

2026/09/18, AI 에이전트 세미나

이론부터 실전까지 AI 에이전트 완벽 마스터

- 9장 단일 · 다중 에이전트 시스템 만들기 -

김 지 혜 (jihye@pel.sejong.ac.kr)

세종대학교 프로토콜공학연구실

목 차

- 자율 에이전트
- 허깅GPT 활용
- 다중 에이전트 시스템
- SaaS, MaaS, DaaS, RaaS

목 차

- 자율 에이전트
- 허깅GPT 활용
- 다중 에이전트 시스템
- SaaS, MaaS, DaaS, RaaS

자율 에이전트

- 정의

- 인간의 개인 없이 스스로 작업을 수행하거나 의사결정을 내릴 수 있는 시스템 또는 개체

- 특징

- 주변 환경을 인지하고, 이를 기반으로 추론 수행
- 주어진 목표에 따라 의사결정을 내리고 행동 수행
- 범용 인공지능(AGI, Artificial General Intelligence)으로 발전하기 위한 중요한 단계

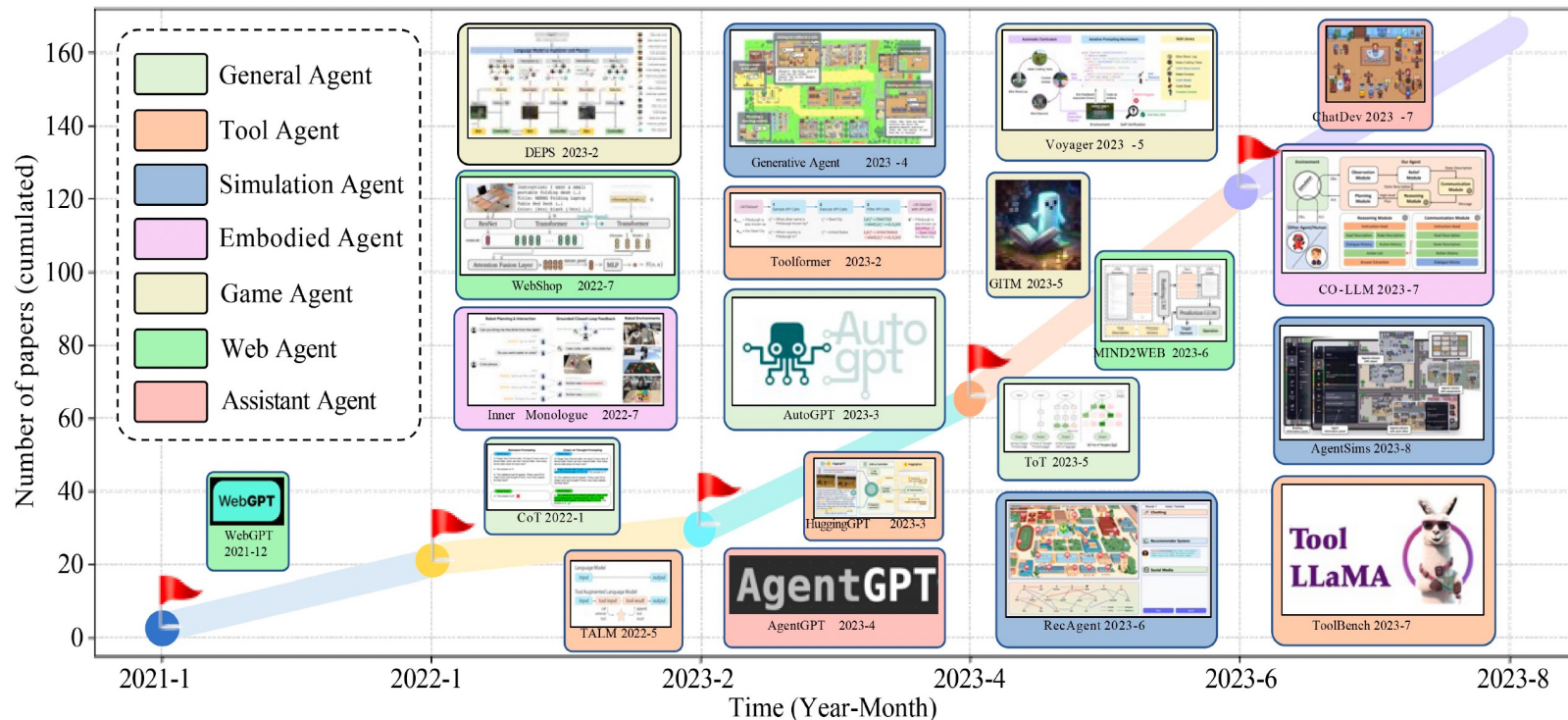
자율 에이전트

- LLM 자율 에이전트

- 단계별 추론을 통해 방정식 풀이, 코드 디버깅 등 복잡한 문제 해결 가능
 - 연역적 추론: 일반적인 규칙을 구체적인 사례에 적용
 - 귀납적 추론: 여러 사례로부터 공통된 패턴 학습
 - 가설적 추론: 불완전한 데이터로부터 가능한 설명을 도출
- 작업을 여러 행동으로 분해 후, 각 행동 결과를 예측 및 조정
- 사용자 제공 컨텍스트와 사전 학습 중 습득한 지식에 의존
 - 의료/금융처럼 전문적인 분야에서는 LLM만으로 문제 해결의 한계 존재
 - LLM 능력을 외부 도구로 확장하거나 외부 환경과 연결하여 자율 에이전트로 확장

자율 에이전트

- LLM 자율 에이전트 진화
- 연구 분야의 급격한 성장 및 에이전트 카테고리의 다양화



<그림 9.1> LLM 자율 에이전트 진화^[1]

[1] WANG, Lei, et al. A survey on large language model based autonomous agents. Frontiers of computer science, 2024, 18.6: 186345.

자율 에이전트

• LLM 자율 에이전트 고려사항

• 아키텍처 설계

- 자율 에이전트가 환경을 인식하고 학습하며, 상황에 따라 적절한 역할을 수행할 수 있도록 아키텍처 설계 필요
- 설계된 아키텍처를 실제 작업 수행 과정에서 어떻게 활용할 것인지 고려 필요

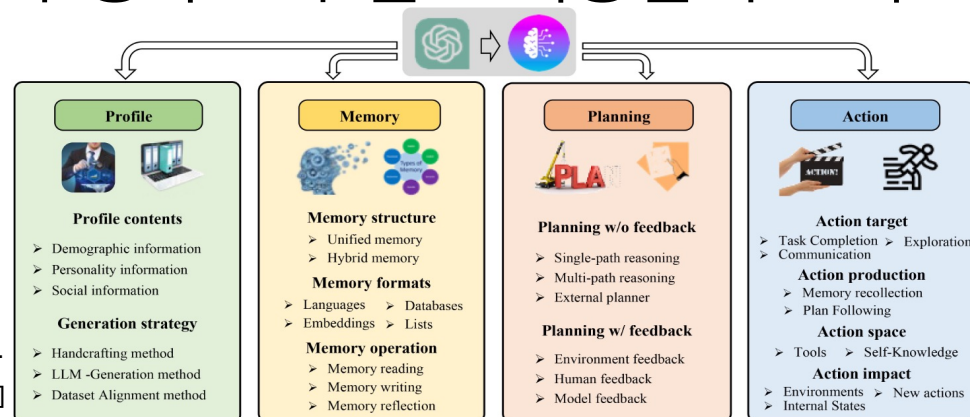
• 아키텍처 목적

- LLM이 에이전트로서의 최대 역량을 발휘하도록 지원
- 복잡한 작업을 효과적으로 수행하도록 필요 기능을 구조화

• 핵심 구성 모듈

- 프로파일링(Profiling)
- 메모리(Memory)
- 계획(Planning)
- 행동(Action)

<그림 9.2> LLM 자율 에이전트 필요 모듈[1]



자율 에이전트

- LLM 자율 에이전트 아키텍처 (1/5)
 - 프로파일링 모듈
 - 프롬프트를 통해 에이전트 역할 및 특성 정의
 - 특성, 직무, 심리/사회적 정보, 다른 에이전트와의 관계 포함
 - 개발자나 도메인 전문가가 수작업으로 작성한 프로필 활용
 - e.g., 소프트웨어 개발 시스템에서는 “당신은 코드 리뷰를 담당하는 소프트웨어 엔지니어이다.” 같은 역할을 만들 수 있음
 - 프로필 작성 시간이 오래 걸리고, 확장성이 떨어지는 한계 존재
 - 최근 LLM 기반 자동 프로파일링 시스템이 연구됨
 - 동적 시스템이나 사용자 피드백이 존재하는 환경에서 여러 상황에 맞게 확장 및 적응 가능
 - 프롬프트 엔지니어링에 영향을 받으며, 여전히 인간 검증 필요

자율 에이전트

- LLM 자율 에이전트 아키텍처 (2/5)
 - 메모리 모듈 (1/2)
 - 시스템이 환경이나 다른 출처로부터 인지한 정보를 저장하고, 향후 행동 및 의사결정에 활용하도록 지원
 - 지각, 단기, 장기 정보에 특화된 형태로 메모리 구성 가능
 - 지각 메모리: 현재 환경에서 인지한 정보 저장
 - 단기 메모리: 최근 사건이나 현재 작업과 관련된 정보 저장
 - 장기 메모리: 이후에도 반복적으로 활용할 수 있는 과거 정보 저장
 - 컨텍스트 길이가 에이전트 사용 가능한 메모리 범위를 제한할 수 있음
 - e.g., 게임 개발을 지원하는 에이전트는 직전에 발생한 이벤트나 관련 설명을 단기 메모리로 활용 가능
 - 과거 생각을 재활용하여 에이전트 행동을 보조하는 하이브리드 메모리 방식이 등장함

자율 에이전트

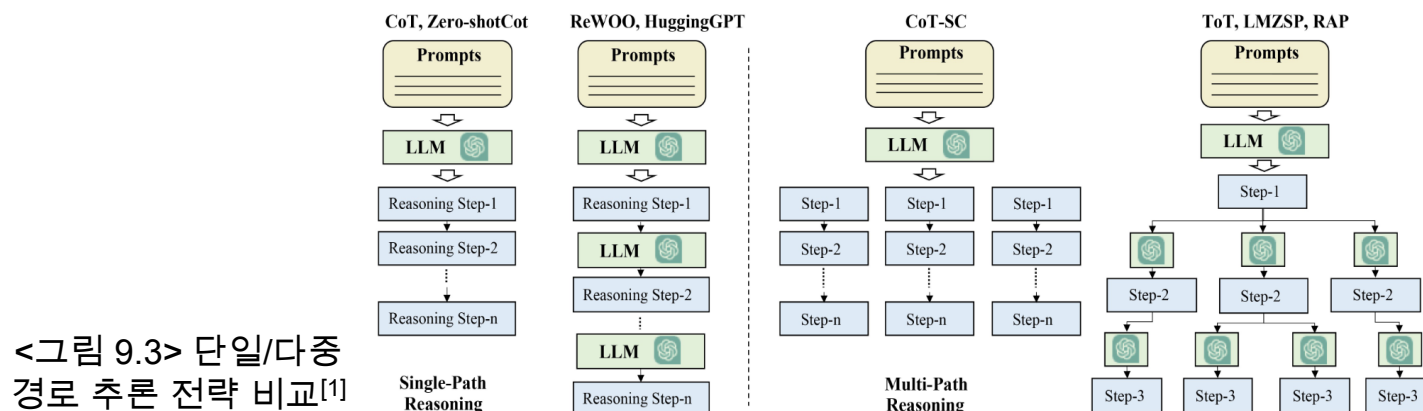
- LLM 자율 에이전트 아키텍처 (3/5)
 - 메모리 모듈 (2/2)
 - 메모리는 읽기, 쓰기, 성찰 방식으로 활용됨
 - 메모리 읽기: 에이전트 행동에 유용한 정보를 검색 및 추출, 필요한 정보를 검색하여 문맥의 연속성과 일관성을 높임
 - 메모리 쓰기: 미래에 활용할 수 있는 환경 및 상호작용 정보 저장, 중복 정보 및 메모리 초과를 줄이며 행동에 필요한 정보를 지속 축적
 - 메모리 성찰: 저장된 정보를 통해 추상적이고 복잡한 정보를 평가 및 추론, 과거 정보를 분석하여 새로운 통찰을 도출하고 목표 달성을 위해 이후 행동을 조정하도록 지원

자율 에이전트

• LLM 자율 에이전트 아키텍처 (4/5)

• 계획 모듈

- 복잡한 작업을 관리하기 쉬운 하위 작업으로 분해하여 LLM이 합리적이고 신뢰성 있게 동작하도록 지원
- 계획 과정은 피드백 여부에 따른 두 가지 유형으로 구분
 - 피드백 없는 계획: 미래 행동에 영향을 주는 피드백을 반영하지 않음
 - 단일 경로 추론에서는 작업을 일련의 중간 단계로 나눠 순차적으로 연결함
 - 다중 경로 추론에서는 각 단계가 여러 후속 단계를 탐색하는 트리 구조 사용
 - 피드백 있는 계획: 피드백을 받아, 이후 계획과 행동을 조정함
 - 초기에 완전한 계획 수립이 어렵거나 환경이 변화하는 장기 작업에 적합함



자율 에이전트

- LLM 자율 에이전트 아키텍처 (5/5)

- 행동 모듈

- 계획을 구체적 결과로 전환 및 외부 환경과의 상호작용 담당
 - 특정 목표를 가진 작업의 실행과 행동에 중점을 둠
 - 다른 에이전트와의 소통, 환경 탐색, 필요 메모리 탐색, 계획 실행 등
- 사전 학습 중 습득한 지식이나 외부 도구를 사용함
 - 사전 학습된 지식은 텍스트 생성, 질의응답, 기존 데이터 기반 의사 결정 등에 활용 가능
 - 동적 환경, 실시간 작업, 특화된 작업에는 외부 도구(e.g., API, 데이터베이스, 소프트웨어, 외부 모델 등)를 활용해야 함

자율 에이전트

- 더 나은 에이전트 개발 전략

- 파인튜닝

- 범용 LLM을 특정 작업, 도메인, 혹은 행동 목표에 맞게 조정하여 에이전트 성능 향상
 - 인간의 가치에 맞는 행동을 유도하거나, 명령 수행 능력 개선 및 특정 분야에 특화된 에이전트로 발전 가능

