

강사 소개

주식회사 토스트 대표

하동호

biz@project404.co.kr

010-5170-5136

프로젝트 404 · AI 교육 · AI 업무 자동화 SI · AI/AX 컨설팅

AI · LLM · AGENT 핵심 용어와 개념

용어를 외우는 강의가 아니라

AI가 실제 업무를 수행하게 되는 과정을 따라갑니다.

● ● ● agent.run()

```
goal = "내일 미팅 준비해줘"  
reason → 일정 확인이 필요하다  
action → calendar.list(date="+1d")  
observe ← 14:00 고객 미팅  
reason → 담당자 정보도 필요하다  
action → contacts.find(...)  
✓ final answer
```

LLM

Prompt

RAG

Tool / MCP

Agent

오늘의 큰 질문

**"LLM에 질문하는 것"과
"AI Agent가 실제 일을 수행하는 것" 사이에는
무엇이 추가되어야 하는가?**

Instructions

+

Context

+

Tools

+

Loop

+

Guardrails

PART 12에서 다시

여덟 개의 주제, 여덟 개의 질문

PART	주제	핵심 질문
01	AI 역사 / Agent의 등장	왜 지금 Agent인가?
02	AX	AI를 어디에 넣는 것이 AX인가?
03	LLM	모델은 실제로 무엇을 하는가?
04	Prompt / Fine-tuning	지시와 학습은 무엇이 다른가?
05	RAG 맞보기	왜 모든 문서를 Prompt에 넣으면 안 되는가?
06	Tool / MCP	AI는 어떻게 외부 세계와 연결되는가?
07	LangChain / LangGraph	Agent를 어떤 프레임워크로 만드는가?
08	전체 Agent 구조 / n8n	Workflow와 Agent는 어떻게 결합되는가?

※ RAG의 Chunking / Embedding / Vector DB / Retriever / Metadata / Reranker는 2일차에서 본격적으로 다룹니다.

강의 진행 순서

용어를 하나씩 외우는 것이 아니라 "AI가 실제 업무를 수행하게 되는 과정"을 따라간다.

01 AI 역사
Transformer → Agent 시대

02 AI Agent / AX
답변하는 AI와 일을 끝내는 AI

03 LLM
Token, Context Window, Temperature

04 CoT / ToT / ReAct
추론·탐색·도구 사용 패턴

05 Prompt
System / User / Assistant

06 Fine-tuning
Pre-training / Fine-tuning / Prompt

07 RAG 맛보기
필요한 자료만 찾아 넣기

08 Tool / Function Calling
모델이 외부 함수를 선택하는 방식

09 MCP
Tool-Data 연결을 표준화

10 LangChain / LangGraph
고수준 Agent vs Orchestration

11 전체 구조 + n8n
결국 Agent 시스템은 어떻게 연결되는가?

"여러분이 지금까지 AI를 쓸 때 했던 일은 Chatbot 사용이었나요, Copilot 사용이었나요, Agent 사용이었나요?"

A

Chatbot

물어보고 답을 받는다.
실행은 내가 한다.

B

Copilot

내 작업 공간 안에서
제안을 받는다.

C

Agent

목표를 받고 스스로
판단·실행·반복한다.

PART 01

AI 역사

중요한 것은 모델 버전 암기가 아니라, "인터페이스가 어떻게 바뀌었는가"이다.

Transformer



ChatGPT



멀티모달



AI IDE



Agent 시대

출처/참고: Transformer: Vaswani et al. (2017); ChatGPT 2022-11-30; GPT-4 2023-03-14; GPT-4o 2024-05-13; Claude 4 2025-05-22

"복붙해서 쓰는 AI" → "내 작업공간 안에 들어온 AI"

과거 · 사람이 Context를 운반

- 1 GPT 웹 페이지를 연다
- 2 코드를 복사한다
- 3 Prompt와 함께 붙여넣는다
- 4 답을 다시 IDE로 복사한다
- 5 오류가 나면 다시 붙여넣는다

AI는 프로젝트 전체를 보지 못하고, 파일을 수정하거나 테스트를 실행하지도 못한다.

