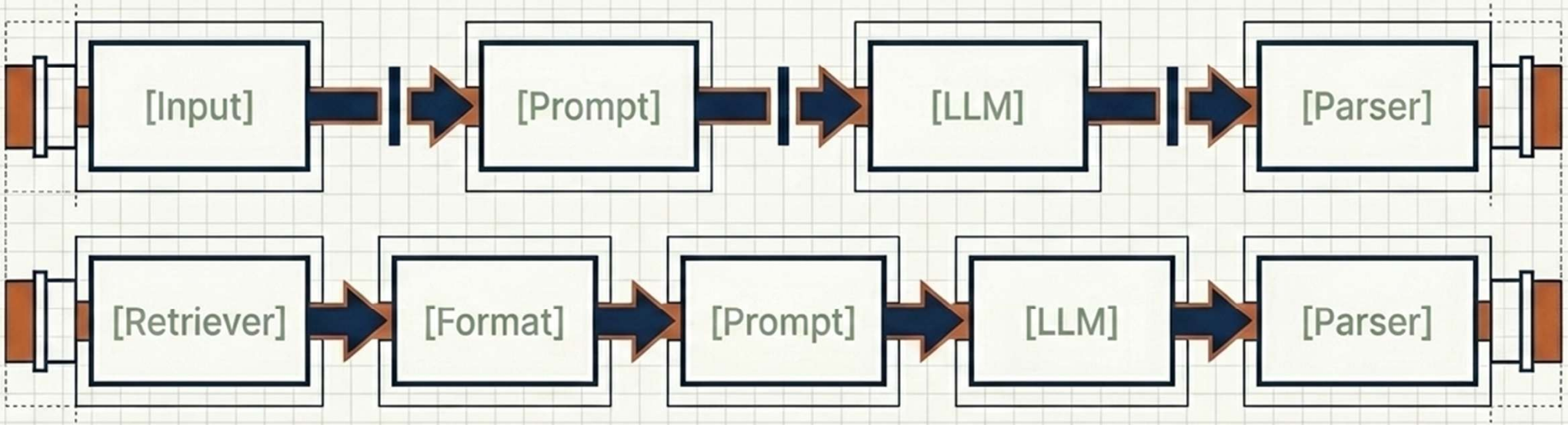
2026



Harness

- 입체적 다중 레이어 (3D)
- DeepAgents / Skills
- 외골격 하네스 통합

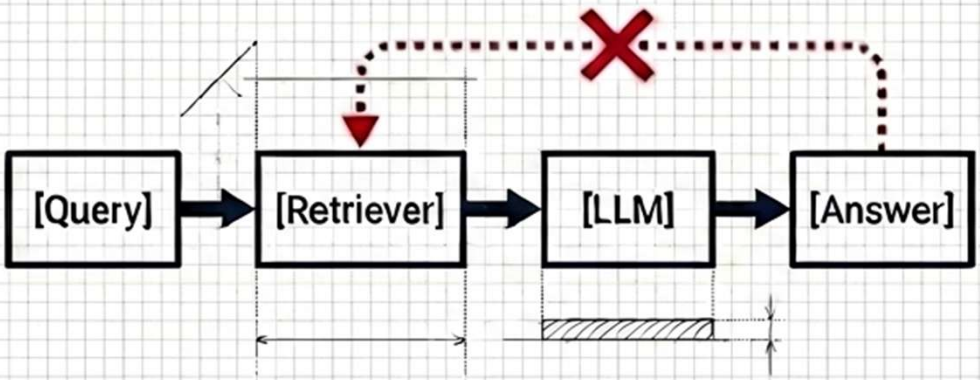
2024년 — LCEL, 체인을 파이프로 잇다



2024년의 LCEL 앱들은 데이터가 입력에서 출력으로 향하는 단방향의 결정론적 구조였습니다. 독립적으로 존재하는 여러 체인들을 파이프(|)를 사용해 하나의 거대한 선언적 워크플로우로 용접하여, 뒤로 되돌아가는 흐름 없이 데이터를 순차적으로 처리했습니다.