- 파인튜닝 데이터셋

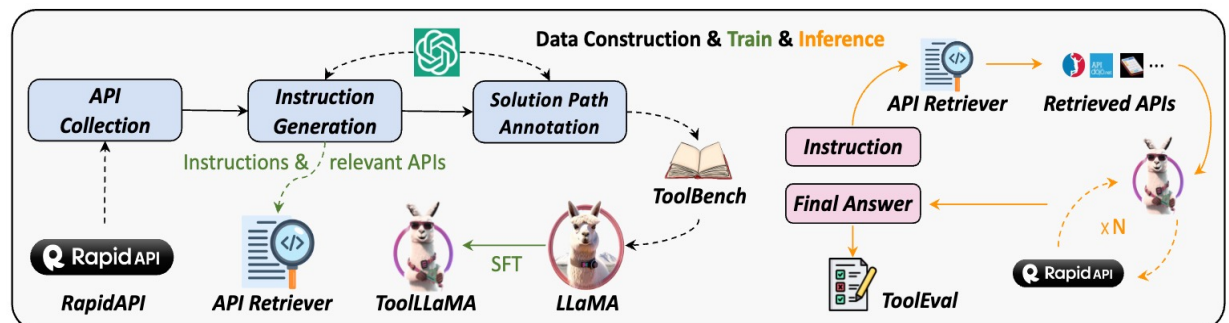
- 주로 특정 작업이나 도메인에 맞는 학습 데이터를 수집하고, 인간이 직접 주석을 추가하여 구성
 - 예시 – Webshop
 - Amazon에서 120만 개의 상품 수집 및 가상 전자상거래 웹사이트 구축
 - 사용자의 검색 과정을 상품 선택 에이전트 파인튜닝 데이터로 활용
 - 예시 – EduChat
 - 교육 특화 에이전트를 위한 다양한 교육 시나리오 포함 데이터셋 구축
 - 심리학자 등 전문 인력이 데이터의 품질을 평가하고 편집

자율 에이전트

- 데이터셋 수집 비용 한계 개선안 (1/2)
 - LLM 기반 주식 데이터셋
 - 인간 주식 방식과 LLM 주식 방식은 데이터 품질과 구축 비용 사이에 절충 관계가 존재
 - 데이터 품질은 인간 주식에 비해 떨어지나, 대규모 데이터셋을 상대적으로 낮은 비용으로 구축 가능
 - 예시 – ToolBench
 - LLM을 API에 연결한 에이전트 시스템임
 - 16,000개 이상의 실제 API 수집 후, ChatGPT를 통해 수집된 데이터에 주석 생성하여 에이전트 학습용 데이터셋 구축
 - LLaMA 파인튜닝 결과, 이는 API 활용에서 향상된 성능을 보임

<그림 9.4>
ToolBench의 구성^[2]

[2] QIN, Yujia, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In: International Conference on Learning Representations. 2024.



자율 에이전트

- 데이터셋 수집 비용 한계 개선안 (2/2)
 - 인간 주식 데이터셋과의 혼합 접근 방식
 - 주식/비주식 데이터셋 간 절충을 해결하기 위해 등장함
 - 인간 주식 데이터셋은 정확성, 세부 표현, 신뢰성 측면에서 높은 품질을 제공하나 구축 비용이 높음
 - 비주식 데이터셋은 상대적으로 품질이 낮을 수 있으나, 넓은 범위의 데이터를 빠르게 구축하고, 반복적 데이터 생성이 가능함
 - LLM을 통해 대규모 데이터에 레이블링 수행 후, 인간이 결과를 검증하거나 중요한 작업에 직접 참여함
 - 인간 주식의 품질과 LLM 기반 주식의 확장성을 함께 활용
 - 데이터 품질과 구축 비용의 균형을 맞추면서 에이전트 파인튜닝을 현실적이고 효과적으로 수행

자율 에이전트

- 툴포머(Toolformer) (1/2)
 - 정의
 - LLM이 외부 도구를 언제, 어떤 방식으로 사용할지 스스로 학습하여 문제 해결에 활용하도록 하는 방법
 - 특징
 - 도구 사용을 특정 작업에 한정되지 않고 일반화되도록 학습
 - 인간 주석에 의존하지 않고 기존 LLM 범용성을 유지하면서 도구 활용하는 능력 획득
 - 특정 프롬프트에 도구 사용을 하드코딩하지 않고, 필요한 상황에 선택적으로 도구 호출

자율 에이전트

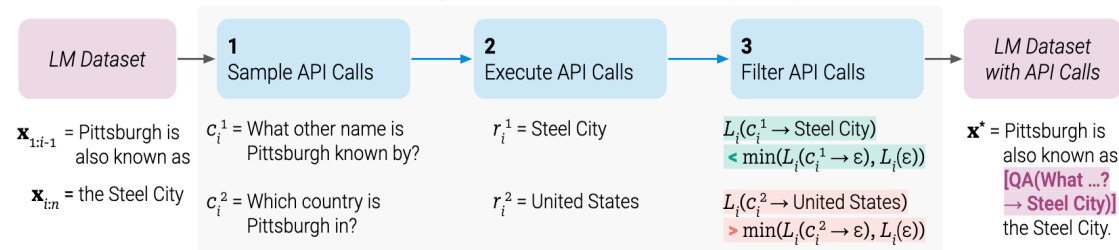
• 툴포머(Toolformer) (2/2)

• 접근 방식

- 외부 도구를 API 호출 형태로 정의하여 기존 언어 모델링 과정 안에서 처리
 - 모델은 입력 문맥을 바탕으로 도구 호출 필요 여부 판단 후, 적절한 도구와 입력 인자를 선택하여 API 호출 생성
 - 실행 결과를 다시 문맥에 포함하여 이후 토큰 생성 및 문제 해결에 활용
 - 해당 과정을 통해 새로운 도구도 동일한 API 기반 인터페이스로 추가할 수 있어 확장성이 높음

• 활용 예시

- 수학적 계산: 계산기 API 호출 후 결과로 답변 생성
- 외부 지식: 검색 엔진 호출 후 문맥에 반영하여 답변 생성



<그림 9.5> 툴포머 접근 방식[3]

[3] SCHICK, Timo, et al. Toolformer: Language models can teach themselves to use tools. Advances in neural information processing systems, 2023, 36: 68539-68551.

자율 에이전트

- 허깅GPT (1/8)

- 등장 배경

- LLM은 텍스트 기반 작업에 강하나, 시각·음성 등 다양한 모달리티 정보를 직접 처리하는 데 한계 존재
 - 실제 환경의 복잡한 작업은 여러 개의 하위 작업으로 분해 가능
 - LLM은 다양한 작업 처리가 가능하나, 특정 작업에서는 해당 작업에 특화된 전문 모델보다 성능이 낮을 수 있음

자율 에이전트

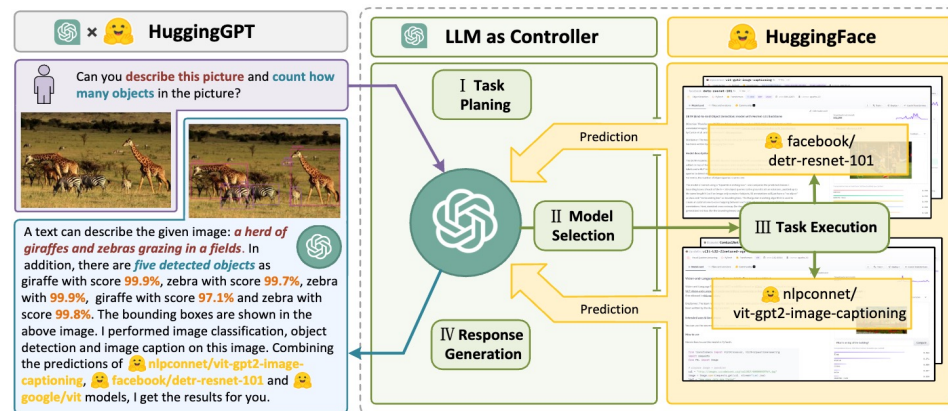
• 허깅GPT (2/8)

• 정의

- 다양하고 복잡한 작업을 자율적으로 해결하도록 설계된 LLM 기반 에이전트

• 특징

- LLM을 허깅페이스 같은 머신러닝 커뮤니티와 연결
 - 연결된 LLM은 여러 모달리티를 입력으로 받아 다양한 작업 수행 가능
- API 호출과 유사한 방식을 사용하나 파인튜닝 필요하지 않음
- LLM은 작업 이해 및 행동 계획하나, 실제 행동은 전문 모델이 수행
- 모델별 기능을 텍스트 설명으로 프롬프트에 제공



<그림 9.6> 허깅GPT의 일반 구조[4]

[4] SHEN, Yongliang, et al. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. Advances in Neural Information Processing Systems, 2023.

자율 에이전트

- 허깅GPT (3/8)
 - LLM과의 상호작용
 - 모델 기능 설명을 LLM 프롬프트에 입력할 수 있음
 - LLM은 계획, 스케줄링, 협력 차원에서 다른 모델을 관리하는 두뇌로 볼 수 있음
 - 직접 작업하는 대신 특정 모델을 호출하여 문제를 해결함
 - e.g., 사용자가 이미지 속 동물에 관한 질문 시, LLM은 모델을 호출 및 실행하여 출력을 반환하며, 해당 동물을 텍스트 답변으로 생성
 - 특정 작업을 수행 가능한 모델에 대한 양질의 설명을 확보하는 것이 중요
 - Hugging Face 같은 머신러닝 커뮤니티의 모델 및 모델 설명 활용

자율 에이전트

• 허깅GPT (4/8)

• 동작과정 개요

1. 작업 계획(Task Planning)

- LLM이 사용자 요청 분석 후, 질문을 해결 가능한 하위 작업으로 변환

2. 모델 선택(Model Selection)

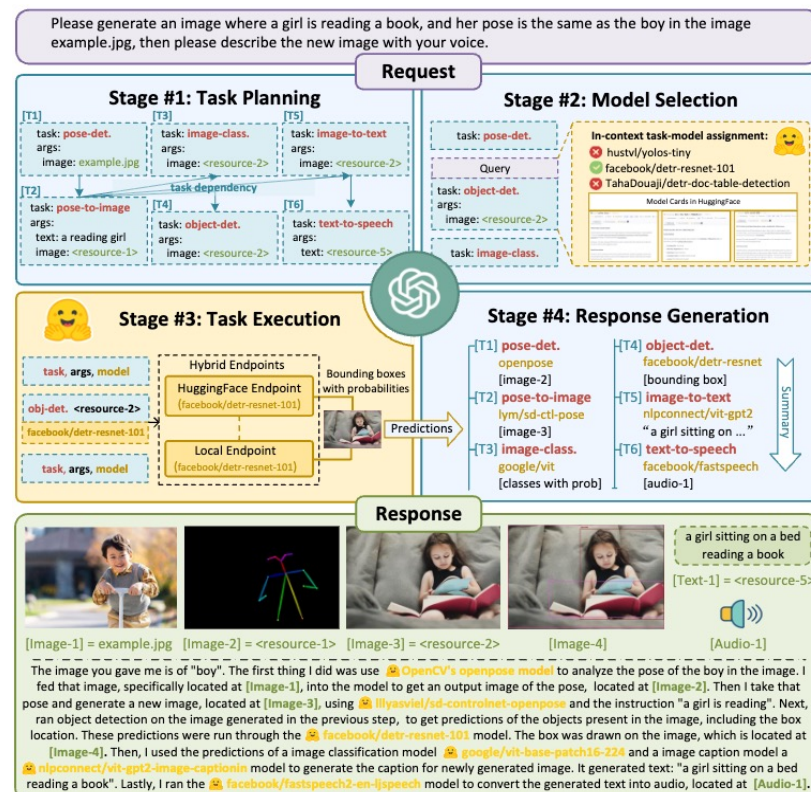
- 허깅페이스에서 적절한 모델 선택

3. 작업 실행(Task Execution)

- 모델 호출 및 실행 후, 결과를 LLM으로 반환

4. 응답 생성(Response Generation)

- 모델 결과 통합 후 답변 생성



<그림 9.7> 허깅GPT 프로세스[4]

자율 에이전트

• 허깅GPT (5/8)

• 동작과정 – 1) 작업 계획

- LLM은 사용자 요청 분석 후, 하위 작업으로 분해함
- 슬롯 채우기(slot filling)*로 하위 작업의 필요 정보 구조화
- 하위 작업 간 관계 분석으로 작업 간 의존성, 실행 순서 파악
- 프롬프트는 LLM에 필요한 모든 정보를 제공함
 - 작업에 대한 지침, 정보 검색 위치, 작업 수행 예시, 기대 출력 포함

• 허깅GPT 표준화된 템플릿 구조

항목	설명
작업 ID	각 작업과 생성된 리소스를 구분하기 위한 고유 식별자
작업 유형	언어, 비디오, 오디오 등 수행할 작업의 유형
작업 의존성	현재 작업 수행 전에 완료되어야 하는 선행 조건 정의
작업 인자	텍스트, 이미지 등 작업 수행에 필요한 입력 정보

*슬롯 채우기(slot filling)

사용자 요청에서 작업 수행에 필요한 핵심 정보를 정해진 슬롯(slot)에 맞춰 구조화하는 방식

Task Planning	Prompt	
	#1 Task Planning Stage - The AI assistant performs task parsing on user input, generating a list of tasks with the following format: [{"task": task, "id": task_id, "dep": dependency_task_ids, "args": {"text": text, "image": URL, "audio": URL, "video": URL}}]. The "dep" field denotes the id of the previous task which generates a new resource upon which the current task relies. The tag "<resource>-task_id" represents the generated text, image, audio, or video from the dependency task with the corresponding task_id. The task must be selected from the following options: [{ Available Task List }]. Please note that there exists a logical connections and order between the tasks. In case the user input cannot be parsed, an empty JSON response should be provided. Here are several cases for your reference: [{ Demonstrations }]. To assist with task planning, the chat history is available as [{ Chat Logs }], where you can trace the user-mentioned resources and incorporate them into the task planning stage.	
	Demonstrations	
	Can you tell me how many objects in e1.jpg?	[{"task": "object-detection", "id": 0, "dep": [-1], "args": {"image": "e1.jpg"}}, {"task": "image-to-text", "id": 0, "dep": [-1], "args": {"image": "e2.jpg"}}, {"task": "image-cls", "id": 1, "dep": [-1], "args": {"image": "e2.jpg"}}, {"task": "object-detection", "id": 2, "dep": [-1], "args": {"image": "e2.jpg"}}, {"task": "visual-question-answering", "id": 3, "dep": [-1], "args": {"text": "what's the animal doing?", "image": "e2.jpg"}}]

<그림 9.8> 허깅GPT 프롬프트 설계 세부 사항^[4]

자율 에이전트

• 허깅GPT (6/8)

• 동작과정 – 2) 모델 선택

- 작업 계획 단계에서 확인된 작업 유형에 따라 모델 필터링
 - 하위 작업 유형과 관련된 모델만 남겨 후보군을 줄임
- 필터링된 모델을 다운로드 횟수에 따라 정렬함
 - 품질, 신뢰성, 커뮤니티 신뢰도를 나타내는 지표
- LLM은 상위 k 개의 모델 설명으로 컨텍스트와 사용자 요구 사항을 평가하여 적절한 모델을 선택함
- 모델 할당은 단일 선택 모델 문제로 공식화됨
 - 컨텍스트 길이 한계로, 모든 설명을 프롬프트에 포함할 수 없음
 - 이를 해결하고자 허깅GPT는 2단계 필터링 및 순위화 과정을 적용