AI IDE / CODING AGENT가 바꾼 것

AI IDE

현재 파일, 프로젝트 검색, 코드 수정 제안을 개발 환경 안에서 수행

Coding Agent

파일 읽기, 명령 실행, 테스트, 수정, 반복 — "작업 루프"를 수행

핵심 변화

사람의 복사/붙여넣기 횟수가 줄고, AI가 직접 Context와 Tool을 사용하기 시작

2025년 5월을 왜 하나의 분기점으로 설명할 수 있는가?

Anthropic은 2025년 5월 22일 Claude Sonnet 4와 Claude Opus 4를 공개하면서 코딩, 고급 추론, 장시간 **Agent workflow**를 핵심 메시지로 내세웠다.

"그 전 모델은 쓸 수 없었다"는 뜻은 아니다. Claude 3.7 Sonnet도 2025년 2월에 이미 hybrid reasoning과 tool use 개선을 제공했다. 다만 2025년 중반부터는 모델 회사들이 단순 Chat 성능보다 "**Tool을 쓰고 오래 일하는 Agent**"를 제품 메시지의 중심으로 올리기 시작했다.

2022 - 2024

대화형 AI와 Copilot 확산



2025 년 전후

Tool 사용과 장시간 작업을 수행하는
Agent 경험이 본격적으로 확산

출처/참고: Anthropic, Introducing Claude 4, 2025-05-22 · anthropic.com/news/claude-4

AI Agent란 무엇인가?

한 문장 정의

AI Agent는 목표를 받고, 현재 상황을 판단하고, 필요한 Tool을 선택·실행하고, 결과를 관찰한 뒤 다음 행동을 반복하여 목표를 달성하는 시스템이다.

Agent는 "답변을 잘하는 모델"이 아니라 "목표를 향해 행동을 반복하는 시스템"이다.

가장 단순한 AGENT LOOP



"다음 행동을 누가 결정합니까?"

개발자가 고정했다면

A → B → C

Workflow

모델이 상황을 보고 선택한다면

A / B / C

Agent 적 성격

실무 시스템은 대부분 둘을 섞는다. 고정되어야 할 부분은 Workflow로 두고, 판단이 필요한 지점만 Agent에게 맡긴다.

DX와 AX를 구분해서 보기

구분	DX · Digital Transformation	AX · AI Transformation
핵심	업무와 데이터를 디지털화	디지털 업무에 AI의 판단·생성·행동을 결합
대표 기술	ERP, CRM, SaaS, API, RPA	LLM, Agent, RAG, Tool Calling, AI Workflow
질문	"이 업무를 시스템으로 옮길 수 있는가?"	"이 업무 중 어떤 판단을 AI에게 맡길 수 있는가?"
예	종이 휴가신청 → 전자결재	AI가 잔여 연차 조회 → 규정 판단 → 신청 초안 → 승인 요청
목표	속도·표준화·데이터화	의사결정 자동화·지식 활용·업무 재설계

AI는 어디에 넣는 것이 좋은가

중요

AX라고 해서 모든 것을 Agent로 만들 필요는 없다. 단순 반복은 기존 코드/Workflow가 더 싸고 정확할 수 있다.

AI를 넣기 좋은 지점

규칙이 너무 많은 업무

자연어 · 비정형 데이터 이해가 필요한 업무

상황별 판단이 필요한 업무

모델 선택보다 어떤 판단을 AI에 맡길지 먼저 정의한다.

LLM의 가장 단순한 동작 원리

LLM은 데이터베이스에서 정답 문장을 꺼내는 시스템이 아니다. 현재까지 들어온 Token을 바탕으로 "다음 Token에 어떤 값이 올 확률이 높은가"를 계산하고, 이 과정을 반복해 출력을 만든다.

