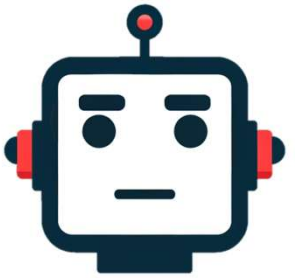
RAG와 AI Agent의 이해

LLM과 Agent

훌륭하지만... 언어 모델일 뿐이잖아요?

LLM의 구조적 한계

- 다량의 언어 데이터 학습을 바탕으로, 언어 패턴과 지식을 습득
- 구조적으로는 자연어 입력→ 자연어 출력이 전부

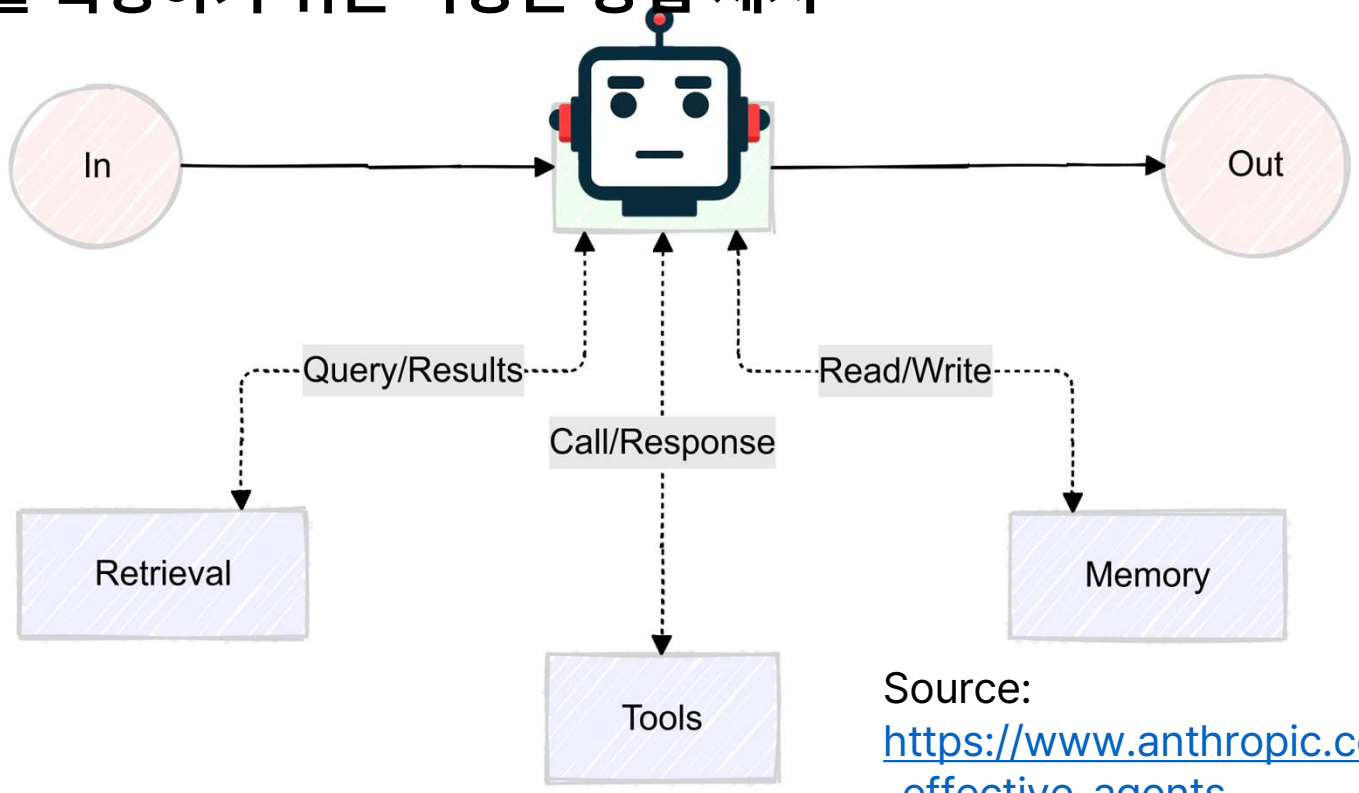


LLM의 능력은?