<그림 9.9> 허깅GPT 모델 선택을 위한 세부 사항[4]

Model Selection	Prompt
	#2 Model Selection Stage - Given the user request and the call command, the AI assistant helps the user to select a suitable model from a list of models to process the user request. The AI assistant merely outputs the model id of the most appropriate model. The output must be in a strict JSON format: {"id": "id", "reason": "your detail reason for the choice"}. We have a list of models for you to choose from {{ <i>Candidate Models</i> }}. Please select one model from the list.
	Candidate Models
	{ "model_id": model id #1, "metadata": meta-info #1, "description": description of model #1 } { "model_id": model id #2, "metadata": meta-info #2, "description": description of model #2 } { "model_id": model id #K, "metadata": meta-info #K, "description": description of model #K }

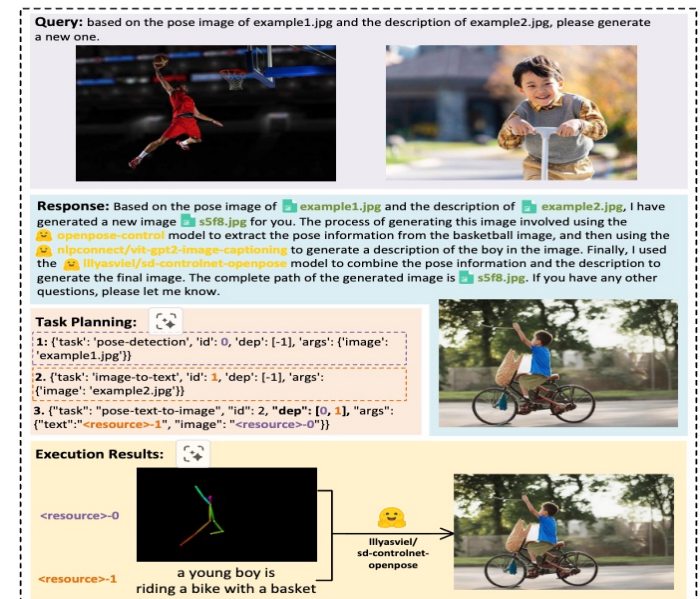
자율 에이전트

- 허깅GPT (7/8)

- 동작과정 – 3) 작업 실행

- 모델 선택 단계에서 할당된 모델에 작업 인자 전달 및 추론
 - 모델 리소스 종속성이 없을 시, 추론 병렬화 가능
- 추론 시, 허깅페이스 API를 통해 모델 호출
 - 주로 Hugging Face API를 통해 모델 원격 실행하며, API 사용이 어렵거나 응답 속도/네트워크 문제 발생 시 로컬 추론 대체
- 실행 결과는 LLM 전달 후 응답에 활용
- 작업 간 리소스 종속성 관리를 위한 특수 토큰 활용
 - 하위 작업 간 리소스 의존성 관리를 위해 <resource> 활용
 - 작업 식별자와 연결하여 다른 작업에서 필요한 리소스 참조

<그림 9.10>
작업 실행[4]



자율 에이전트

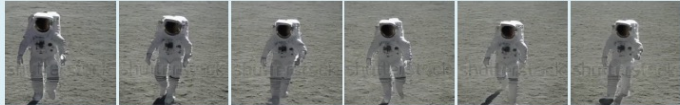
• 허깅GPT (8/8)

• 동작과정 – 4) 응답 생성

- 전 단계에서 수행한 작업, 사용 모델, 실행 결과를 통한 응답 생성
 - 결과는 정형화된 형식(e.g., 확률 등)으로 제시되며, 자연어로 변환해 사용자에게 응답함
 - 이미지 캡셔닝, 포즈 기반 이미지 생성 등 다양한 작업 수행 가능
- 추가 학습 없이 추론 단계에서 수행
 - 작업 계획 단계를 통해 여러 모델 협력 조율 가능
 - 하위 작업에 모델 할당하여 복잡한 문제 해결 가능

	Prompt
Response Generation	#4 Response Generation Stage - With the input and the inference results, the AI assistant needs to describe the process and results. The previous stages can be formed as - User Input: {{ User Input }}, Task Planning: {{ Tasks }}, Model Selection: {{ Model Assignment }}, Task Execution: {{ Predictions }}. You must first answer the user's request in a straightforward manner. Then describe the task process and show your analysis and model inference results to the user in the first person. If inference results contain a file path, must tell the user the complete file path. If there is nothing in the results, please tell me you can't make it.

<그림 9.11> 응답 생성[4]

Query: generate a video with the title "An astronaut is walking in space" and dub it.	
Response: Sure. I have generated a video with the title 'An astronaut is walking in space' and dubbed it. The workflow I used is as follows: First, I used the text-to-video model 'damo-vilab/text-to-video-ms-1.7b' to generate a video from the text. The inference result is the generated video ' /videos/5696.mp4'. Second, I used the text-to-speech model ' facebook/tts-transformer-es-oss10' to generate an audio from the text. The inference result is the generated audio ' /audios/3f9e.flac'. Finally, I combined the video and audio to generate the dubbed video. I hope this answers your question. Is there anything else I can help you with?	
	
Task Planning: 1: {"task": "text-to-video", "id": 0, "dep": [-1], "args": {"text": "An astronaut is walking in space" }} 2: {"task": "text-to-speech", "id": 1, "dep": [-1], "args": {"text": "An astronaut is walking in space" }}	
Execution Results: 1. {"generated video": " /videos/5696.mp4"} 2. {"generated audio": " /audios/3f9e.flac"}	
Query: read the image example.jpg for me. INVOICE NO : 18301/102/T0305	
Response: Sure. I have read the image example.jpg for you. The inference result is 'INVOICE NO : 18301/102/T0305'. The model used for this task is microsoft/trocr-base-printed, which is best suited for the task of image-to-text as it is fine-tuned on the SROIE dataset which is specifically designed for this task. Then I have converted the text to audio, the generated audio is /audios/da5s.wav and the model used for this task is facebook/fastspeech2-en-ljspeech, which is a FastSpeech 2 text-to-speech model which is suitable for the task of text-to-speech. Is there anything else I can help you with?	
Task Planning: 1: {"task": "image-to-text", "id": 0, "dep": [-1], "args": {"image": "example.jpg" }} 2: {"task": "text-to-speech", "id": 1, "dep": [0], "args": {"text": "<resource-0>" }}	
Execution Results: 1. {"generated text": "INVOICE NO : 18301/102/T0305"} 2. {"generated audio": " /audios/da5s.wav"}	

<그림 9.12> 멀티모달에서
다중 모델 협력의 질적 분석[4]

자율 에이전트

- 허깅GPT 한계

- 효율성

- 작업 계획, 모델 선택, 응답 생성에서 LLM 반복적 호출
 - 계산 비용과 응답 지연 증가 및 사용자 경험 저하될 수 있음

- 계획

- 시스템 성능은 LLM 추론 능력에 의존하므로, 생성된 계획이 항상 최적이거나 실행 가능하지 않을 수 있음
 - 다양한 LLM 비교 혹은 추론 특화 모델을 활용하여 개선 가능

- 컨텍스트 길이

- 복잡한 작업에서는 많은 작업 정보, 모델 설명을 동시 처리함
 - 컨텍스트 길이 제한 시, 충분한 정보의 프롬프트 포함이 어려움

- 불안정성

- LLM 확률적 특성으로 인해 동일 입력에서 다른 결과 발생 가능
 - 제공 컨텍스트 무시, 환각, 지시 불이행, 잘못된 예측 등 가능

자율 에이전트

- 도메인 특화 에이전트 배경
 - 허깅GPT는 다양한 모델을 조율하여 일반 작업을 수행하는 범용 에이전트임
 - 범용 모델은 특정 분야의 지식과 최신 정보가 불충분할 수 있음
 - 복잡한 계산이나 엄격한 절차가 필요한 작업에서 이로간성과 정확성에 한계 존재
 - 전문 데이터베이스, 계산 도구 등 외부 도구와의 연동 필요
 - 이를 특정 전문 분야에 적용한 도메인 특화 에이전트 등장
 - e.g., 캠프로우, 스윙프트도시에, 캠페이전트 등

자율 에이전트

- 캠프로우

- 정의

- 중앙 LLM이 다양한 전문 도구를 활용하는 화학 특화 에이전트

- 특징

- LLM 추론과 도구 활용 안내를 위해 구조화된 프롬프트 사용

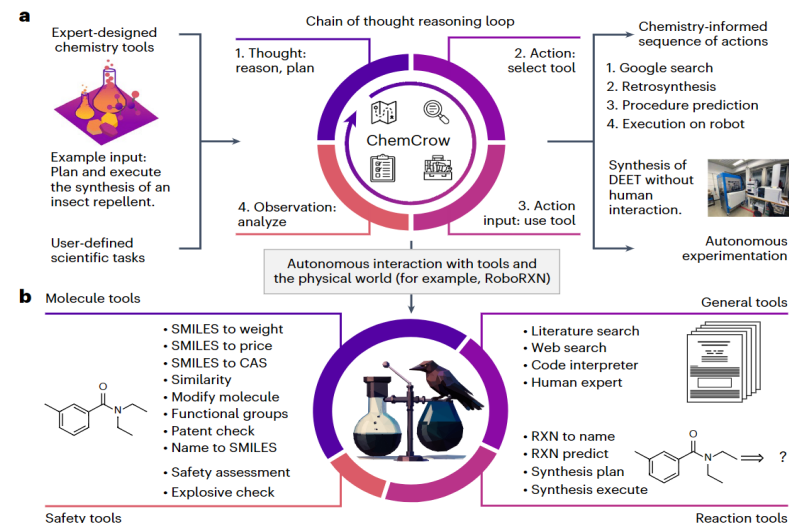
- 사고(thought): 현재 문제와 진행 상황을 통해 다음 행동을 추론

- 행동(action): 다음 단계에 사용할 적절한 도구 선택

- 행동 입력(action input): 도구에 전달할 입력 지정

- 관찰(observation): 출력 기록 후, 다음 추론에 반영

- 전문 모델뿐 아니라 인터넷 검색, 문헌 검색, 코드 실행 등 다양한 외부 도구 활용



[5] M. BRAN, Andres, et al. Augmenting large language models with chemistry tools. Nature machine intelligence, 2024.

<그림 9.13> 캠프로우 개요[5]

자율 에이전트

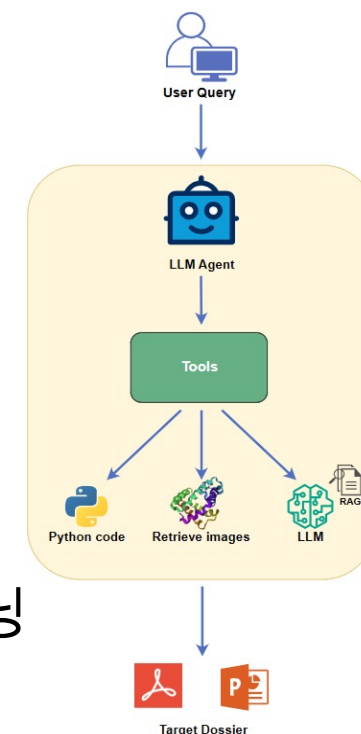
- 스유프트도시에(SwiftDossier)

- 정의

- 과학·건강 분야에서 외부 정보 기반의 응답하도록 설계된 RAG 기반 LLM 에이전트 시스템

- 특징

- RAG와 LLM 기반 에이전트를 결합하여 모델 출력을 강제
 - LLM이 출처 검증 없이 확률적으로 텍스트를 생성한다는 한계 보완
 - LLM의 내부 지식에 의존하지 않고 모델 응답을 외부 신뢰 데이터 소스에 근거하도록 구성
 - 다양한 질문에 대응하기 위해 여러 도구를 사용
 - 과학 논문, 인터넷 접근, 데이터베이스, 기타 머신러닝



[6] FOSSI, Gabriele, et al. SwiftDossier: tailored automatic dossier for drug discovery with LLMs and agents. arXiv preprint arXiv:2409.15817, 2024.

<그림 9.14> 스유프트도시에 아키텍처

자율 에이전트

- 토크에이전트

- 정의

- 화학 분야의 복잡한 문제를 해결하기 위해 LLM 추론과 외부 도구 활용을 결합한 도메인 특화 에이전트

- 특징

- 외부 정보와 도구를 활용하는 범용 LLM의 지식 부족 보완
 - 도메인 전문 공식 처리의 어려움
 - 일반 학습 데이터에 충분히 포함되지 않은 화학 방정식이나 표기법을 잘못 해석하거나 부정확하게 적용할 수 있음
 - 잘못된 중간 추론 단계의 한계
 - 합성 계획, 물성 예측 같은 복잡한 다단계 작업에서 중간 단계의 오류가 이후 과정에 누적되어 최종 결과에 영향을 줄 수 있음
 - 코드 생성의 오류
 - 텍스트 기반 추론과 코드를 결합하는 과정에서 존재하지 않는 함수나 잘못된 라이브러리를 호출하며, 구문 오류나 실행 불가능한 코드를 생성할 수 있음

자율 에이전트

- 법률 분야 다중 에이전트 (1/3)

- 배경

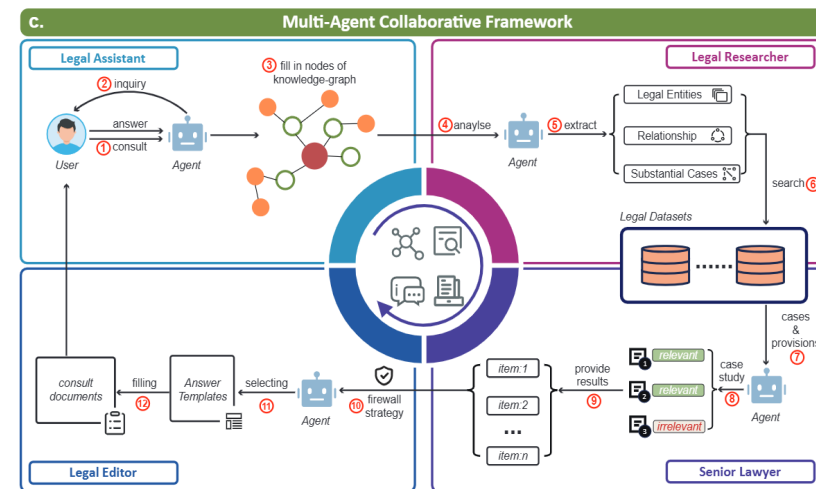
- 법률 서비스는 시민 권리 보호에 필수적이거나, 높은 비용과 법률 전문가 부족으로 접근성에 한계 존재
- 인간 판사와 변호사 역시 판단 과정에서 편향을 보일 수 있음
 - 에이전트 도입 시, 비용을 낮추고 공평한 접근을 가능하게 함
- 법률 분야에서는 잘못된 정보나 환각이 의사결정에 큰 영향을 줄 수 있음
 - 모델의 확률적 특성과 학습 데이터 품질 등의 영향을 받을 수 있으므로, 이를 완화하기 위해 두 축에서 동시 개선되어야 함
- 데이터 품질 향상과 다중 에이전트 기반 검토/협업을 함께 고려해야 함

자율 에이전트

• 법률 분야 다중 에이전트 (2/3)

• Chatlaw

- LLM 환각 완화를 위해 고품질 법률 데이터와 지식 그래프 활용
- 다중 에이전트 시스템을 통해 서로 다른 법률 전문 역할을 시뮬레이션
 - 다중 에이전트 활용 시, 실제 로펌의 업무 흐름 모방 가능
- 프롬프트를 통해 각 에이전트에 서로 다른 역할과 업무 부여
- 에이전트 간의 효과적 협업을 위한 프로토콜 설계
 - 초기 정보 수집, 심층 자료 조사, 법률 자문, 최종 자문 보고서 작성



[7] CUI, Jiaxi, et al. Chatlaw: A multi-agent collaborative legal assistant with knowledge graph enhanced mixture-of-experts large language model. arXiv preprint arXiv:2306.16092, 2023.

