2026/08/24, AI 에이전트 세미나

이론부터 실전까지 AI에이전트 완벽 마스터

- 4장 LLM을 활용한 웹 스크래핑 에이전트 구축 -

이 진 수 (jinsue@pel.sejong.ac.kr)

세종대학교 프로토콜공학연구실

목 차

- 두뇌, 지각, 행동 패러다임
- AI 에이전트 분류
- 단일 에이전트와 다중 에이전트
- 주요 라이브러리

목 차

- 두뇌, 지각, 행동 패러다임
- AI 에이전트 분류
- 단일 에이전트와 다중 에이전트
- 주요 라이브러리

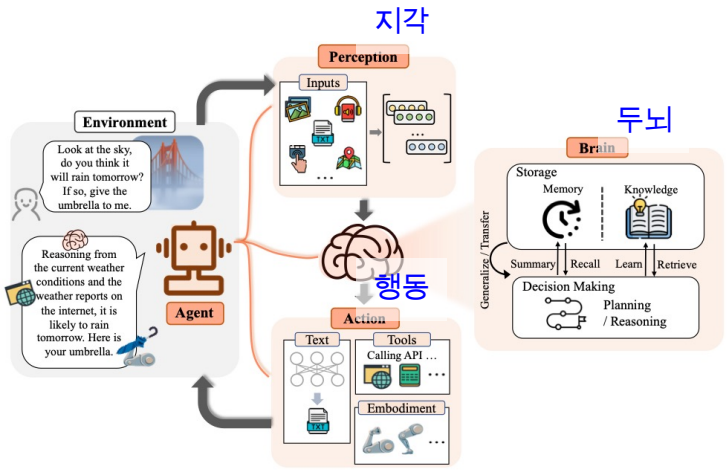
두뇌, 지각, 행동 패러다임

- LLM 기반 에이전트*
  - 정의
    - LLM을 의사결정 주체로 삼아, 외부 모듈을 통해 환경을 인지하고 도구를 사용해 행동을 수행하는 시스템
  - 특징
    - 단일 추론이 아닌, 지각-판단-행동 사이클의 반복으로 동작
      - 목표 달성 여부를 모델이 판단하여 사이클을 종료
    - 모델 재학습 없이 프롬프트 수준에서 행동 능력을 확보
      - 인-컨텍스트 러닝(in-context learning) 방식을 통한 파라미터 업데이트 없이 프롬프트에 주어진 지시와 예시만으로 도구 사용
      - 지시-출력 쌍 학습으로 다양한 작업 수행 능력 확보 가능
      - 중간 추론 단계(e.g., 사고의 사슬*) 단계를 통해 행동 계획 수립 가능

* 에이전트(Agent)
  - 행동할 수 있는 능력을 지닌 실체
* 사고의 사슬(Chain-of-Thought)
  - 중간 추론 단계를 통해 복잡한 추론을 가능하게 하는 프롬프팅

두뇌, 지각, 행동 패러다임

- LLM 기반 에이전트
  - 핵심속성
    - 자율성(autonomy)
      - 인간의 개입이나 명시적 지시 없이 스스로 작업을 수행
    - 반응성(reactivity)
      - 외부 환경의 변화를 인지하고 즉각 대응
    - 능동성(pro-activeness)
      - 반응에 그치지 않고 목표 달성을 위해 추론하고 계획
    - 사회적 능력(social ability)
      - 인간 및 다른 에이전트와 소통 및 협력
  - 구성요소
    - 두뇌(brain)
    - 지각(perception)
    - 행동(action)



<그림 4.1> 두뇌, 지각, 행동으로 구성된 LLM 기반 에이전트 프레임워크[1]

[1] Xi, Zhiheng, et al. "The rise and potential of large language model based agents: A survey. arXiv 2023." arXiv preprint arXiv:2309.07864 10 (2025).

두뇌, 지각, 행동 패러다임

- 두뇌(Brain)
  - 정보를 저장하고 탐색하며 추론과 계획을 통해 의사결정을 수행하는 시스템 구성요소

  - 모델 지식(model knowledge)
    - 언어 지식
      - 언어의 의미론과 구문론에 관한 지식을 통한 소통 기반
    - 상식 지식
      - 명시하지 않아도 통용되는 일반적 규칙과 사실
    - 도메인 지식
      - 의학, 수학 등 특정 분야의 작업 수행에 필요한 전문 지식

두뇌, 지각, 행동 패러다임

- 두뇌(Brain)
  - API 기반 폐쇄형 LLM 모델
    - 사업자가 제공하는 API를 호출해 사용하며 모델 내부에 접근 불가
    - 별도 인프라 없이 높은 성능의 모델 사용 가능하나, 호출 비용 발생과 외부 의존이 발생함
    - e.g., GPT-4, Claude 등
  - 오픈소스 LLM 모델
    - 가중치가 공개되어 자체 환경에 배포하고 파인튜닝 가능
    - 데이터를 내부에 두고 비용을 통제할 수 있으나 자원과 운영 부담이 비교적 높음
    - e.g., Mistral, LLaMa 등

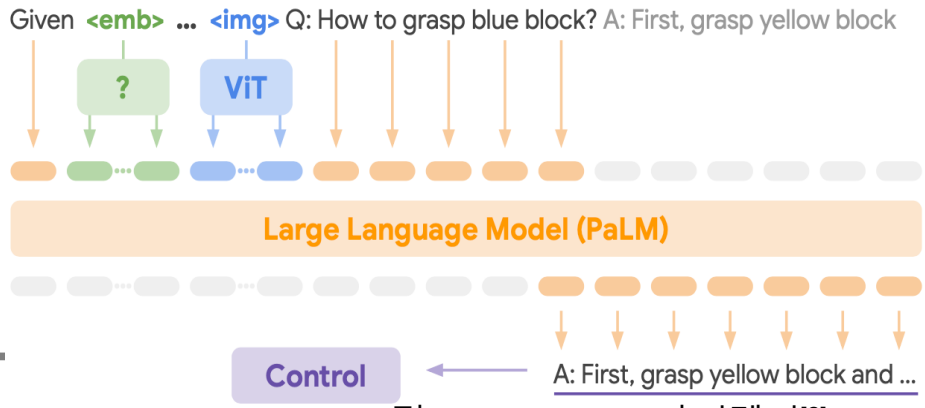
두뇌, 지각, 행동 패러다임

- 지각(Perception)
  - 정의
    - 다양한 모달리티로부터 정보를 획득하도록 입력범위를 확장하는 구성요소
  - 종류
    - PaLM-E
    - Flamingo
    - AudioGPT

두뇌, 지각, 행동 패러다임

* 체화된 추론(Embodied Reasoning)
  - 실제 환경의 센서 관측 값에 근거하여, 물리적으로 실행 가능한 판단이나 계획을 도출하는 추론
* ViT(Vision Transformer)
  - 이미지를 시퀀스 데이터에 변환하여 트랜스포머에 적용한 모델

- PaLM-E
  - 정의
    - 이미지, 상태 벡터 등 연속적인 입력을 언어 토큰과 동일한 임베딩 공간에 삽입하여 체화된 추론*을 수행하는 범용 멀티모달 언어 모델
  - 특징
    - 멀티모달 문장을 입력으로 사용
      - 텍스트 토큰 사이에 이미지·3D 표현 등의 임베딩을 포함
      - 삽입된 임베딩은 트랜스포머 블록에서 텍스트 토큰과 동일 처리
    - 사전 학습된 LLM(PaLM)과 ViT*를 결합한 구조
      - ViT가 이미지를 임베딩 시퀀스로 변환하고, 언어 임베딩 공간에 투영



<그림 4.2> PaLM-E 아키텍처[2]

[2] Driess, Danny, et al. "Palm-e: An embodied multimodal language model." *arXiv preprint arXiv:2303.03378* (2023).

두뇌, 지각, 행동 패러다임

- PaLM-E

* 어파인 변환
  - 입력 벡터에 행렬을 곱하고 편향을 더하는 선형 변환

  - 동작과정(1/3)

    1. 관측값 인코딩
       - 연속 관측값(e.g., 이미지)을 인코더가 언어 토큰과 동일한 차원의 임베딩 시퀀스로 변환
         - 관측 종류에 따라 서로 다른 인코더를 사용(e.g., 상태 벡터(MLP), 2D 이미지(ViT), 3D 이미지(OSRT) 등)
       - ViT($\tilde{\phi}_{ViT}$)로 이미지(I)를 토큰 임베딩으로 변환 후, 어파인 변환*(ψ)을 거쳐 언어 임베딩 공간의 벡터(xi) 생성: $xi = \phi ViT(I)_i = \psi(\tilde{\phi}_{ViT}(I)_i)$

    2. 멀티모달 문장 구성
       - 변환된 관측 임베딩을 텍스트 토큰 임베딩 사이 임의 위치에 삽입하여 멀티모달 문장 형태의 접두사 구성
       - 텍스트 토큰(w_i)은 워드 임베딩(γ)으로, 관측값(O_j)은 인코더(ϕ_j)로 변환하여 동일한 시퀀스의 i번째 위치(x_i)에 배치:

$$x_i = \begin{cases} \gamma(w_i) \; If \; i \; is \; a \; text \; token, or \\ \phi_j(O_j) \; If \; i \; corresponds \; to \; observation \; O_j \end{cases}$$

두뇌, 지각, 행동 패러다임

- PaLM-E
  - 동작과정(2/3)

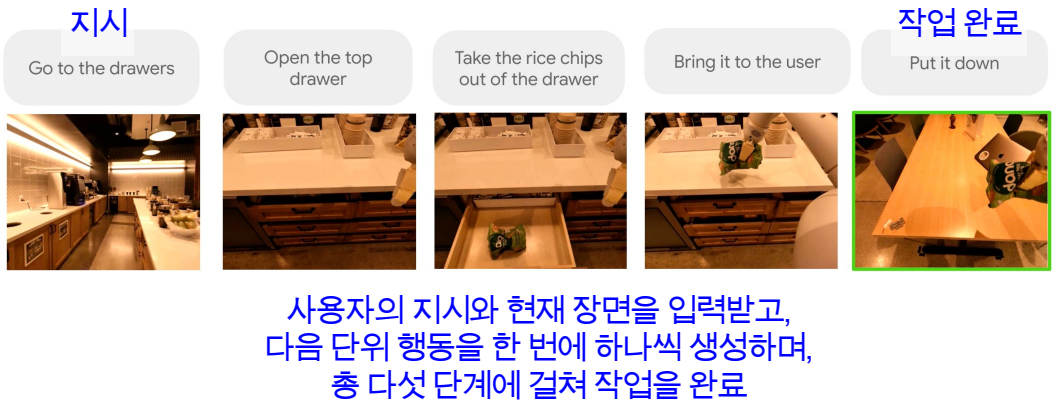
3. 자기회귀 텍스트 생성
   - 디코더 전용 LLM이 멀티모달 접두사에 이어질 텍스트를 자기회귀적으로 생성함
   - 접두사($w_{1:n}$)가 주어졌을 때 이어질 각 토큰(w_l)의 확률을 LLM(p_{LM})이 순차적으로 산출하고, 이를 곱해 전체 시퀀스의 확률(P) 계산:

$$P(w_{n+1:L}|w_{1:n}) = \prod_{l=n+1}^{L} p_{LM}(w_l|w_{1:l-1})$$

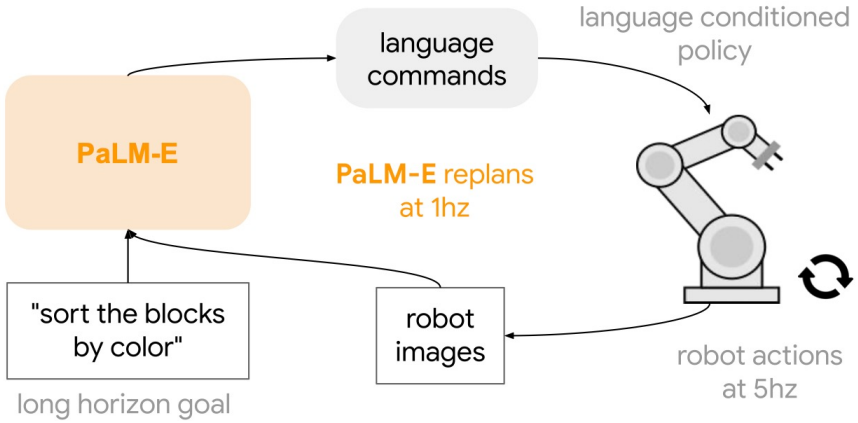
4. 출력 처리
   - 질의응답 작업에서는 생성된 텍스트가 최종 답변으로 산출됨
   - 계획·제어 작업에서는 생성된 텍스트가 단위 행동(e.g., 서랍을 연다, 과자를 집는다 등)의 목록이 됨
     - 수행 가능한 행동은 학습 데이터와 프롬프트를 근거로 모델이 스스로 판단

두뇌, 지각, 행동 패러다임

- PaLM-E
  - 동작과정(3/3)
    5. 제어 루프 실행
      - 생성된 단위 행동을 로봇 제어 모델(e.g., RT-1)에 한 문장씩 전달하면, 제어 모델이 이를 실제 로봇의 움직임 명령으로 변환
      - 로봇이 움직인 뒤의 장면이 다시 이미지로 입력되어 PaLM-E가 다음 행동을 재결정
        - PaLM-E는 로봇을 직접 움직이지 않고 상위 계획자 역할을 수행함



<그림 4.3> PaLM-E에서 생성한 단위 행동 실제 수행 과정[2]

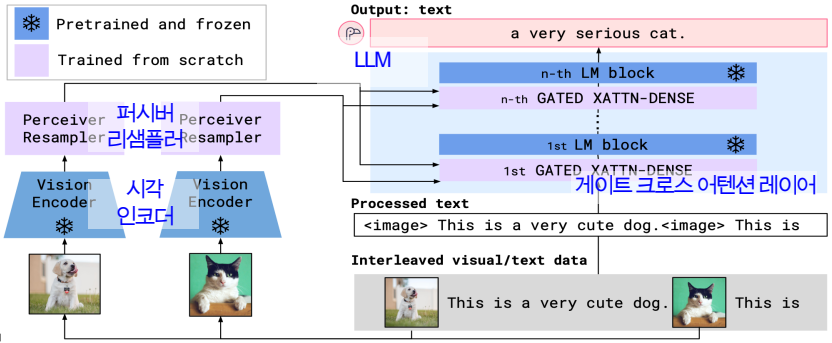


<그림 4.4> PaLM-E의 제어 루프[2]

[2] Driess, Danny, et al. "Palm-e: An embodied multimodal language model." *arXiv preprint arXiv:2303.03378* (2023).

두뇌, 지각, 행동 패러다임

- Flamingo
  - 정의
    - 동결된 시각 인코더와 LLM을 연결하여, 이미지·비디오와 텍스트가 교차된 시퀀스를 입력받아 텍스트를 생성하는 퓨샷(few-shot) 시각 언어 모델
  - 특징
    - 동결된 LLM에 크로스 어텐션으로 시각 정보를 주입하여, 언어 능력 훼손 없이 지각 범위만 확장함
      - 시각 인코더(NFNET-F6)와 LLM의 가중치는 갱신하지 않고 연결 모듈만 학습
    - 이미지와 비디오를 고정 길이 토큰으로 압축하여 처리
    - 이미지와 텍스트가 교차된 시퀀스를 시각 작업에서도 퓨샷 학습 적용
      - 각 텍스트 토큰은 직전 이미지 하나에만 어텐션하여, 학습시보다 많은 이미지에 일반화 가능

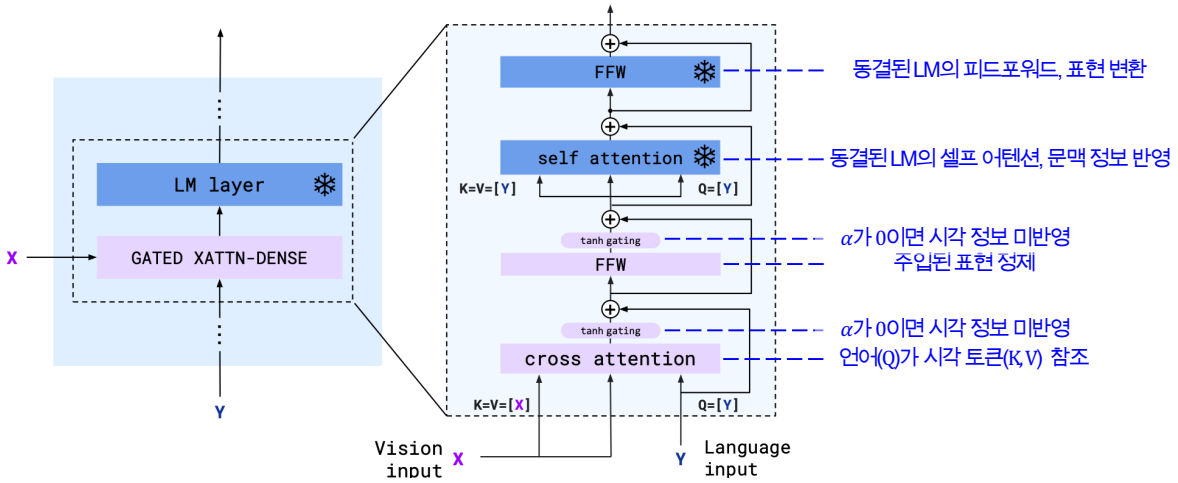


<그림 4.5> Flamingo 아키텍처[3]

[3] Alayrac, Jean-Baptiste, et al. "Flamingo: a visual language model for few-shot learning." *Advances in neural information processing systems* 35 (2022): 23716-23736.

두뇌, 지각, 행동 패러다임

- Flamingo
  - 학습 과정(1/4)
    1. 학습 대상 결정
       - 사전 학습된 시각 인코더와 LM 블록은 동결하고, 그 사이를 잇는 퍼시버 리샘플러와 게이트 크로스 어텐션 레이어만 무작위 초기화 상태로 학습
       - 언어표현(y)에 시각 토큰(x)을 참조한 크로스 어텐션과 피드포워드 결과를 더하되, 각 항에 tanh 게이트(α)를 곱하여 반영 정도 조절
         - α는 0으로 초기화되므로, 학습 초기에는 동결된 두 항만 남아 기존 LM과 동일하게 동작, 학습이 진행되며 α가 커져 시각 정보가 점차 반영




```
def gated_xattn_dense(
    y,  # input language features
    x,  # input visual features
    alpha_xattn, # xattn gating parameter – init at 0.
    alpha_dense, # ffw gating parameter – init at 0.
):
  """Applies a GATED XATTN-DENSE layer."""

  # 1. Gated Cross Attention   #학습 대상: 시각 정보 주입
  y = y + tanh(alpha_xattn) * attention(q=y, kv=x)
  # 2. Gated Feed Forward (dense) Layer  #학습 대상: 주입된 표현 정제
  y = y + tanh(alpha_dense) * ffw(y)

  # Regular self-attention + FFW on language
  y = y + frozen_attention(q=y, kv=y) #동결: 문맥 반영
  y = y + frozen_ffw(y) #동결: 표현 변환
  return y  # output visually informed language features
```

<그림 4.6> 게이트 크로스 어텐션 레이어 동작[3]

[3] Alayrac, Jean-Baptiste, et al. "Flamingo: a visual language model for few-shot learning." *Advances in neural information processing systems* 35 (2022): 23716-23736.

두뇌, 지각, 행동 패러다임

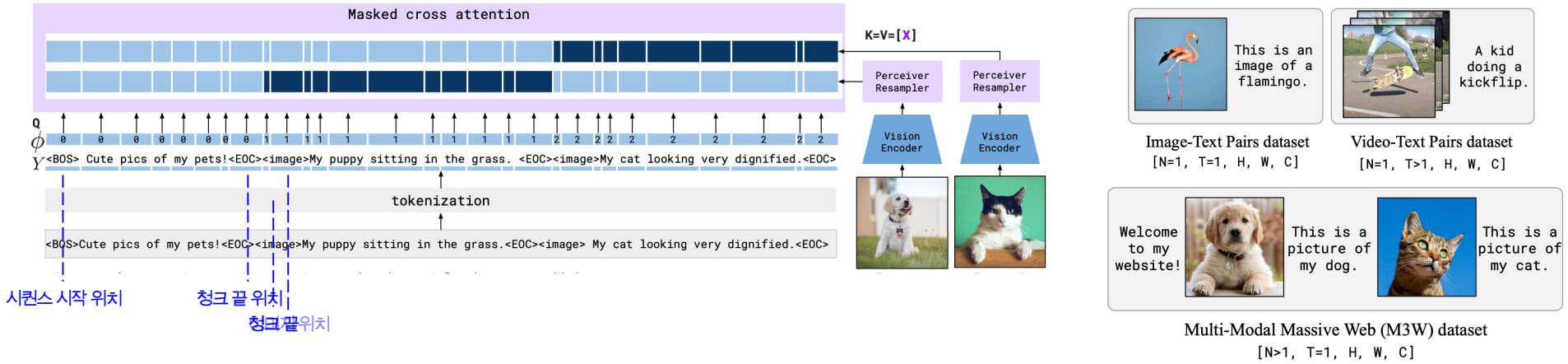
- Flamingo

* M3W(MultiModal MassiveWeb)
  - Flamingo 학습을 위해 구축한 데이터셋

  - 학습 과정(2/4)

    2. 학습 데이터 구성

      - 교차 데이터와 쌍 데이터를 함께 사용하며, 데이터셋마다 시각 입력 수와 프레임 수가 다름
      - 시퀀스 시작에 <BOS>, 이미지 위치에 <Image>, 청크 끝에 <EOC> 토큰을 삽입해 모든 데이터셋의 입력 형식 통일
        - M3W* 문서에서 256개 토큰의 부분 시퀀스를 무작위로 뽑고, 그 안의 이미지를 최대 5개까지만 사용함



<그림 4.7> 데이터셋 통합을 위한 특수 토큰 삽입 형식[3]

[3] Alayrac, Jean-Baptiste, et al. "Flamingo: a visual language model for few-shot learning." *Advances in neural information processing systems* 35 (2022): 23716-23736.