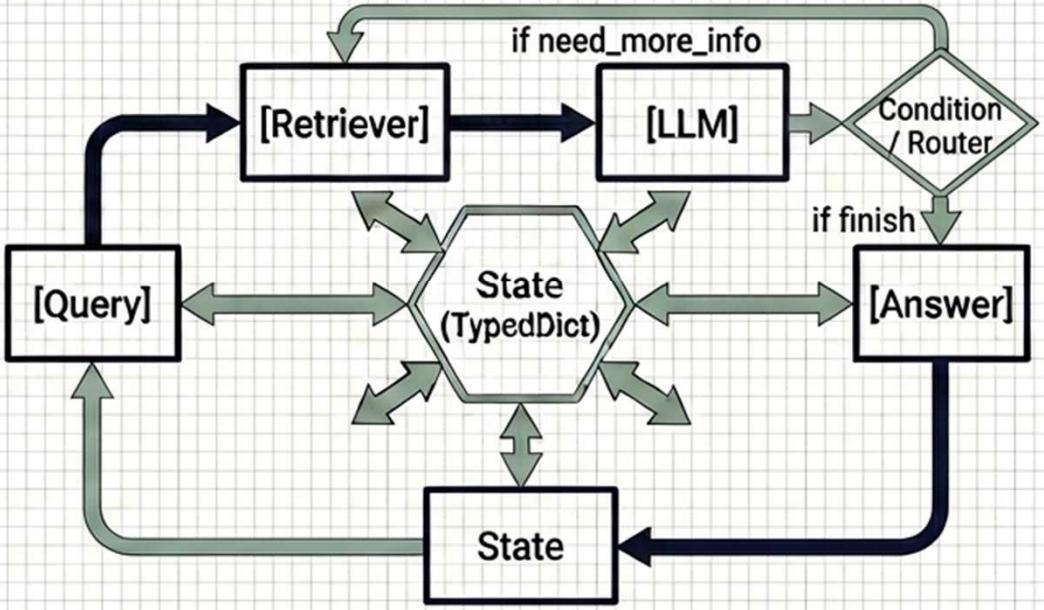
LangChain에서 LangGraph로: 순환형 State 구조의 등장

Pipe 문법의 한계



- 데이터 흐름 (Data Flow) 전용
- 조건 분기, 루프 불가능

StateGraph (제어 흐름 + 공유 상태)



- 다방향 제어 흐름 (Control Flow)
- 노드가 중앙 상태 버퍼를 읽기/쓰기로 공유
- 조건 분기, 루프 기능 추가

복잡한 Workflow 에서 Agent로: LLM + Tool + Loop 표준화

모델 능력의 임계점 돌파

GPT-4o와 Claude 3.5 이후의 모델들에서, Tool Calling 능력이 크게 향상되며, LLM에게 도구를 연결하고 자율적으로 사용하는 방식이 제대로 작동하기 시작함.

코딩 에이전트의 실전 검증

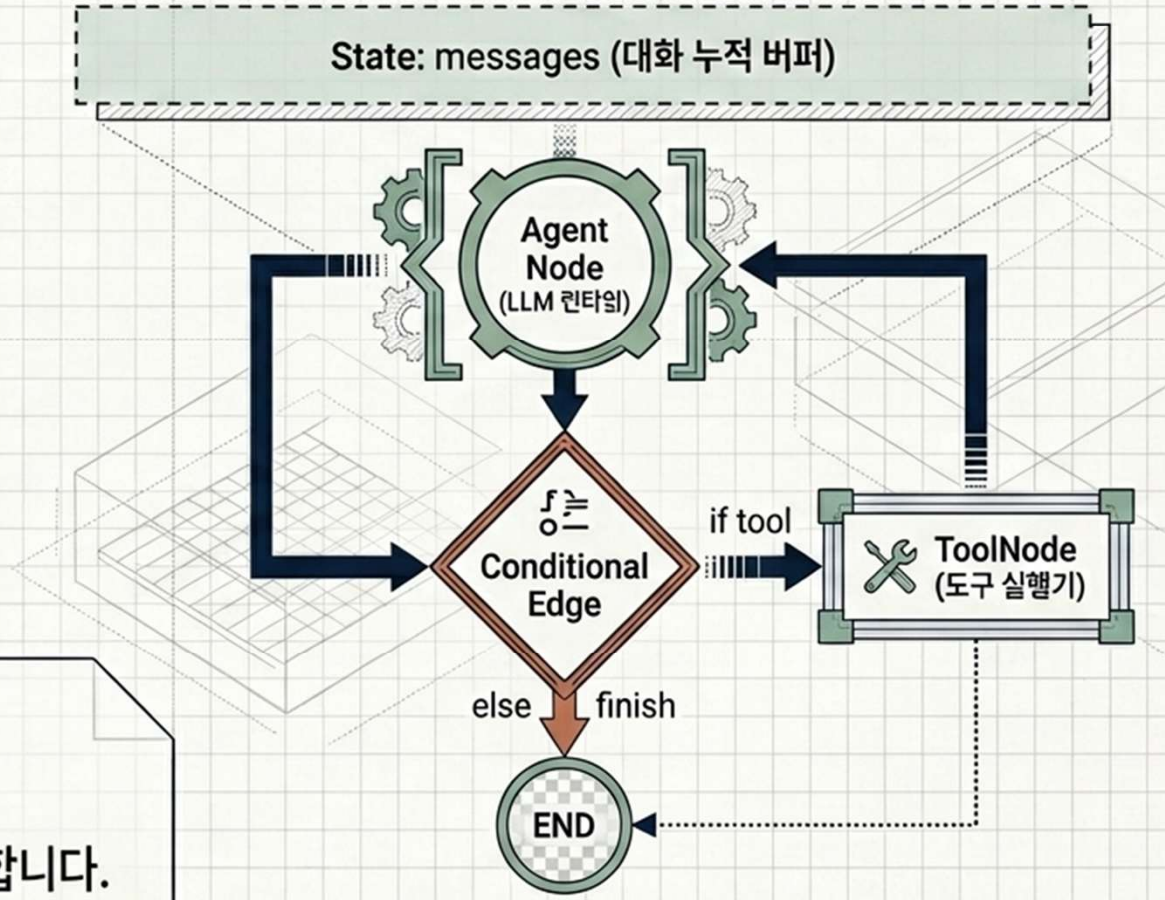
Claude Code, Cursor 등이 증명한 사실:
복잡한 워크플로우를 인간이 하드코딩하는 것보다,
풍부한 도구를 쥐어준 자율 루프가 압도적으로 효율적임.