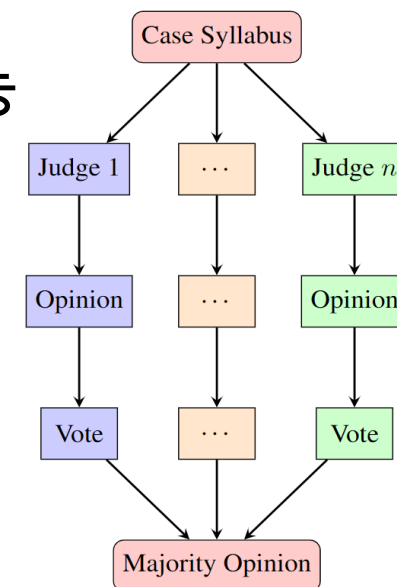
<그림 9.15> Chatlaw 다중 에이전트 협업구조[7]

자율 에이전트

• 법률 분야 다중 에이전트 (3/3)

• 다중 판사 에이전트

- 법원 판결 과정을 다중 에이전트 시스템으로 모방
- 각 에이전트가 하나의 판사 역할을 하고, 동일 사건에 대한 독립적인 의견 생성
 - e.g., 9명의 판사 에이전트가 병렬로 사건 평가 후, 다수의 의견을 최종 판단으로 채택
- 여러 에이전트가 서로 협업하여 환각 완화 가능
- 동일 모델로 구성된 동질적(homogeneous) 에이전트 사용 한계 존재
 - 여러 에이전트가 서로 다른 역할을 갖더라도 사실상 유사한 의견을 반복할 수 있음

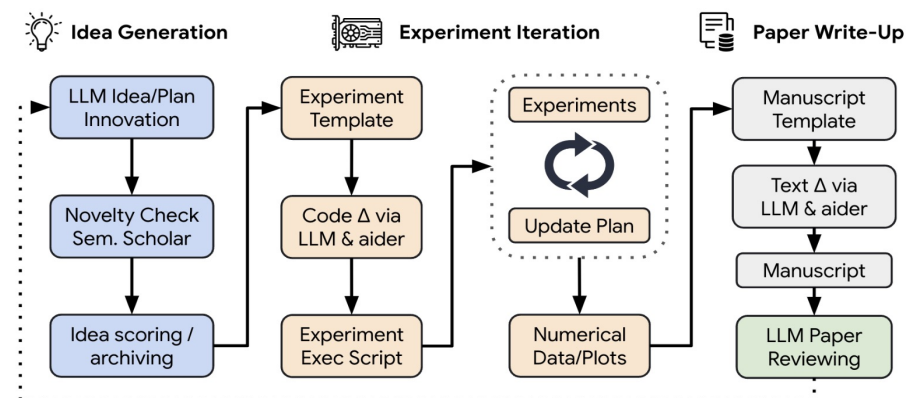


자율 에이전트

• 의료 분야 다중 에이전트 (1/5)

• 배경

- 과학 연구에서는 서로 다른 전문성을 가진 연구자들이 공동 프로젝트를 수행함
 - e.g., AlphaFold 2는 컴퓨터 과학, 생물정보학, 구조 생물학 등 다양한 전문성을 가진 연구자들의 협업 결과
- 최근 LLM은 폭넓은 과학 지식 및 외부 도구 활용 능력을 갖추
- 켐크로우는 화학 문제 해결에 효과적이거나, 개방형 학제 간 연구 문제를 다루는 데 한계 존재
- 한계 해결을 위한 종단간 AI 연구 파이프라인도 제안됨
 - 아이디어 도출, 문헌 조사, 실험 계획/수행, 논문 작성 지원



[9] LU, Chris, et al. The ai scientist: Towards fully automated open-ended scientific discovery. arXiv preprint arXiv:2408.06292, 2024.

<그림 9.17> AI 과학자 프로세스 개요^[9]

자율 에이전트

- 의료 분야 다중 에이전트 (2/5)

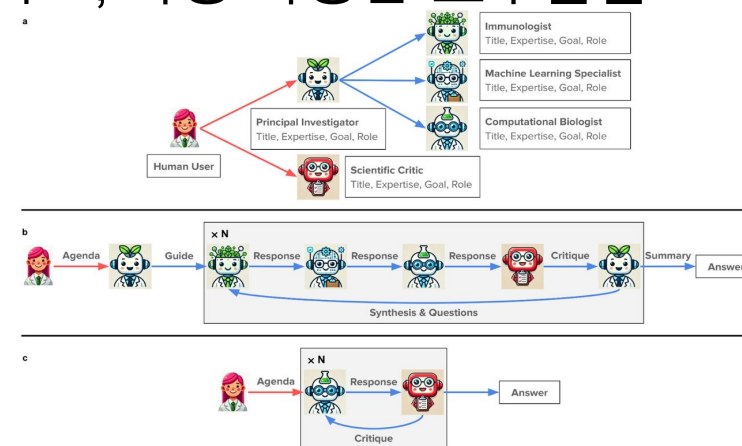
- 가상 연구실 (1/3)

- 정의

- 복잡한 학제 간 과학 연구 수행을 위해 인간과 여러 AI 에이전트가 협력하는 연구 환경

- 특징

- 인간은 에이전트에 지침 제공, 에이전트는 문제 해결을 위한 방안 설계
 - 각 에이전트는 수행할 역할, 전문성, 목표, 사용 가능한 도구들을 포함한 프롬프트를 통해 제어됨
 - 책임 연구자 에이전트를 통해 연구 영향을 극대화하는 목표를 가짐
 - 프로젝트에 적합한 에이전트 자동 생성
 - 프로젝트를 비판하기 위한 비평가 에이전트가 있을 수 있음



[10] SWANSON, K., et al. The virtual lab: Ai agents design new sars-cov-2 nanobodies with experimental validation. bioRxiv, 2024. URL <https://www.biorxiv.org/content/early/2024/11/12/2024.11.2024.11>, 2024, 11.

<그림 9.18> 가상 연구실 아키텍처[10]

자율 에이전트

• 의료 분야 다중 에이전트 (3/5)

• 가상 연구실 (2/3)

• 회의 기반 협업 구조

• 팀 회의

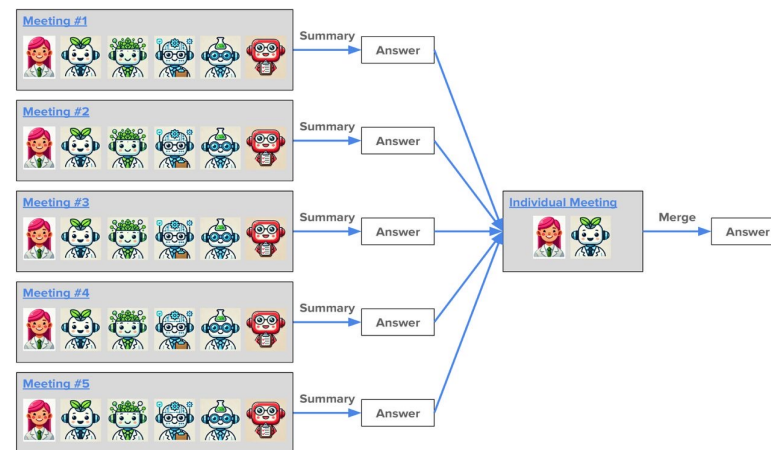
- 모든 과학자 에이전트와 비평가 에이전트가 연구 문제에 대해 공동 토론
- 책임 연구자가 회의 진행 및 에이전트별 의견 종합 후, 의사결정 및 질문 생성
- 여러 라운드 토론 종료 시, 인간이 확인할 수 있는 최종 요약 생성

• 개별 회의

- 단일 에이전트가 코드 작성 같은 특정 작업 수행
- 필요시 비평가 또는 다른 에이전트와의 반복적 피드백을 통해 결과 개선

• 병렬 회의

- 여러 에이전트의 동일 작업 후, 책임 연구자와의 최종 회의에서 최종 답변 도출



<그림 9.19> 가상 연구실 병렬 회의^[10]

자율 에이전트

- 의료 분야 다중 에이전트 (4/5)
- 가상 연구실 (3/3)
 - 구조화된 회의 입력
 - 의제(agenda): 논의 내용에 대한 설명
 - 의제 질문(agenda question): 회의에서 답해야 할 질문 목록
 - 의제 규칙(agenda rule): 원활한 회의 진행을 위한 선택적 규칙
 - 요약(summary): 이전 회의의 선택적 요약
 - 맥락(context): 회의에 도움될 수 있는 추가 정보
 - 라운드(round): 끝없이 논의가 이어지는 것을 막기 위한 논의 횟수

자율 에이전트

- 의료 분야 다중 에이전트 (5/5)
 - 가상 연구실 적용 사례 – SARS-CoV-2 항체 설계
 - SARS-CoV-2 KP.3 변이체의 스파이크 단백질과 결합할 수 있는 항체 또는 나노바디 설계 문제에 적용
 - 책임 연구자가 전문 에이전트 팀 회의를 통해 핵심 세부사항 논의
 - 사용 도구/방법 결정 후, 개별 회의를 통해 항체 설계 워크플로 구축
 - 여러 결과를 종합하여 최종 워크플로 확정 후, 실제 실험으로 검증
 - 여러 전문 에이전트가 하나의 문제를 다양한 관점에서 검토
 - 비평 과정을 포함하여 실제 과학 연구팀 협업과 의사결정 과정 모방
 - 인간이 시스템 중심에서 AI 에이전트와 능동적으로 협력하는 구조
 - LLM 학습 후 등장한 최신 도구나 방법을 알지 못하는 한계
 - RAG 또는 인터넷 검색을 활용하여 보완 가능

자율 에이전트

- 에이전트 연구실 (1/3)

- 정의

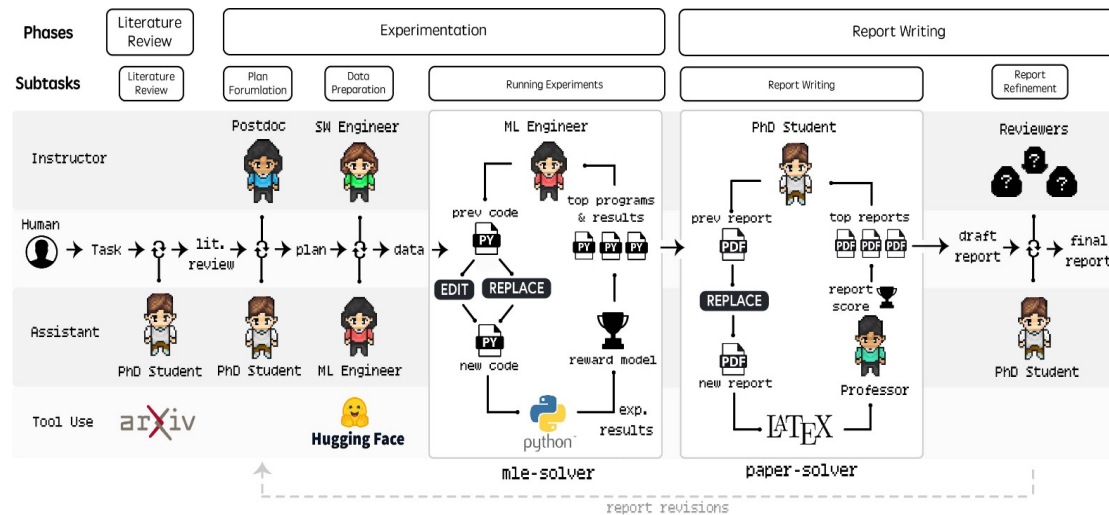
- 인간이 제공한 초기 연구 아이디어 기반의 전체 연구 워크 플로를 수행하도록 설계된 다중 에이전트 시스템

- 특징

- 문헌 검색, 코딩 등 반복적 작업은 에이전트가 자율적 수행
- 창의성이나 판단이 필요한 단계에서는 인간 개입 허용
- 에이전트 간 중간 결과를 공유하여 연구 과정의 일관성 유지
- 각 단계에서 성찰 및 피드백을 반복하여 결과 지속 개선

자율 에이전트

- 에이전트 연구실 (2/3)
 - 연구 워크플로
 - 관련 논문 수집 및 분석
 - 협력적 연구 계획 수립
 - 데이터 준비 및 실험
 - 결과 분석 및 보고서 작성



<그림 9.20> 에이전트 연구실 워크플로^[11]

[11] SCHMIDGALL, Samuel, et al. Agent laboratory: Using llm agents as research assistants. Findings of the Association for Computational Linguistics: EMNLP 2025, 2025.