두뇌, 지각, 행동 패러다임

- Flamingo
  - 학습 과정(3/4)

* 우도(likelihood)
  - 모델이 주어진 데이터를 생성할 확률로, 값이 클수록 데이터를 잘 설명함

3. 조건부 우도* 정의
   - 각 텍스트 토큰의 확률을 이전 토큰과 그 앞에 등장한 이미지에만 조건화함
   - 텍스트 토큰(y_l)의 확률로 이전 토큰($y < 1$)과 선행 이미지($x \leq 1$)에 조건화하고, 이를 모두 곱해 전체 텍스트의 확률(p) 계산:

$$p(y|x) = \prod_{l=1}^{L} p(y_l | y < l, x \leq l)$$

4. 목적 함수 정의
   - 형식이 서로 다른 데이터셋을 하나의 손실로 묶어 동시 학습함
   - 데이터셋(D_m)별 음의 로그 가능도 기대값에 가중치(λ_m)를 곱해 합산한 값을 손실로 사용:

$$\sum_{m=1}^{M} \lambda_m \cdot \mathbb{E}_{(x,y)\sim D_m} \left[-\sum_{\ell=1}^{L} \log\left(y_\ell | y < y_{<\ell} l, x_{\leq \ell}\right) \right]$$

[3] Alayrac, Jean-Baptiste, et al. "Flamingo: a visual language model for few-shot learning." *Advances in neural information processing systems* 35 (2022): 23716-23736.

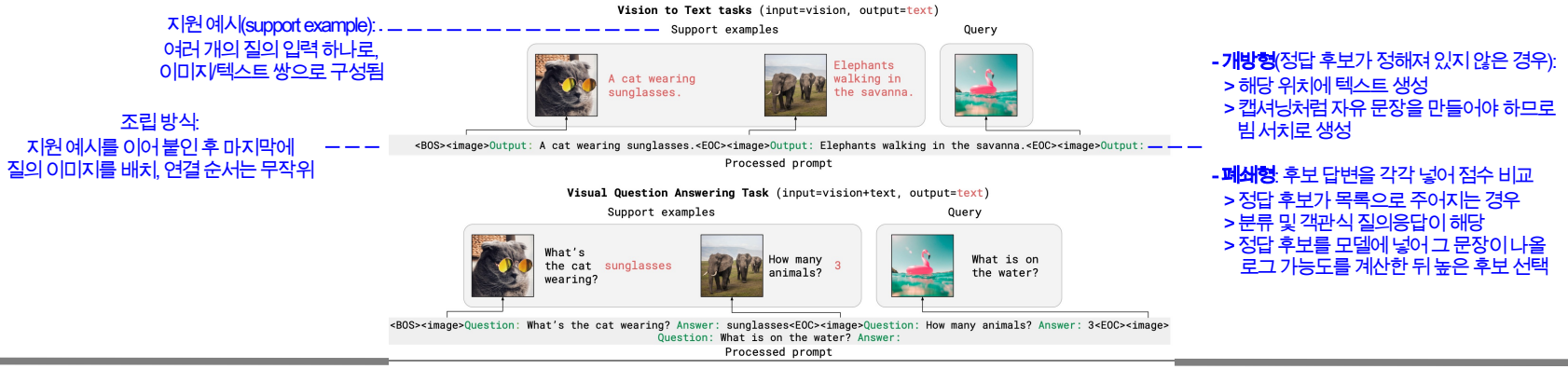
두뇌, 지각, 행동 패러다임

- Flamingo
  - 학습 과정(4/4)

5. 최적화
    - 데이터셋을 번갈아 학습하는 라운드 로빈* 방식 대신, 모든 데이터셋의 기울기를 누적하여 한 번에 갱신함

6. 작업 적응
    - 학습 이후 가중치를 갱신하지 않고, (이미지, 텍스트) 예시 쌍과 질의 입력을 교차 배치한 프롬프트만으로 새로운 작업에 적응함
    - 개방형 작업은 빔 서치*로 생성하고, 폐쇄형 작업은 후보 답변의 로그 가능도를 비교해 답변을 선택함

* 라운드 로빈
  - 여러 데이터셋을 정해진 순서대로 번갈아 가며 학습하는 방식
* 빔 서치
  - 매 시점 확률이 높은 후보 문장 여러 개를 동시에 유지하며 탐색하는 생성 방식

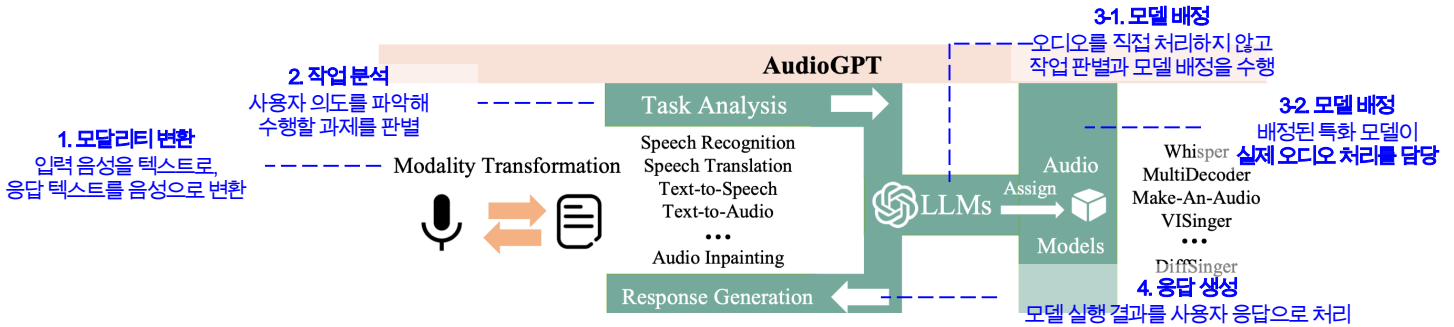


<그림 4.8> 퓨샷 프롬프트의 구성[3]

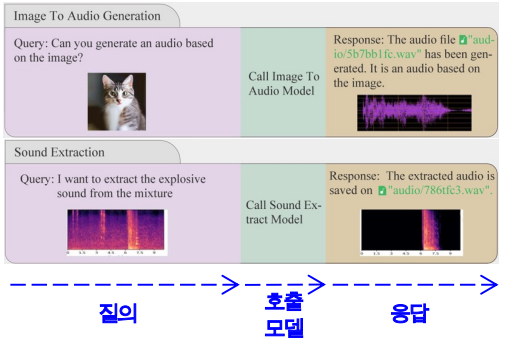
[3] Alayrac, Jean-Baptiste, et al. "Flamingo: a visual language model for few-shot learning." *Advances in neural information processing systems* 35 (2022): 23716-23736.

두뇌, 지각, 행동 패러다임

- AudioGPT
  - 정의
    - LLM을 오디오 파운데이션 모델 및 음성 입출력 인터페이스와 결합하여, 오디오 모달리티를 생성하는 프롬프트 기반 멀티모달 AI 시스템
  - 특징
    - LLM을 학습하지 않고 외부 특화 모델을 호출하는 방식으로 오디오 모달리티를 확장
    - 음성 입출력 인터페이스(i.e, ASR, TTS)를 통해 음성 대화 지원



<그림 4.9> AudioGPT 구조[4]



<그림 4.10> AudioGPT 수행 예시[4]

[4] Huang, Rongjie, et al. "Audiogpt: Understanding and generating speech, music, sound, and talking head." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 38. No. 21. 2024.

두뇌, 지각, 행동 패러다임

- AudioGPT
  - 동작과정(1/2)

    1. 모달리티 변환
       - 질의 설명이 음성이면 텍스트로 변환하여 일관된 형식의 질의 생성
       - 질의 설명($q^{(d)}_n$)이 텍스트면 유지하고, 음성이면 변환기(T)를 적용하되, 질의 관련 자원($\{q^{(s_1)}_n, \ldots\}$)은 변환하지 않고 유지함:

$$q'_n = (q'^{(d)}_n, \{q^{(s_1)}_n, \ldots, q^{(s_k)}_n\}) = \begin{cases} (q^{(d)}_n, \{q^{(s_1)}_n, \ldots, q^{(s_k)}_n\}) & \text{if } q^{(d)}_n \text{ is text,} \\ (\mathcal{T}(q^{(d)}_n), \{q^{(s_1)}_n, \ldots, q^{(s_k)}_n\} & \text{if } q^{(d)}_n \text{ is audio.} \end{cases}$$

    2. 작업 분석
       - 작업 처리기가 입출력 모달리티로 작업군을 판별한 뒤, 대화 엔진과 프롬프트 관리자가 구조화된 인자를 추출함
       - 작업 처리기($\mathcal{H}$)가 선택한 작업군과 질의 설명을 프롬프트 관리자($\mathcal{M}$)에 전달하고, 대화 엔진($\mathcal{L}$)이 맥락($\mathcal{C}$)과 함께 처리하여 사용할 모델($\mathcal{P}_p$)과 그 인자($h_{\mathcal{P}_p}$)를 결정함: $(\mathcal{P}_p, h_{\mathcal{P}_p}) = \mathcal{L}(\mathcal{M}(\mathcal{H}(q'_n), q'^{(d)}_n), \mathcal{C})$

두뇌, 지각, 행동 패러다임

- AudioGPT
  - 동작과정(2/2)

* 시간축 사후확률 그래프
  - 가로축을 시간, 세로축을 범주로 두고 각 지점의 예측 확률을 색으로 나타낸 그래프

3. 모델 배정
   - 선택된 모델에 질의 자원과 인자를 넘겨 실행하고 결과를 얻음
   - 모델($\mathcal{P}_p$)이 질의 자원($\{q^{(s_1)}{}_n, \ldots\}$)과 인자($\mathcal{h}_{\mathcal{P}_p}$)를 입력 받아 작업 출력($o_{\mathcal{P}_p}$)을 산출함: $o_{\mathcal{P}_p} = \mathcal{P}_p(\{q_n^{(s_1)}, q_n^{(s_2)}, \ldots, q_n^{(s_k)}\}, \mathcal{h}_{\mathcal{P}_p})$

4. 응답 생성
   - 대화 엔진이 변환된 질의, 맥락, 모델 출력을 결합해 최종 응답을 생성
   - 작업 유형에 따라 표현 방식이 달라지며, 오디오 생성은 파형 이미지와 음성 파일 제시하고, 분류는 시간축 사후확률 그래프*를 제시함

두뇌, 지각, 행동 패러다임

- 행동(Action)
  - 정의
    - 모델의 판단을 도구 호출로 변환해 외부 환경에 실제로 적용시키는 구성요소
  - 출력 형태
    - 텍스트 응답
      - LLM이 본래 갖춘 텍스트 생성 능력에 해당하는 기본적인 출력
    - 도구 사용
      - 도구를 호출해 LLM 자체의 한계를 넘어선 작업을 수행
    - 체화된 행동
      - 가상 환경에 머물던 상호작용을 물리적 세계로 확장

두뇌, 지각, 행동 패러다임

- Toolformer

* 자기지도 학습(Self-supervised Learning)
  - 사람이 붙인 정답 없이, 데이터 자체에서 학습 신호를 만들어 학습하는 방식

  - 정의
    - 사람이 만든 주석 없이, 어떤 도구를 언제 어떤 인자로 호출할지 스스로 학습하는 자기지도* 기반 언어 모델

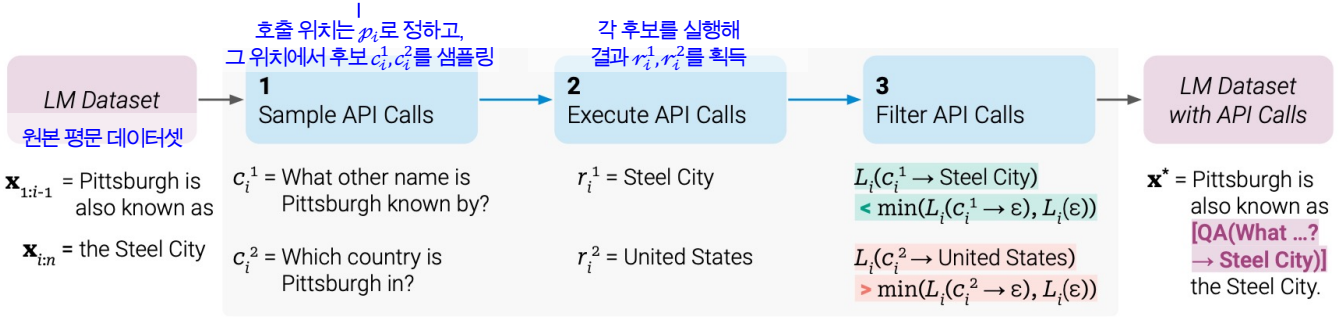
  - 특징
    - 도구 사용법을 사람이 지정하지 않고 모델이 스스로 결정
      - 어느 위치에서, 어떤 API를, 어떤 인자로 호출할지 모델이 판단
    - 입출력을 텍스트로 표현할 수 있는 모든 도구 연결 가능
      - 질의응답, 위키피디아 검색, 계산기, 기계 번역, 달력

<그림 4.11> Toolformer가 생성한 API 호출 예시[5]

[5] Schick, Timo, et al. "Toolformer: Language models can teach themselves to use tools." *Advances in neural information processing systems* 36 (2023): 68539-68551.

두뇌, 지각, 행동 패러다임

- Toolformer
  - 학습과정(1/2)
    1. API 호출 후보 샘플링
       - 도구별 프롬프트로 텍스트의 각 위치에서 호출 후보 생성
       - 위치(i)에서 호출 시작 토큰이 나올 확률(p_i)이 임계값(τ_s)을 넘는 위치만 남기고 상위 k개를 고른 뒤, 각 위치마다 최대 m개의 호출 샘플링: $p_i = p_m(< \text{API} > | P(\text{x}), x_{1:i-1})$
    2. API 호출 실행
       - 생성된 호출을 실제로 실행하여 결과를 텍스트 한 줄로 받음
       - 도구에 따라 다른 신경망 호출, 파이썬 스크립트 실행, 검색 시스템 조회 등 수행



<그림 4.12> Toolformer 학습 파이프라인[5]

[5] Schick, Timo, et al. "Toolformer: Language models can teach themselves to use tools." *Advances in neural information processing systems* 36 (2023): 68539-68551.

두뇌, 지각, 행동 패러다임

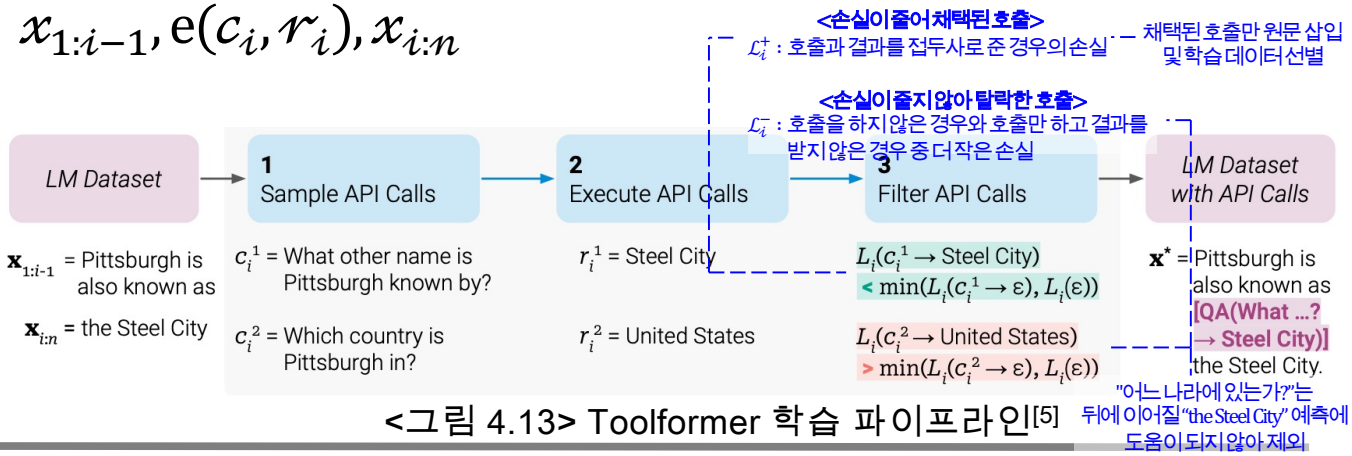
- Toolformer
  - 학습과정(2/2)

3. 호출 필터링
   - 호출 결과가 이후 토큰 예측에 실제로 도움이 된 경우만 남김
   - 호출과 결과를 모두 준 손실($\mathcal{L}_i^+$)이, 호출을 하지 않거나 결과를 주지 않은 손실($\mathcal{L}_i^-$)보다 임계값(τ_f) 이상 작은 경우 채택

4. 파인 튜닝
   - 원본 텍스트(x)의 i번째 위치에 호출과 결과를 추가해 새로운 시퀀스(x^*)를 구성하며, 삽입된 호출 외에는 원본과 동일한 내용 유지:

$$x^* = x_{1:i-1}, \mathrm{e}(c_i, r_i), x_{i:n}$$

5. 추론
   - 텍스트 생성 중 "→" 토큰 생성시 API 호출 결과 삽입, 생성 재개



<그림 4.13> Toolformer 학습 파이프라인[5]

[5] Schick, Timo, et al. "Toolformer: Language models can teach themselves to use tools." *Advances in neural information processing systems* 36 (2023): 68539-68551.

목 차

- 두뇌, 지각, 행동 패러다임
- AI 에이전트 분류
- 단일 에이전트와 다중 에이전트
- 주요 라이브러리

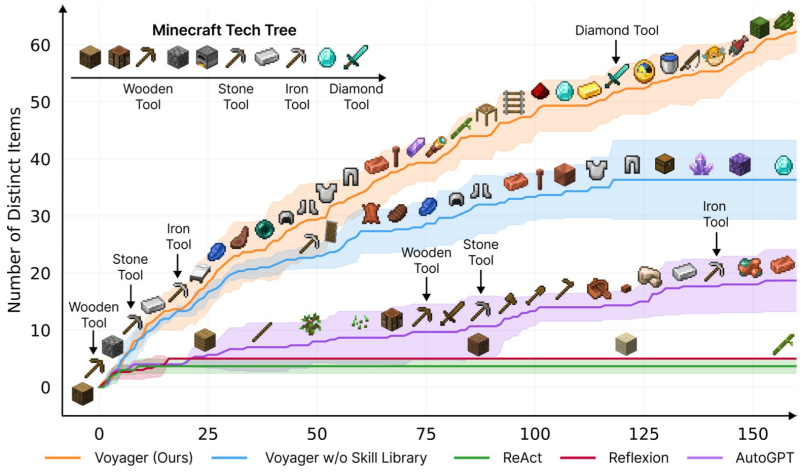
AI 에이전트 분류

- 분류 기준#1: 상호작용 범위
  - 구분
    - 에이전트가 활동하고 영향을 미치는 환경을 기준으로 분류
  - 종류
    - 디지털 에이전트(Digital Agent)
      - 가상 환경에서만 활동하며, 가상 세계와 상호작용
      - e.g., Voyager 등
    - 체화 에이전트(Embodied Agent)
      - 물리적 세계와 직접 상호작용하는 에이전트
      - e.g., PaLM-E 등

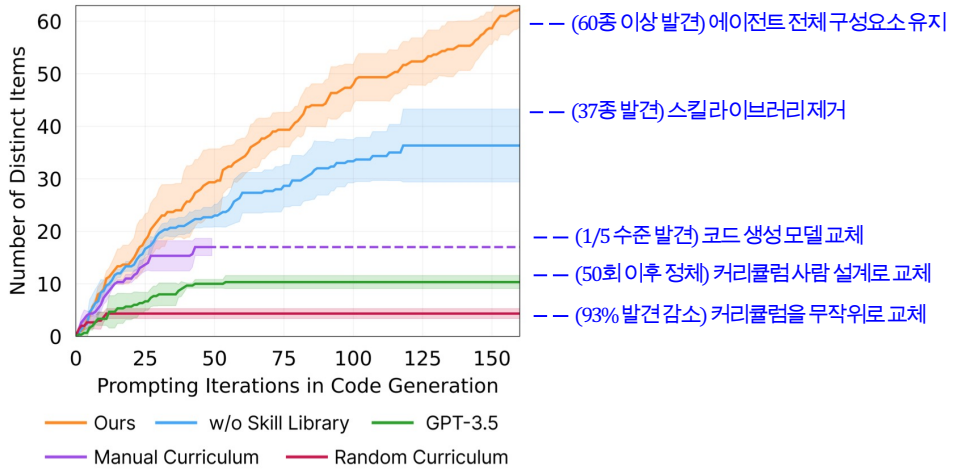
AI 에이전트 분류

- Voyager
  - 정의
    - 코드 형태의 스킬을 자율적으로 축적하며 마인크래프트 내부를 탐험하는 LLM 기반 에이전트
  - 특징
    - 파라미터 갱신 없이 문맥 내 학습으로만 능력 확장
    - 자동 커리큘럼*이 작업을 스스로 제안해 개방형 탐색을 지속

* 커리큘럼 학습(Curriculum Learning)
  - 에이전트가 수행할 과제를 난이도 순으로 배열하여 제시하는 방식



<그림 4.14> 프롬프트 반복 횟수에 따른 신규 아이템 발견 수[6]



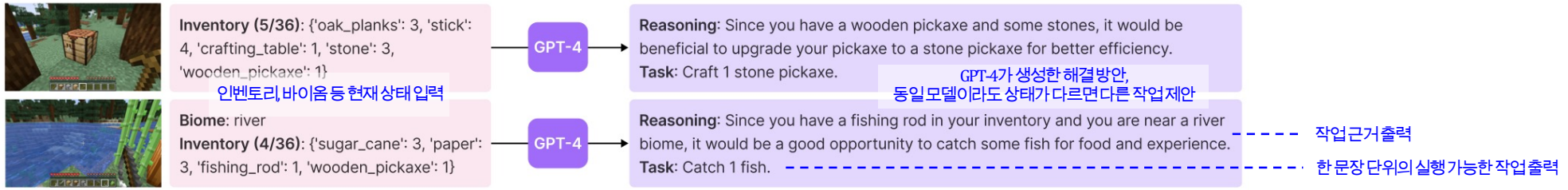
<그림 4.15> Voyager 구성요소 제거에 따른 탐색 성능 변화[6]

[6] Wang, Guanzhi, et al. "Voyager: An open-ended embodied agent with large language models." *arXiv preprint arXiv:2305.16291*(2023).