- 텍스트 형식의 응답을 생성한다
- 학습하지 않은 지식에 대해서는 부정확할 수 있다
- 대화가 끝나면 Context를 잊어버린다

LLM의 한계 극복 3요소: Retrieval, Tools, Memory

LLM의 외연을 확장하기 위한 다양한 방법 제시

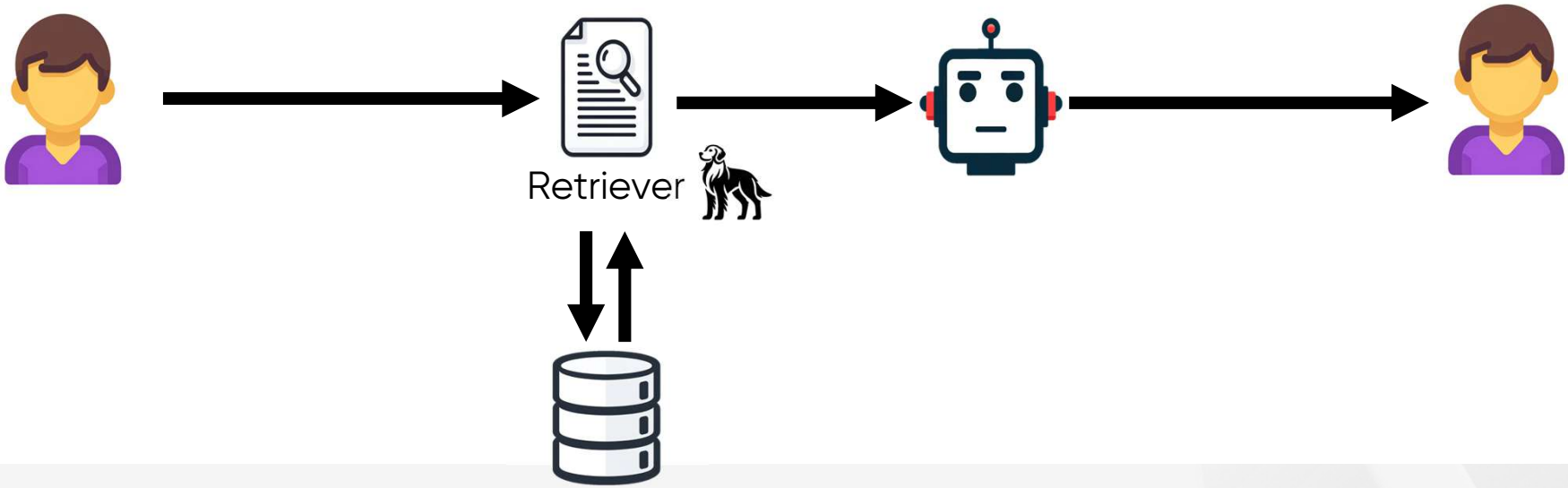


Source: https://www.anthropic.com/research/building-effective-agents

1) Retrieval: 검색 기능 추가하기

데이터베이스 검색 결과를 프롬프트에 추가하여 활용하자 → RAG

- 학습은 되지 않으나, Context의 내용을 활용하여 답변
- 할루시네이션을 최소화하고, 모르는 질문에 대한 답변도 가능



2) Tool: LLM에게 도구 들려주기

LLM이 사용할 수 있는 외부 함수나 모듈

- 검색, 파이썬 코드 실행, API 호출 등
- 함수(Function)로 정의하기도 함

대표적 Tool

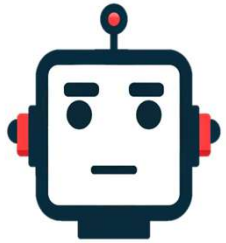
- SerpAPI, Tavily: 웹 검색 기능
- Code Interpreter: 파이썬 코드 실행 기능
- Slack, Github, Gmail, ...

