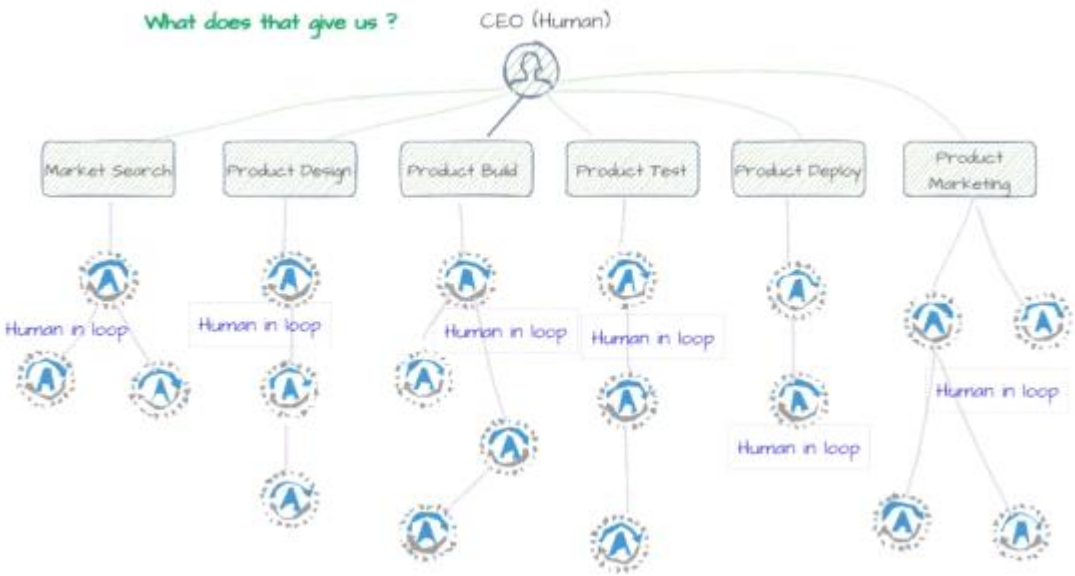
AI Agent (LLM-powered autonomous agent system)

1. Fine-tuning과 RAG의 여전한 문제들

- Hallucination and self-inconsistency
- LLM의 FT나 RAG는 텍스트 기반의 입력에 대한 텍스트 생성 반응에 국한된다 (uni-modal)
  : **실세계**의 다양한 문제 해결에는 부족
  (the gap between linguistic understanding and real-world execution)
- 이미 정해진(Fine-tuned, Augmented) 데이터에 대한 고정된 정보만 사용
  : 지식이나 **환경이 동적(realtime or dynamic)으로 변하는 경우**의 문제 해결이 어렵다
  : 기본적으로 모델내부+RAG 데이터에만 의존
- 복잡한 다단계 **절차(procedures)가 수반된 문제의 해결**이 어렵다
- 독립적으로 알아서(**자율적으로 판단**해서) 문제를 해결할 수 없다는 점

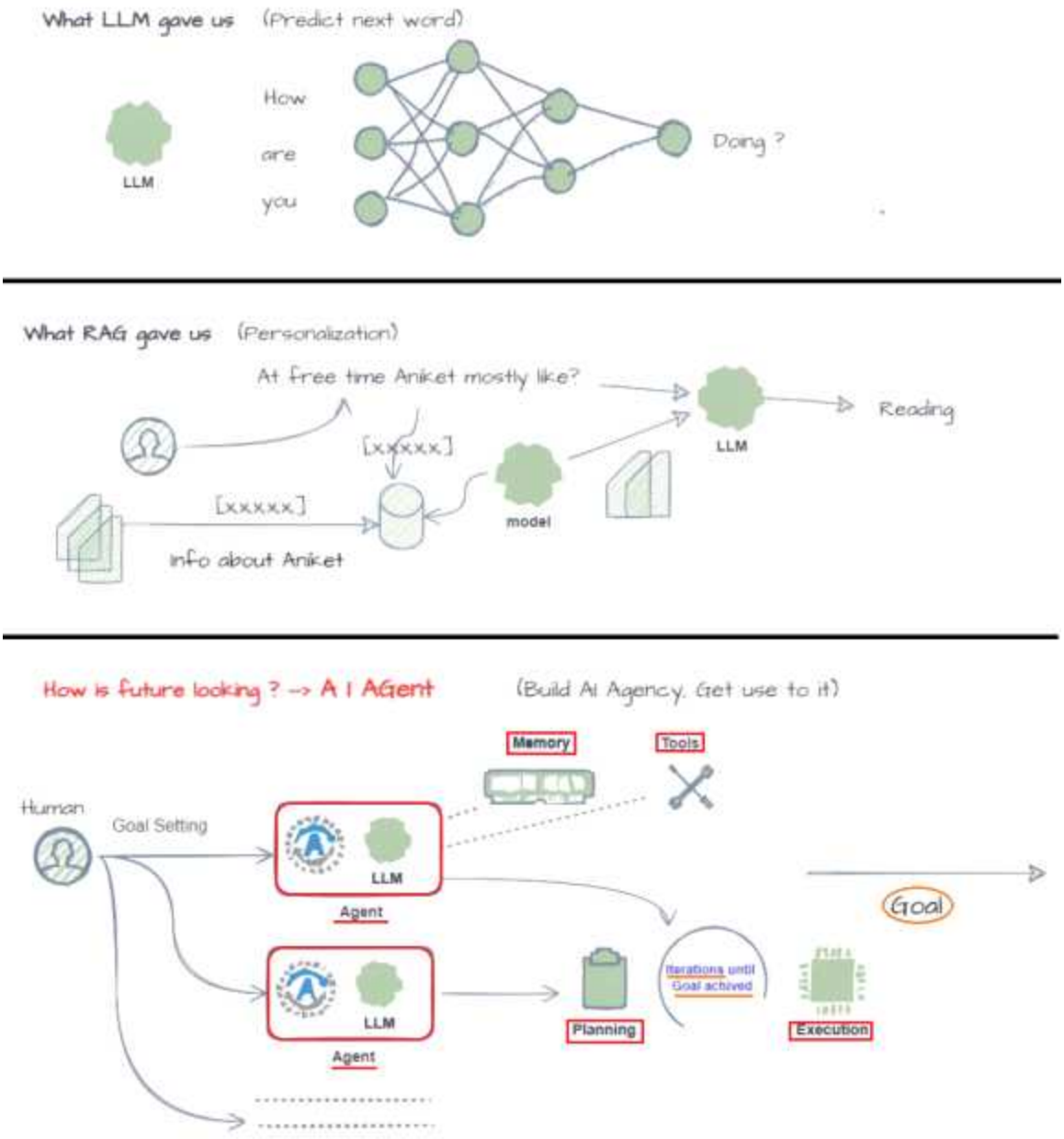
※ 추가로 필요한 기능
: **다른 (생성형) AI 모델과의 상호작용** 능력
: **환경을 인식/인지** 하는 능력
: 인터넷 데이터의 자유로운 활용, PC내 자료의 활용 등 **모델 외부의 동적인 데이터 활용**
: **자율적 의사결정**을 통한 문제해결 능력