AI 에이전트 분류

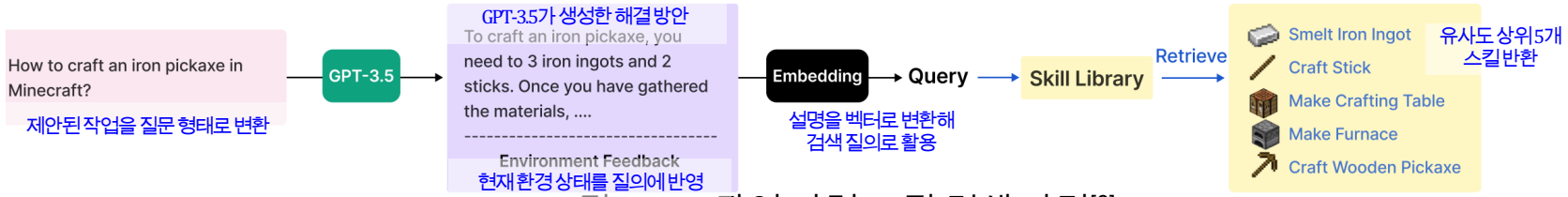
- Voyager
  - 동작 과정(1/3)
    1. 작업 제안
       - 인벤토리, 바이옴*, 주변 개체, 체력 등 현재 상태를 GPT-4에 제시
       - GPT-4가 상태를 근거로 판단한 이유와 다음 작업을 출력

* 바이옴(Biome)
  - 마인크래프트 세계를 구성하는 지형·기후 단위
  - 숲, 사막 등으로 구분되며 획득 가능한 자원이 상이함



<그림 4.16> 상태정보에 따라 차순 작업을 제안하는 과정 예시[6]

    2. 스킬 검색
       - 제안된 작업을 GPT-3.5에 제공 및 해결 방안 설명 생성
       - 설명을 임베딩해 질의로 삼고, 스킬 라이브러리에서 상위 5개 검색

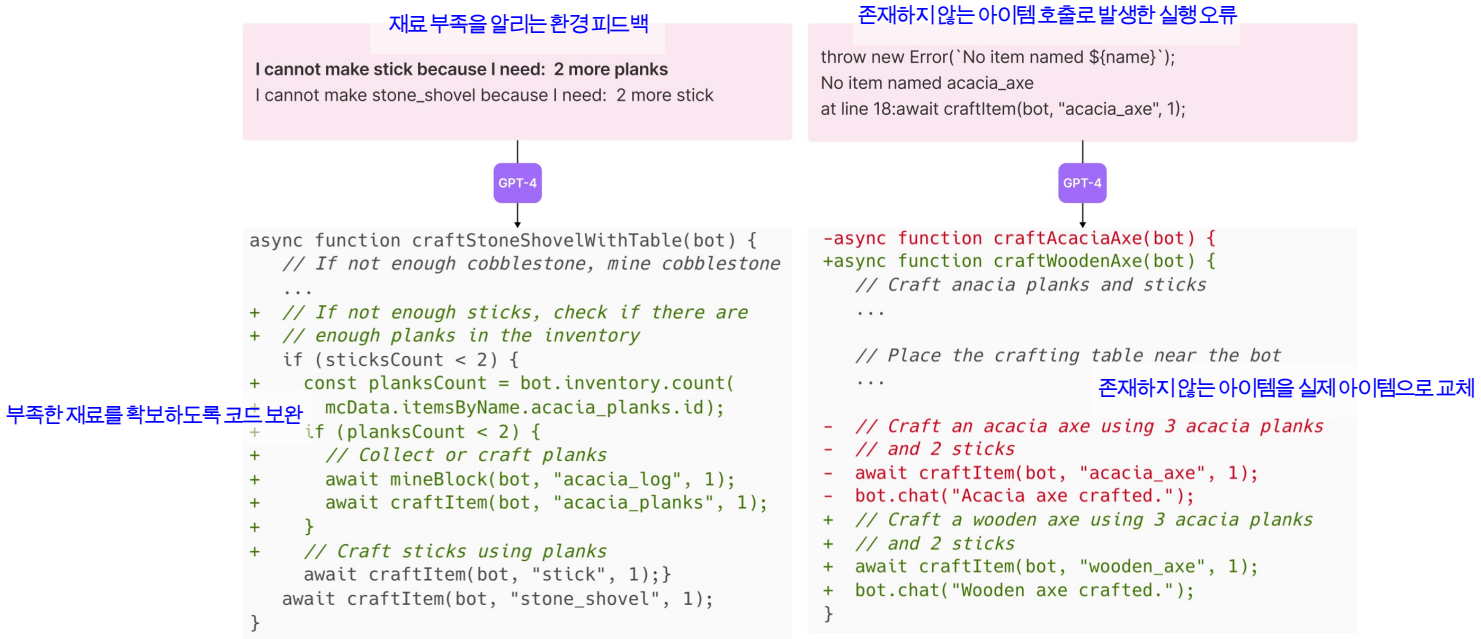


<그림 4.17> 작업 관련 스킬 검색 과정[6]

[6] Wang, Guanzhi, et al. "Voyager: An open-ended embodied agent with large language models." *arXiv preprint arXiv:2305.16291*(2023).

AI 에이전트 분류

- Voyager
  - 동작 과정(2/3)
    3. 코드 생성과 수정
       - GPT-4가 검색된 스킬과 제어 API를 통해 작업 수행 코드 생성 및 실행
       - 재료 부족을 알리는 환경 피드백과 잘못된 호출로 인한 실행 오류를 프롬프트에 넣어 코드 재생성

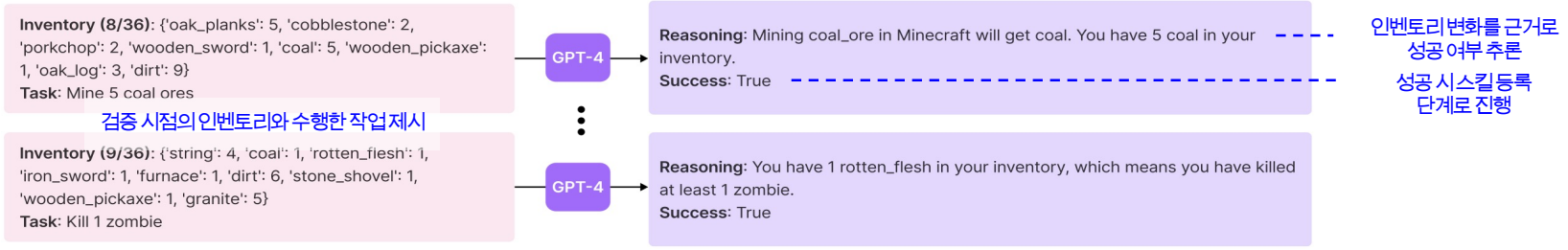


<그림 4.18> 환경 피드백과 실행 오류에 따른 코드 수정[6]

[6] Wang, Guanzhi, et al. "Voyager: An open-ended embodied agent with large language models." *arXiv preprint arXiv:2305.16291*(2023).

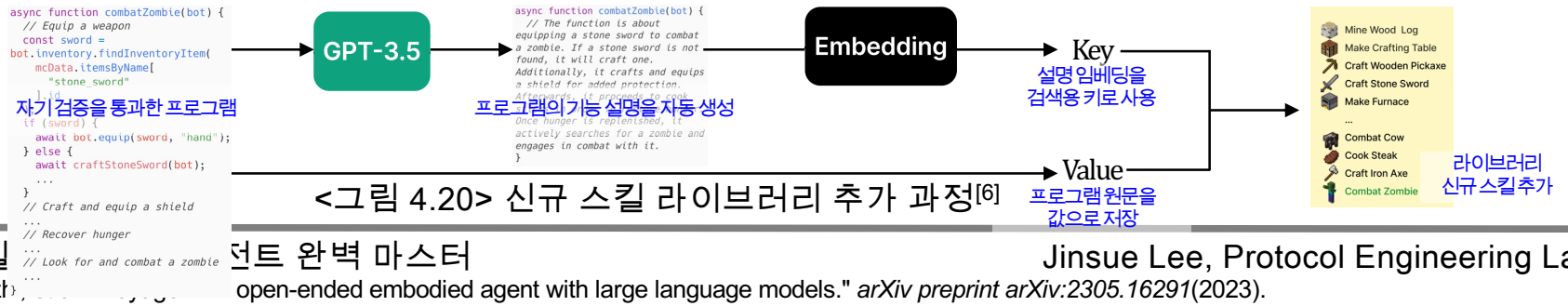
AI 에이전트 분류

- Voyager
  - 동작 과정(3/3)
    4. 자기 검증
       - 현재 상태와 작업을 다른 GPT-4에 제시해 작업 성공 여부를 판정
       - 실패 판정 시 완료에 필요한 조치를 지적으로 반환하여 최대 4회 반복



<그림 4.19> 자기검증 동작 예시[6]

    5. 스킬 등록
       - 성공한 코드의 기능 설명을 GPT-3.5가 생성하고, 임베딩으로 변환
       - 프로그램을 값으로 저장한 뒤 1단계로 복귀



<그림 4.20> 신규 스킬 라이브러리 추가 과정[6]

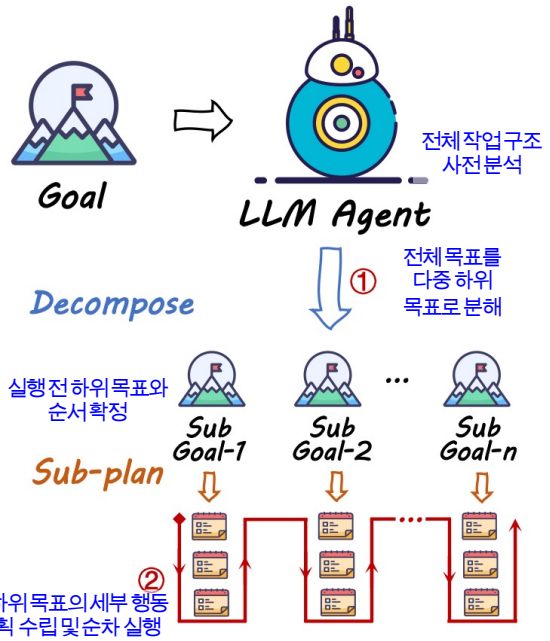
[6] Wang, Guanzh... open-ended embodied agent with large language models." *arXiv preprint arXiv:2305.16291*(2023).

AI 에이전트 분류

- 분류 기준#2: 작업 계획 방식
  - 구분
    - 목표를 분해하고 실행 계획을 수립·수정하는 방식을 기준으로 분류
  - 특징
    - 복잡하고 추상적인 목표를 모델이 수행할 수 있는 작업으로 변환함
  - 종류
    - 작업 분해 시점 관련 방식
      - 작업 분해와 계획·실행의 순서를 결정
      - e.g., 선분해 방식, 교차 분해 방식 등 포함
    - 계획 성능 보완 관련 방식
      - 작업 분해만으로 부족한 계획의 선택·검증·수정 및 경험 활용 능력 보완
      - e.g., 다중 계획 선택, 외부 플래너 지원 계획, 성찰 및 개선, 메모리 증강 계획 등 포함

* 제로샷 연쇄 추론(Zero-Shot CoT)
  - 정답 예시 없이 "단계별로 생각해 보자"와 같은 지시만으로 모델의 중간 추론 과정을 유도하는 기법

- 선분해 방식(Decomposition – first)
  - 정의
    - 전체 목표를 여러 개의 하위 목표로 먼저 분해한 후, 미리 정한 순서에 따라 각 작업을 실행하는 방식
  - 특징
    - 제로샷 연쇄 추론*의 영향을 받은 방식
    - 실행 전에 전체 하위 목표와 수행 순서를 결정
    - 각 하위 목표에 대한 세부 계획을 순차적으로 수립 및 실행

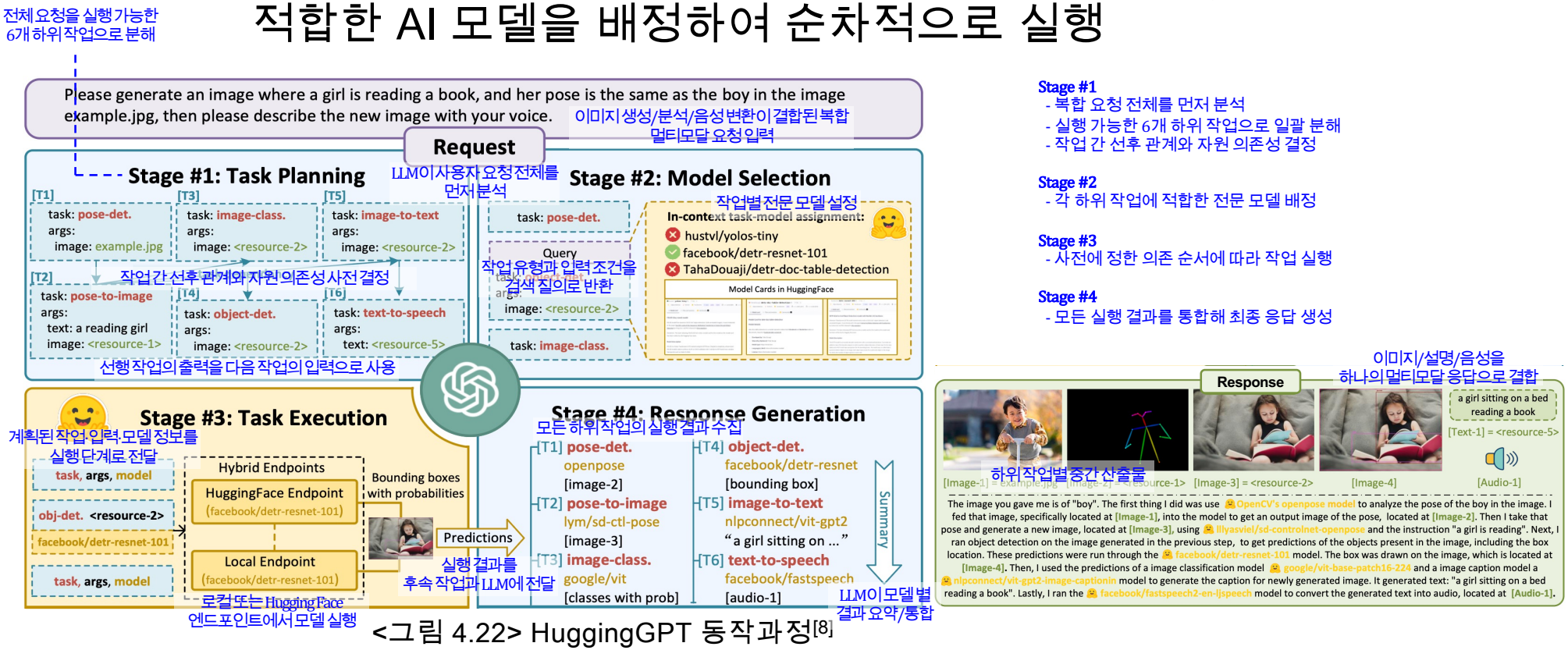


<그림 4.21> 선분해 방식의 작업 분해 및 하위 계획 과정[7]

[7] Huang, Xu, et al. "Understanding the planning of LLM agents: A survey." *arXiv preprint arXiv:2402.02716* (2024).

AI 에이전트 분류

- 선분해 방식(Decomposition – first)
  - 활용 예시
    - HuggingGPT
      - 복잡한 요청을 여러 하위 작업으로 먼저 분해하고, 각 작업에 적합한 AI 모델을 배정하여 순차적으로 실행



<그림 4.22> HuggingGPT 동작과정[8]

[8] Shen, Yongliang, et al. "Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face." *Advances in Neural Information Processing Systems* 36 (2023): 38154-38180.

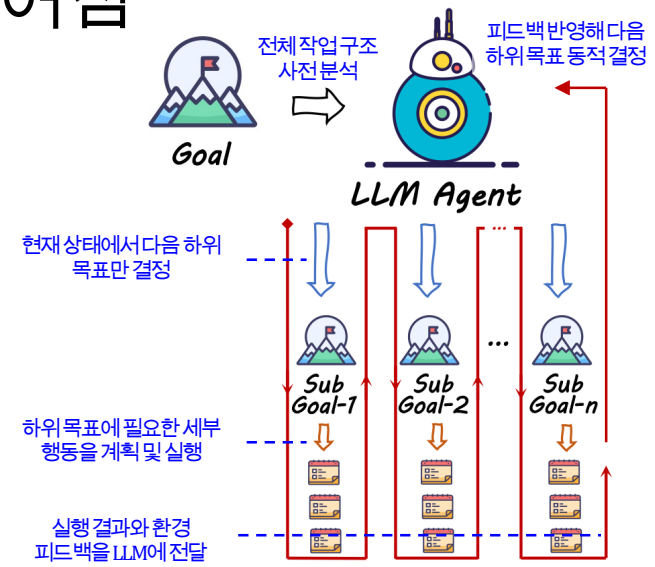
AI 에이전트 분류

- 선분해 방식(Decomposition – first)
  - 장점
    - 전체 작업의 구조와 진행 경로의 사전 파악 가능
    - 기존 목표와 하위 작업 사이의 연관성 유지 용이
    - 장기 작업에서 목표 망각과 환각 감소 가능
  - 단점
    - 초기 계획을 실행 중에 수정하려면 별도의 조정 과정 필요
    - 앞 단계의 오류가 이후 단계까지 전달 가능
    - 예상하지 못한 환경 변화에 유연한 대응 어려움

[7] Huang, Xu, et al. "Understanding the planning of LLM agents: A survey." *arXiv preprint arXiv:2402.02716* (2024).

AI 에이전트 분류

- 교차 분해 방식(Interleaved Decomposition)
  - 정의
    - 하나의 하위 작업을 계획하고 실행 후, 그 결과를 바탕으로 다음 하위 작업을 결정하는 과정을 반복하는 방식
  - 특징
    - 작업 분해, 계획 및 실행이 번갈아 이루어짐
    - 현재 단계에서 필요한 한두 개의 작업만 결정
    - 실행 결과와 환경 피드백을 다음 작업 분해에 반영

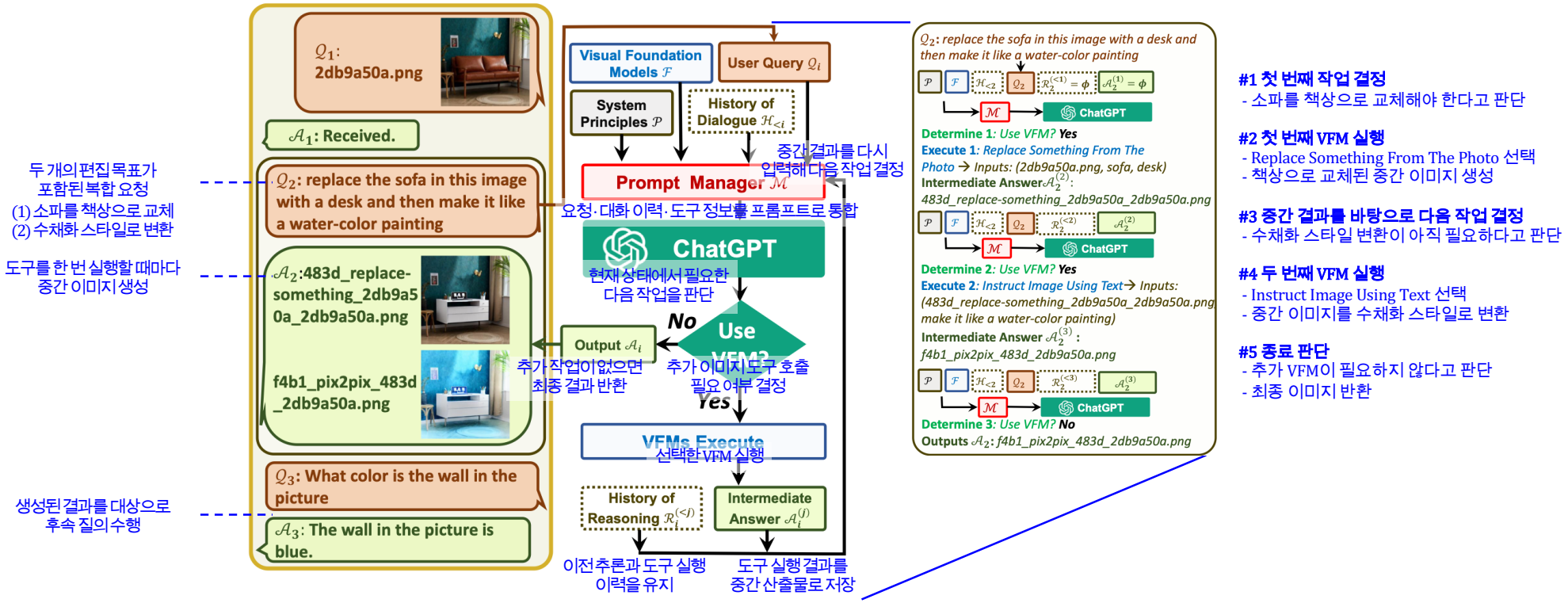


<그림 4.23> 교차 분해 방식의 반복적 작업 분해 및 하위 계획 과정[7]

[7] Huang, Xu, et al. "Understanding the planning of LLM agents: A survey." *arXiv preprint arXiv:2402.02716* (2024).