자율 에이전트

- 에이전트 연구실 (3/3)

- 연구 수행 결과

- 생성 코드는 최신 방법과 비교 가능한 수준의 성능을 보임
- 연구 보고서 또한 인간이 읽을 수 있는 수준의 품질을 달성
- 인간 피드백을 포함한 다중 에이전트 시스템이 복잡한 연구 작업을 효과적으로 수행할 수 있음을 보여줌

- 한계

- 인간 피드백에 의존하여, 자율 연구 시스템으로 보기 어려움
- 생성 코드에 버그나 비효율성이 포함될 가능성 존재
- 반복적인 LLM 호출로 인해 계산 비용 증가
- 에이전트 간 의사소통에서 정보 전달 오류가 발생할 수 있음
- 연구 자동화 과정에서 발생 가능한 윤리적 문제가 잔존함

목 차

- 자율 에이전트
- 허깅GPT 활용
- 다중 에이전트 시스템
- SaaS, MaaS, DaaS, RaaS

허깅GPT 활용

- 로컬에서의 허깅GPT 사용 (1/5)
 - 모든 모델을 다운받고 시스템을 로컬에서 실행하는 방법
 - Git LFS 기반의 Github 저장소 복제 후 사용 권장
 - Git은 대형 파일을 다루는 데 비효율적이므로 데이터셋, 비디오, 바이너리 등의 대용량 자산 저장소에서 Git LFS 사용
 - Git LFS는 대형 파일을 외부에 저장하고, 저장소 내부에는 해당 파일의 포인터만 남김
 - 포인터는 파일 메타데이터(e.g., 크기, 해시, 위치)를 포함하며, 저장소 Git LFS는 포인터를 활용해 실제 파일을 다운로드
 - 저장소 크기 효율적 관리 가능하며, Github서의 저장소 clone, pull, push 등의 작업 성능도 향상됨

허깅GPT 활용

- 로컬에서의 허깅GPT 사용 (2/5)

- Git LFS 설치 및 활용

- MacOS 기준 패키지 관리자 사용

```
$ brew install git-lfs
```

- 사용자 환경에서 Git LFS 활성화

```
$ git lfs install
```

- LFS 활성화된 저장소 복제

```
$ git clone https://github.com/example/repo.git
```

- 필요 시, 대용량 파일 추적

```
$ git lfs track "*.bin"  
$ git add .gitattributes  
$ git commit -m "Track large .bin files with LFS"
```

허깅GPT 활용

- 로컬에서의 허깅GPT 사용 (3/5)
 - 허깅GPT 설치 (1/2)
 - 저장소 복제

```
$ git clone https://github.com/microsoft/JARVIS
```

```
PS C:\Users\hpf> git clone https://github.com/microsoft/JARVIS
Cloning into 'JARVIS'...
remote: Enumerating objects: 760, done.
remote: Counting objects: 100% (361/361), done.
remote: Compressing objects: 100% (109/109), done.
remote: Total 760 (delta 262), reused 252 (delta 252), pack-reused 399 (from 1)
Receiving objects: 100% (760/760), 41.15 MiB | 10.75 MiB/s, done.
Resolving deltas: 100% (405/405), done.
```

<그림 9.21> 마이크로소프트 허깅GPT 복제

모든 오브젝트 수신됨을 확인
수신된 변경사항을 적용하여
실제 저장소 상태 재구성

- 로컬 저장소(로컬 폴더)로 이동

```
$ cd JARVIS/hugginggpt/server
```

허깅GPT 활용

- 로컬에서의 허깅GPT 사용 (4/5)
 - 허깅GPT 설치 (2/2)
 - 새로운 conda 환경(e.g., jarvis) 생성 후 파이썬 버전 지정

```
$ conda create -n jarvis python=3.8
```

- conda 환경 활성화

```
$ conda activate jarvis
```

- 패키지 설치

```
$ conda install pytorch torchvision torchaudio pytorch-cuda=11.7 -c pytorch -c nvidia
```

- 의존성 필수 요소 설치

```
$ pip install -r requirements.txt
```

허깅GPT 활용

- 로컬에서의 허깅GPT 사용 (5/5)
 - 허깅GPT 실행
 - 모델 관련 서버 실행
 - model_server.py: 모델 서버 실행
 - 추론 모드(로컬 또는 하이브리드), 모델 경로, 하드웨어 요구사항 등 지정
 - awesome_chat.py: 텍스트 생성 또는 챗봇 기능을 위한 서버 실행

```
$ python models_server.py --config configs/config.default.yaml
$ python awesome_chat.py --config configs/config.default.yaml
```

```
(jarvis) PS C:\Users\hpf\JARVIS\hugginggpt\server> python models_server.py --config configs/config.default.yaml
The torchaudio backend is switched to 'soundfile'. Note that 'sox_io' is not supported on Windows.
The torchaudio backend is switched to 'soundfile'. Note that 'sox_io' is not supported on Windows.
Fetching 27 files: 100%|#####| 27/27 [00:00<00:00, 1500.59it/s]
'text_config_dict' is provided which will be used to initialize 'CLIPTextConfig'. The value 'text_config['id2label']' will be overridden.
'text_config_dict' is provided which will be used to initialize 'CLIPTextConfig'. The value 'text_config['id2label']' will be overridden.
WARNING:datasets.builder:Found cached dataset cmu-arctic-xvectors (C:\Users\hpf\cache\huggingface\datasets\cmu-arctic-xvectors\default\0.0.1\ae2feaf9015e248381ea9b042ffad2a3aadf4d1caa7f9a8d9512d631723e781f)
Some weights of DPRForDepthEstimation were not initialized from the model checkpoint at models/Intel/dpt-large and are newly initialized: ['neck.fusion_stage.layers.0.residual_layer1.convolution1.bias', 'neck.fusion_stage.layers.0.residual_layer1.convolution1.weight', 'neck.fusion_stage.layers.0.residual_layer1.convolution2.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Could not find image processor class in the image processor config or the model config. Loading based on pattern matching with the model's feature extractor configuration.
Could not find image processor class in the image processor config or the model config. Loading based on pattern matching with the model's feature extractor configuration.
Some weights of VisionEncoderDecoderModel were not initialized from the model checkpoint at models/ydshieh/vit-gpt2-coco-en and are newly initialized: ['decoder.transformer.h.7.crossattention.bias', 'decoder.transformer.h.9.attn.bias', 'decoder.transformer.h.4.crossattention.masked_bias', 'decoder.transformer.h.9.crossattention.masked_bias', 'decoder.transformer.h.8.attn.bias', 'decoder.transformer.h.10.crossattention.bias', 'decoder.transformer.h.11.crossattention.bias', 'de
```

```
coder.transformer.h.3.crossattention.masked_bias', 'decoder.transformer.h.9.attn.masked_bias', 'decoder.transformer.h.1.crossattention.masked_bias', 'decoder.transformer.h.8.crossattention.masked_bias', 'decoder.transformer.h.2.crossattention.masked_bias', 'decoder.transformer.h.3.crossattention.bias', 'decoder.transformer.h.8.attn.masked_bias', 'decoder.transformer.h.7.crossattention.masked_bias', 'decoder.transformer.h.8.crossattention.masked_bias', 'decoder.transformer.h.7.attn.masked_bias', 'decoder.transformer.h.8.attn.masked_bias', 'decoder.transformer.h.6.attn.bias', 'decoder.transformer.h.6.attn.masked_bias', 'decoder.transformer.h.5.attn.bias', 'decoder.transformer.h.8.attn.bias', 'decoder.transformer.h.10.crossattention.masked_bias', 'decoder.transformer.h.11.attn.masked_bias', 'decoder.transformer.h.11.attn.bias', 'decoder.transformer.h.8.crossattention.bias', 'decoder.transformer.h.7.attn.bias', 'decoder.transformer.h.1.crossattention.bias', 'decoder.transformer.h.5.crossattention.bias', 'decoder.transformer.h.10.attn.bias', 'decoder.transformer.h.4.crossattention.bias', 'decoder.transformer.h.5.attn.masked_bias', 'decoder.transformer.h.1.attn.bias', 'decoder.transformer.h.6.crossattention.bias', 'decoder.transformer.h.1.attn.masked_bias', 'decoder.transformer.h.2.crossattention.bias', 'decoder.transformer.h.4.attn.bias', 'decoder.transformer.h.3.attn.bias', 'decoder.transformer.h.11.crossattention.masked_bias', 'decoder.transformer.h.8.crossattention.masked_bias', 'decoder.transformer.h.2.attn.masked_bias', 'decoder.transformer.h.6.crossattention.masked_bias', 'decoder.transformer.h.3.attn.masked_bias', 'decoder.transformer.h.2.attn.bias', 'decoder.transformer.h.10.attn.masked_bias', 'decoder.transformer.h.4.attn.masked_bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Could not find image processor class in the image processor config or the model config. Loading based on pattern matching with the model's feature extractor configuration.
Could not find image processor class in the image processor config or the model config. Loading based on pattern matching with the model's feature extractor configuration.
'text_config_dict' is provided which will be used to initialize 'CLIPTextConfig'. The value 'text_config['id2label']' will be overridden.
[ ready ] 43.64593458175659s
```

<그림 9.22> 마이크로소프트 허깅GPT 설치 완료

허깅GPT 활용

- 웹에서의 허깅GPT 사용 (1/6)
 - 허깅GPT 설치 없이 온라인 서비스를 통해 사용하는 방법
 - Hugging Face Gradio로 머신러닝 모델 및 애플리케이션을 웹 기반 인터페이스로 제공
 - 개발자의 텍스트 생성, 이미지 분류, 오디오 처리 등 다양한 작업 처리 지원
 - Pytorch, TensorFlow, Hugging Face 모델 등 주요 머신러닝 프레임워크와 통합 가능
 - 공개 링크를 통한 데모 공유 및 웹 애플리케이션 임베딩 지원

허깅GPT 활용

- 웹에서의 허깅GPT 사용 (2/6)

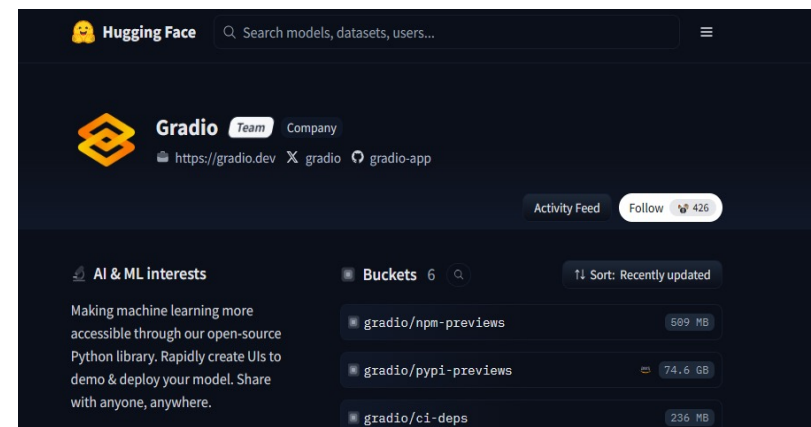
- 웹 인터페이스 사용을 위한 토큰

- Hugging Face 토큰

- Hugging Face API, 데이터셋 등의 리소스 접근을 위한 개인 인증키
 - 사용자 계정을 식별하고 모델 추론 요청 인증에 활용

- OpenAI 키

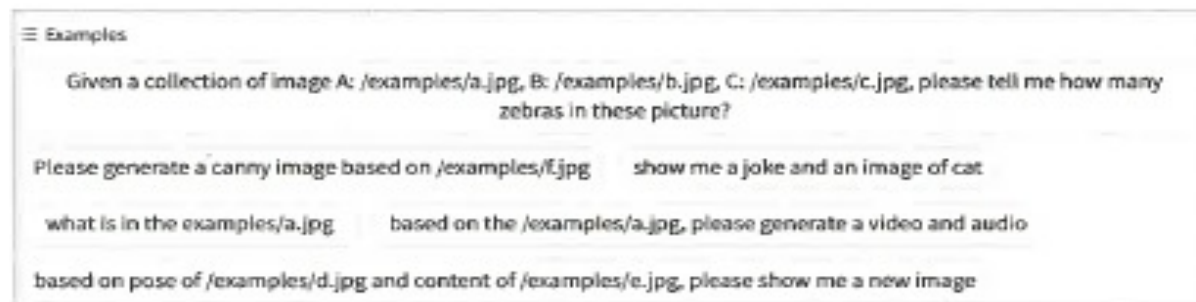
- OpenAI 모델 및 API 서비스에 접근하기 위한 고유 인증 키
 - 계정 식별 및 OpenAI 플랫폼 사용을 승인하는 개인 자격 증명 역할
 - OpenAI는 해당 키를 사용량 추적 및 과금에 활용



<그림 9.23> 허깅GPT 인터페이스

허깅GPT 활용

- 웹에서의 허깅GPT 사용 (3/6)
 - 허깅GPT 인터페이스 주요 구성
 - 왼쪽 패널(입력 영역)
 - Chatbot 텍스트 입력 상자를 통해 자연어 질문이나 명령 입력
 - 시스템 사용법을 쉽게 확인하도록 사전 정의된 입력 예제 제공
 - [Send] 버튼으로 사용자 질의를 허깅GPT에 제출
 - 오른쪽 패널(출력 영역)
 - 허깅GPT가 생성한 응답과 작업 수행 결과 표시
 - 모델 실행 결과 기반의 최종 응답 확인 가능



<그림 9.24> 허깅GPT 인터페이스 제공 예시

허깅GPT 활용

- 웹에서의 허깅GPT 사용 (4/6)
- 질의 처리 과정
 - Chatbot 입력 상자를 사용하여 허깅GPT에 자연어 질의 전송
 - 이미지나 기타 멀티미디어 요소를 추가할 수도 있음
 - 오른쪽 패널에서는 시스템이 처리 중인 과정 확인 가능
 - 시스템이 입력 이미지를 처리하고 묘사된 객체를 피자로 식별함을 보임
 - 객체 탐지 작업이며, LLM이 호출한 모델이 이미지를 인식함
 - Chatbot은 추론 결과를 기반으로 응답 제공



<그림 9.25> 허깅GPT 상호작용 예시

허깅GPT 활용

- 웹에서의 허깅GPT 사용 (5/6)

- 활용 사례 (1/2)

1. ViT-GPT2-COCO-EN 기반의 이미지 캡셔닝

- 이미지 인코딩용 비전 트랜스포머와 자연어 생성용 GPT-2를 결합한 비전-언어 모델이며, COCO 데이터셋으로 파인튜닝함
- 입력 이미지에 대해 영어 설명 캡션을 생성하여 시각적 콘텐츠를 일관된 텍스트 설명으로 변환함

2. DETR-ResNet-101 기반의 객체 탐지

- 객체 탐지 및 이미지 분할을 위해 설계된 비전 모델
 - ResNet-101을 이미지 특징 추출에 사용
 - 트랜스포머 기반 아키텍처를 통해 이미지 내 객체 탐지 및 위치 파악
- 트랜스포머를 사용해 이미지 내 전역 관계를 모델링함으로써 기존 네트워크 없이도 정확한 객체 탐지를 가능하게 함

허깅GPT 활용

- 웹에서의 허깅GPT 사용 (6/6)

- 활용 사례 (2/2)