StateGraph의 표현력은 무한하지만, 대다수의 실전 에이전트는 결국 이 강력한 단일 표준 루프(ReAct)로 수렴했습니다.

LangChain의 create_react_agent

```
create_react_agent(model, tools)
```

이 한 줄의 코드가 호출되는 순간...

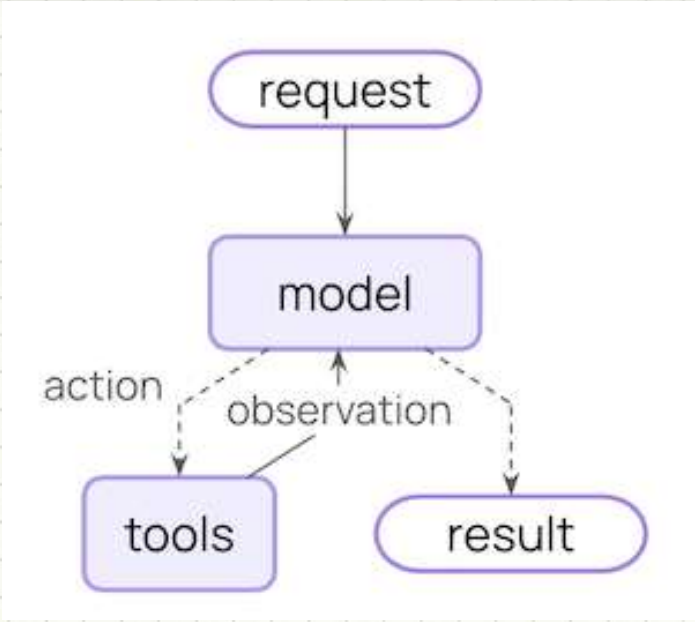


LangChain Agent의 역할

높은 추상화 패턴을 바탕으로,
LLM은 도구를 반복적으로 실행해 목표에 도달합니다.