입력

"대한민국의 수도는"

다음 Token 후보 (개념 예시)

서울 0.94

부산 0.02

대한민국 0.01

기타 0.03

→ 하나를 선택 → 다시 다음 Token 예측 → 반복

서울



0.94

부산



0.02

대한민국



0.01

기타



0.03

여기서 생기는 중요한 결과

LLM의 목표는 "사실 DB에서 정답 조회"가 아니라 언어 생성이다. 그래서 모르는 것에도 그럴듯한 문장을 만들 수 있고, 이것이 Hallucination 문제로 이어진다.

GPT / Claude / Gemini는 무엇인가?

OPENAI

GPT / ChatGPT

범용 LLM · 멀티모달 · Tool/Agent API 생태계

Function Calling

Web/File Search

Agents

ANTHROPIC

Claude

코딩 · 문서 처리 · 긴 작업 흐름, Claude Code

Tool Use

Claude Code

MCP 생태계

GOOGLE

Gemini

멀티모달 · Google 생태계와의 결합

Function Calling

Cloud/Workspace

모델 순위는 시점·업무·평가 방법에 따라 바뀐다. "누가 무조건 최고인가"보다 각 공급자가 어떤 Tool/플랫폼 생태계를 제공하는지 함께 보는 것이 중요하다.

수강생에게 질문 — "같은 질문을 GPT, Claude, Gemini에 모두 던져본 적 있나요? 결과 차이가 모델 때문인지, 웹 서비스의 Tool/검색/지침 차이 때문인지 구분할 수 있나요?"

Token

Token은 모델이 텍스트를 처리할 때 사용하는 기본 조각이다. "단어 1개 = Token 1개"가 아니다. 언어와 tokenizer에 따라 하나의 단어가 여러 조각으로 나뉠 수도 있고, 짧은 기호가 하나의 Token이 될 수도 있다.

같은 문장, 다른 조각 수 (개념 예시)

"Agent" → [Ag][ent]

"에이전트" → [에이][전][트]

"{" → [{]

조각내기(Tokenize) ≠ 의미 비교(Embedding)

키친 / 치킨 → 글자·발음이 비슷

치킨 / 닭 → 의미가 비슷

핵심 포인트

Tokenization은 텍스트를 조각내는 문제이고, 의미가 비슷한지를 찾는 것은 Embedding 같은 별도 개념이다.

글자·발음의 축

의미의 축

Embedding은 2일차

Context Window = 모델의 작업 책상

한 번의 호출에 올릴 수 있는 것

System / Developer Instructions

사용자 질문

이전 대화

첨부 문서 / 이미지의 처리 결과

RAG 검색 결과

Tool 실행 결과

코드 / 로그

모델이 생성한 출력

책상이 크다고 모든 자료를 올려놓는 것이 좋은 것은 아니다.

대화가 너무 길어지면?

제품/Agent는 오래된 내용을 일부 제외하거나 **요약·압축(compaction)**해 Context 한계 안에 맞출 수 있다.

요약 과정에서는 세부 문구, 숫자, 예외조건 등 일부 정보가 덜 보존될 수 있다.

따라서 "중요한 지식은 매번 대화 전체에 의존하지 말고, 필요할 때 다시 가져 오는 구조"가 필요해진다.

→ PART 08 RAG로 이어짐

Temperature / Hallucination / Multimodal

개념	쉽게 말하면	주의점
Temperature	출력 다양성을 조절하는 샘플링 설정	모든 최신 모델/모드에서 개발자가 자유롭게 쓰는 것은 아님
Hallucination	근거 없는 내용을 그럴듯하게 생성하는 현상	RAG/Tool을 써도 0이 되는 것은 아님
Multimodal	Text 외에 Image/Audio/Video 등 여러 입력을 함께 이해	"OCR만 하는 것"보다 문서 구조의 미까지 해석 가능

Temperature 낮게

정형 추출, 분류, 반복 가능한 업무 — 더 보수적으로

Temperature 높게

아이디어, 카피, 스토리 — 더 다양한 후보를 허용

HALLUCINATION을 줄이는 방향

좋은 Prompt

+ 외부 근거 (RAG)

+ 실시간 Tool

+ Structured Output

+ Validation / Eval

+ Human Approval

웹에서 쓰는 ChatGPT/Claude와 API 호출은 왜 다르게 느껴질까?