AI 에이전트 분류

- 교차 분해 방식(Interleaved Decomposition)
  - 활용 예시
    - Visual ChatGPT
      - 이미지 처리 결과를 확인하면서 다음 도구와 작업을 동적으로 선택



<그림 4.24> Visual ChatGPT 동작과정[9]

[9] Wu, Chenfei, et al. "Visual chatgpt: Talking, drawing and editing with visual foundation models." *arXiv preprint arXiv:2303.04671*(2023).

AI 에이전트 분류

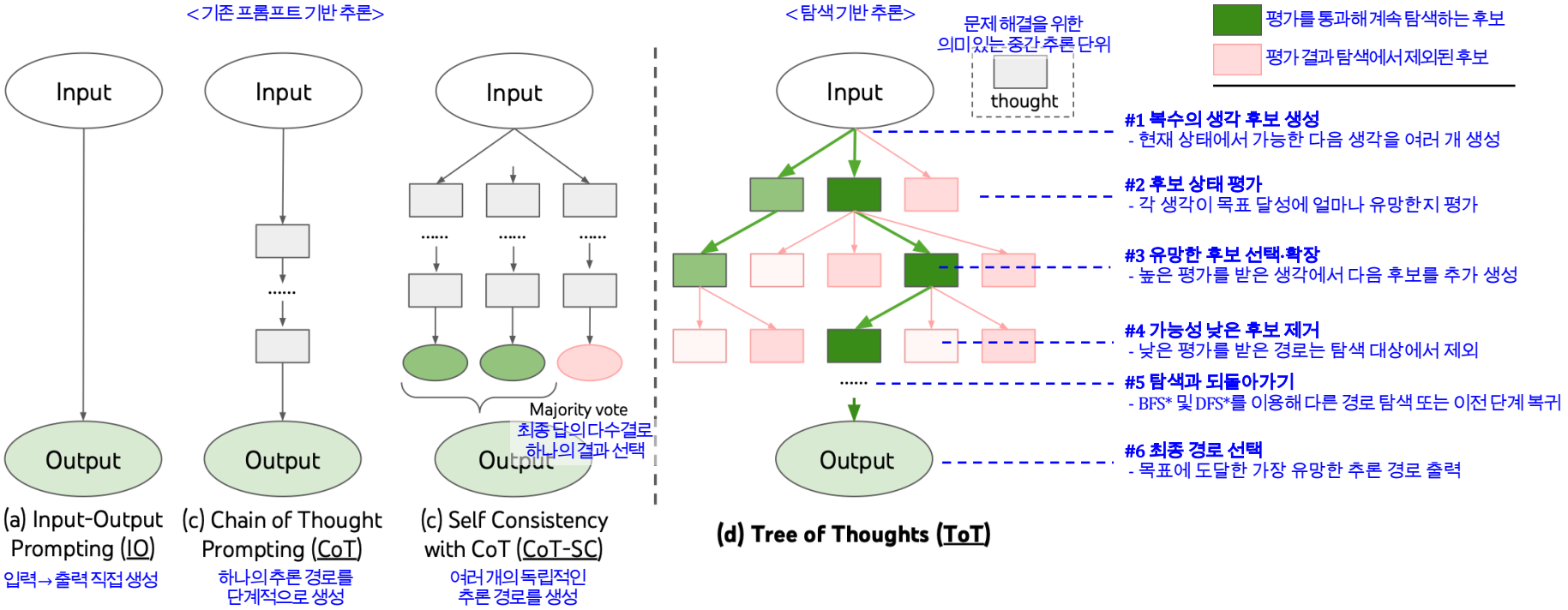
- 교차 분해 방식(Interleaved Decomposition)
  - 장점
    - 환경 변화에 따라 다음 계획을 동적으로 조정 가능
    - 실행 중 오류가 발생해도 이후 단계에서 복구 가능
    - 오류 허용성과 환경 적응성 높음
  - 단점
    - 복잡한 작업에서 실행 경로가 지나치게 길어질 수 있음
    - 긴 실행 과정에서 환각이 발생하거나 원래 목표에서 벗어날 수 있음
    - 반복적인 추론으로 계산 비용과 응답 시간이 증가할 수 있음

AI 에이전트 분류

- 다중 계획 선택(Multi-Plan Selection)
  - 정의
    - 하나의 목표에 대해 여러 계획 후보를 생성하고, 평가·탐색을 통해 가장 적합한 계획을 선택하는 방식
  - 특징
    - 계획 후보 생성과 최적 계획 선택의 두 단계로 구성
    - 샘플링, 프롬프트를 이용해 서로 다른 계획 경로 구성
    - 다수결, 점수 평가, 트리 탐색 등을 이용해 계획 후보 비교
    - 선택된 하나의 계획을 실제 실행 단계로 전달

* BFS(Breadth-First Search, 너비 우선 탐색)
  - 같은 단계의 후보를 먼저 모두 탐색한 후 다음 단계로 확장하는 방식
* DFS(Depth-First Search, 깊이 우선 탐색)
  - 하나의 후보 경로를 끝까지 탐색하고, 실패하면 이전 분기로 돌아가 다른 경로를 탐색하는 방식

- 다중 계획 선택(Multi-Plan Selection)
  - 활용 예시
    - Tree of Thoughts(ToT)
      - 여러 생각 후보를 트리 형태로 생성하고, 각 후보의 유망성을 평가하면서 가장 유망한 추론 경로를 탐색



<그림 4.25> 단일 추론에서 다중 경로 탐색으로의 확장[10]

[10] Yao, Shunyu, et al. "Tree of thoughts: Deliberate problem solving with large language models." *Advances in neural information processing systems* 36 (2023): 11809-11822.

AI 에이전트 분류

- 다중 계획 선택(Multi-Plan Selection)

* 단일 계획: 하나의 경로를 생성해 실행
* 다중 계획: 여러 경로를 생성·비교한 후 하나를 실행

  - 장점
    - 하나의 계획에 의존할 때 발생하는 오류와 불확실성 감소
    - 더 넓은 탐색 공간에서 실행 가능성이 높은 계획 찾기 가능
    - 복잡한 문제에서 다양한 해결 경로 비교 가능

  - 단점
    - 여러 계획을 생성·평가하여 모델 호출과 계산 비용 증가
    - 계획 후보가 많아지면 응답 시간과 토큰 사용량 증가
    - LLM의 평가 능력에 의존하므로 최적 계획 선택 보장 불가
    - 확률적 생성과 평가로 선택 결과의 일관성 감소

AI 에이전트

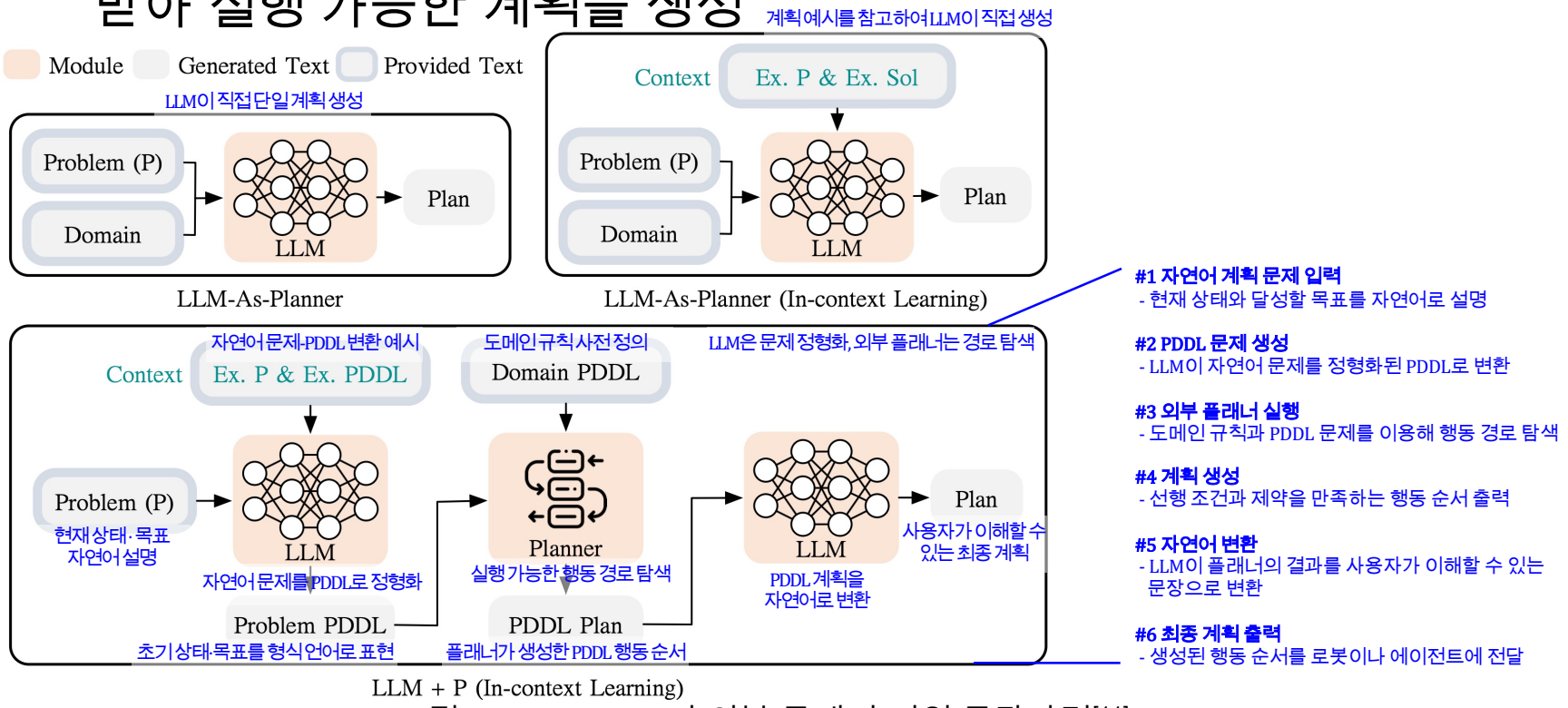
* 플래너(planner)
  • 목표 달성을 위한 행동 순서 생성 모듈
* 심볼릭 플래너
  • 형식 규칙과 탐색으로 계획을 생성하는 방식
* 뉴럴 플래너
  • 학습된 신경망으로 계획을 예측·생성하는 방식

- 외부 플래너 지원 계획(External Planner-Aided Planning)
  - 정의
    - LLM이 작업과 제약 조건을 정형화하고, 외부 플래너*의 지원을 받아 실행 가능한 계획을 생성하는 방식
  - 특징
    - LLM과 외부 플래너가 계획 수립 과정의 역할을 분담
    - LLM은 자연어 목표, 환경, 행동, 제약 조건을 형식 언어로 변환
    - 외부 플래너는 정형화된 문제에서 실행 가능한 경로 탐색
    - 생성된 계획을 에이전트가 수행할 수 있는 행동 순서로 변환
    - 심볼릭 플래너(symbolic planner)*, 신경망 기반 뉴럴 플래너(neural planner)* 지원 가능

AI 에이전트 분류

- 외부 플래너 지원 계획(External Planner-Aided Planning)
  - 활용 예시
    - LLM+P
      - LLM이 자연어 계획 문제를 PDDL*로 변환하고, 고전적 플래너의 지원을 받아 실행 가능한 계획을 생성

* PDDL(Planning Domain Definition Language)
  - 계획 문제의 상태·행동·목표·제약을 형식적으로 표현하는 언어



<그림 4.26> LLM+P의 외부 플래너 지원 동작과정[11]

[11] Liu, Bo, et al. "Llm+ p: Empowering large language models with optimal planning proficiency." *arXiv preprint arXiv:2304.11477*(2023).

AI 에이전트 분류

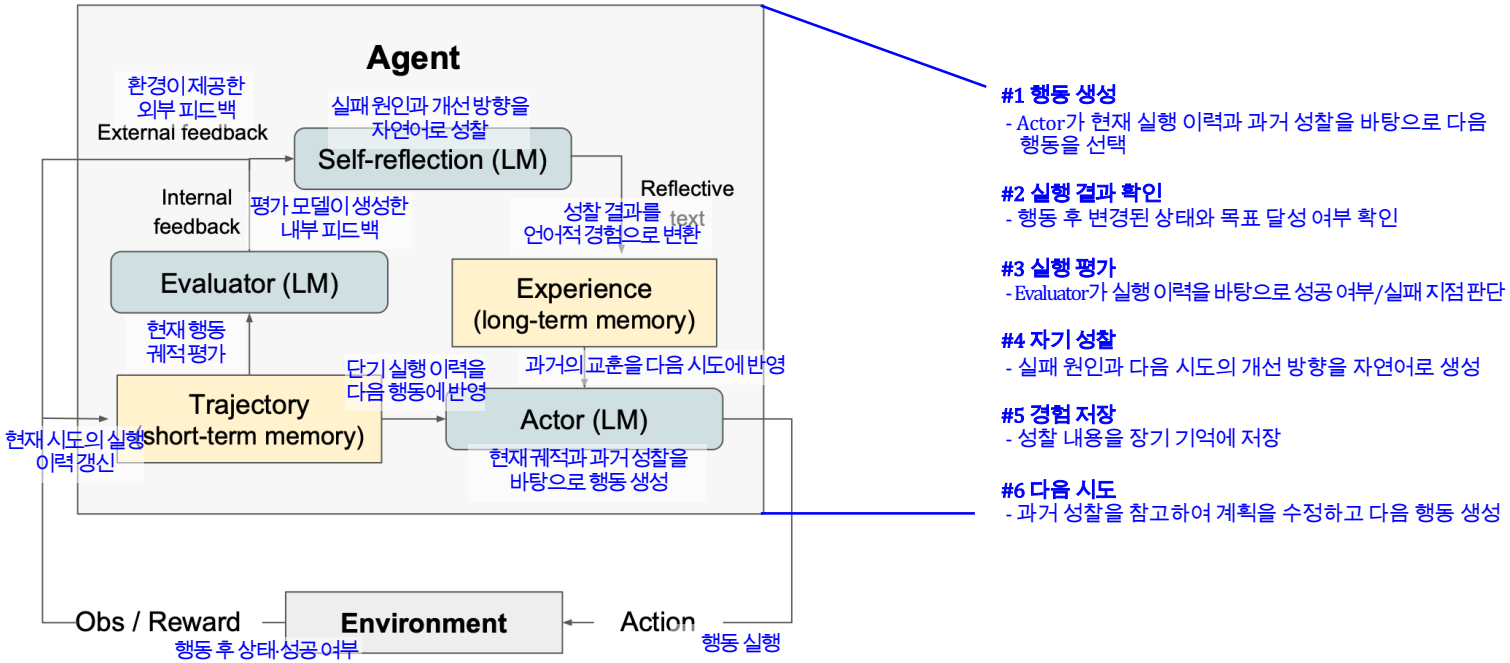
- 외부 플래너 지원 계획(External Planner-Aided Planning)
  - 장점
    - 행동의 선행 조건과 제약을 검사하여 실행 불가능한 계획 감소
    - 긴 계획에서 높은 정확도를 가진 행동 순서 생성 가능
    - 목적 함수가 정의되면 비용이 낮거나 최적인 경로 탐색 가능
    - LLM의 자연어 이해 능력과 전문 플래너 탐색 능력 결합
  - 단점
    - LLM이 작업을 잘못 정형화하면 외부 플래너도 잘못된 계획 생성
    - 사전에 정의되지 않은 행동·예외 상황 처리 어려움
    - 환경이 변하면 문제 재정형화 및 계획 재수립 필요
    - LLM과 외부 플래너 연결을 위한 추가 시스템 구성 필요

* 성찰(Reflection)
  • 실행 결과를 평가하고 실패 원인을 분석하는 과정
* 개선(Refinement)
  • 성찰 결과를 반영해 계획이나 행동을 수정하는 과정

- 성찰 및 개선(Reflection and Refinement)
  - 정의
    - 계획 실행 결과와 피드백을 바탕으로 오류 원인을 성찰*하고, 기존 계획이나 행동을 반복적으로 수정하는 방식
  - 특징
    - '계획 수립 → 실행 → 결과 관찰 → 성찰 → 계획 수정' 순환구조
    - 환경·도구·평가 모델 또는 자기 평가로부터 피드백 수집
    - 성찰 단계에서 실패 원인과 개선* 방향을 자연어로 생성
    - 개선 단계에서 전체 계획을 재수립 또는 실패한 일부 단계만 수정
    - 목표를 달성하거나 종료 조건을 만족할 때까지 반복

AI 에이전트 분류

- 성찰 및 개선(Reflection and Refinement)
  - 활용 예시
    - Reflexion
      - 실행 결과를 평가하여 실패 원인을 자연어로 성찰하고, 이를 메모리에 저장해 다음 시도의 계획과 행동을 개선



<그림 4.27> Reflexion의 언어적 성찰을 통한 반복적 계획 개선 과정[12]

[12] Shinn, Noah, et al. "Reflexion: Language agents with verbal reinforcement learning." *Advances in neural information processing systems* 36 (2023): 8634-8652.

* 자기 평가(Self-evaluation)
  • 모델이 자신의 계획이나 실행 결과를 직접 판단하는 과정

AI 에이전트 핵심

- 성찰 및 개선(Reflection and Refinement)
  - 장점
    - 실행 결과를 반영하여 초기 계획의 오류를 발견·수정 가능
    - 환경 변화, 예상하지 못한 상황에 맞추어 계획을 동적으로 조정
    - 전체 계획 폐기 없이 실패한 단계만 수정하여 기존 성과 유지
    - 별도의 모델 학습 없이 반복적인 피드백으로 계획 성공률 향상
  - 단점
    - 자기 평가가 부정확할 경우 잘못된 원인 도출에 따른 계획 악화
    - 피드백 부족 및 모호성 발생 시 실제 실패 원인 식별 어려움
    - 반복적인 생성/평가/수정으로 모델 호출 횟수 및 계산 비용 증가

AI 에이전트 분류

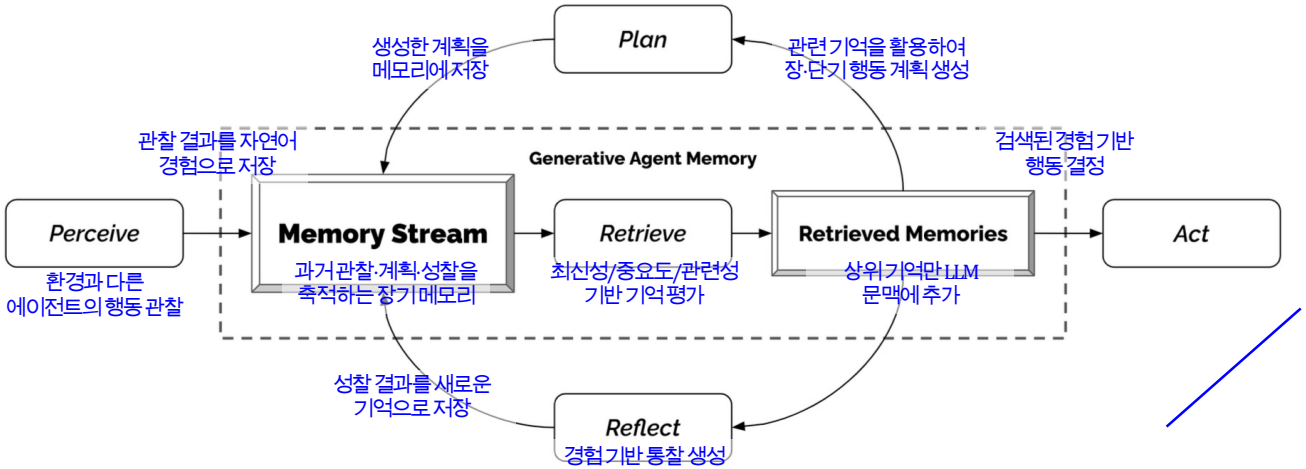
- 메모리 증강 계획(Memory-Augmented Planning)
  - 정의
    - 과거 경험과 지식을 메모리에 저장하고, 현재 작업과 관련된 정보를 검색하여 계획 수립에 활용하는 방식

  - 특징
    - 성공·실패 경험, 행동 궤적 등을 메모리에 저장
    - RAG* 구조로 현재 작업과 관련된 기억을 검색하고, 이를 프롬프트에 추가하여 계획과 행동 생성을 지원
    - 새로운 경험을 지속적으로 저장·갱신하여 장기적인 성능 향상 지원

* 검색 증강 생성(RAG, Retrieval-augmented generation)
  - 외부 저장소에서 관련 정보를 검색해 LLM 입력에 추가하는 방식

AI 에이전트 분류

- 메모리 증강 계획(Memory-Augmented Planning)
  - 활용 예시
    - Generative Agents
      - 관찰과 경험을 자연어 메모리로 저장하고, 현재 상황과 관련된 기억을 검색하여 계획과 행동 생성에 활용



#1 환경 관찰
- 주변 상황과 다른 에이전트의 행동 인식

#2 경험 저장
- 관찰 결과를 자연어로 Memory Stream에 기록

#3 RAG 기반 기억 검색
- 현재 상황과 목표를 질의로 사용해 관련된 과거 경험 검색

#4 프롬프트 문맥 증강
- 최신성·중요도·관련성이 높은 기억을 LLM 입력에 추가

#5 계획 및 행동 생성
- 증강된 문맥을 바탕으로 계획과 다음 행동 결정

#6 메모리 갱신
- 새로운 계획·행동·성찰을 다시 메모리에 저장

<그림 4.28> Generative Agent의 RAG형 경험 검색 및 계획 생성 구조[13]<그림 4.29> Generative Agents 행동 사례[13]

[13] Park, Joon Sung, et al. "Generative agents: Interactive simulacra of human behavior." *Proceedings of the 36th annual acm symposium on user interface software and technology*. 2023.