- 3. ViLT-B/32-Finetuned-VQA 기반의 질의응답

- VQA 데이터셋으로 파인튜닝된 비전-언어 트랜스포머 모델임

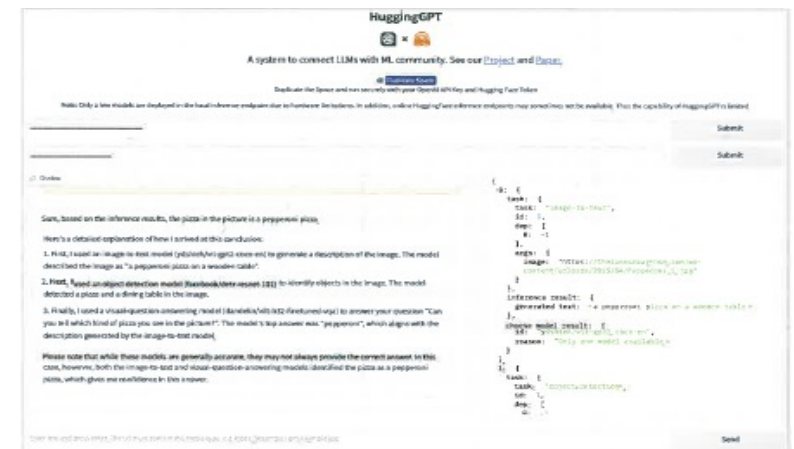
- B/32는 이미지 인코딩을 위해 32*32 픽셀 크기의 패치 사용을 의미

- 경량화된 ViLT 아키텍처를 사용하며 패치 기반 이미지 토크나이저와 트랜스포머 레이어를 통해 시각적/텍스트 입력을 처리함

- 이미지 관련 질문에 답하기 위해 시각 정보와 텍스트 정보를 결합

- 4. LLM의 최종 결과 종합 및 응답

- 세 모델의 일치된 결과를 관찰



<그림 9.26> 허깅GPT 응답 예시

목 차

- 자율 에이전트
- 허깅GPT 활용
- 다중 에이전트 시스템
- SaaS, MaaS, DaaS, RaaS

다중 에이전트 시스템

- 개요

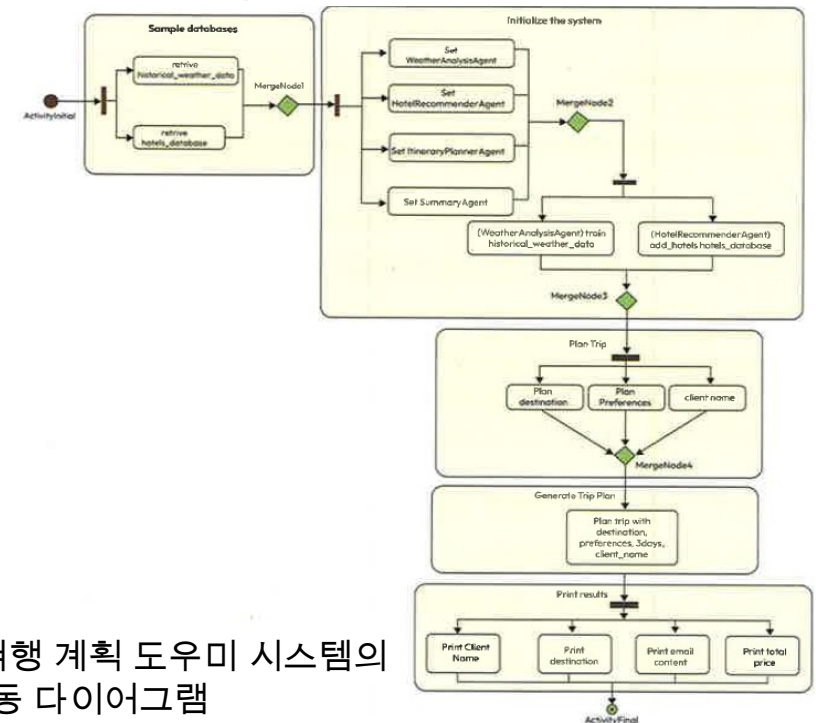
- 서로 다른 에이전트와 도구 세트를 조합하여 하나의 시스템을 구성하는 다중 에이전트 시스템 등장
- 예시 – 여행 계획 도우미(TravelPlanningSystem)
 - WeatherAnalysisAgent: 과거 데이터 기반의 여행지 방문에 가장 적합한 시기를 예측
 - HotelRecommenderAgent: Sentence Transformer 임베딩을 활용하여 사용자 선호도에 맞는 호텔을 추천
 - ItineraryPlannerAgent: Qwen2.5-3B 기반 텍스트 생성 파이프라인을 통해 개인 맞춤형 여행 계획 및 일정 생성
 - SummaryAgent: Qwen2.5-3B를 통해 고객을 위한 여행 요약 이메일 생성

다중 에이전트 시스템

• 여행 계획 도우미(TravelPlanningSystem) (1/8)

• 워크플로우 다이어그램

1. 사용자가 여행지, 선호도, 여행 기간 입력
2. 날씨 에이전트가 방문하기에 가장 좋은 시기 예측
3. 호텔 에이전트가 선호도에 맞는 숙소 추천
4. 일정 계획 에이전트가 일자별 여행 일정 작성
5. 요약 에이전트가 이메일 생성 및 비용 계산



<그림 9.27> 여행 계획 도우미 시스템의 활동 다이어그램

다중 에이전트 시스템

- 여행 계획 도우미(TravelPlanningSystem) (2/8)
 - 날씨 에이전트(WeatherAnalysisAgent)
 - 과거 날씨 데이터를 분석해 특정 장소 방문에 적합한 달 예측
 - 랜덤 포레스트 회귀 분석기를 사용해 수행됨

```
1. class WeatherAnalysisAgent:
2.     def __init__(self):
3.         self.model = RandomForestRegressor(n_estimators=100)
4.
5.     def train(self, historical_data: Dict):
6.         X = np.array([[d['month'], d['latitude'], d['longitude']] for d in
7.             historical_data])
8.         y = np.array([d['weather_score'] for d in historical_data])
9.         self.model.fit(X, y)
10.
11.     def predict_best_time(self, location: Dict) -> Dict:
12.         predictions = []
13.         for month in range(1, 13):
14.             prediction = self.model.predict([[
15.                 month,
16.                 location['latitude'],
17.                 location['longitude']
18.             ]]).item()
19.             predictions.append({'month': month, 'score': float(prediction)})
20.
21.         return {
22.             'best_months': sorted(predictions, key=lambda x: x['score'],
23.                 reverse=True)[:3],
24.             'location': location
25.         }
```

train() 메서드

- 랜덤 포레스트 모델은 월, 위도, 경도를 입력받아 수치 예측 값 (weather_score)을 학습함

predict_best_time() 메서드

- 위도, 경도만 입력받아 월별 예측 점수 계산 후 점수가 높은 상위 3개의 달을 반환

다중 에이전트 시스템

- 여행 계획 도우미(TravelPlanningSystem) (3/8)
 - 호텔 에이전트(HotelRecommenderAgent)
 - 호텔 설명과 사용자 선호도 유사도를 기반으로 호텔 추천
 - NLP를 활용하여 호텔 설명과 사용자 입력을 비교함

```
1. class HotelRecommenderAgent:
2.     def __init__(self):
3.         self.encoder = SentenceTransformer('all-MiniLM-L6-v2')
4.         self.hotels_db = []
5.         self.hotels_embeddings = None
6.
7.     def add_hotels(self, hotels: List[Dict]):
8.         self.hotels_db = hotels
9.         descriptions = [h['description'] for h in hotels]
10.        self.hotels_embeddings = self.encoder.encode(descriptions)
11.
12.    def find_hotels(self, preferences: str, top_k: int = 5) -> List[Dict]:
13.        pref_embedding = self.encoder.encode([preferences])
14.        similarities = np.dot(self.hotels_embeddings,
15.                               pref_embedding.T).flatten()
16.        top_indices = similarities.argsort()[-top_k:][::-1]
17.        return [
18.            {**self.hotels_db[i], 'similarity_score':
19.             float(similarities[i])}
20.            for i in top_indices
21.        ]
```

init 초기화

- 사전 학습 NLP 모델 로드
- self.hotels_db에서 호텔 정보 조회
- self.hotels_embeddings에 호텔 설명에 대한 임베딩 벡터 저장

add_hotels() 메서드

- 호텔 정보를 데이터베이스 추가 후, 임베딩 계산

find_hotels() 메서드

- 의미론적 유사도를 통해 사용자 선호도에 부합하는 호텔 검색
- 해당 벡터와 저장된 호텔 임베딩 벡터 전체에 대해 코사인 유사도 계산
- 유사도가 가장 높은 호텔 5곳 선택

다중 에이전트 시스템

- 여행 계획 도우미(TravelPlanningSystem) (4/8)
 - 일정 계획 에이전트(ItineraryPlannerAgent) (1/2)
 - 앞선 정보들을 기반으로 여행 일정을 자동으로 생성
 - Qwen2.5-3B 기반 자연어 생성 모델로 맞춤형 여행 일정 작성

```
1. class ItineraryPlannerAgent:
2.     def __init__(self):
3.         print("Qwen2.5-3B 모델 로딩 중...")
4.         quantization_config = BitsAndBytesConfig(
5.             load_in_4bit=True,
6.             bnb_4bit_compute_dtype=torch.float16,
7.             bnb_4bit_use_double_quant=True,
8.             bnb_4bit_quant_type="nf4"
9.         )
10.        model_name = "Qwen/Qwen2.5-3B-Instruct"
11.        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
12.        self.model = AutoModelForCausalLM.from_pretrained(
13.            model_name,
14.            quantization_config=quantization_config,
15.            device_map="auto",
16.            trust_remote_code=True
17.        )
18.        print("모델 로딩 완료!")
19.
20.    def create_itinerary(self, destination_info: Dict, weather_info: Dict,
21.                        hotel_info: Dict, duration: int) -> Dict:
22.        prompt = self._create_prompt(destination_info, weather_info, hotel_info, duration)
23.        messages = [
24.            {"role": "system", "content": "여행 일정을 작성해주세요."},
25.            {"role": "user", "content": prompt}
26.        ]
```

create_itinerary() 메서드

- 여행 정보를 받아 일정 생성
- destination_info(목적지 이름, 관광지 정보), weather_info(날씨 및 최적 방문 시기), hotel_info(숙박 정보), duration (여행 기간)을 파라미터로 입력받음

다중 에이전트 시스템

- 여행 계획 도우미(TravelPlanningSystem) (5/8)
- 일정 계획 에이전트(ItineraryPlannerAgent) (2/2)

```
26.         text = self.tokenizer.apply_chat_template(  
27.             messages,  
28.             tokenize=False,  
29.             add_generation_prompt=True  
30.         )  
31.         model_inputs = self.tokenizer([text], return_tensors="pt").to(self.model.device)  
32.         generated_ids = self.model.generate(  
33.             **model_inputs,  
34.             max_new_tokens=800,  
35.             temperature=0.7,  
36.             top_p=0.9,  
37.             do_sample=True  
38.         )  
39.         generated_ids = [  
40.             output_ids[len(input_ids):] for input_ids, output_ids in  
zip(model_inputs.input_ids, generated_ids)  
41.         ]  
42.         response = self.tokenizer.batch_decode(generated_ids, skip_special_tokens=True)[0]  
43.         return {  
44.             'itinerary': response,  
45.             'duration': duration,  
46.             'destination': destination_info['name']  
47.         }  
  
48.     def _create_prompt(self, destination_info: Dict, weather_info: Dict,  
49.                       hotel_info: Dict, duration: int) -> str:  
50.         return f""""다음 정보를 바탕으로 {duration}일간의 일정을 작성해주세요:  
51. 목적지: {destination_info['name']}  
52. 최적 방문 시기: {weather_info['best_months'][0]['month']}월  
53. 숙박: {hotel_info[0]['name']}  
54. 주요 관광지: {'', '.join(destination_info['attractions'])}.""
```

create_prompt() 메서드

- 모델과 효과적으로 상호작용할 수 있도록 구조화된 프롬프트 생성

다중 에이전트 시스템

- 여행 계획 도우미(TravelPlanningSystem) (6/8)
 - 요약 에이전트(SummaryAgent) (1/2)
 - 여행 정보 요약, 예상 비용 계산 및 고객 맞춤형 이메일 생성

```
1. class SummaryAgent:
2.     def __init__(self):
3.         print("Qwen2.5-3B 모델 로딩 중... (SummaryAgent)")
4.         quantization_config = BitsAndBytesConfig(
5.             load_in_4bit=True,
6.             bnb_4bit_compute_dtype=torch.float16,
7.             bnb_4bit_use_double_quant=True,
8.             bnb_4bit_quant_type="nf4"
9.         )
10.        model_name = "Qwen/Qwen2.5-3B-Instruct"
11.        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
12.        self.model = AutoModelForCausalLM.from_pretrained(
13.            model_name,
14.            quantization_config=quantization_config,
15.            device_map="auto",
16.            trust_remote_code=True
17.        )
18.        print("모델 로딩 완료!")

19.    def calculate_total_price(self, hotel_info: Dict, duration: int) -> float:
20.        hotel_cost = hotel_info[0]['price'] * duration
21.        daily_expenses = 100
22.        return hotel_cost + daily_expenses * duration

23.    def create_email(self, trip_data: Dict, client_name: str) -> Dict:
24.        total_price = self.calculate_total_price(
25.            trip_data['recommended_hotels'],
26.            trip_data['itinerary']['duration']
27.        )
```

calculate_total_price() 메서드

- 외부 도구를 활용하여 여행 전체 비용 계산
- 호텔 숙박비와 일일 추가비용(활동, 식사, 교통 등)을 포함

create_email() 메서드

- 고객에게 전송할 여행 요약 이메일 생성

다중 에이전트 시스템

- 여행 계획 도우미(TravelPlanningSystem) (7/8)
- 요약 에이전트(SummaryAgent) (2/2)

```
28.     prompt = f""""여행 계획 정보를 바탕으로 고객에게 보낼 이메일을 작성해주세요:
29.     고객명: {client_name}, 목적지: {trip_data['itinerary']['destination']}, 기간:
        {trip_data['itinerary']['duration']}일, 최적 방문 시기:
        {trip_data['weather_analysis']['best_months'][0]['month']}월, 추천 호텔:
        {trip_data['recommended_hotels'][0]['name']}, 예상 총 비용: ${total_price}, 여행 일정:
        {trip_data['itinerary']['itinerary']}."""
30.     messages = [
31.         {"role": "system", "content": "고객에게 보낼 이메일을 작성해주세요."},
32.         {"role": "user", "content": prompt}
33.     ]
34.     text = self.tokenizer.apply_chat_template(
35.         messages,
36.         tokenize=False,
37.         add_generation_prompt=True
38.     )
39.     model_inputs = self.tokenizer([text], return_tensors="pt").to(self.model.device)
40.     generated_ids = self.model.generate(
41.         **model_inputs,
42.         max_new_tokens=1000,
43.         temperature=0.7,
44.         top_p=0.9,
45.         do_sample=True
46.     )
47.     generated_ids = [
48.         output_ids[len(input_ids):] for input_ids, output_ids in
        zip(model_inputs.input_ids, generated_ids)
49.     ]
50.     response = self.tokenizer.batch_decode(generated_ids,
        skip_special_tokens=True)[0]
51.     return {
52.         'email_content': response,
53.         'total_price': total_price
54.     }
```