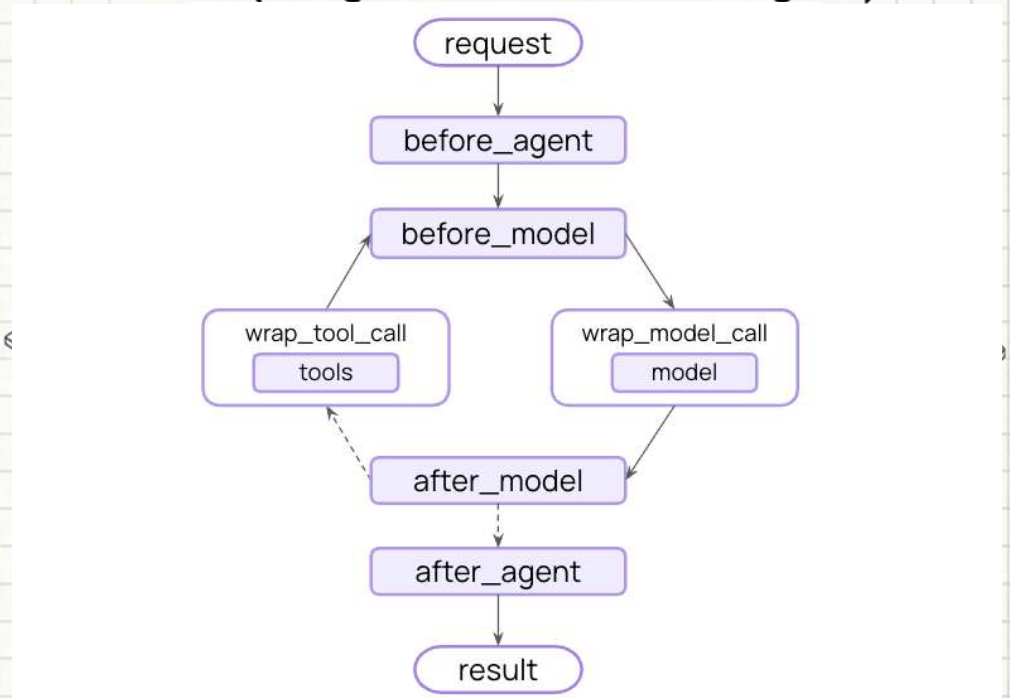
create_agent로의 재설계: 루프를 열어 젖힌 미들웨어 메인보드

Before (create_react_agent)



외부에서 개입할 틈이 없는 완전한 블랙박스.
코어 제어가 불가능함.

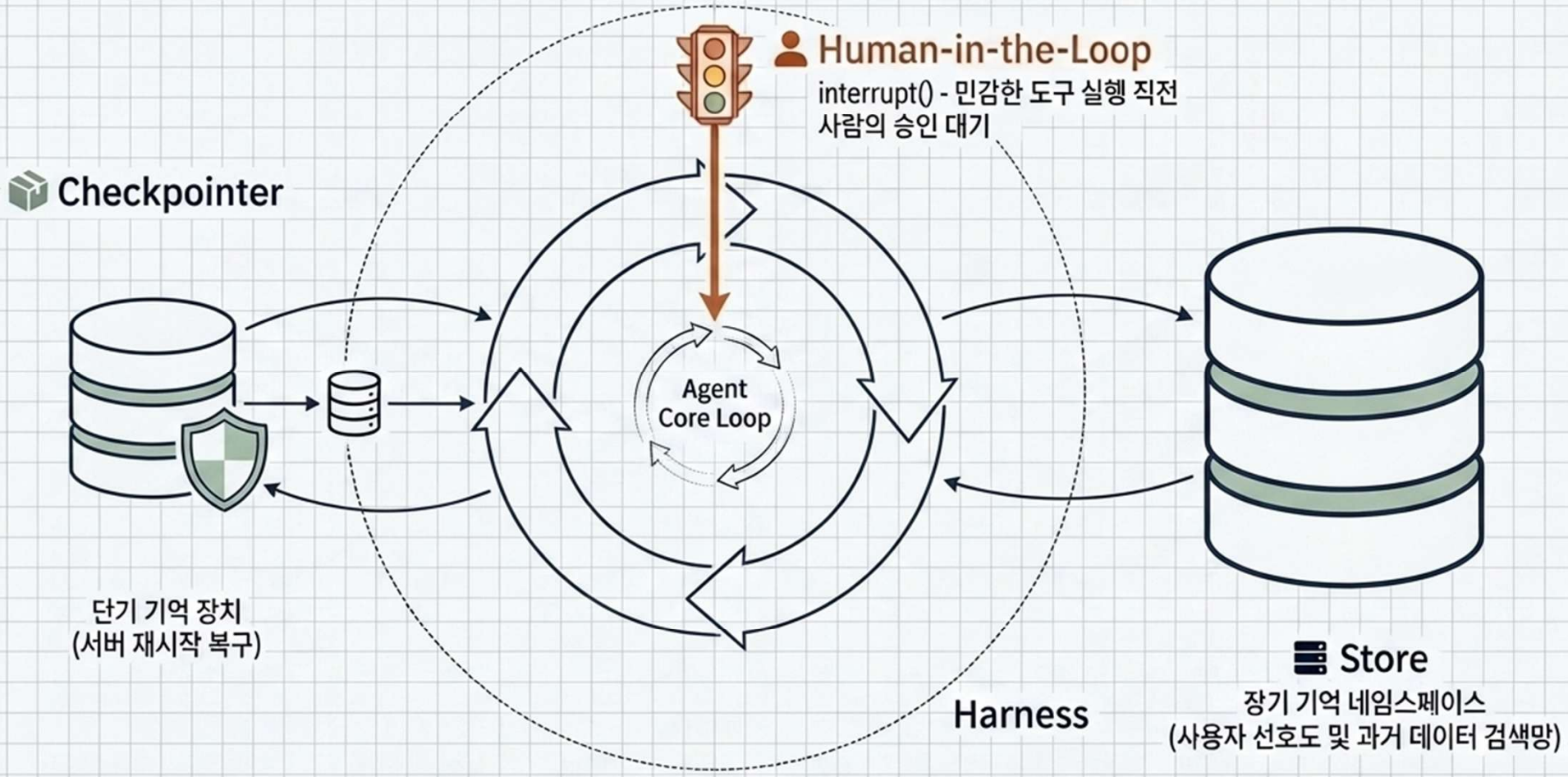
After (LangChain 1.0, create_agent)