2) Tool: LLM에게 도구 들려주기

LLM이 제어를 할 수 있을까?

- 언어 모델이므로 제어는 불가능하다.
- 그러나, 제어 명령은 내릴 수 있다.

User: 요즘 새로 개봉한 영화 있나요?



Call {Tool: 검색, Query: 영화}

2) Tool: LLM에게 도구 들려주기

Tool**이 탑재된** LLM**의 판단**

• Tool이 필요한 경우, Tool 실행을 요청

→ 프로그램 상에서, Tool 요청을 파싱하여 실제 실행 가능

Tool Message: **툴 실행 결과**

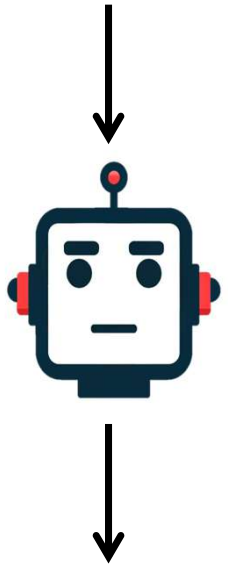
• 툴 실행 결과를 LLM에게 전달

• LLM은 답변을 생성

User: 요즘 새로 개봉한 영화 있나요?

LLM: Call {Tool: 검색, Query: 영화}

Tool: (영화 검색 결과)



네! 새로 개봉한 영화는 …

Tool Calling 예시: 웹 검색 증강 생성

LLM의 시스템 프롬프트 설정

- “너는 뉴스 검색을 할 수 있어.”
- “실행 방법은 get_news(query) 야.”
- “query는 문자열이고,
  세 단어 이하로 작성해.”

```python
from pydantic import BaseModel, Field

class get_news(BaseModel):
    """query를 이용하여 뉴스 검색"""
    query: str= Field(
        description="검색어(3단어 이하)")

tools = [openai.pydantic_function_tool(get_news)]
```

Tool Calling 예시: 웹 검색 증강 생성

Tool의 Schema 형식

- 툴의 사용법과 형식에 대한 정보
- 시스템 메시지에 포함

```
{
    'type': 'function',
    'function': {
        'name': 'get_news',
        'strict': True,
        'parameters': {
            'description': 'query를 이용하여 뉴스 검색',
            'properties': {'query': {'description': '검색어(3단어
            'required': ['query'],
            'title': 'get_news',
            'type': 'object',
            'additionalProperties': False
        },
        'description': 'query를 이용하여 뉴스 검색'
    }
}
```

Tool Calling 예시: 웹 검색 증강 생성

Prompt, Tool Call, Tool Msg, Answer

- 유저 메시지: “오픈AI 뉴스 검색 결과 요약해줘.”
- LLM 출력:

  <FUNCTION CALL> {function: get_news, argument: {query:오픈AI}} <END OF TEXT>
- Python 레벨: 파싱하여 get_news(query=오픈AI) 실행 → LLM에게 전달
- LLM 출력: (오픈AI 검색 결과 요약)

Tool Calling의 효과

LLM의 출력이 실제 행동으로 연결된다면?

- 직접 작업을 지시하고, 제어할 수 있는 능력
- 기존의 지능형 판단이 필요하던 대부분의 영역에서,
  LLM을 컨트롤 타워의 역할로 쓸 수 있다