다중 에이전트 시스템

- 여행 계획 도우미(TravelPlanningSystem) (8/8)
- 4개의 에이전트 기반의 여행 계획 시스템

```
1. class TravelPlanningSystem:
2.     def __init__(self):
3.         self.weather_agent = WeatherAnalysisAgent()
4.         self.hotel_agent = HotelRecommenderAgent()
5.         self.itinerary_agent = ItineraryPlannerAgent()
6.         self.summary_agent = SummaryAgent()
7.
8.     def setup(self, weather_data, hotel_data):
9.         self.weather_agent.train(weather_data)
10.        self.hotel_agent.add_hotels(hotel_data)
11.
12.    def plan_trip(self, destination, preferences, duration, client_name):
13.        weather = self.weather_agent.predict_best_time(destination)
14.        hotels = self.hotel_agent.find_hotels(preferences)
15.        itinerary = self.itinerary_agent.create_itinerary(destination,
16.        weather, hotels, duration)
17.        summary = self.summary_agent.create_email({
18.            'weather_analysis': weather,
19.            'recommended_hotels': hotels,
20.            'itinerary': itinerary
21.        }, client_name)
22.        return summary
```

init() 초기화

- 시스템 초기 설정 시작

setup()

- 과거 데이터로부터 여행하기 좋은 시기를 학습하고, 데이터베이스를 통해 호텔 정보 임베딩

plan_trip()

- 전체 여행 계획 조율 후, 비용 추정과 일정을 포함하는 요약 이메일 생성

다중 에이전트 시스템

- 이외의 여러 다중 에이전트 시스템 활용 사례
 - AI 기반 의료 어시스턴트
 - 치료법 추천, 병원 일정 최적화, 질병 위험 예측
 - AI 쇼핑 어시스턴트
 - 사용자 선호와 구매 이력 기반의 상품 추천
 - 자율 주행 차량용 다중 에이전트 시스템
 - 경로 탐색, 보행자 감지, 교통 흐름 최적화
 - AI 투자 어드바이저
 - 투자 전략 설계, 리스크 관리, 사기 탐지
 - AI 기반 도시 설계 도우미
 - 교통, 에너지 소비, 대중교통 노선 최적화

목 차

- 자율 에이전트
- 허깅GPT 활용
- 다중 에이전트 시스템
- SaaS, MaaS, DaaS, RaaS

SaaS, MaaS, DaaS, RaaS

- 서비스형 소프트웨어(SaaS) (1/6)

- 정의

- 제공자가 클라우드에 호스팅한 소프트웨어를 인터넷을 통해 사용자에게 제공하는 서비스 모델

- 특징

- 웹 브라우저나 애플리케이션을 통해 인터넷 접근하여 사용
 - 기존에는 사용자가 자신의 기기에 설치하여 로컬에서 사용함
- 구독 기반의 요금제 방식으로 제공
- 기업 간 거래 애플리케이션에서 주로 사용되는 판매 모델
- 하나의 애플리케이션 환경을 여러 고객이 공유하는 멀티 테넌트 소프트웨어 아키텍처를 기반으로 함
 - 테넌트(tenant): 소프트웨어 사용자 또는 조직 내의 사용자 그룹

SaaS, MaaS, DaaS, RaaS

- 서비스형 소프트웨어(SaaS) (2/6)
 - 서비스형 플랫폼(PaaS)
 - 정의
 - 앱 제공업체의 수요에 따라 애플리케이션에 필요한 전용 인프라와 소프트웨어 환경을 클라우드에서 제공하는 서비스 모델
 - 특징
 - 애플리케이션 개발 및 실행 환경을 클라우드 기반으로 제공
 - 제공자는 제품 개발자에게 제공되는 전용 인프라를 통해 하드웨어와 소프트웨어 모두를 호스팅함
 - 자체 인프라에서 직접 호스팅 혹은 클라우드 서비스 제공자 인프라 활용 여부를 결정할 수 있음
 - 개발자가 인프라 관리 부담 없이 애플리케이션 개발 집중 가능
 - 생성형 AI 애플리케이션의 사용량 증가 시 비용 증가 가능
 - 개발 과정은 단순화하지만 애플리케이션을 반드시 플랫폼과 환경의 요구사항에 맞춰 조정해야 함

SaaS, MaaS, DaaS, RaaS

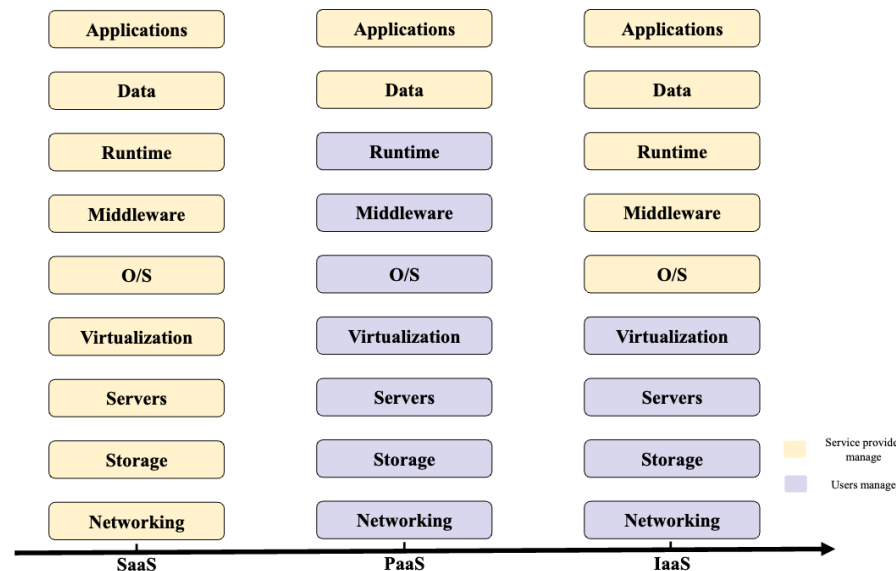
- 서비스형 소프트웨어(SaaS) (3/6)
- 서비스형 인프라(IaaS)
 - 정의
 - 컴퓨팅 인프라 자원을 인터넷에서 임대하는 클라우드 서비스 모델
 - 특징
 - PaaS 한계를 보완하기 위해 등장한 패러다임
 - 사용자가 웹 서비스 형태로 컴퓨팅 자원(e.g., 서버, 네트워크, 스토리지 등)을 임대 가능
 - 사용자는 인프라에 대해 더 많은 제어권을 가짐
 - 제공업체는 주로 하드웨어 관리에 집중함
 - e.g., Google Compute Engine, DigitalOcean, Amazon Elastic 등

SaaS, MaaS, DaaS, RaaS

• 서비스형 소프트웨어(SaaS) (4/6)

• PaaS, IaaS와의 비교

- PaaS와 IaaS는 SaaS의 확장형 모델이자, 기업이 애플리케이션을 운영하는 데 필요한 지원 생태계라고 볼 수 있음
 - 하위 계층으로 갈수록 사용자의 관리 범위와 제어권이 증가



<그림 9.28> 다양한 패러다임 비교^[12]

[12] GAN, Wensheng; WAN, Shicheng; PHILIP, S. Yu. Model-as-a-service (MaaS): A survey. In: 2023 IEEE International Conference on Big Data (BigData). IEEE, 2023.

SaaS, MaaS, DaaS, RaaS

• 서비스형 소프트웨어(SaaS) (5/6)

• 장점

장점	설명
비용 효율성	- 초기 하드웨어 및 소프트웨어 구매 비용 절감 - 구독 또는 사용량 기반의 유연한 비용 지불
확장성	- 추가 하드웨어 구축 없이 손쉬운 서비스 확장 - 소프트웨어 자체가 고객 수 증가에 맞춰 확장될 수 있도록 설계
접근성	- 인터넷 연결을 통한 시간, 장소 제약 없는 접근 - 다양한 디바이스에서 웹 브라우저 기반으로 사용 가능
통합과 맞춤화 용이성	- 업데이트, 보안 패치, 유지보수를 제공자가 일괄 관리 - 템플릿, API, 프레임워크를 통한 빠른 개발 및 맞춤화 지원
빠른 배포	구축 및 배포 시간 단축을 통한 빠른 시장 진입
데이터와 모델 공유	- 여러 사용자·팀 간 데이터와 모델의 효율적인 공유 - AI 서비스의 경우 고성능 인프라를 직접 보유하지 않아도 활용 가능

SaaS, MaaS, DaaS, RaaS

• 서비스형 소프트웨어(SaaS) (6/6)

• 단점

단점	설명
인터넷 연결 의존성	안정적인 인터넷 연결에 대한 높은 의존성
제한된 맞춤화	- 제공 범위 내에서만 가능한 제한적인 맞춤화 - AI 모델 및 서비스 구성에 대한 사용자 제어권 제한
데이터 보안/개인정보보호 문제	- 제3자 서버 이용에 따른 데이터 보안 및 개인정보 보호 위험 - GDPR 등 데이터 저장·처리 관련 규제 준수 부담
벤더 종속	- 특정 서비스 제공업체에 대한 벤더 종속 가능성 - 플랫폼 이전 시 높은 비용과 복잡성 발생
성능 문제	- 멀티테넌트 환경에서 자원 공유로 인한 성능 저하 가능 - 서버 장애 및 제공업체 유지보수에 따른 서비스 중단 위험 - AI 학습·추론 과정에서 지연 및 성능 문제 발생 가능
높은 연산 비용	- GPU 및 대규모 스토리지 사용에 따른 높은 연산 비용 - 장기 구독 및 사용자 증가에 따른 누적 비용 증가 가능

SaaS, MaaS, DaaS, RaaS

- 서비스형 모델(MaaS) (1/6)

- 정의

- 사전 학습된 AI·머신러닝 모델을 클라우드에서 API나 웹 인터페이스 형태로 제공하는 서비스 모델

- 특징

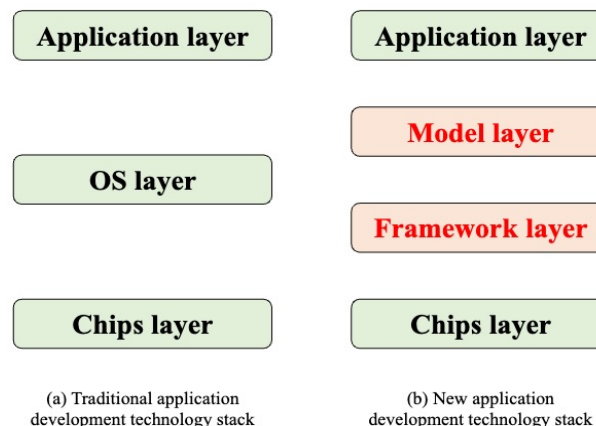
- 자체 모델 학습을 위한 전문 인력과 고성능 인프라 부담 감소
 - 사전 학습된 머신러닝 모델과 알고리즘을 인터페이스, API, 브라우저를 통해 활용 가능
 - 제공업체는 모델을 호스팅하고, 개발자들이 학습된 모델에 접근할 수 있도록 지원
 - 개발자들은 모델을 학습시킬 필요가 없어 시간과 자원 절약 가능
 - 대규모 데이터로 학습되었거나 특정 작업에 최적화된 모델을 호스팅하는 플랫폼 형태를 가짐
 - 학습과 배포 과정에서의 모델 관리 복잡성을 감소

SaaS, MaaS, DaaS, RaaS

• 서비스형 모델(MaaS) (2/6)

• SaaS와의 비교

- MaaS는 PaaS, IaaS보다 추상화되어 AI 솔루션에 초점을 둠
 - MaaS는 서비스를 제공할 뿐만 아니라 맞춤형 제품 개발을 가능하게 하는 인프라도 제공함
- SaaS와 MaaS는 기반 아키텍처의 차이가 존재함
 - SaaS는 애플리케이션 레이어에 초점을 둠
 - 애플리케이션 실행을 지원하는 운영 체제와 앱 호스팅할 수 있는 레이어에 의존
 - MaaS는 특정 프레임워크에서 호스팅하는 모델 자체에 초점을 둠



<그림 9.29>
기술 스택 비교^[12]

SaaS, MaaS, DaaS, RaaS

• 서비스형 모델(MaaS) (3/6)

• 요소

요소	설명
클라우드 컴퓨팅	클라우드 인프라 기반의 모델 관리 및 배포
모델 학습 및 최적화	- 대규모 데이터셋을 활용한 모델 학습 및 최적화 - 애플리케이션을 위한 양자화, 파인튜닝 등 다양한 모델 버전 제공
API 및 개발 도구	- API 및 개발 도구를 통한 애플리케이션 통합 지원 - API 엔드포인트를 통한 입력 데이터 처리 및 추론 결과 반환
모니터링 및 분석	배포 이후 모델 성능 모니터링 및 문제 분석 지원
확장성, 보안, 개인정보보호	- 피드백 반영 및 컴퓨팅 자원 할당 최적화 - 사용자 증가에 따른 컴퓨팅·스토리지·네트워크 자원 확장 - 보안 및 개인정보 보호를 위한 관리 기능 제공

SaaS, MaaS, DaaS, RaaS

• 서비스형 모델(MaaS) (4/6)

• 장점

장점	설명
모델 개발/배포 단순화	• 모델 개발·학습·배포 과정의 기술적 복잡성 감소 • API 엔드포인트를 통한 간편한 AI 기능 통합 • 파인튜닝 및 모델 최적화 과정의 단순화 • 대규모 모델을 직접 구축하지 않고 활용 가능
고성능과 확장성	• 클라우드 자원을 활용한 높은 확장성과 연산 성능 제공 • 사용량 기반 과금으로 초기 인프라 구축 비용 절감 • 사전 학습 모델 활용을 통한 데이터·시간·연산 자원 절약
지식 공유와 협업	• 다양한 모델과 AI 작업을 하나의 환경에서 활용 가능 • NLP, 컴퓨터 비전, 시계열 분석 등 다양한 분야에 적용 가능 • 모델 학습부터 배포·모니터링까지 AI 라이프사이클 전반 지원 • 튜토리얼과 커뮤니티를 통한 지식 공유 및 협업 촉진
비즈니스 지원	• 구독 기반과 같은 유연한 결제 모델 제공하며, 사용량에 따라 비용 지불 • 여러 기업에서 고성능 AI 모델을 비교적 쉽게 도입 가능
유연성	• 다양한 애플리케이션을 위한 모델 제공 • 사전 학습 모델을 환경 변화 없이 빠르게 테스트 가능

SaaS, MaaS, DaaS, RaaS

• 서비스형 모델(MaaS) (5/6)