웹 제품

대화 관리

파일 처리

검색 / 도구

서비스 지침

메모리

모델 호출

직접 API를 쓸 때

이 레이어를 직접 설계해야 한다

모델 호출

"같은 모델을 API로 불렀는데 웹에서보다 못한다"는 느낌은 모델 자체보다 주변 **Harness 차이**에서 생기는 경우도 많다.

정확한 표현 — 웹 제품 내부 구현은 공급자가 계속 바꾸므로 "항상 OCR을 먼저 한다"처럼 단정하지 않는다. 강의에서는 "파일/이미지를 모델이 이해할 수 있도록 제품 레이어에서 처리·구성해 줄 수 있다" 정도로 설명한다.

OCR은 글자를 뽑고, 멀티모달 LLM은 구조를 해석한다

입력 · 스캔된 거래명세서 이미지

거래명세서

합계

멀티모달 LLM의 출력

```
{
  "doc_type": "거래명세서",
  "supplier": "쌍용건설(가상)",
  "issued_at": "2026-03-14",
  "items": [
    { "name": "철근 D13", "qty": 120, "unit": "EA" },
    { "name": "레이콘", "qty": 18, "unit": "m3" }
  ],
  "total": 12480000,
  "confidence": 0.92
}
```

✓ 문서 종류 · 필드 관계 · 표 구조 · 의미까지 해석

CoT · ToT · ReAct

이 셋을 "Agent Framework"라고 외우기보다 문제 해결 패턴으로 이해한다.

5-1

Chain of Thought

복잡한 문제를 한 번에 결론내기보다 중간 단계로 나누어 추론한다.

문제

- 조건 정리
- 중간 계산
- 검토
- 결론

5-2

Tree of Thought

한 경로만 밀어붙이지 않고 여러 후보 접근을 만들고 평가하여 더 나은 경로를 선택한다.

후보 A

후보 B ✓ 선택

후보 C

5-3

ReAct

Reason + Act. 판단 → 행동(Tool) → 관찰을 반복한다. Agent와 가장 직접적으로 연결된다.

Reason

Act

Observation

↺ 반복

현대 Reasoning 모델에서는 내부 추론이 모델 안에서 수행될 수 있다. 실무 Prompt에서는 내부 생각을 길게 공개하도록 요구하기보다 "단계별로 검토하고, 최종 결론과 검증 가능한 근거를 제시하라"처럼 결과 중심으로 지시하는 편이 좋다.

판단 → 행동 → 관찰의 반복

Reason: 일정 확인이 필요하다

Action: Calendar Tool 호출

Observation: 14:00 고객 미팅 발견

Reason: 담당자 정보도 필요하다

Action: Contacts Tool 호출

Observation: 담당자 정보 발견

→ Final Answer

R 무엇이 더 필요한지 판단

A Tool을 골라 실행

O 결과를 관찰하고 다시 판단

09장 Agent Loop와 같은 그림

System / User / Assistant 역할

Prompt는 단순한 "질문"이 아니라 모델에게 전달되는 지시·맥락·입력의 설계다.

ROLE	역할	예시
System / Developer	애플리케이션이 모델에 주는 상위 지침	"너는 HR 상담 Agent다. 사내 규정 밖의 내용은 추측하지 마라."
User	실제 사용자의 요청	"내 연차 몇 개 남았어?"
Assistant	모델이 이전에 생성한 응답 / 대화 맥락	"직원번호를 확인하겠습니다."

OpenAI 최신 API 참고 — 최근 OpenAI API에서는 developer 역할이 애플리케이션 지침에 사용되는 경우가 많다. 강의에서는 개념적으로 "상위 지침(System/Developer) → User 요청 → Assistant 응답" 계층을 잡아주면 된다.

좋은 Prompt의 구성

[Role]

너는 영업팀 AI 비서다.

[Goal]

고객 미팅 준비 정보를 정리한다.

[Context]

시간대는 KST 다.

[Tools]

Calendar, Contacts

[Rules]

데이터에 없는 정보는 추측하지 않는다.