에이전트 루프의 핵심 지점(시작, LLM 호출, 도구 실행 전후)에
커스텀 로직을 삽입할 수 있는 강력한 훅(Hook) 시스템으로 진화했습니다.

재설계의 진짜 이유는 바로 이 비어 있는 슬롯(Slot)들에 있습니다.

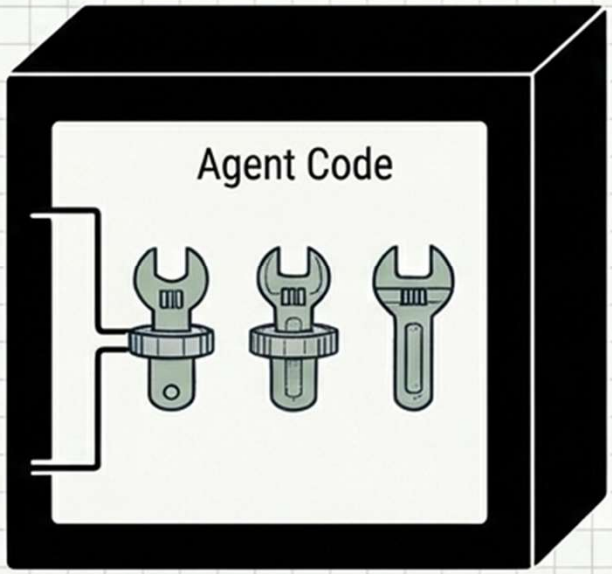
LangChain Memory와 HITL: 루프를 감싸는 하네스(Harness)의 등장



루프 자체가 에이전트가 아닙니다. 코어 루프를 둘러싼 방어적이고 영속적인 시스템 전체(Harness)가 비로소 실전용 에이전트입니다.

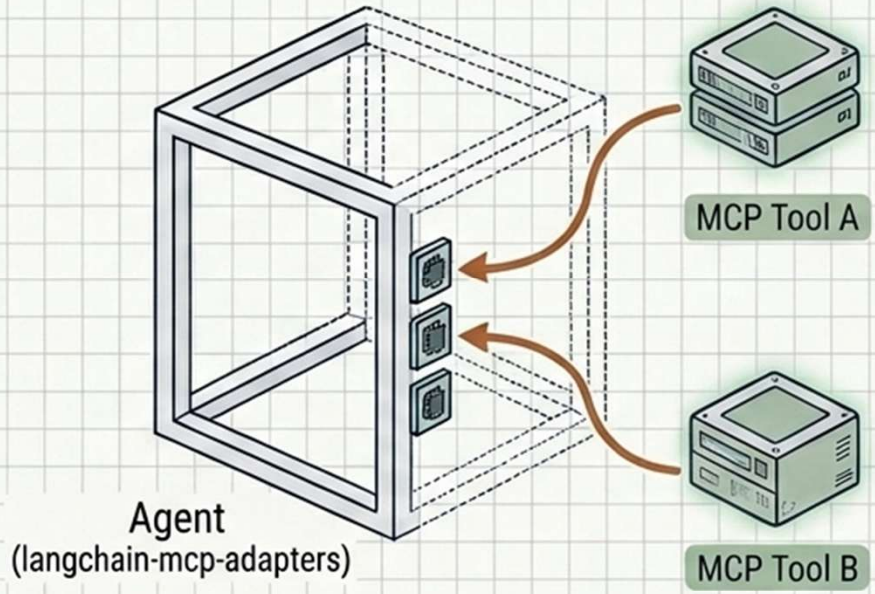
LangChain-mcp-adapters: Model Context Protocol 지원

과거: Hard-coded (용접된 부품)