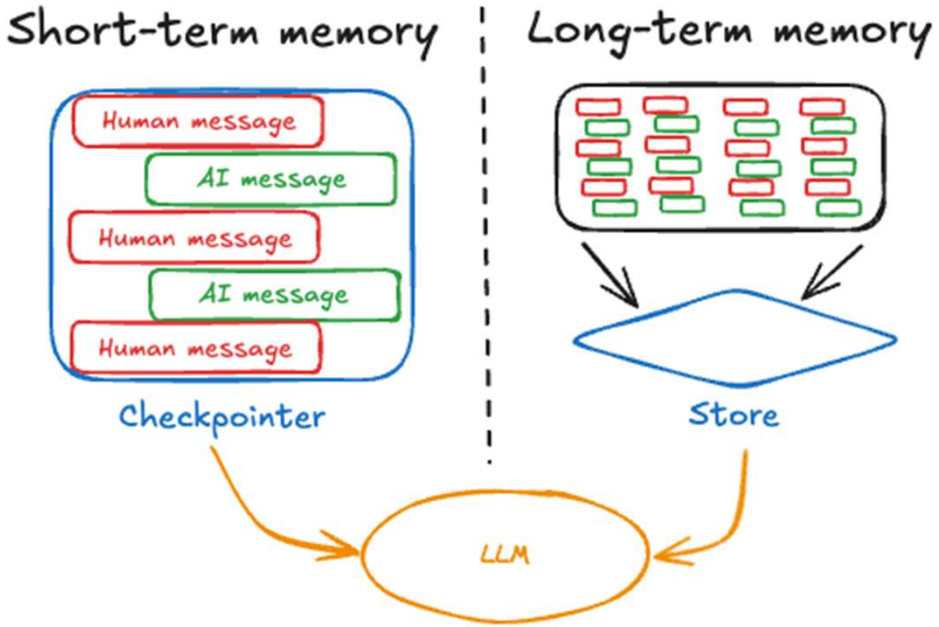
→ 예시: Robotics

Tool Calling의 효과

(나쁜) 예시: Realtime API와 Tool Calling의 결합

3) Memory: 서비스화를 위한 필수 요소

Context의 짧은 메모리 + Store 형식의 긴 메모리



Source: https://langchain-ai.github.io/langgraph/concepts/img/memory/short-vs-long.png

Agent = LLM + 3요소 + Reflection

Reflection: **자기 반성**

- 현재 상황을 판단하고, 다음 단계로 넘어가기 위한 '생각'이 필요
- 높은 추론 능력과도 연관

여러 단계의 Tool Call**이 필요한 문제라면?**

- 예시) 계산기와 인터넷 검색

  Q) 디카프리오의 탄생년도를 검색한 뒤, 각 숫자를 순서대로 곱해서 알려줘.

다단 Tool Call 예시: 올바른 전개

- 프롬프트:
  1. 유저) “디카프리오의 탄생년도를 검색해서, 각 숫자를 곱해봐.”
  2. LLM) <FUNCTION CALL> (검색) <END OF TEXT>
  3. Tool) “레오나르도 디카프리오는 1974년 …”
  4. LLM) <FUNCTION CALL> (곱하기: 1*9) <END OF TEXT>
  5. Tool) “9”
  6. LLM) <FUNCTION CALL> (곱하기: 9*7) <END OF TEXT>
  7. Tool) “63”
  8. LLM) <FUNCTION CALL> (곱하기: 63*4) <END OF TEXT>
  9. Tool) “252”
  10. LLM) “디카프리오의 생년월일을 모두 곱한 결과는 252입니다!”

다단 Tool Call 예시: 실제로 발생하는 전개

- 프롬프트:
  1. 유저) “디카프리오의 탄생년도를 검색해서, 각 숫자를 곱해봐.”
  2. LLM) <FUNCTION CALL> (검색) <END OF TEXT>
  3. Tool) “레오나르도 디카프리오는 1974년 …”
  4. LLM) <FUNCTION CALL> (곱하기: 1*9) <END OF TEXT>
  5. Tool) “9”
  6. LLM) <FUNCTION CALL> (곱하기: 1*9) <END OF TEXT>
  7. Tool) “9”
  8. LLM) <FUNCTION CALL> (곱하기: 1*9) <END OF TEXT>
  9. Tool) “9” (이하 반복)

다단 Tool Call: 왜 실패했는가?