* 메모리 오염(Memory Contamination)
  • 부정확하거나 불필요한 정보가 누적되어 이후 판단에 영향을 주는 현상

AI 에이전트 분류

- 메모리 증강 계획(Memory-Augmented Planning)
  - 장점
    - 성공 경험을 재사용하여 효율적인 계획 수립 가능
    - 실패 경험을 참고하여 동일한 오류 반복 방지
    - RAG 기반 외부 메모리를 활용하여 LLM의 제한된 문맥 길이 보완
    - 모델을 재학습 없이 새로운 경험과 지식 지속 추가 가능
  - 단점
    - 관련 없는 기억이 검색되면 현재 작업에 부적절한 계획 수립 가능
    - 메모리 오염* 문제 발생 가능
    - 메모리가 증가할수록 비용 및 프롬프트 토큰 사용량 증가
    - 사용자 정보 및 행동 이력 저장에 따른 개인정보보호 문제 발생

AI 에이전트

* 작업 분해 시점 방식
  - 작업을 언제 분해하고 다음 단계를 결정하는지에 따른 구분
* 계획 성능 보완 방식
  - 작업 분해만으로 해결하기 어려운 계획의 오류와 불확실성 보완 방식

- 정리
  - 작업 분해 시점 방식과 계획 성능 보완 방식은 대체 관계가 아니며, 에이전트의 목적과 환경에 따라 함께 적용할 수 있음

[표 4.1] AI 에이전트 작업 계획 방식별 특징 비교

구분	방식	핵심 원리	장점	단점	활용 예시
작업 분해 시점	선분해 방식	실행 전에 전체 작업을 하위 작업으로 분해	전체 구조와 작업 순서 파악 가능	초기 계획 오류와 환경 변화에 취약	HuggingGPT[8]
	교차 분해 방식	한 작업을 실행한 후 다음 작업을 동적으로 결정	실행 결과와 환경 변화 즉시 반영 가능	높은 반복 호출 비용 및 전체 계획의 일관성 감소	Visual ChatGPT[9]
계획 성능 보완	다중 계획 선택	여러 계획 후보를 생성 및 평가하여 하나를 선택	다양한 해결 경로 비교 통해 단일 계획 오류 감소	높은 후보 생성/평가 비용 및 평가결과에 의존	Tree of Thoughts[10]
	외부 플래너 지원 계획	정형화된 문제를 외부 플래너로 탐색·검증	규칙과 제약을 만족하는 실행 가능한 계획 생성	자연어의 정형화 오류와 외부 플래너 의존	LLM+P[11]
	성찰 및 개선	실행 결과를 평가하고 실패 원인을 반영한 계획 수정	실행 과정에서 오류를 발견하고 반복적 개선 가능	잘못된 평가 반영 가능성 및 반복 비용 증가	Reflexion[12]
	메모리 증강 계획	관련된 과거 경험을 검색하여 계획 생성에 활용	성공/실패 경험을 재사용하여 장기적 계획 개선 가능	부적절한 기억 검색 및 메모리 오염 가능성	Generative Agents[13]

[8] Shen, Yongliang, et al. "Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face." *Advances in Neural Information Processing Systems* 36 (2023): 38154-38180.
[9] Wu, Chenfei, et al. "Visual chatgpt: Talking, drawing and editing with visual foundation models." *arXiv preprint arXiv:2303.04671*(2023).
[10] Yao, Shunyu, et al. "Tree of thoughts: Deliberate problem solving with large language models." *Advances in neural information processing systems* 36 (2023): 11809-11822.
[11] Liu, Bo, et al. "Llm+ p: Empowering large language models with optimal planning proficiency." *arXiv preprint arXiv:2304.11477*(2023).
[12] Shinn, Noah, et al. "Reflexion: Language agents with verbal reinforcement learning." *Advances in neural information processing systems* 36 (2023): 8634-8652.

[13] Park, Joon Sung, et al. "Generative agents: Interactive simulacra of human behavior." *Proceedings of the 36th annual acm symposium on user interface software and technology*. 2023.

목 차

- 두뇌, 지각, 행동 패러다임
- AI 에이전트 분류
- 단일 에이전트와 다중 에이전트
- 주요 라이브러리

단일 에이전트와 다중 에이전트

- 에이전트 시스템
  - 구분
    - 목표 수행에 참여하는 에이전트의 수와 상호작용 구조 기준
  - 종류
    - 단일 에이전트 시스템
      - 하나의 에이전트가 환경을 인식하고 계획·행동을 수행
    - 다중 에이전트 시스템
      - 여러 에이전트가 정보를 교환하며 공동 또는 경쟁적 목표 수행

* 배치(Deployment)
  - 에이전트를 특정 목표와 환경에 투입하여 운용하는 형태

단일 에이전트와 다중 에이전트

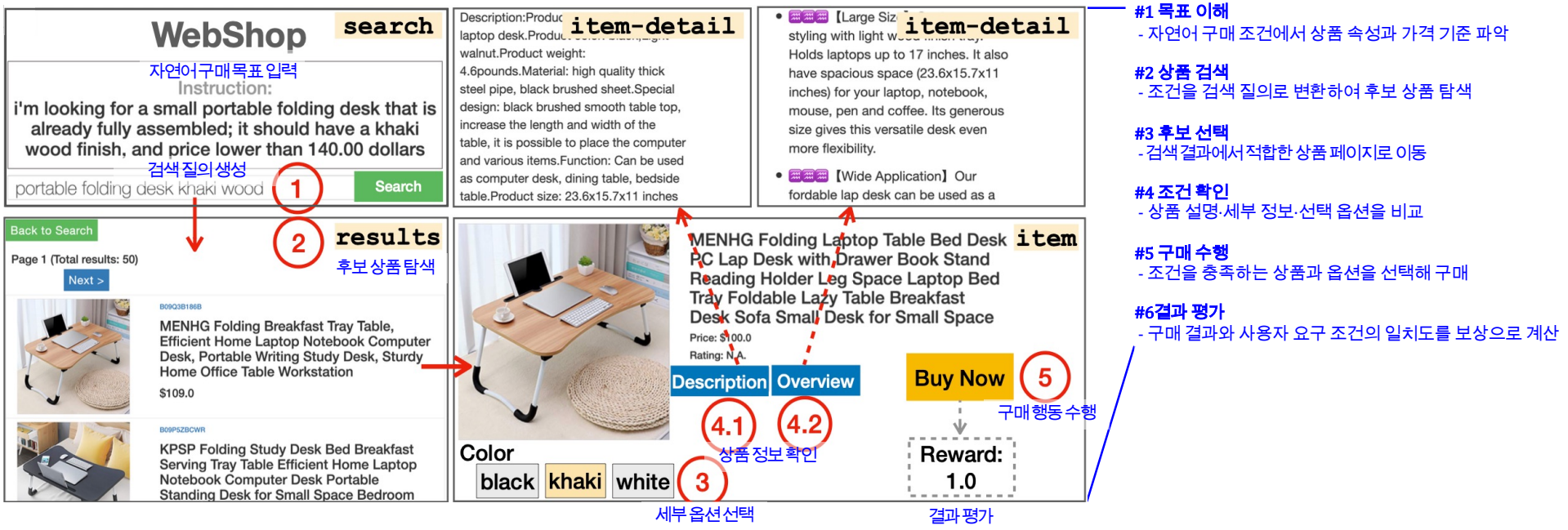
- 단일 에이전트 시스템
  - 정의
    - 하나의 에이전트가 환경을 인식하고, 목표 달성을 위한 계획과 행동을 독립적으로 수행하는 시스템
  - 특징
    - 하나의 에이전트가 계획·행동·결과 평가를 통합 관리
    - 여러 도구 및 외부 모델 사용 환경이어도 의사결정 주체가 하나이면 단일 에이전트에 해당함
  - 종류
    - 작업 지향 배치(task-oriented deployment)
    - 혁신 지향 배치(innovation-oriented deployment)
    - 생애 주기 배치(lifecycle-oriented deployment)

단일 에이전트와 다중 에이전트

- 작업 지향 배치(Task-Oriented Deployment)
  - 정의
    - 사용자가 제시한 명확한 목표나 지시를 분석하고, 필요한 작업을 순차적으로 수행하여 결과를 제공하는 배치 형태
  - 특징
    - 목표 달성에 필요한 작업을 하위 작업으로 분해
    - 검색 엔진·웹사이트 등의 외부 도구 활용
    - 웹 환경과 생활 환경의 구체적인 작업 수행에 주로 활용
    - ‘작업을 어떻게 분해하는가’보다 에이전트를 구체적인 목표 달성에 활용하는 배치 목적을 가짐

단일 에이전트와 다중 에이전트

- 작업 지향 배치(Task-Oriented Deployment)
  - 활용 예시
    - WebShop
      - 자연어 구매 조건을 이해하고 전자상거래 환경에서 검색·비교·구매를 수행하는 작업 지향 에이전트



<그림 4.30> WebShop의 작업 지향 구매 과업 수행 과정[14]

[14] Yao, Shunyu, et al. "Webshop: Towards scalable real-world web interaction with grounded language agents." *Advances in Neural Information Processing Systems* 35 (2022): 20744-20757.

단일 에이전트와 다중 에이전트

- 작업 지향 배치(Task-Oriented Deployment)
  - 장점
    - 목표와 성공 조건이 명확하여 실행 결과 평가 용이
    - 반복적이고 정형화된 작업 자동화 가능
    - 외부 도구를 활용하여 LLM의 지식·행동 범위 확장
    - 사용자의 세부 지시 없이도 하위 작업을 자율 수행
  - 단점
    - 모호하거나 범위를 벗어난 요청에 대한 대응 어려움
    - 초기 작업 분해나 도구 선택 오류의 전파 가능성
    - 웹사이트 변경·도구 오류 등 예상하지 못한 환경 변화에 취약
    - 복잡한 장기 작업에서 문맥과 진행 상태 유지 어려움

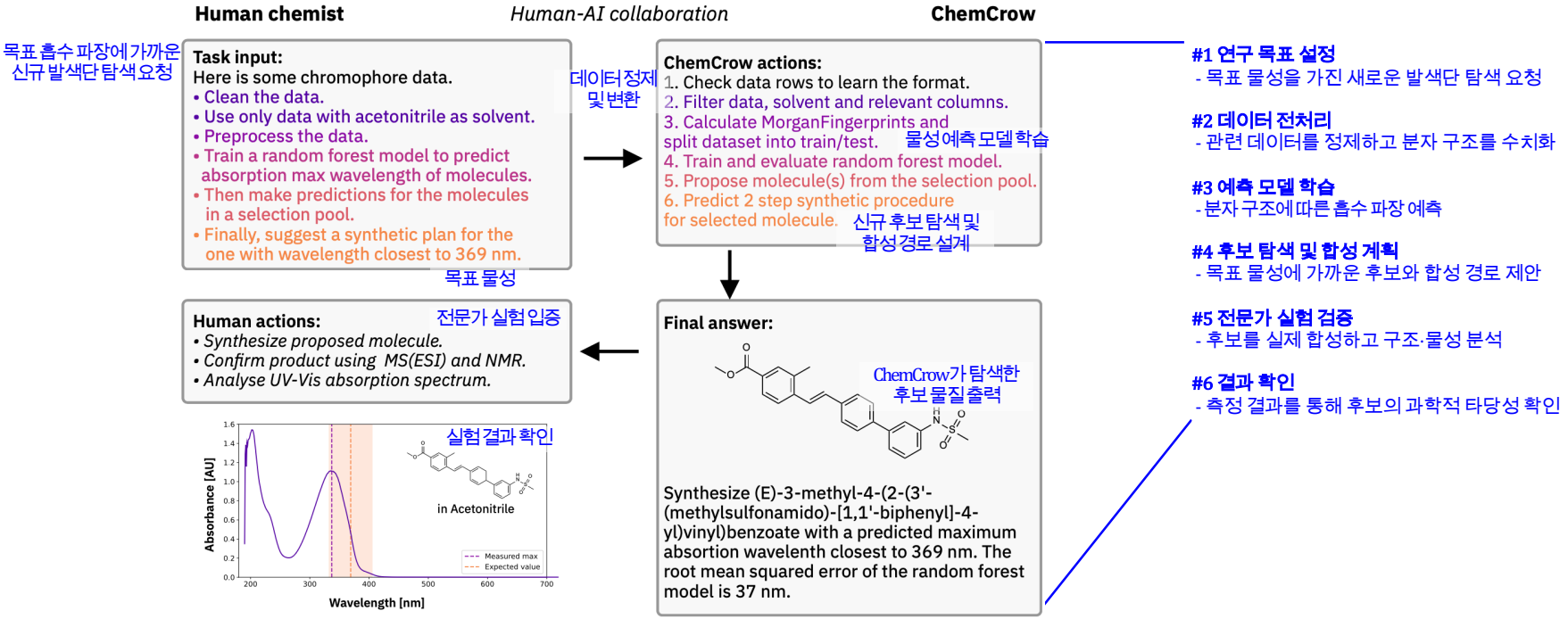
단일 에이전트와 다중 에이전트

- 혁신 지향 배치(Innovation-Oriented Deployment)
  - 정의
    - 에이전트가 정해진 작업의 반복 수행을 넘어, 해결책을 탐색하고 새로운 아이디어를 생성하도록 배치하는 형태
  - 특징
    - 목표는 제시하나 해결 절차와 정답은 사전 정의되지 않음
    - 문헌 검색·가설 생성·실험 설계 등 반복 통해 해결책 탐색
    - 생성된 결과를 평가하고 피드백을 통해 탐색 방향 수정
    - 연구·신약·소재 등 개발에 활용

단일 에이전트와 다중 에이전트

* 합성 경로(Synthetic Route)
  - 원료 물질을 목표 화합물로 변환하는 화학 반응 단계의 순서

- 혁신 지향 배치(Innovation-Oriented Deployment)
  - 활용 예시
    - ChemCrow
      - 전문 도구와 데이터를 활용해 새로운 분자 후보와 합성 경로*를 탐색하는 혁신 지향 에이전트



<그림 4.31> ChemCrow 기반 신규 발색단 후보 및 검증 과정[15]

[15] M. Bran, Andres, et al. "Augmenting large language models with chemistry tools." *Nature machine intelligence* 6.5 (2024): 525-535.

단일 에이전트와 다중 에이전트

* 해결 공간(Solution Space)
  • 문제를 해결하기 위해 고려할 수 있는 모든 해결 방법과 후보의 범위

- 혁신 지향 배치(Innovation-Oriented Deployment)
  - 장점
    - 사람이 직접 검토하기 어려운 넓은 해결 공간*을 빠르게 탐색
    - 참고문헌 및 도구를 통합한 연구 과정 지원 체계
    - 기존 지식의 새로운 조합을 통한 아이디어 발굴 지원
  - 단점
    - 생성된 아이디어의 신규성과 타당성 보장 한계
    - 부정확한 지식 또는 도구 사용 결과의 오류 초래
    - 결과 검증에 전문 지식과 실제 실험 필요

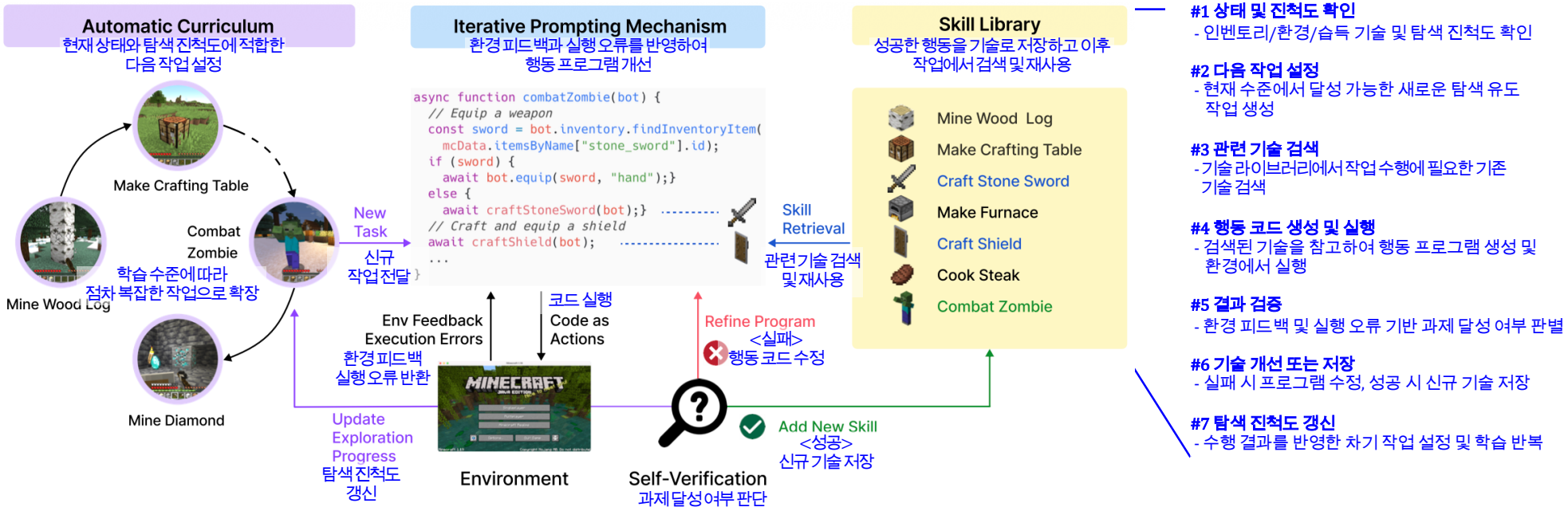
단일 에이전트와 다중 [illegible]

* 개방형 환경(Open-ended Environment)
  • 정해진 최종 목표 없이 새로운 상태나 작업을 지속적으로 탐색할 수 있는 환경

- 생애주기 배치(Lifecycle-Oriented Deployment)
  - 정의
    - 에이전트가 개방형 환경*에서 장기간 활동하며 경험과 기술을 지속적으로 학습하도록 배치하는 형태
  - 특징
    - 하나의 고정된 과업보다 지속적인 학습과 능력 확장 목표
    - 현재 상태와 학습 수준에 따라 새로운 목표 및 작업 설정
    - 저장된 기술을 검색 및 조합하여 새로운 작업에 활용
    - 모델 재학습 과정 없이 경험 축적을 통한 수행 범위 확장

단일 에이전트와 다중 에이전트

- 생애주기 배치(Lifecycle-Oriented Deployment)
  - 활용 예시
    - Voyager
      - 개방형 환경(e.g., 마인크래프트)을 계속 탐색하고 학습한 기술을 축적·재사용하며 능력을 확장하는 생애주기 지향 에이전트



<그림 4.32> Voyager의 생애주기 지향 학습 과정[6]

[6] Wang, Guanzhi, et al. "Voyager: An open-ended embodied agent with large language models." *arXiv preprint arXiv:2305.16291*(2023).