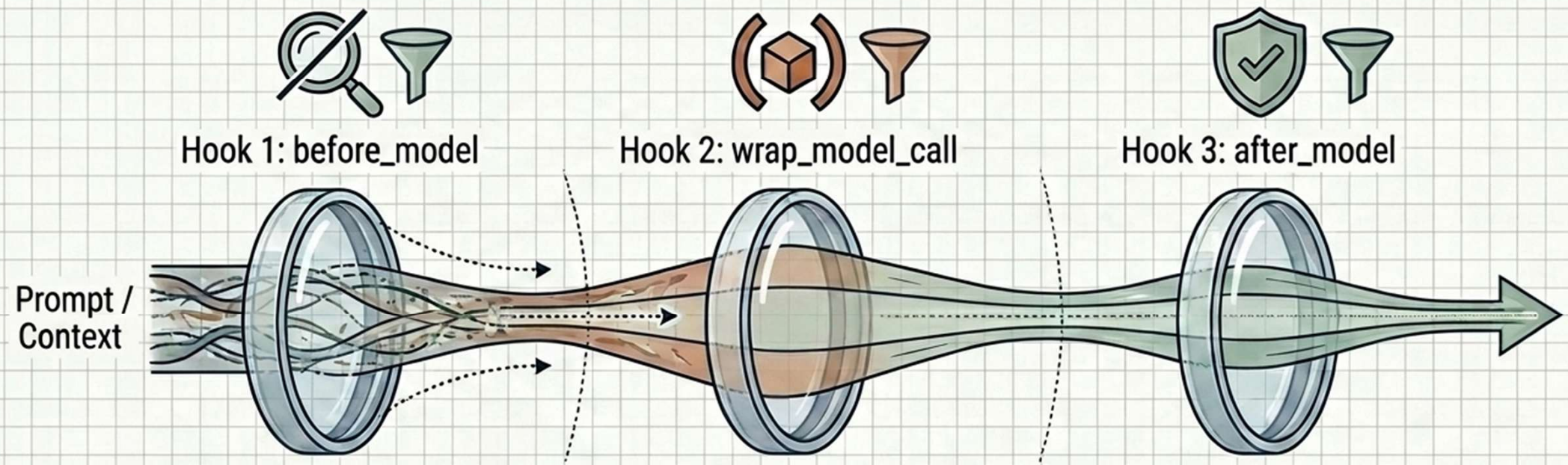
- 코드 안에 미리 바인딩된 스키마
- 새 도구 추가 시 코드 수정 및 시스템 재배포 필수

현재: Hot-swappable (플러그 앤 플레이)



- 런타임에 동적으로 발견(Discovery)되는 도구 스키마
- 도구는 이제 정적 코드가 아니라 런타임에 주입되는 데이터

미들웨어 렌즈: 정적 프롬프트에서 동적 '컨텍스트 엔지니어링'으로



- 개인정보(PII) 마스킹 필터링
- 오래된 대화 기록(History Trim) 압축
- 사용자 정보와 같은 동적 컨텍스트 주입
- 에러 발생 시 재시도(Retry) 적용
- 출력 결과 가드레일 통과 검증
- 위험도에 따른 HITL(사람 승인) 분기 대기

컨텍스트 엔지니어링(Context Engineering)의 재정의

프롬프트 엔지니어링이 1회성 정적 텍스트 호출이라면, 컨텍스트 엔지니어링은
모델이 보는 모든 정보를 매 스텝마다 동적으로 통제하고 재가공하는 런타임 프로세스입니다.

DeepAgents: 파편화된 최신 패턴들을 하나로 조립한 '오피니언드 스택'