[Output]

시간 / 고객사 / 담당자 / 장소 / 준비사항

요소

질문

Role

누구처럼 행동해야 하는가?

Goal

무엇을 끝내야 하는가?

Context

판단에 필요한 배경은 무엇인가?

Rules

하면 안 되는 것은 무엇인가?

Tools

어떤 외부 기능을 쓸 수 있는가?

Output

어떤 형태로 결과를 줘야 하는가?

Completion Criteria

언제 "완료"라고 볼 것인가?

수강생에게 질문 — Prompt가 길면 좋은가? → 아니다. "지금 판단에 필요한 정보"가 명확한 Prompt가 좋다. 이 지점에서 Context Engineering으로 이어진다.

Pre-training / Fine-tuning / Prompt

Prompt로 지시하는 것과 모델의 Weight를 바꾸는 것은 완전히 다른 층위의 작업이다.

Pre-training

모델의 기본 Weight 대규모 학습

인터넷·책·코드 등 대규모 corpus로 기반 모델 학습

매우 큼

Fine-tuning

기존 Weight를 특정 방향으로 추가 최적화

상담 문체, 분류 패턴, 특정 작업 행동

중 ~ 큼

Prompt

Weight는 그대로, 이번 호출의 Context만 변경

"JSON으로 답하라", 역할·규칙 지정

작음

Pre-training — 모델 자체를 "처음부터" 학습

Fine-tuning — 이미 학습된 모델의 Weight를 추가 조정

Prompt — Weight는 그대로, 입력 Context로 행동을 유도

왜 Fine-tuning은 단순 Prompt 수정과 다른가?

Fine-tuning은 모델 내부 파라미터(Weight)의 최적화 방향을 바꾼다. 즉 이번 대화 한 번의 지시가 아니라, 이후 다양한 입력에 대한 모델의 행동 경향 자체를 조정하려는 것이다.

제대로 하려면 필요한 일

학습 목적 정의

고품질 예제 데이터 수집·정제

Train / Validation 분리

학습 실행 및 비용 관리

기존 능력 저하 여부 확인

Eval과 회귀 테스트

모델 버전 관리 / 재학습 전략

뉘앙스

"Fine-tuning은 무조건 초거대 프로젝트"라고 단정할 필요는 없다.

LoRA/PEFT나 Managed SFT로 기술적 비용은 낮아질 수 있다. 하지만 데이터 설계와 평가까지 포함하면 Prompt 몇 줄 수정과는 비교할 수 없는 프로젝트가 된다.

구분 포인트

자주 바뀌는 사실 지식이나 사내 문서는 모델 자체를 다시 학습시키기보다 RAG로 외부 지식을 연결하는 편이 적합한 경우가 많다.

Prompt에 모든 문서를 넣으면 안 되는 이유

매 질문마다 들어나는 **CONTEXT**

프로젝트 파일

+ 로그

+ 대화 기록

+ 지침

+ Tool 결과 → 비용·지연 증가

모든 사내 문서를 매 질문마다 넣으면 입력 Token이 계속 커진다.

모델의 Context Window에는 한계가 있다.

긴 대화를 유지하는 제품/Agent는 일부 내용을 요약 압축하거나 이전 내용을 덜 포함하는 전략을 사용할 수 있다.

그 과정에서 "예외조항, 정확한 숫자, 특정 문장"이 희미해질 수 있다.

모든 정보를 모델에 외우게 하는 것이 아니라, 필요한 순간에 관련 자료를 검색해 Context로 제공한다.

RAG = Retrieval-Augmented Generation

답변하기 전에 외부 지식에서 관련 자료를 찾아서, 그 결과를 LLM의 Context에 추가해 답변하게 하는 방식.