단일 에이전트와 다중 에이전트

- 생애주기 배치(Lifecycle-Oriented Deployment)
  - 장점
    - 장기간 경험 축적을 통해 수행 능력 지속적 향상
    - 학습한 기술 재사용 및 조합을 통한 새로운 작업 대응
    - 모델 재학습 없이 지식과 기술 확장 가능
  - 단점
    - 잘못 학습한 행동 및 정보의 장기간 누적 발생 문제
    - 메모리 및 기술 라이브러리 증가에 따른 비용 증가
    - 장기간 실행에 따른 모델 호출 비용 및 응답시간 증가

단일 에이전트와 다중 에이전트

- 단일 에이전트 시스템 정리
  - 작업·혁신·생애주기 지향 배치는 에이전트가 추구하는 성과와 운용 기간에 따른 구분이며, 실제 시스템에서 함께 적용 가능

[표 4.2] 단일 에이전트 배치 목적별 특징 비교

방식	핵심 원리	장점	단점	활용 예시
작업 지향 배치	명확한 목표와 종료 조건이 있는 특정 작업의 완료에 집중	목표와 평가 기준이 명확하며 안정적인 작업 수행 가능	새로운 환경에 대한 일반화 및 장기적 경험 축적 제한	WebShop[14]
혁신 지향 배치	전문 도구와 데이터를 활용하여 새로운 해결 후보 탐색 및 검증	기존 해법을 넘어 다양한 후보 및 새로운 해결책 발견 가능	결과의 불확실성이 높고 전문적 검증에 많은 비용 소요	ChemCrow[15]
생애주기 지향 배치	장기간 환경과 상호작용하며 경험과 기술을 축적 및 재사용	학습 경험 누적에 따른 적응력 및 수행 능력 지속적 향상	오류 누적 가능성 및 장기 운용·메모리 관리 비용 증가	Voyager [6]

[14] Yao, Shunyu, et al. "Webshop: Towards scalable real-world web interaction with grounded language agents." *Advances in Neural Information Processing Systems* 35 (2022): 20744-20757.

[15] M. Bran, Andres, et al. "Augmenting large language models with chemistry tools." *Nature machine intelligence* 6.5 (2024): 525-535.

[6] Wang, Guanzhi, et al. "Voyager: An open-ended embodied agent with large language models." *arXiv preprint arXiv:2305.16291*(2023).

단일 에이전트와 다중 에이전트

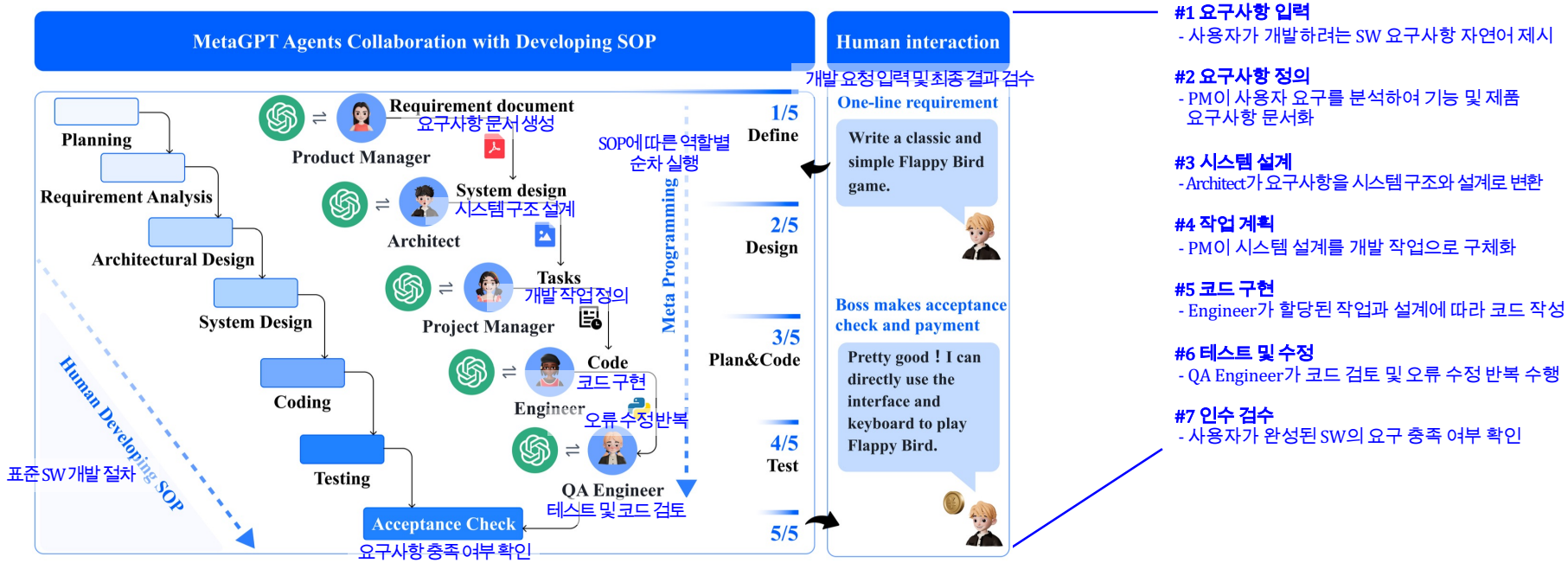
- 다중 에이전트 시스템
  - 정의
    - 여러 에이전트가 정보를 교환하고 상호작용하며 공동 또는 상반된 목표를 수행하는 시스템
  - 특징
    - 서로 다른 역할과 능력을 가진 에이전트 간 분업 가능
    - 자연어로 의견·정보·실행 결과 교환
    - 각 에이전트의 결과를 종합하여 집단적 의사결정 수행
  - 종류
    - 에이전트 – 에이전트 상호작용
      - 협력적 상호작용, 적대적 상호작용
    - 인간 – 에이전트 상호작용
      - 불균등 상호작용, 균등 상호작용

단일 에이전트와 다중 에이전트

- 협력적 상호작용(Cooperative Engagement)
  - 정의
    - 여러 에이전트가 공동 목표를 달성하기 위해 역할 분담 및 정보와 작업 결과를 공유하며 협력하는 방식
  - 특징
    - 에이전트별 역할 및 능력에 따라 작업 분담
    - 개별 결과를 종합하여 최종 결론 및 행동 결정
  - 종류
    - 질서형 상호작용(ordered interaction)
      - 사전에 정의한 역할·규칙에 따라 에이전트 간 협력
    - 비질서형 상호작용(disordered interaction)
      - 고정된 순서 없이 에이전트 간 자유로운 의견 교환

단일 에이전트와 다중 에이전트

- 질서형 상호작용(Ordered Interaction)
  - 활용예시
    - MetaGPT
      - 정해진 역할과 표준 작업 절차에 따라 요구사항 분석부터 설계·구현·검증까지 순차적으로 수행하는 질서형 에이전트

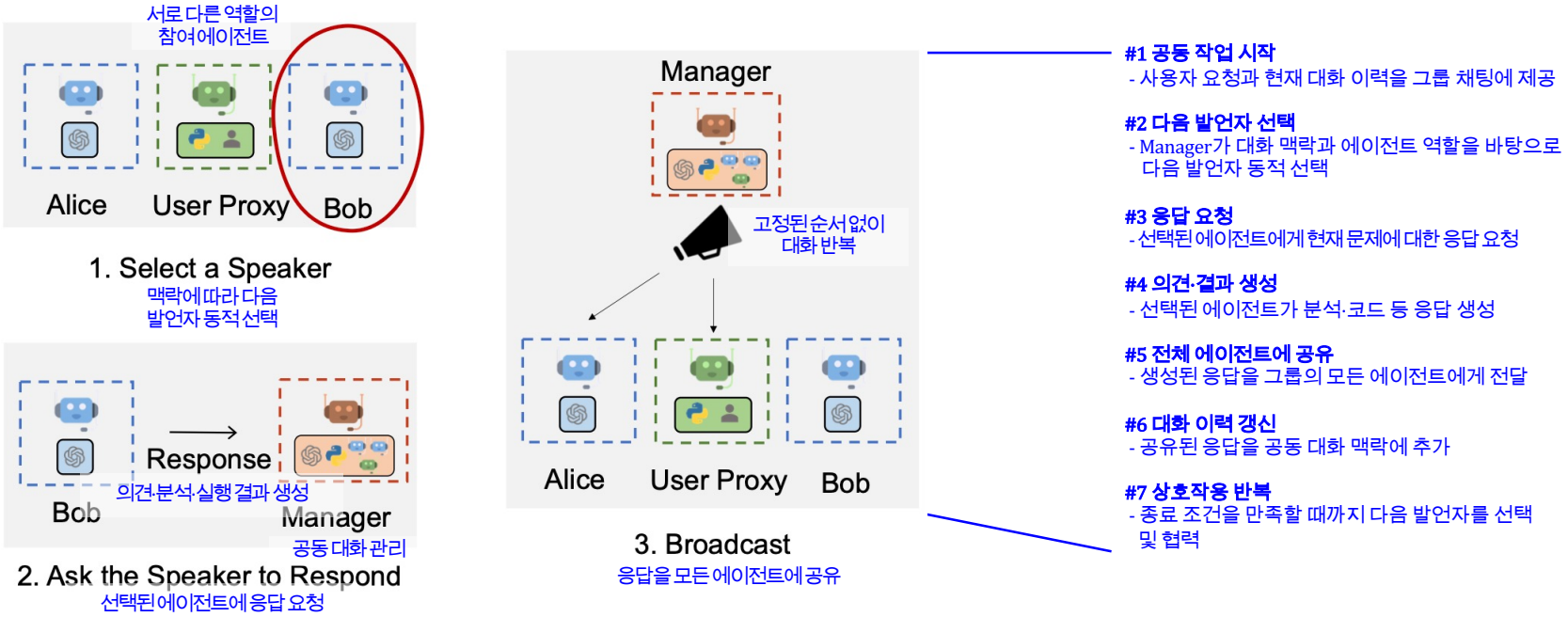


<그림 4.33> MetaGPT의 SOP 기반 질서형 다중 에이전트 협업 과정[16]

[16] Hong, Sirui, et al. "MetaGPT: Meta programming for a multi-agent collaborative framework." *International Conference on Learning Representations*. Vol. 2024. 2024.

단일 에이전트와 다중 에이전트

- 비질서형 상호작용(Disordered Interaction)
  - 활용예시
    - AutoGen Dynamic Group Chat
      - 고정된 발언 순서 없이 대화 맥락에 따라 다음 에이전트를 선택하고, 의견과 결과를 공유 및 협력하는 비질서형 다중 에이전트



<그림 4.34> AutoGen의 동적 발언자 선택 기반 비질서형 다중 에이전트 협업 과정[17]

[17] Wu, Qingyun, et al. "Autogen: Enabling next-gen llm applications via multi-agent conversation." *arXiv preprint arXiv:2308.08155*(2023).

단일 에이전트와 다중 에이전트

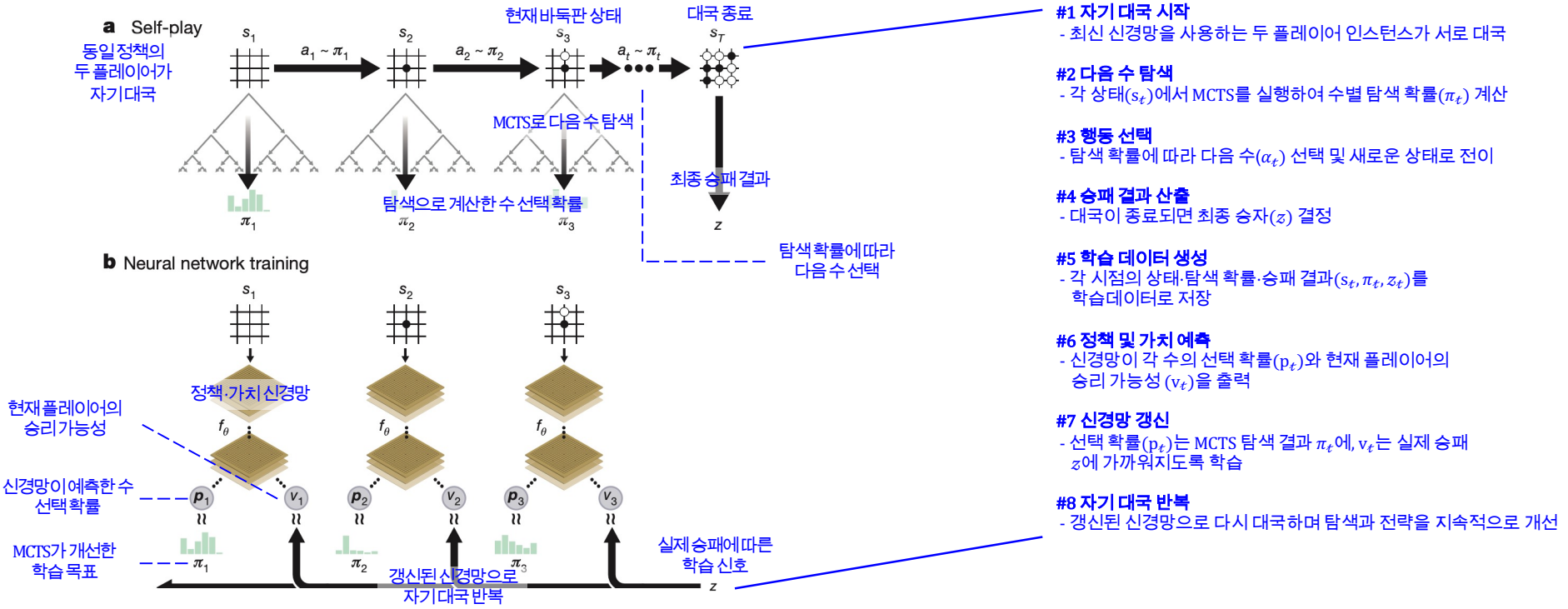
* 에이전트 인스턴스(Agent Instance)
  • 동일한 모델 혹은 정책을 기반으로 생성되어 각각 독립적으로 행동하는 에이전트 실행 단위

- 적대적 상호작용(Adversarial Engagement)
  - 정의
    - 여러 에이전트가 서로 상반된 목표 또는 보상 구조에서 경쟁하고, 그 결과를 바탕으로 전략과 성능을 개선하는 방식
  - 특징
    - 한 에이전트의 목표 달성이 다른 에이전트의 실패 의미
    - 승패·점수·비판 결과 등을 피드백으로 활용
    - 동일한 정책을 활용하는 에이전트 인스턴스* 혹은 상호 다른 에이전트 간 경쟁 가능
    - 경쟁을 반복하며 전략적 취약점 발견 및 성능 향상 기대

단일 에이전트와 다중

* 자기 대국(Self-Play)
  - 동일한 정책을 사용하는 플레이어 인스턴스 간 경쟁하며 학습 경험을 생성하는 방식
* MCTS(Monte Carlo Tree Search)
  - 가능한 수의 결과를 반복적 탐색을 통해 유망한 행동을 선택하는 기법

- 적대적 상호작용(Adversarial Engagement)
  - 활용예시
    - AlphaGo Zero
      - 동일한 신경망을 공유하는 흑·백 플레이어 인스턴스가 자기 대국을 반복하고, 바둑 전략을 스스로 개선하는 적대적 상호작용 에이전트



#1 자기 대국 시작
- 최신 신경망을 사용하는 두 플레이어 인스턴스가 서로 대국

#2 다음 수 탐색
- 각 상태(s_t)에서 MCTS를 실행하여 수별 탐색 확률(π_t) 계산

#3 행동 선택
- 탐색 확률에 따라 다음 수(α_t) 선택 및 새로운 상태로 전이

#4 승패 결과 산출
- 대국이 종료되면 최종 승자(z) 결정

#5 학습 데이터 생성
- 각 시점의 상태·탐색 확률·승패 결과(s_t, π_t, z_t)를 학습데이터로 저장

#6 정책 및 가치 예측
- 신경망이 각 수의 선택 확률(p_t)와 현재 플레이어의 승리 가능성 (v_t)을 출력

#7 신경망 갱신
- 선택 확률(p_t)는 MCTS 탐색 결과 π_t에, v_t는 실제 승패 z에 가까워지도록 학습

#8 자기 대국 반복
- 갱신된 신경망으로 다시 대국하며 탐색과 전략을 지속적으로 개선

<그림 4.35> AlphaGo Zero의 자기 대국 기반 적대적 상호작용 및 학습 과정[18]

[18] Silver, David, et al. "Mastering the game of go without human knowledge." *nature* 550.7676 (2017): 354-359.

단일 에이전트와 다중 에이전트

- 불균등 상호작용(Unequal Interaction)
  - 정의
    - 인간과 에이전트 사이에 권한과 책임의 위계가 존재하며, 일반적으로 인간이 목표와 최종 결정을 통제하는 방식
  - 특징
    - 인간이 목표·제약 조건을 제시하고 작업 위임
    - 에이전트는 지시에 따라 정보 제공 또는 작업 수행
    - 인간이 실행 결과 검토 및 피드백 수행
    - 에이전트를 비서 등 의사결정 지원 도구로 활용

단일 에이전트와 다중

* 어포던스(affordance)
  - 현재 환경과 로봇 상태에서 특정 행동을 수행할 수 있는 가능성

- 불균등 상호작용(Unequal Interaction)
  - 활용 예시
    - SayCan
      - 인간이 자연어로 상위 목표 지시하고 에이전트가 실행 가능한 로봇 기술을 선택 및 수행하는 인간 주도형 불균등 상호작용 에이전트

<그림 4.36> SayCan의 인간 지시 기반 로봇 기술 선택 과정[19]

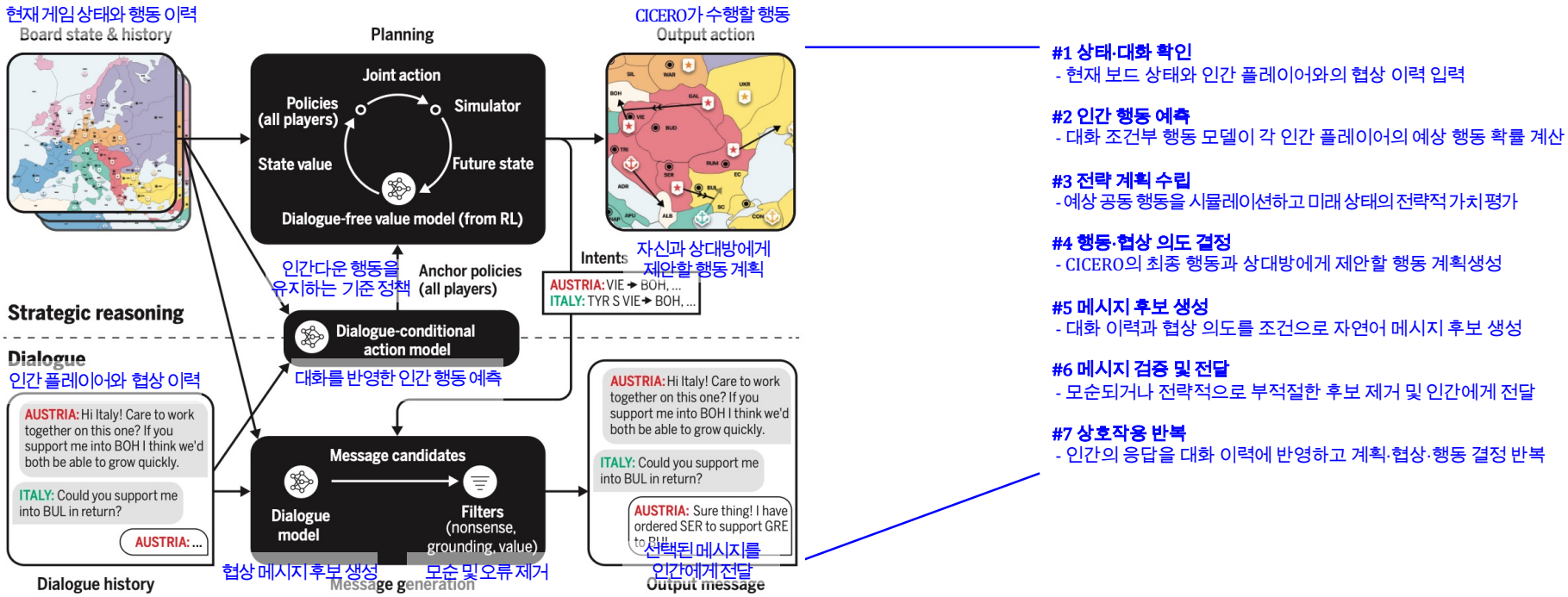
[19] Ahn, Michael, et al. "Do as i can, not as i say: Grounding language in robotic affordances." *arXiv preprint arXiv:2204.01691* (2022).

단일 에이전트와 다중 에이전트

- 균등 상호작용(Equal Interaction)
  - 정의
    - 인간과 에이전트가 독립적인 참여자로서 의견을 제안 및 평가, 상호 조정을 통한 결론을 도출하는 방식
  - 특징
    - 인간과 에이전트가 모두 작업을 능동적으로 제안
    - 서로의 의견과 결과를 평가하고 피드백 교환
    - 대화 과정에서 역할과 주도권이 동적으로 전환
    - 에이전트를 단순한 도구가 아닌 협업 파트너로 활용

단일 에이전트와 다중 에이전트

- 균등 상호작용(Equal Interaction)
  - 활용 예시
    - CICERO
      - 인간과 동일한 게임 규칙과 플레이어 권한을 가지고, 자연어 협상과 전략적 행동 결정을 수행하는 균등 상호작용 에이전트



<그림 4.37> CICERO의 인간-AI 협상 및 전략 결정 과정[20]

[20] Meta Fundamental AI Research Diplomacy Team (FAIR)†, et al. "Human-level play in the game of Diplomacy by combining language models with strategic reasoning." *Science* 378.6624 (2022): 1067-1074.