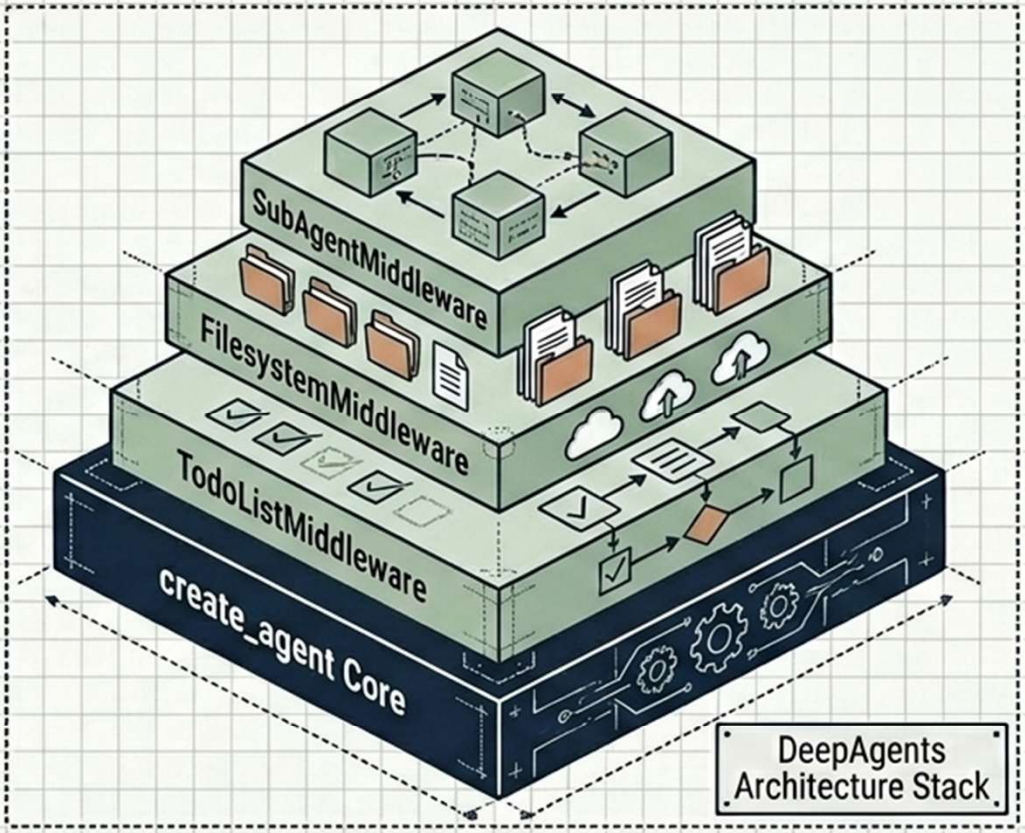
현대 에이전트의 4대 공통 요구사항

1. 상세한 System Prompt
(지시어 풀)

2. Planning Tool
(구체적 계획 수립 후 실행)

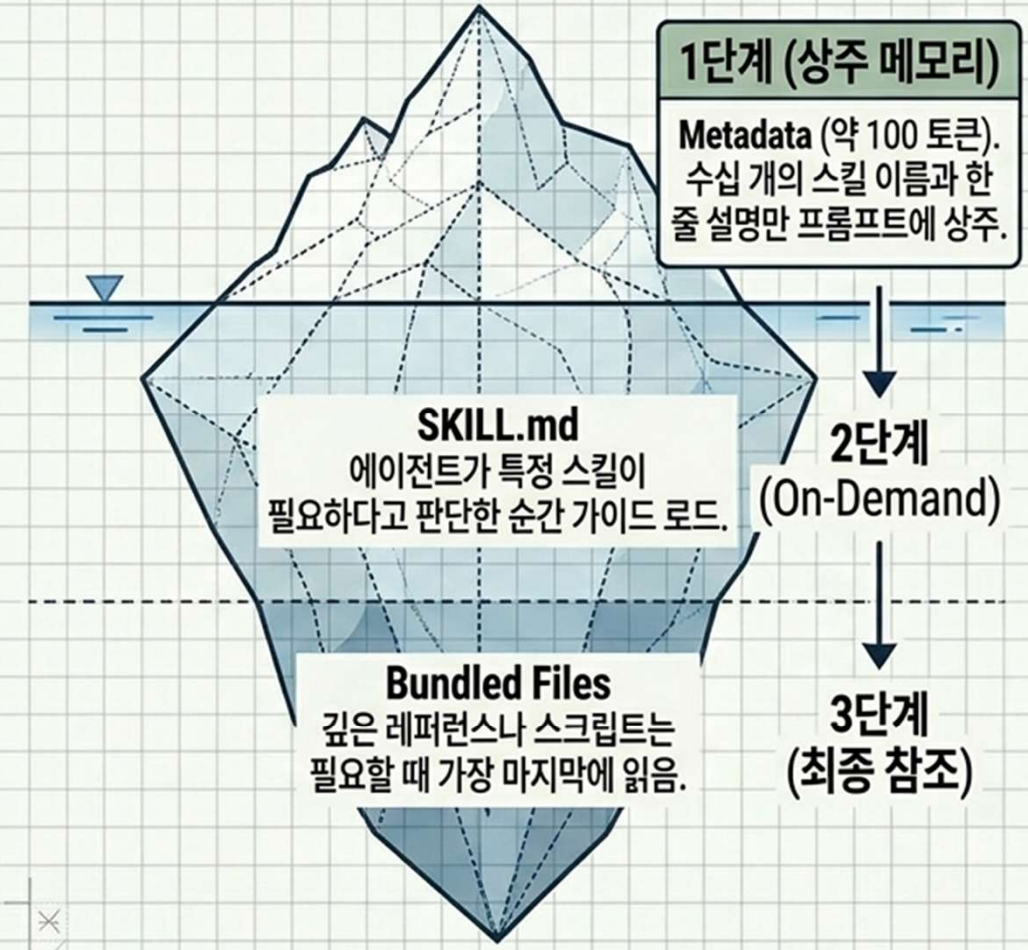
3. Filesystem
(거대 컨텍스트 파일 오프로드)

4. SubAgent
(하위 태스크의 Context 격리)



Claude Code, Manus 등에서 증명된 최고 수준의 패턴들을 바닥부터 짜지 마십시오.
DeepAgents는 한 줄의 코드(`create_deep_agent`)로 미리 정밀하게 조립된 최고급 하네스 스택을 즉각 제공합니다.