오늘 이해할 것

왜 모든 문서를 Prompt에 넣으면 안 되는가

왜 필요한 자료를 다시 찾아야 하는가

RAG는 "검색 + 생성"이라는 것

RAG가 Agent의 지식 도구가 될 수 있다는 것

2일차에 배울 것

Parsing / Chunking

Embedding / Vector DB

Retriever / Top-K / Metadata

Hybrid Search / Reranker / 평가

"그럼 AI가 수천 개 문서 중에서 '필요한 부분'을 어떻게 찾을까요? 내일은 그 검색 시스템을 직접 만들어봅시다."

Function Calling의 본질

개발자가 사용할 수 있는 함수/도구의 이름, 설명, 입력 Schema를 모델에게 알려준다. 모델은 사용자 요청을 보고 필요한 도구와 인자를 구조화해서 반환한다.

```
# Tool 정의 (개념)
{
  "name": "get_weather",
  "description": "도시의 날씨를 조회",
  "parameters": {
    "city": "string",
    "date": "string"
  }
}
```

사용자: "내일 서울 날씨 알려줘"



모델: `get_weather(city="Seoul", date="...")`

GPT, Claude, Gemini 모두 이와 유사한 Tool/Function Calling 기능을 제공한다. 명칭과 API 형식은 조금씩 다르다.

수업에서 반드시 짚을 것 — 모델이 함수 코드를 "직접 실행"한다기보다, 일반적으로는 어떤 함수와 인자를 쓸지 결정하고, 개발자 애플리케이션이 실제 함수를 실행한 후 Tool Result를 다시 모델에 전달한다.

Tool Calling과 Agent의 관계

TOOL CALLING만 있음



AGENT LOOP



Function Calling은 Agent가 행동하기 위한 핵심 부품이지만, Function Calling 하나가 곧 Agent 전체는 아니다.

수강생에게 질문 — "Calendar 조회 함수를 모델이 한 번 호출했다. 이것은 Agent인가?" → 상황에 따라 Tool-enabled LLM일 수 있다. 여러 단계의 목표, 상태, 반복 판단까지 갖추면 Agent적 성격이 훨씬 강해진다.

API를 비개발자에게 3분 만에 설명하기

한 프로그램이 다른 프로그램에 "이 기능을 실행해줘"라고 요청하고 결과를 받는 공식적인 창구.



MCP가 필요한 이유 — API는 프로그램끼리 통신하는 약속이다. Agent가 여러 서비스와 연결될수록 서로 다른 연결 방식을 표준화할 필요가 커진다.

MCP • Model Context Protocol

AI 애플리케이션이 외부 Tool과 Context를 표준 방식으로 발견하고 사용할 수 있게 하는 프로토콜.

MCP SERVER가 대표적으로 노출하는 것

01 Tools

모델이 실행할 수 있는 기능

02 Resources

파일, DB Schema 등 Context 데이터

03 Prompts

재사용 가능한 Prompt Template

비개발자용 한 문장 — "MCP는 API 자체를 없애는 기술이 아니라, AI가 여러 API·도구·데이터를 비슷한 방식으로 연결해서 쓰도록 중간 규격을 만들어주는 장치"

MCP Server를 붙이는 설정 (개념 예시)

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "server-github"],
      "env": { "GITHUB_TOKEN": "****" }
    }
  }
}
```

연결되면 Host가 Tool 목록을 발견한다

✓ `list_issues` • `create_pull_request` • `search_code`

출처: modelcontextprotocol.io/specification

Function Calling vs MCP

구분	Function / Tool Calling	MCP
핵심 질문	모델이 어떤 Tool을 호출할까?	Tool/Data를 어떤 표준 방식으로 연결·노출할까?
범위	모델 ↔ 애플리케이션 내부 Tool 선택	Host/Client ↔ MCP Server 연결 규약
예	<code>get_calendar_events()</code> 호출 결정	GitHub MCP Server에서 issue tool 발견·호출
관계	Agent 행동의 실행 방식	Agent가 Tool을 얻는 연결/표준화 방식

Function Calling = AI가 사용할 기능을 선택·호출하는 방식

MCP = 외부 Tool과 Data를 표준 방식으로 연결하기 위한 프로토콜

LangChain

서로 다른 모델 공급자와 Tool을 공통 인터페이스로 다루고, Agent를 빠르게 구성할 수 있도록 하는 **상위 레벨 프레임워크**.