단일 에이전트와 다중 에이전트

- 다중 에이전트 시스템 정리
  - 에이전트 간은 목표·관계·협력 구조, 인간–에이전트 간은 권한 배분에 따른 구분이며, 여러 유형이 함께 나타날 수 있음

[표 4.3] 다중 에이전트 상호작용 방식별 특징 비교

구분	방식	핵심 원리	장점	단점	활용 예시
에이전트–에이전트	협력적 상호작용 (질서형 협력)	정해진 역할·순서에 따른 협력	작업 과정의 명확성, 결과 재현 용이	변화 대응 어려움 및 오류 전파 가능성 존재	MetaGPT[16]
에이전트–에이전트	협력적 상호작용 (비질서형 협력)	맥락에 따른 동적 상호작용	특화 에이전트 선택을 통한 협력 유연성 확보	상호작용 충돌 가능성 및 실행 비용 증가	AutoGen Dynamic Group Chat[17]
에이전트–에이전트	적대적 상호작용	상반된 목표 아래 반복 경쟁	취약점 발견 및 전략 및 성능의 견고성 향상	반복 경쟁에 따른 연산 비용 증가 및 승패 목표 전략 편향 가능	AlphaGo Zero[18]
인간–에이전트	불균등 상호작용	인간 지시·감독, 에이전트 수행	통제 및 책임 명확, 안전한 작업 위임 가능	인간의 지시로 인한 에이전트 자율성 제한	SayCan[19]
인간–에이전트	균등 상호작용	인간과 에이전트가 독립적으로 제안·협상 후 행동결정	공동 주도형 의사결정 가능	의견 충돌 및 조정 비용 발생에 따른 책임 소재 불명확성 존재	CICERO[20]

[16] Hong, Sirui, et al. "MetaGPT: Meta programming for a multi-agent collaborative framework." *International Conference on Learning Representations*. Vol. 2024. 2024.

[17] Wu, Qingyun, et al. "Autogen: Enabling next-gen llm applications via multi-agent conversation." *arXiv preprint arXiv:2308.08155*(2023).

[18] Silver, David, et al. "Mastering the game of go without human knowledge." *nature* 550.7676 (2017): 354-359.

[19] Ahn, Michael, et al. "Do as i can, not as i say: Grounding language in robotic affordances." *arXiv preprint arXiv:2204.01691* (2022).

[20] Meta Fundamental AI Research Diplomacy Team (FAIR)†, et al. "Human-level play in the game of Diplomacy by combining language models with strategic reasoning." *Science* 378.6624 (2022): 1067-1074.

목 차

- 두뇌, 지각, 행동 패러다임
- AI 에이전트 분류
- 단일 에이전트와 다중 에이전트
- 주요 라이브러리

주요 라이브러리

- 라이브러리
  - 정의
    - LLM, 데이터, 메모리, 도구를 연결하여 에이전트 애플리케이션 개발을 지원하는 SW 프레임워크
  - 특징
    - 공통 인터페이스와 모듈화를 통한 도구 조합·교체 가능
    - 라이브러리별 에이전트 관련 특화 영역이 상이함
  - 주요 라이브러리
    - LangChain
    - Haystack
    - LlamaIndex
    - Semantic Kernel
    - AutoGen

주요 라이브러리

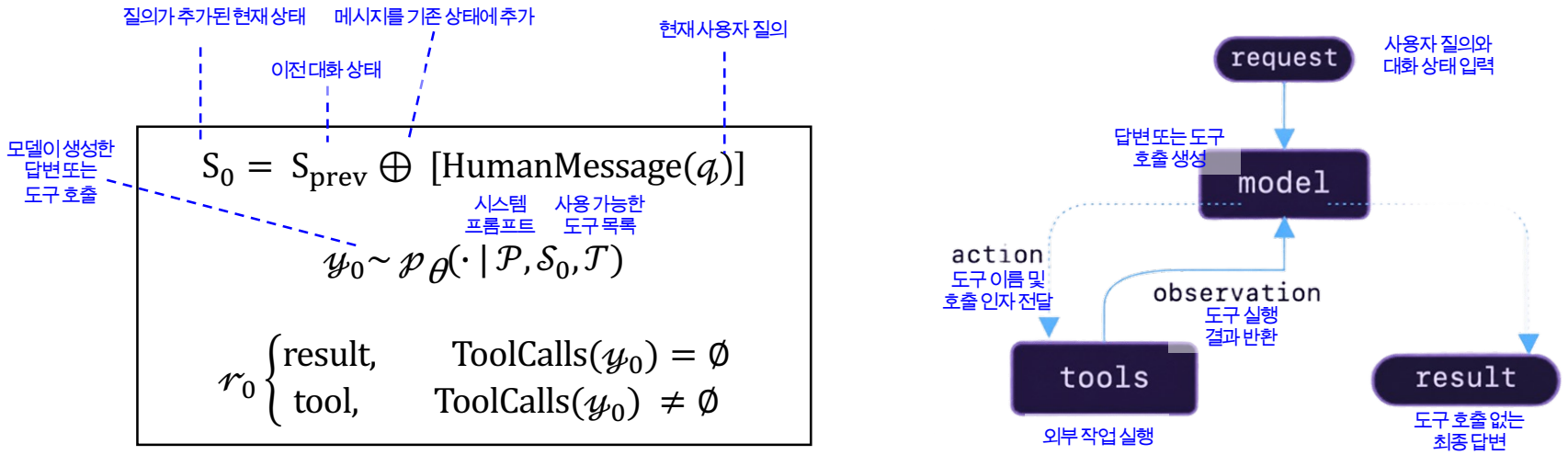
* 추상화(Abstraction)
  • 서로 다른 모델·도구의 세부 구현을 감추고 공통 방식으로 사용할 수 있도록 표현

- LangChain
  - 정의
    - LLM과 외부 데이터 및 도구를 연결하여 에이전트와 생성형 AI 애플리케이션 개발을 지원하는 오픈소스 프레임워크
  - 특징
    - 다양한 모델과 도구를 공통 인터페이스로 추상화*하여 조합·교체 가능
    - 사전 구축된 에이전트 구조와 상태 기반 반복 실행 지원

주요 라이브러

* AgentState
  - 대화 메시지와 사용자 정의 실행 정보를 보관하는 형식화된 자료구조

- LangChain
  - 동작과정(1/3) – 요청 입력과 경로 결정
    1. 사용자 질의 q를 AgentState* 메시지 목록에 추가
    2. 시스템 프롬프트($\mathcal{P}$), 상태(S_0), 도구 목록($\mathcal{T}$)을 모델에 전달
    3. 모델이 최종 답변 또는 도구 호출(y_0) 생성
    4. 도구 호출 유무에 따라 result 또는 tools로 경로 선택

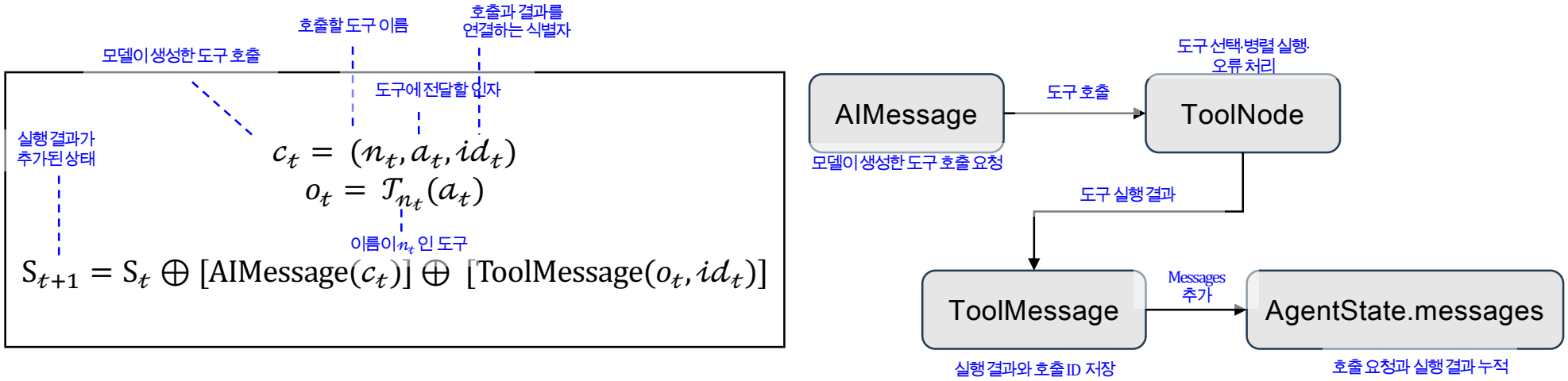


<그림 4.38> LangChain 에이전트 요청 처리 및 실행 경로 결정 과정 [21]

[21] LangChain, "Agents," *LangChain Documentation*, accessed Aug. 23, 2026.

주요 라이브러리

- LangChain
  - 동작과정(2/3) – 도구 실행과 상태 갱신
    1. 모델 출력에서 도구 이름(n_t)과 호출 인자 (a_t) 추출
    2. 지정된 도구($\mathcal{T}_{n_t}$)에 호출 인자를 전달하여 실행
    3. 도구 실행 결과(o_t)를 ToolMessage로 변환
    4. 도구 호출과 실행 결과를 AgentState에 추가

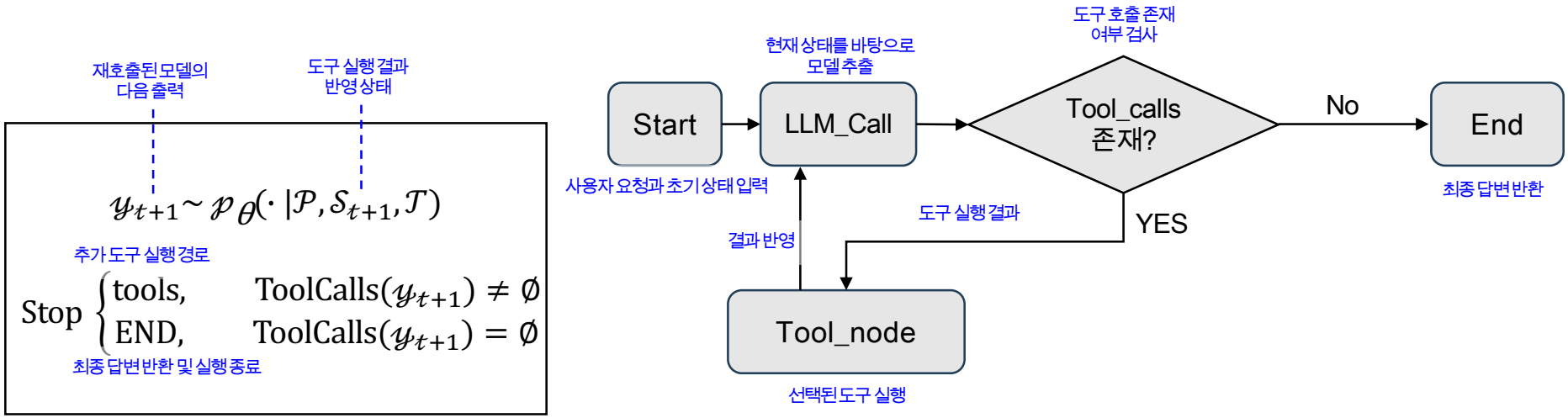


<그림 4.39> LangChain 도구 호출 및 메시지 기반 상태 갱신 과정 [21]

[21] LangChain, "Agents," *LangChain Documentation*, accessed Aug. 23, 2026.

주요 라이브러리

- LangChain
  - 동작과정(3/3) – 반복 호출과 종료
    1. 갱신된 상태(S_{t+1})를 모델에 전달
    2. 모델이 추가 도구 호출 또는 최종 답변을 생성
    3. 도구 호출이 있을 경우, 모델-도구 실행 과정 반복
    4. 최종 답변 또는 최대 실행 조건 도달 시 종료



<그림 4.40> LangChain 에이전트의 조건부 반복 및 종료 구조 [21]

[21] LangChain, "Agents," *LangChain Documentation*, accessed Aug. 23, 2026.

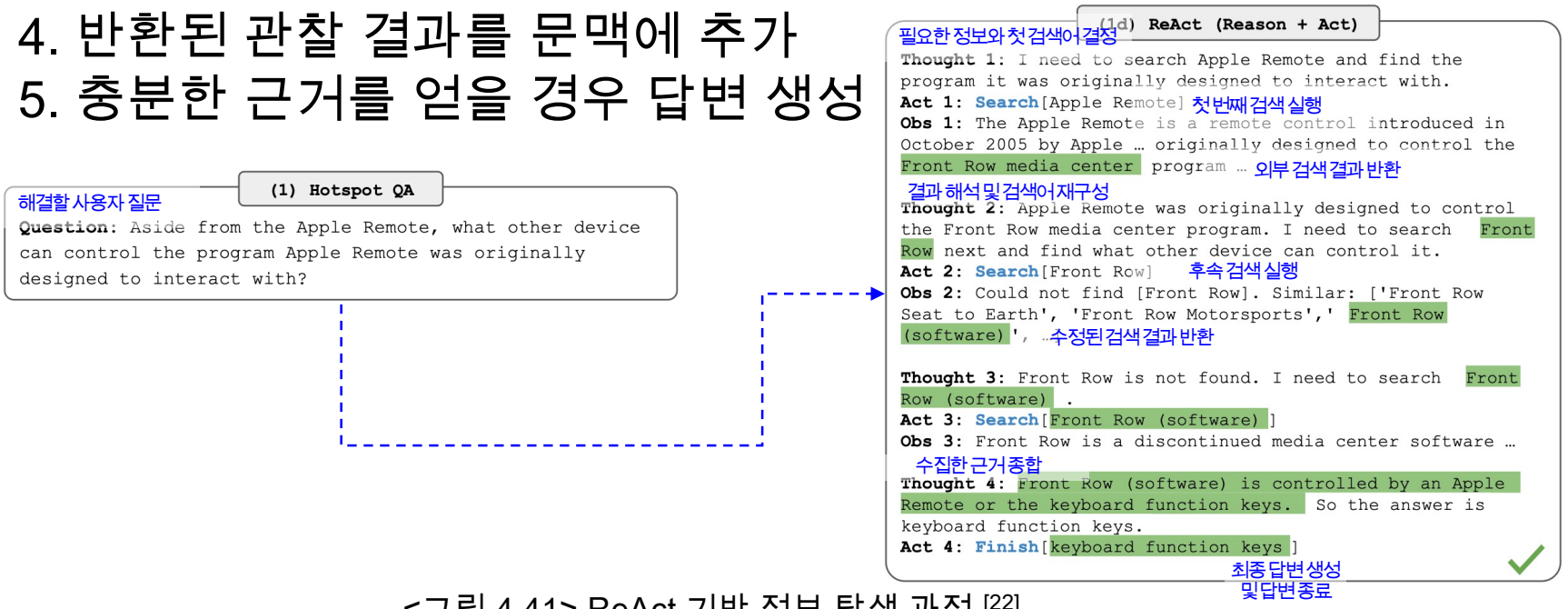
주요 라이브러

* 추론(Reasoning)
  • 현재 상태를 해석하고 다음 행동을 결정
* 행동(Action)
  • 검색·계산·API 등의 도구 호출 수행
* 관찰(Observation)
  • 도구 실행 결과를 상태에 반영

- LangChain
  - ReAct 에이전트(1/3)
    - 정의
      - LLM이 추론과 도구 사용을 번갈아 수행하고, 관찰 결과를 다음 판단에 반영하여 최종 답변을 생성하는 에이전트 방식
    - 특징
      - '추론*→행동*→관찰*' 순환 구조로 동작
      - 도구 결과를 바탕으로 계획과 다음 행동을 동적으로 수정
      - 충분한 정보 획득 및 종료 조건 달성까지 수행 반복

주요 라이브러리

- LangChain
  - ReAct 에이전트(2/3)
    - 동작 과정
      1. 사용자 질문과 현재까지의 실행 이력을 문맥으로 구성
      2. 모델이 필요한 정보와 다음 검색 행동을 추론
      3. 검색·조회 등의 행동을 외부 환경에 전달
      4. 반환된 관찰 결과를 문맥에 추가
      5. 충분한 근거를 얻을 경우 답변 생성



<그림 4.41> ReAct 기반 정보 탐색 과정 [22]

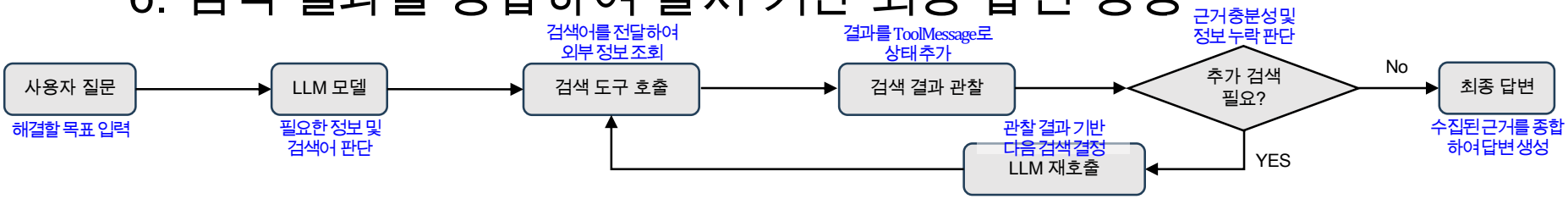
[22] Yao, Shunyu, et al. "React: Synergizing reasoning and acting in language models." *arXiv preprint arXiv:2210.03629* (2022).

주요 라이브러리

- LangChain

부록 #1, 부록 #2 코드 추가

  - ReAct 에이전트(3/3)
    - 활용 예시 – 검색 에이전트 설계 및 실행
      - LangChain의 모델·검색 도구·AgentState를 결합하여 필요한 정보를 반복적으로 검색하고 근거 기반 답변 생성
    - 설계 과정
      1. 검색 질의를 입력받아 결과를 반환하는 검색 도구 정의
      2. 도구의 이름·설명·입력 형식을 모델에 제공
      3. 검색 조건과 답변 원칙을 시스템 프롬프트에 설정
      4. 모델과 검색 도구를 create_agent로 연결
      5. 사용자 질문을 입력하여 ReAct 반복 실행
      6. 검색 결과를 종합하여 출처 기반 최종 답변 생성



<그림 4.42> LangChain 기반 ReAct 검색 에이전트 실행 과정 [22]

[22] Yao, Shunyu, et al. "React: Synergizing reasoning and acting in language models." *arXiv preprint arXiv:2210.03629* (2022).

주요 라이브러리

- LangChain
  - 장점
    - 모델·도구의 조합과 교체가 쉬움
    - 에이전트 구조를 빠르게 구현 가능
    - 상태 관리·스트리밍·오류 복구 지원
    - 미들웨어를 통한 기능 확장 가능
  - 단점
    - 추상화로 내부 동작 파악 어려움
    - 반복 호출로 시간·비용 증가
    - 버전·모델별 호환성 관리 필요

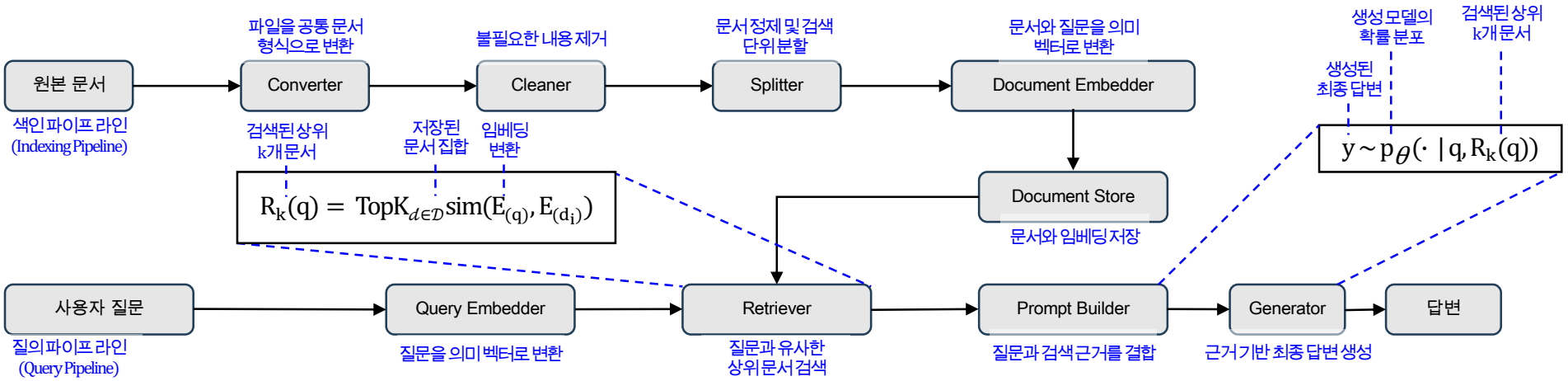
주요 라이브러리

* 분기(Branching)
  - 조건이나 입력에 따라 서로 다른 실행 경로를 선택하는 구조

- Haystack
  - 정의
    - 검색과 생성 컴포넌트를 연결하여 RAG와 AI 에이전트를 구축하는 오픈소스 프레임워크
  - 특징
    - 컴포넌트를 조합해 모델과 기능을 쉽게 교체하고 확장함
    - 분기*와 반복이 가능한 파이프라인으로 복잡한 실행 흐름 구성
    - 문서 색인부터 검색과 답변 생성까지 하나의 과정으로 연결
    - 다양한 모델과 문서 저장소 및 외부 도구 연동 지원

주요 라이브러리

- Haystack
  - 동작과정
    1. 원본 문서를 Haystack 문서 형식으로 변환
    2. 불필요한 내용을 정제하고 검색 단위로 분할
    3. 문서 임베딩을 생성해 document store에 저장
    4. 사용자 질문을 임베딩하여 관련 문서 검색
    5. 질문과 검색 문서를 하나의 프롬프트로 구성
    6. 생성 모델이 검색 근거를 반영한 답변 생성



<그림 4.43> Haystack의 문서 색인 및 RAG 질의 처리 과정 [23]

[23] deepset, “Pipelines,” Haystack Documentation, accessed Aug. 23, 2026.