Skills: 컨텍스트 폭발을 막는 '단계적 지식 로딩 (Progressive Disclosure)'

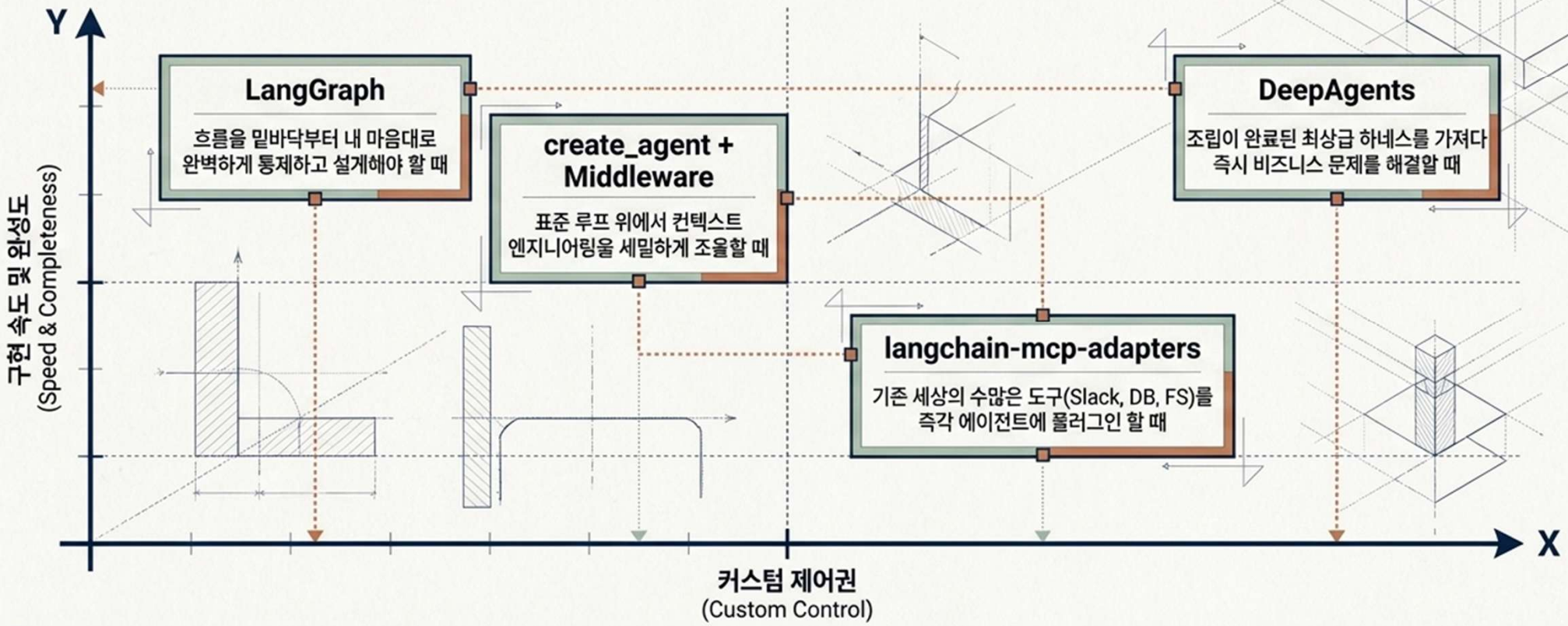


Tools (MCP)	Skills
• 실행(Action)의 영역	• 준비(Knowledge)의 영역
• 외부 시스템 API 연결	• 내부 절차적 지식 주입
• 런타임에 동적으로 호출되는 무기	• 점진적으로 뇌에 로드되는 전술서

+

경쟁 관계가 아닌 완벽한 보완 아키텍처

2026 개발자의 나침반 : 내 프로젝트에 무엇을 꺼내 써야 하는가



프레임워크의 진정한 가치

이 기술들은 특정 틀에 종속된 유행이 아닙니다. 에이전트 엔지니어링 전체가 수렴하고 있는 공용어(Lingua Franca)입니다.
LangChain 생태계는 이 2026년형 표준 인프라를 가장 명시적으로 학습하고 조립할 수 있는 나침반입니다.