※ 사람은 어떻게 일하는가?



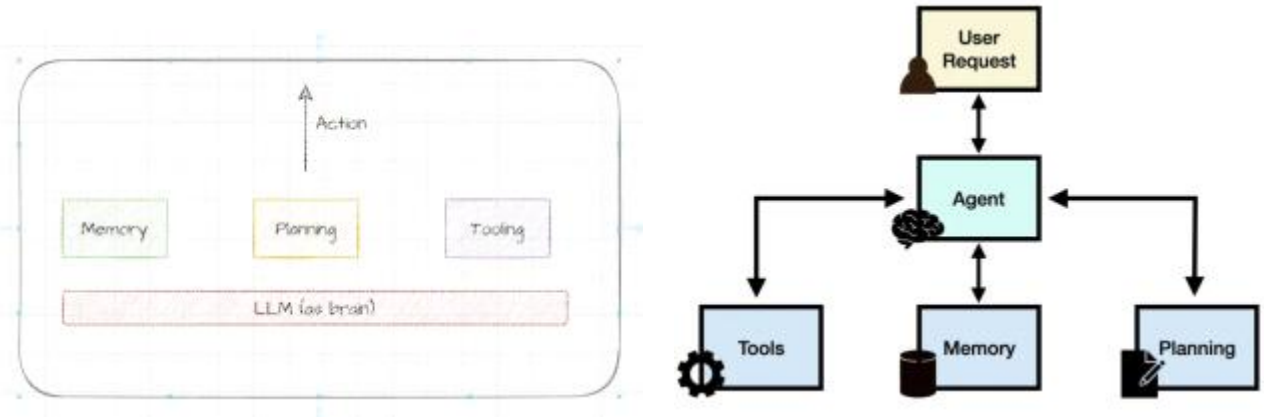
2. AI Agent : automated reasoning and decision engine

- Agent: **AI systems** programmed to **perform task**, **make decisions**,
  and **interact with their environment**
- Agent can learn from data, adapt to new information,
  and execute complex functions autonomously
- ex: chat-bots, sophisticated robots, 마케팅 전략가, ...
- prompt에 대응해 반응하는 전통적인 상호작용과 달리, agent는 목표에 기반하여
  새로운 정보와 환경에 적응하면서 독자적으로 문제를 해결.-Dynamic Problem Solver
- In agent systems, an LLM functions as the agent's brain,
  utilizing different components to act out in the digital world