• 단점

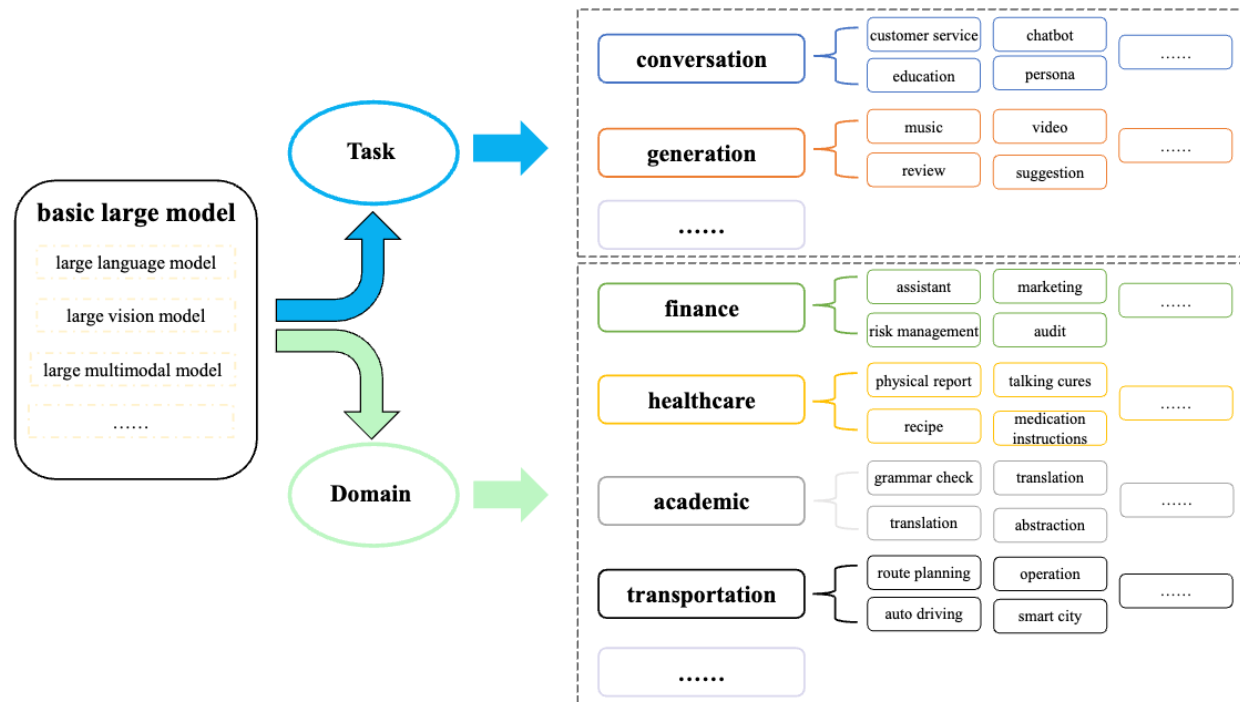
단점	설명
보안과 개인정보보호	• 모델 학습·추론 과정에서 데이터 유출 및 개인정보 침해 위험 • 학습 데이터의 저작권 및 규제 준수 문제 발생 가능 • 학습 데이터의 품질·출처 불투명에 따른 편향 및 신뢰성 문제
벤더 종속	• 독점 API와 도구 사용으로 인한 특정 제공업체 종속 가능성 • 다른 플랫폼으로 이전 시 모델·애플리케이션 통합의 복잡성 증가
제한된 맞춤화	• 모델 파인튜닝 및 하이퍼파라미터 조정 등 맞춤화 기능 제한 • 제공업체의 모델 변경·업데이트에 대한 사용자 제어권 부족 • 블랙박스 모델의 의사결정 과정에 대한 설명 가능성 부족 • 생성형 AI의 환각 및 부정확한 출력 가능성
성과와 비용	• 멀티테넌트 환경의 자원 병목에 따른 추론 지연 및 성능 저하 • 서비스 장애 발생 시 이를 활용하는 애플리케이션에도 영향 • 대규모 학습·추론 시 사용량 기반 비용의 급격한 증가 가능

SaaS, MaaS, DaaS, RaaS

- 서비스형 모델(MaaS) (6/6)

- 적용 사례

- 헬스케어, 의료, 금융, 블록체인, 웹 3.0 등의 분야에서 활용



<그림 9.30> MaaS 내 산업 분야 적용 사례[12]

SaaS, MaaS, DaaS, RaaS

- 서비스형 데이터(DaaS) (1/3)

- 정의

- 사용자의 지리적 위치나 조직적 경계에 관계없이 데이터를 온디맨드 방식으로 제공하는 비즈니스 모델

- 특징

- 데이터가 클라우드에 저장되며, 고객은 구독을 통한 비용 지불 후 접근 가능
 - 데이터는 보안 포털을 통해 제공되며, 건별 결제 혹은 3가지 구독 유형에 따른 과금
 - 시간 기반 모델, 사용량 기반 모델, 호출당 과금 기반 모델
 - 제공업체는 원시 데이터뿐만 아니라 기계 판독 혹은 활용이 가능하도록 정규화된 데이터도 제공 가능
 - 제공업체는 데이터 수집 및 정규화 후 자사 데이터와 통합하여 고객에게 제공하는 양방향 데이터 흐름도 가능

SaaS, MaaS, DaaS, RaaS

• 서비스형 데이터(DaaS) (2/3)

• 장점

장점	설명
비용 효율성	• 고객의 데이터 인프라 및 팀 구축/유지 필요성 감소 • 데이터 저장 없이 필요 시, 데이터 스트림에 접근 가능하여 비용 절감
접근 용이성	• 데이터의 온디맨드를 통해 데이터 처리에 필요한 시간 및 전문성 최소화 • 사용자는 데이터 구조를 몰라도 인터넷 연결만 있으면 접근 가능
확장성	• 고객은 인프라에 추가 투자 없이도 증가하는 데이터 수요에 맞춰 확장 가능 • 필요한 데이터 워크로드의 유연한 선택 가능
중앙화된 데이터 관리	• 데이터가 일관되고 중앙화된 방식으로 저장되므로 불일치와 중복이 줄고 데이터 거버넌스와 규제 준수가 쉬워짐
핵심 활동에 집중	• 기업이 데이터를 관리하는 대신 데이터에서 가치 추출하는 데 집중 가능 • 동일 데이터에 동일한 형식으로 접근할 수 있어 팀 간 협업 개선
다른 서비스와의 통합	• 데이터를 분석 플랫폼, 시각화 도구, 기타 클라우드 서비스와 통합 가능 • 데이터셋을 정기적으로 업데이트하여 사용자가 최신 데이터 활용 가능
데이터 품질 향상	• 데이터가 중앙화되면 품질이 높아지는 경향이 있음 • 한 번 데이터 검증 시, 업데이트가 없는 한 추가 검증 불필요

SaaS, MaaS, DaaS, RaaS

• 서비스형 데이터(DaaS) (3/3)

• 단점

단점	설명
데이터 보안과 개인정보보호	• 데이터가 클라우드에 있어, 민감 정보가 제3자에게 유출될 수 있음 • 제공업체는 엄격해지는 최신 규제를 반드시 준수해야 함 • 데이터 해킹 공격의 증가에 따른 인프라 보안 비용도 증가함 • 데이터를 익명화해도 경우에 따라 역추적으로 원본 복원할 가능성 존재
벤더 종속	• 제공업체 측의 서비스 중단이나 장애는 클라이언트와 데이터에 의존하는 모든 서비스에 직접적인 영향을 미침 • 고객은 주로 데이터에 접근하고 다운받지 않으므로 기업 운영의 필수 데이터와 연결이 끊길 수 있음
제한된 맞춤화	• 특정 고객이 원하는 형식이나 세분화된 수준을 제공하지 않을 수 있음 • 부적절한 데이터 형식은 기존 시스템이나 워크플로에 통합하기 어려움 • 기업은 시스템이나 데이터를 적응시키는 데 추가 비용이 요구될 수 있음
품질 보증 문제	• 데이터 품질이 낮으면 잘못된 의사결정이나 관련 서비스의 오류 초래 가능 • 제공업체는 데이터가 관련성 있고 품질 보장된 상태임을 확인해야 함
지연과 성능 문제	• 인터넷 연결 불안정 혹은 큰 데이터셋 활용 시 필연적인 지연 유발 가능 • 해당 지연은 데이터 스트림이 타서비스와 결합 시 전체 성능 저하 유발

SaaS, MaaS, DaaS, RaaS

- 서비스형 결과(RaaS) (1/3)

- 정의

- 서비스 제공업체가 도구, 플랫폼, 원시 데이터를 제공하는 대신 특정 결과물이나 성과를 제공하는 비즈니스 모델

- 특징

- 데이터 분석, 자동화, AI 분야에서 주목받고 있음
 - 제공업체가 AI를 활용하여 고객에게 개인화된 인사이트 제공
 - 고객은 서비스에 대한 구독을 통해 일정 간격으로 분석 결과 수신
 - 모델 대신 결과 제공에 중점을 둠
 - 기업의 초기 비용 부담 최소화의 목표를 가짐
 - 고객은 기술 도입 비용을 줄이면서 가치 유지를 선호함

SaaS, MaaS, DaaS, RaaS

• 서비스형 결과(RaaS) (2/3)

• 장점

장점	설명
성과 중심 접근	• 제공업체는 도구, 인프라, 전문 지식이 아닌 결과에 대한 비용만 지불 • 소프트웨어 사용이나 분석 수행에 대한 책임이 없으므로 기업 위험을 감소
비용 효율성	• 고객은 인프라와 전문 지식을 구축하는 데 비용 들이지 않아도 됨 • 제공업체가 프로세스를 자동화해 비용을 절감 가능 • 고객은 합의된 가격으로 구독형 요금제 선택 가능 • 성과 기반 요금제를 통해 비용이 실제 성과와 직접적으로 연동됨
핵심 역량 집중	• 제공업체는 시스템 구축, 프로세스 관리에 자원 소모가 적어 시간 절약 • 기업은 새로운 역량을 구현하면서도 실행은 제공업체에 맡길 수 있음 • 제공업체는 역량에 집중하면서 결과를 즉시 업무 파이프라인 통합 가능
확장성, 정확성, 유연성	• 확장 가능하고 유연한 구조로, 제공업체가 동일 기술을 고객에게 재사용 가능 • 제공업체 보상/평판이 서비스 성공과 관련 있어 고품질 결과 제공할 유인이 큼

SaaS, MaaS, DaaS, RaaS

• 서비스형 결과(RaaS) (3/3)

• 단점

단점	설명
통제력 상실	• 고객은 결과에 대한 통제권만 가지므로, 프로세스 추적이나 문제 진단 불가능 • 제공업체가 규정 준수나 품질, 윤리적 문제가 있어도 고객이 알아채기 어려움
벤더 종속	• 고객은 제공업체에 대한 강한 의존성을 가지므로 제공업체 변경의 어려움, 높은 전환 비용으로 이어질 수 있음 • 제공업체의 실패나 비효율성은 고객 운영에 직접적인 영향을 미침
데이터 보안과 개인정보 위험	• 민감 데이터를 제공업체와 공유해야 할 수 있으며, 프라이버시 문제 야기 • 규제로 인해 데이터 공유가 어려울 수 있으며, 위반 시 벌금 부여 가능
결과 측정의 복잡성	• 명확하고 측정 가능한 결과 정의가 어려워, 목표나 분석이 복잡할수록 고객과 제공업체 간 기대 불일치 발생 가능
비용 증가 가능성	• 초기 비용은 감소하나 장기적으로는 기업에 고비용인 선택이 될 수 있음 • 추가 분석이나 성능·목표 불일치 시 발생하는 비용이 누적될 수 있음
제한된 맞춤화	• 제공업체는 가능한 많은 고객에게 적합한 자동화된 솔루션을 설계하므로, 개별 고객의 요구사항을 충족하지 못할 수 있음
품질 보증 문제	• 제공업체는 비용 절감을 위해 자동화되고, 많은 고객에게 적합한 솔루션 제공 • 단기 성과에 치중하여, 장기적 가치가 훼손될 위험 존재

SaaS, MaaS, DaaS, RaaS

• 서비스형 모델 비교

구분		SaaS	MaaS	DaaS	RaaS
제공 업체	장점	- 반복적인 수익 모델 - 확장 가능한 인프라 - S/W 업데이트 용이 - 비용 효율적 개발 라이프사이클	- 모델 수익화 가능 - 컴퓨팅 자원의 확장 가능한 분배 - 헬스, 금융 등 다양한 산업 지원 - 인프라 부담 감소 - AI 애플리케이션 확장	- 데이터 수익화 기회 - 데이터 중앙 집중식 관리 - 예측 가능한 수익 - 기존 데이터셋 활용 능력 - 여러 산업별 서비스 유연성	- 안정적/예측 가능한 수익 흐름 - 결과 기반 가치 지향적 가격 모델 장려 - 제품 판매 대신 결과 제공에 집중 가능 - 고객 유지율 향상
	단점	- 고객 이탈 가능성 - 인프라 및 업데이트에 대한 지속적인 비용 - 지역별 규제 및 규정 준수 문제	- 모델 개발을 위한 높은 초기 비용 - 모델 공정성, 신뢰성, 규정 준수 보장 문제 - 다양한 사용 사례에서의 모델 성능 기대치 관리 필요 - 자원 집약적 모델 업데이트	- 데이터 사용의 개인정보 보호 문제 - 실시간 데이터 전송을 위한 인프라 필요 - 복잡한 데이터 규제에 대한 준수 필요	- 성공적인 결과 제공 여부에 따라 수익 변동 - 성과 보장을 위한 높은 초기 비용 - 결과 제공이 어렵거나 기대치 불일치의 경우 수익 감소
고객	장점	- 낮은 초기 비용 - 최신 S/W 접근 가능 - 어디서든 접근 가능 - 비즈니스 요구에 맞춘 유연한 구독 방식	- 모델 효율 처리 및 확장 가능한 컴퓨팅 파워 - 예측/자동화를 위한 모델 유연성 - 자체 AI 인프라를 구축하지 않아도 되어 비용 절감	- 빠르고 쉬운 데이터 접근 - 시스템 소유 비용 절감 - 대규모 데이터 저장/처리 인프라 불필요 - 유연한 확장성	- 성과 기반 결제로 리스크 감소 - 인프라 걱정 없이 결과 집중 - 예측 가능한 성능/가치 확보 - 대규모 초기 투자 불필요 - 전문가 지원으로 결과 쉽게 달성
	단점	- 인터넷 연결 의존성 - 데이터 보안 및 개인정보 위험 - 장기적으로는 S/W 소유보다 고비용 위험	- 제3자 모델에 대한 의존성 - 모델 편향 또는 오류 가능성 - 특정 요구에 맞춘 모델 커스터마이징의 제한 - 일부 AI 애플리케이션에서의 개인정보보호 우려	- 데이터 소유권 및 벤더 종속에 대한 우려 - 제3자 데이터에 대한 의존 가능성 - 민감 데이터 보안 위험	- 결과 성공 여부에 대한 벤더 의존성 - 결과 도출 과정의 투명성 부족 - 결과물 수정에 대한 유연성 부족 - 특수하거나 비표준적 요구에 부적합 가능성

Thanks!

김 지 혜 (jihye@pel.sejong.ac.kr)