```
from langchain.agents import create_agent

agent = create_agent(
    model="openai:...",
    tools=[get_weather, search_customer],
    system_prompt="너는 영업팀 AI 비서다."
)
```

Model을 바꾸기 쉬운 표준 인터페이스

Python 함수 등을 Tool로 연결

create_agent()로 기본 Agent Loop 생성

Middleware, Structured Output, Memory 등 확장

LangGraph

State, Node, Edge를 직접 정의해 장시간·상태 기반 Agent Workflow를 세밀하게 설계하는 **저수준 Orchestration Runtime**.



현재 LangChain v1의 create_agent는 내부적으로 LangGraph Runtime 위에서 동작한다. 따라서 처음에는 LangChain으로 시작하고, 분기·상태·Human-in-the-loop·장시간 실행이 복잡해지면 LangGraph 수준으로 내려가는 설명이 이해하기 쉽다.

선택 기준

Tool 몇 개를 사용하는 일반 Agent
→ **LangChain부터 검토**

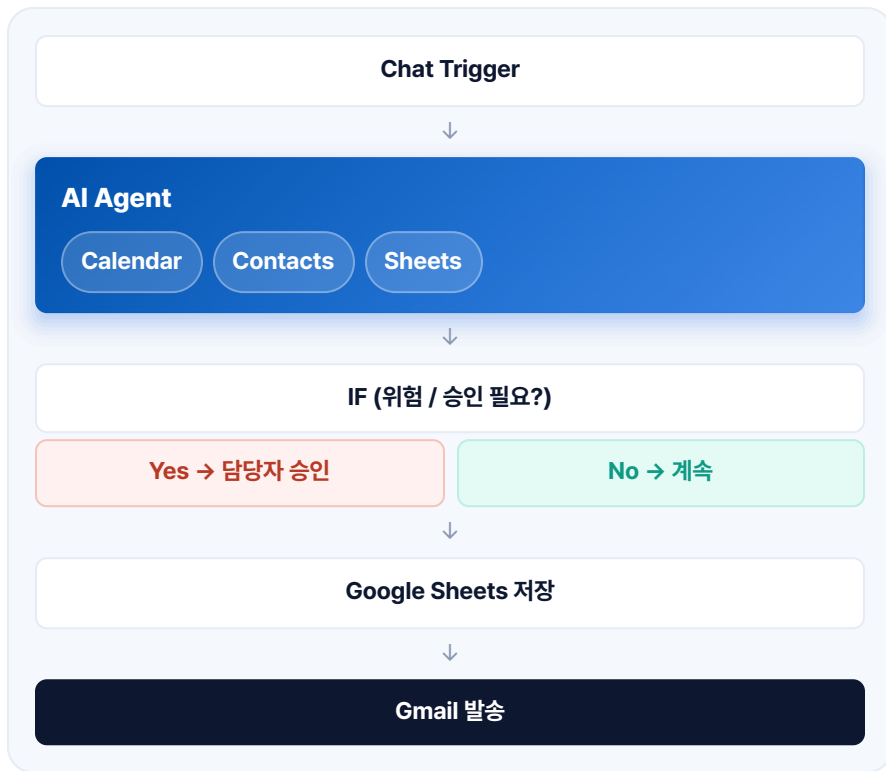
상태·분기·재시도·승인·여러 Agent를 세밀하게 설계
→ **LangGraph 검토**

Agent를 구성하는 요소

요소	역할	오늘 배운 연결
Model	판단 / 생성 엔진	GPT / Claude / Gemini
Instructions	행동 규칙	Prompt, System/Developer
Context	이번 판단에 제공할 정보	대화, RAG 결과, Tool 결과
Tools	외부 행동	Function Calling, MCP
Memory / State	과거 / 현재 작업 상태	대화 기억, Workflow 상태
Loop	판단-행동-관찰 반복	ReAct
Orchestration	순서 · 분기 · 여러 Agent 관리	LangGraph, n8n
Guardrail	실수 / 위험 제한	승인, 권한, 검증

Agent = Model + Instructions + Context + Tools + Memory/State + Loop + Orchestration + Guardrails

n8n에서 어디까지가 Workflow이고 어디부터 Agent인가?



바깥 노드

순서와 조건을 개발자가 설계한 **deterministic workflow**

AI AGENT 노드 내부

모델이 현재 요청에 따라 어떤 Tool을 몇 번 쓸지 판단

수강생에게 질문 — "그럼 n8n 자체가 Agent인가요?" → n8n은 Workflow/Orchestration 플랫폼이다. 그 안에 Agent Node와 LLM/Tool을 넣으면 Agentic Workflow를 만들 수 있다.

Agent 관점에서 GPT와 Claude 다시 비교

성능 순위가 아니라 "실습에서 어떤 경험을 만들기 쉬운가"로 비교한다. 모델 버전과 제품 기능은 계속 바뀐다.