Tool 요청-실행의 나열만으로는 집중력 있는 추론이 어려움

- 툴과 툴 사이의 연결을 강하게 이해시키는 방법이 필요

중간의 상황을 스스로 파악하게 하면 어떨까?

- 툴 실행 결과와 다음 툴 실행 사이에, Reflection을 추가하기

Reflection이 포함된 Agent

- Solution: 4. LLM) 디카프리오의 생년은 1974년이다. 이제 1, 9, 7, 4를 차례대로 곱하자.

5. LLM) <FUNCTION CALL> (곱하기: 1*9) <END OF TEXT>

6. Tool) “9”

7. LLM) 1*9의 값은 9다. 이제 9와 7을 곱하자.

8. LLM) <FUNCTION CALL> (곱하기: 9*7) <END OF TEXT>

9. Tool) “63”

10. LLM) 9*7의 값은 63이다. 이제 63과 4를 곱하자.

11. LLM) <FUNCTION CALL> (곱하기: 63*4) <END OF TEXT>

12. Tool) “252”

13. LLM) 이제 답을 알겠다.

14. LLM) <FUNCTION CALL> (최종답변: 결과는 252입니다!) <END OF TEXT>

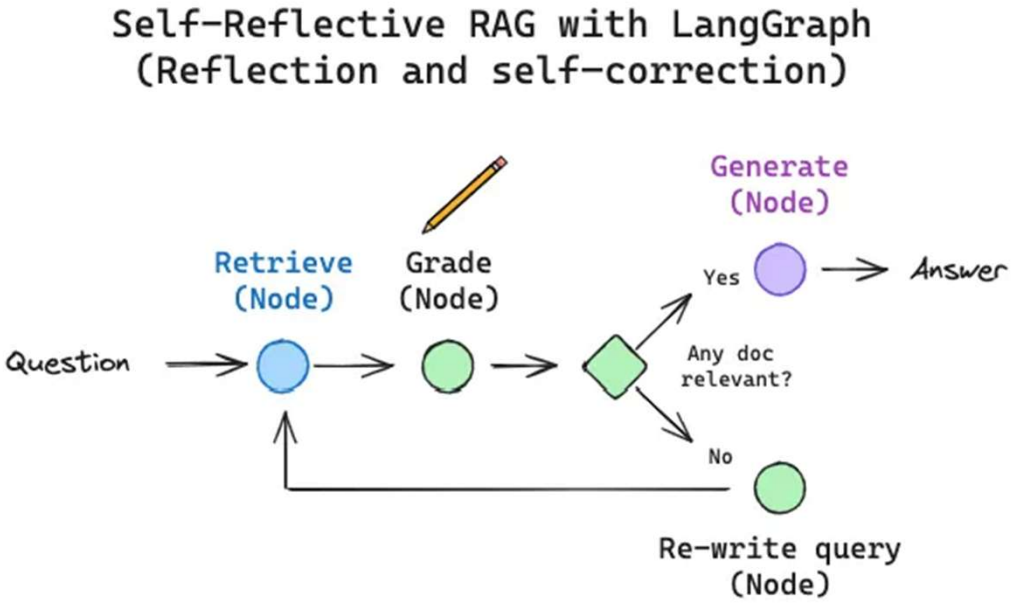
Agent VS Workflow

모든 경로가 자율적일 필요는 없음

- 정해진 경로를 그대로 수행하는 부분도 필요
- 이에 따라 Workflow와 Agent로 분리

Agentic Work

- 두 종류의 어플리케이션을 통합하여 명명
- Andrew Ng 교수의 제안



Summary

LLM의 능력을 증강시키는 (3+1)개의 요소

- Retrieval, Tool, Memory
- Reflection: 다단 추론을 위한 능력

Agent & Workflow → Agentic Work

- 다음 영상에서는 다양한 Agent의 형태에 대해 알아봅니다 :)