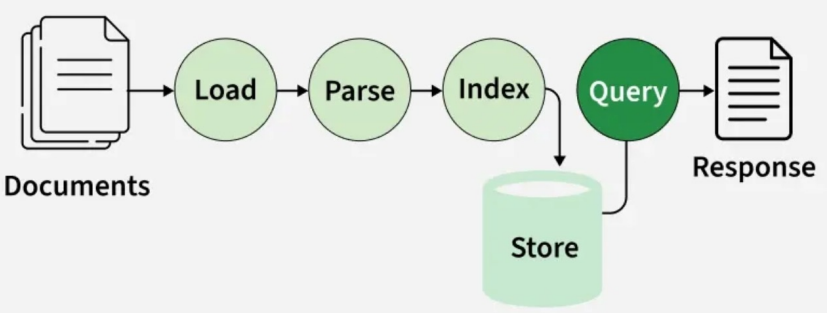
주요 라이브러리

- Haystack
  - 장점
    - 컴포넌트 교체와 확장이 쉬움
    - 색인부터 답변 생성까지 통합 구성 가능
    - 분기와 반복을 활용한 복잡한 흐름 지원
    - 다양한 모델과 문서 저장소 연동 가능
  - 단점
    - 파이프라인이 복잡해지면 추적과 디버깅 어려움
    - 컴포넌트 간 입출력 형식 일관성 확보 필요
    - 검색 성능이 임베딩과 설정 품질에 의존
    - 외부 모델과 저장소 사용에 따른 비용 발생

주요 라이브러리

* 벡터 인덱스(Vector Index)
  - 데이터의 임베딩을 저장하고 의미적 유사성으로 검색하는 구조
* 쿼리 엔진(Query Engine)
  - 질문에 맞는 정보를 검색하고 답변 생성까지 수행하는 질의 처리 모듈

- LlamaIndex
  - 정의
    - 외부 데이터를 검색 가능한 지식 구조로 변환하고 LLM의 질의와 에이전트 활용을 지원하는 오픈소스 프레임워크
  - 특징
    - 다양한 데이터 소스를 문서와 노드로 표준화
    - 벡터 인덱스와 메타데이터를 기반으로 관련 문맥 검색
    - 검색 결과를 필터링하거나 재정렬한 뒤 답변 생성
    - 쿼리 엔진*을 에이전트의 데이터 검색 도구로 활용

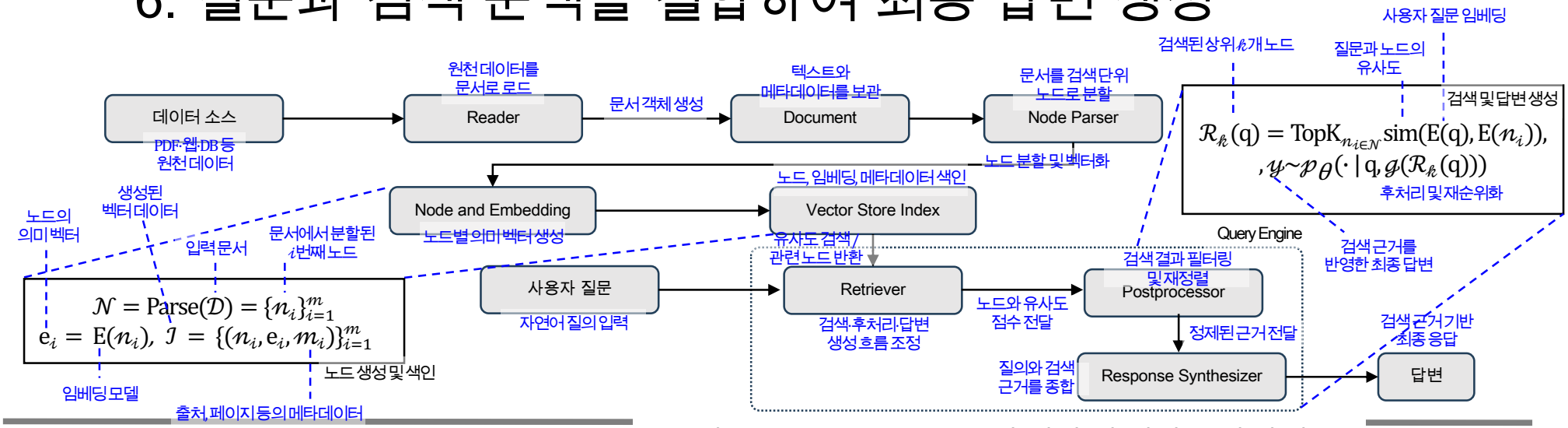


<그림 4.44> LlamaIndex 프레임워크 [24]

[24] LlamaIndex, “High-Level Concepts; Loading Data; Querying,” LlamaIndex Documentation, accessed Aug. 23, 2026.

주요 라이브러리

- LlamaIndex
  - 동작과정
    1. 파일, 웹, DB에서 원본 데이터를 불러옴
    2. 데이터를 문서로 변환한 뒤 노드 단위로 분할
    3. 각 노드의 메타데이터와 임베딩을 생성하여 인덱스에 저장
    4. 사용자 질문과 유사한 상위 노드를 검색
    5. 검색 결과를 필터링하거나 관련성 순으로 재정렬
    6. 질문과 검색 문맥을 결합하여 최종 답변 생성



<그림 4.45> LlamaIndex 문서 색인 및 질의 처리 과정 [24]

[24] LlamaIndex, "High-Level Concepts; Loading Data; Querying," LlamaIndex Documentation, accessed Aug. 23, 2026.

주요 라이브러리

- LlamaIndex

* 검색 증강 생성(RAG, Retrieval-augmented generation)
  - 외부 저장소에서 관련 정보를 검색해 LLM 입력에 추가하는 방식

  - 장점
    - 다양한 데이터를 문서 및 노드 구조로 통합
    - 인덱스와 쿼리 엔진이 검색부터 답변 생성까지의 흐름을 추상화하므로 RAG*를 빠르게 구성 가능
    - 필터링과 후처리로 검색 품질 개선

  - 단점
    - 노드 분할과 검색 설정에 따라 품질 편차 발생
    - 데이터 증가 시 색인·저장·검색 비용 증가
    - 복잡한 질의 흐름은 구성과 디버깅이 어려움

주요 라이브러리

- Semantic Kernel

* 커널(Kernel)
  - AI 서비스와 플러그인을 관리하고 실행을 연결하는 핵심 객체
* 플러그인(Plugin)
  - 모델이 호출할 함수와 API를 묶어 제공하는 기능 단위

  - 정의
    - AI 모델과 기존 코드 및 API를 연결하여 에이전트와 생성형 AI 애플리케이션을 구축하는 오픈소스 프레임워크

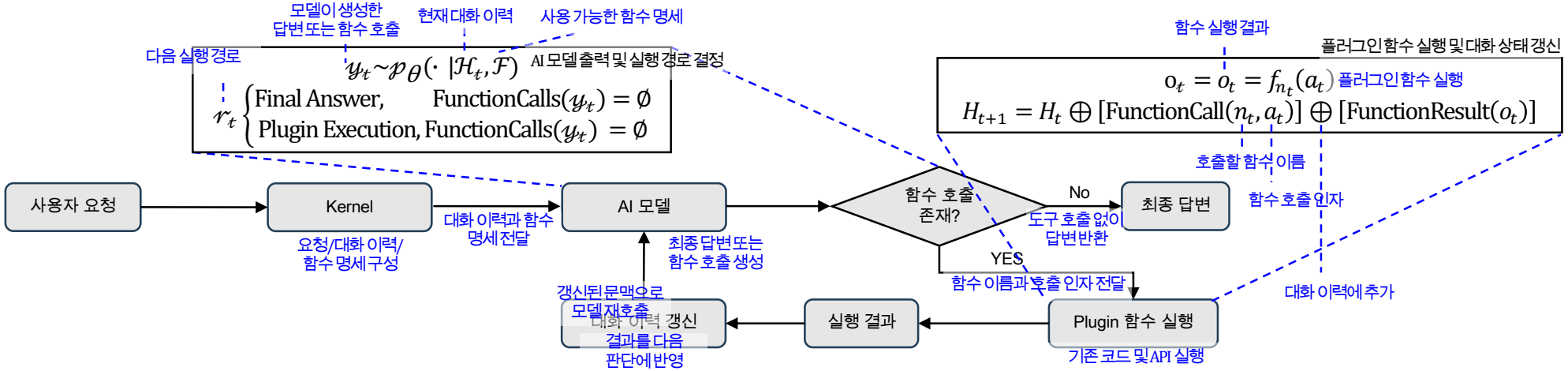
  - 특징
    - Kernel이 AI 서비스와 플러그인을 등록하고 실행을 중재
    - 기존 함수와 API를 플러그인으로 제공해 모델의 자동 함수 호출 지원
    - 모델과 기능을 쉽게 교체하고 확장할 수 있는 모듈형 구조
    - 필터와 로깅을 통해 함수 실행을 관찰하고 통제

주요 라이브러리

- Semantic Kernel

* 플러그인(Plugin)
  - Kernel에 등록되어 모델이 호출할 수 있는 함수 집합

  - 동작과정
    1. Kernel에 AI 모델과 플러그인 함수를 등록
    2. 사용자 요청과 대화 이력을 Kernel에 입력
    3. 함수 이름, 설명, 입력 형식을 JSON Schema로 변환해 모델에 전달
    4. 모델이 최종 답변 또는 실행할 함수와 인자를 생성
    5. 함수 호출 시 Kernel이 플러그인* 실행 및 결과를 이력 추가
    6. 갱신된 이력 모델 전달 및 최종 답변 또는 최대 반복 조건까지 반복



<그림 4.46> Semantic Kernel의 자동 함수 호출 및 반복 실행 과정 [25]

[25] Microsoft, "Introduction to Semantic Kernel," *Microsoft Learn*, accessed Aug. 23, 2026.

주요 라이브러리

- Semantic Kernel
  - 장점
    - 기존 코드와 API를 플러그인으로 재사용 가능
    - 모델과 실행 기능을 분리하여 교체·확장 용이
    - 함수 호출과 대화 상태를 Kernel이 통합 관리
    - 자동 함수 호출로 복잡한 작업 흐름 구현 가능
  - 단점
    - 플러그인 명세와 실행 권한의 별도 설계 필요
    - 잘못된 함수 선택의 실제 작업 오류 연계 가능
    - 반복 호출에 따라 응답 시간과 실행 비용 증가
    - 복잡한 흐름은 상태 추적과 오류 분석이 어려움

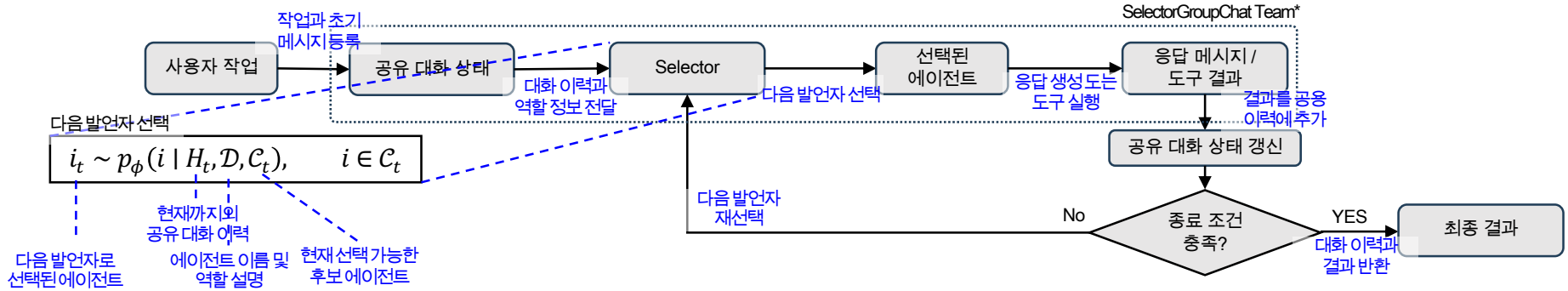
주요 라이브러리

- AutoGen
  - 정의
    - 여러 에이전트간 메시지 교환 및 역할 분담을 통해 모델·도구·사람을 연결해 복합 작업을 수행하는 오픈소스 프레임워크
  - 특징
    - 역할과 능력이 다른 에이전트를 조합하여 협업 구조 구성
    - 메시지 기반 상호작용으로 작업 위임 및 결과 공유
    - 팀 패턴과 종료 조건으로 대화 순서와 반복 범위 제공

주요 라이브러리

* Team
  - 여러 에이전트와 종료 조건을 묶어 공동 작업을 수행하는 실행 단위
* SelectorGroupChat Team
  - 공유 대화 이력과 역할 설명을 바탕으로 모델이 다음 발언자를 동적으로 선택하는 Team

- AutoGen
  - 동작과정
    1. 역할·모델·도구를 지정하여 참여 에이전트 구성
    2. 에이전트를 Team*에 배치하고 종료 조건 설정
    3. 사용자 작업과 기존 대화를 공통 메시지 이력에 입력
    4. 대화 이력과 역할을 바탕으로 다음 발언자 선택
    5. 선택된 에이전트가 응답을 생성하고 필요 시 도구 실행
    6. 응답과 실행 결과를 모든 에이전트에게 공유
    7. 종료 조건을 검사하고, 미충족 시 발언자 선택부터 반복



<그림 4.47> AutoGen 공유 대화 기반 에이전트 선택 및 실행 과정 [26]

[26] Microsoft, “Selector Group Chat,” AutoGen AgentChat Documentation, accessed Aug. 24, 2026.

주요 라이브러리

- AutoGen
  - 장점
    - 역할별 에이전트 분업으로 복잡한 작업 처리
    - 맥락 기반 발언자 선택으로 협업 순서 유연화
    - 공유 대화 상태를 통해 정보와 결과 지속 축적
    - 도구 실행, 코드 실행, 인간 참여를 비교적 쉽게 결합
  - 단점
    - 에이전트와 대화 횟수 증가로 비용과 시간 증가
    - 발언자 선택이나 응답 오류가 이후 과정에 전파
    - 종료 조건이 불명확할 경우, 불필요한 반복 발생
    - 동적 상호작용으로 실행 재현 및 디버깅 어려움

주요 라이브러리

- 정리
  - 데이터·도구·협업 요구에 따라 라이브러리 선택 및 조합 가능

[표 4.4] 주요 라이브러리별 특징 비교

구분	핵심 원리	장점	단점	주요 활용
LangChain[21]	모델·도구·상태를 연결해 에이전트 실행 흐름 구성	다양한 연동 및 높은 범용성	API 변화가 빠르고 복잡한 흐름의 디버깅 어려움	도구 사용 에이전트, ReAct
Haystack[23]	컴포넌트 기반 파이프라인으로 검색 및 생성 연결	RAG 형태 및 분기·반복 지원	파이프라인이 커질수록 구성과 관리 복잡도 증가	문서 검색, 지식 질의응답
LlamaIndex[24]	외부 데이터를 노드와 인덱스로 변환해 질의에 활용	우수한 데이터 수집/색인/검색 기능	성능이 노드 분할과 임베딩 설계에 민감함	데이터 중심 RAG, 문서 질의
Semantic Kernel[25]	Kernel이 모델과 플러그인 함수를 연결해 실행	기존 코드와 업무 시스템 통합에 유리	초기 함수 설계가 필요하고 Microsoft 생태계 의존성 존재	업무 자동화, 기업 시스템 연동
AutoGen[26]	공유 대화 상태에서 여러 에이전트가 역할별 협업	다중 에이전트 구성과 인간 참여가 유연함	대화 비용과 지연이 크고 상호작용 제어 어려움	협업형 문제 해결, 에이전트 팀

[21] LangChain, “Agents,” *LangChain Documentation*, accessed Aug. 23, 2026.

[23] deepset, “Pipelines,” Haystack Documentation, accessed Aug. 23, 2026.

[24] LlamaIndex, “High-Level Concepts; Loading Data; Querying,” LlamaIndex Documentation, accessed Aug. 23, 2026.

[25] Microsoft, “Introduction to Semantic Kernel,” *Microsoft Learn*, accessed Aug. 23, 2026.

[26] Microsoft, “Selector Group Chat,” AutoGen AgentChat Documentation, accessed Aug. 24, 2026.

Thanks!

이 진 수 (jinsue@pel.sejong.ac.kr)

부록 #1

- 그림 4.39 코드
  - ReAct 검색 에이전트 코드 – 도구 등록

```
1.  !pip install ddgs langchain_community wikipedia langchain_openai
2.
3.  from langchain.tools import DuckDuckGoSearchRun
4.  from langchain.tools import WikipediaQueryRun
5.  from langchain.utilities import WikipediaAPIWrapper
6.  from langchain.agents import Tool
7.
8.  ddg_search = DuckDuckGoSearchRun()
9.  wikipedia = WikipediaQueryRun(api_wrapper=WikipediaAPIWrapper())
10.
11. tools = [
12.     Tool(
13.         name="DuckDuckGo Search",
14.         func=ddg_search.run,
15.         description="인터넷에서 정보를 추출하는 웹 검색 도구.",
16.     ),
17.     Tool(
18.         name="Wikipedia Web Search",
19.         func=wikipedia.run,
20.         description="위키피디아를 검색하는 도구.",
21.     ),
22. ]
```

부록 #2

- 그림 4.39 코드
  - ReAct 검색 에이전트 코드 – 에이전트 생성 및 실행

```
23. from langchain_openai import ChatOpenAI
24. from langchain.agents import initialize_agent
25. import os
26.
27. os.environ["OPENAI_API_KEY"] = "발급받은 오픈AI 키 값"
28.
29. llm = ChatOpenAI(
30.     model="gpt-4o",
31.     temperature=0
32. )
33.
34. agent = initialize_agent(
35.     tools, llm, agent="zero-shot-react-description",
36.     verbose=True
37. )
38.
39. agent.run('인터넷에서 2025년 한국의 대통령 검색하고 '
40.           '위키피디아에서 링컨 대통령 검색해서 정리해서 알려줘.')
```

부록 #3

- 오픈소스 LLM 성능 측정 코드
  - 측정 함수(1/2)

```
import pandas as pd
import time
from transformers import AutoTokenizer, AutoModelForCausalLM

def measure_model_performance(model_name, max_length, input_text,
                              model_size_billion_params):
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForCausalLM.from_pretrained(model_name)

    input_tokens = tokenizer(input_text, return_tensors="pt")

    start_time = time.time()
    output = model.generate(**input_tokens, max_length=max_length)
    end_time = time.time()

    output_text = tokenizer.decode(output[0], skip_special_tokens=True)
    time_taken = end_time - start_time
    num_tokens = output.size(1)
    tokens_per_second = num_tokens / time_taken
    tokens_per_second_per_billion_params = (
        tokens_per_second / model_size_billion_params)

```

부록 #3

- 오픈소스 LLM 성능 측정 코드
  - 측정 함수(2/2)

```
23.     data = {
24.         "model_name": model_name.split('/')[1],
25.         "billion_parameters": model_size_billion_params,
26.         "tokens_per_second": tokens_per_second,
27.         "tokens_per_second_per_billion_parameters":
28.             tokens_per_second_per_billion_params,
29.         "answer": output_text
30.     }
31.     return data
```

부록 #4

- 오픈소스 LLM 성능 측정 코드
  - 모델별 측정과 시각화(1/2)

```
models = [
    "mistralai/Mistral-7B-Instruct-v0.1",
    "microsoft/Phi-3-mini-4k-instruct",
    "Qwen/Qwen2-7B-Instruct",
    "tri-ml/mamba-7b-rw",
    "meta-llama/Meta-Llama-3-8B-Instruct"
]
input_texts = [
    "[INST]describe briefly what is an AI agent[/INST]",
    "<|user|>describe briefly what is an AI agent<|end|><|assistant|>",
    "describe briefly what is an AI agent",
    "describe briefly what is an AI agent",
    "describe briefly what is an AI agent"
]
parameters = [7, 3, 7, 7, 8]
max_length = 512

results = []
for model_name, input_text, size in zip(models, input_texts, parameters):
    results.append(
        measure_model_performance(model_name, max_length, input_text, size))
```

부록 #4

- 오픈소스 LLM 성능 측정 코드
  - 모델별 측정과 시각화(2/2)

```
53.
54.  df = pd.DataFrame(results)
55.
56.  import matplotlib.pyplot as plt
57.  plt.figure(figsize=(10, 6))
58.  plt.bar(df['model_name'], df['tokens_per_second'], color='skyblue')
59.  plt.xlabel('Model Name')
60.  plt.ylabel('Tokens per second')
61.  plt.title('Tokens per second by model')
62.  plt.xticks(rotation=45, ha='right')
63.  plt.tight_layout()
64.  plt.show()
```