관점	GPT / OpenAI 계열	Claude / Anthropic 계열
Agent API / Tool 생태계	Function calling, built-in tools, Responses/Agents 계열 API가 한 플랫폼에 잘 모여 있음	Tool use 지원, Claude Code와 개발자 Agent 경험이 강함
Coding Agent 체감	범용 Tool/서비스 Agent 구현이 편한 편	코드 읽기·수정·긴 개발 작업에서 강점을 느끼는 개발자가 많음
MCP	MCP 연결 기능을 지원하는 방향으로 확장	MCP를 처음 공개한 Anthropic 생태계와 Claude Code에서 특히 자연스럽게 사용
수업 선택	Tool calling과 Agent 구조 설명용 메인 모델로 쓰기 편함	Coding Agent / Context / MCP 데모에 매우 적합

수강생에게 질문 — "어떤 모델이 최고인가?"가 아니라 "이 업무에는 어떤 모델 + 어떤 Tool + 어떤 Orchestration 조합이 가장 안정적인가?"가 더 좋은 질문이다.

오늘의 전체 연결



그리고 규모가 커지면

복잡한 상태 / 분기

→ **LangGraph**

빠른 Agent 구성

→ **LangChain**

전체 업무 흐름

→ **n8n / Workflow Orchestration**

용어 한 문장

LLM 언어 패턴을 학습한 판단·생성 엔진

Prompt 이번 작업의 지시와 Context

Fine-tuning 모델 Weight의 행동 경향을 추가 학습으로 조정

RAG 필요한 외부 지식을 검색해 Context에 넣는 방식

Tool Calling 모델이 실행할 기능과 인자를 선택하는 방식

MCP AI와 외부 Tool/Data 연결을 표준화하는 프로토콜

Agent 목표를 위해 판단-행동-관찰을 반복하는 시스템

LangChain Agent를 빠르게 구성하는 상위 레벨 프레임워크

LangGraph 상태·분기·반복을 세밀하게 제어하는 Runtime

n8n 고정 Workflow와 Agent를 함께 묶기 좋은 Orchestration 플랫폼

마지막 확인 질문 10개

1 LLM이 "정답 DB"가 아니라는 말은 무슨 뜻인가?

2 Token과 의미(Embedding)는 왜 다른 개념인가?

3 Context Window가 무한하지 않다는 것이 시스템 설계에 어떤 영향을 주는가?

4 Prompt와 Fine-tuning의 가장 근본적인 차이는 무엇인가?

5 RAG는 왜 모든 문서를 매번 넣는 것보다 유리할 수 있는가?

6 Function Calling에서 실제 함수를 실행하는 주체는 누구인가?

7 MCP와 API는 어떤 관계인가?

8 Tool Calling이 있다고 무조건 Agent인가?

9 LangChain과 LangGraph는 어떤 기준으로 선택할 수 있는가?

10 n8n Workflow 안에서 Agent를 어디에 배치하면 좋은가?

다음 시간 예고

RAG 데이터 준비 → Parsing → Chunking → Embedding → Vector DB → Retrieval → Metadata → Top-K → Hybrid Search → Reranking → 품질 확인