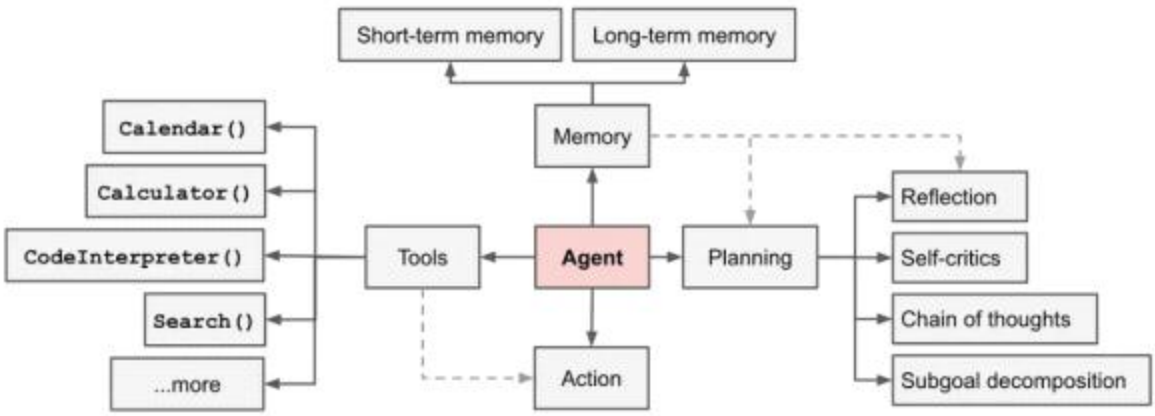


(https://medium.com/@learn-simplified/why-entire-ai-field-is-headed-towards-ai-agents-a268ac9661ed)

2.1 Agent Architecture



(https://medium.com/aiguys/ai-agents-are-all-you-need-0b38e8ee5481)



Overview of an LLM-powered autonomous agent system

Tool : current information access, mathematical engines, code execution capability, access to proprietary information sources, and many more

Memory :

- Short-term memory : prompt의 context window
- long-term memory : vector store(RAG), LLM weights

Planning :

- Subgoal & task decomposition : complex task -> smaller tasks
- Reflection and refinement : self-criticism, learn from mistakes

2.2 Reasoning (Planning)

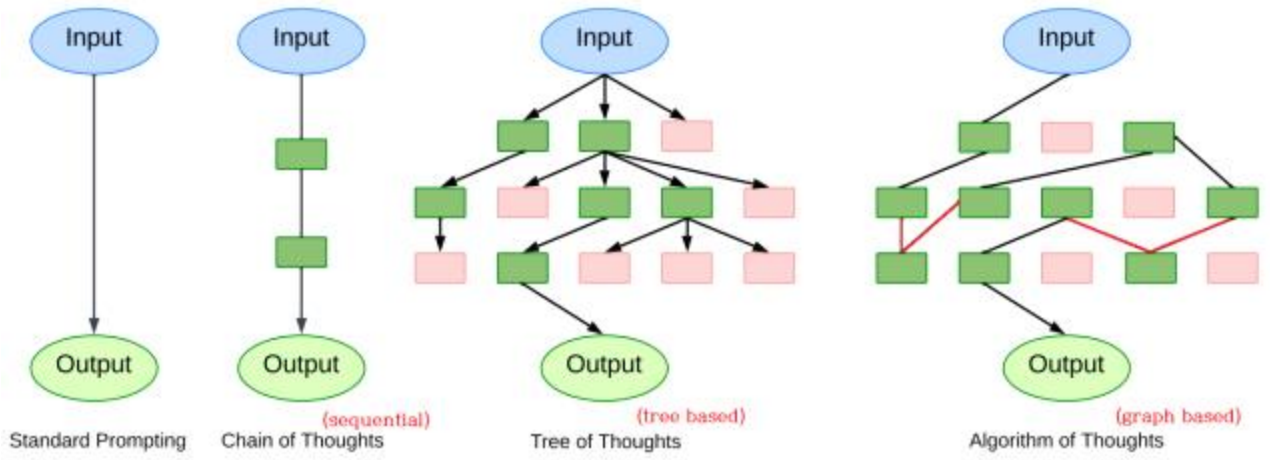


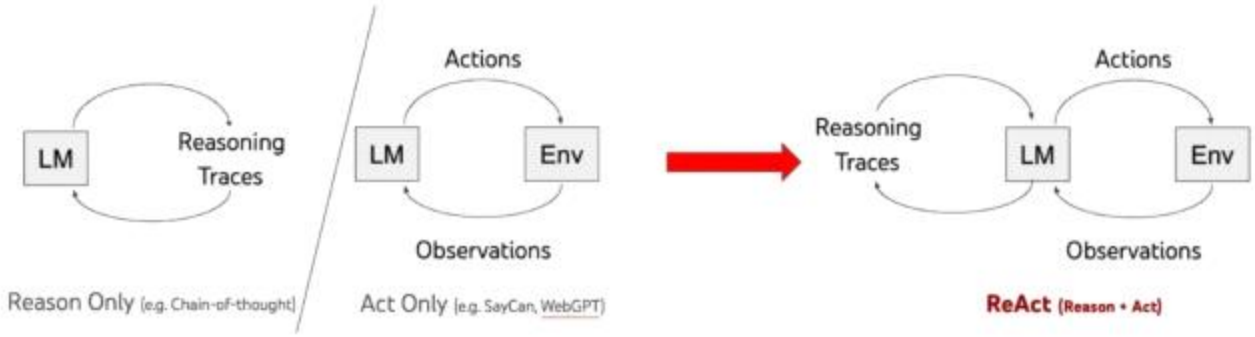
Figure 2: Illustration outlining various strategies for tackling reasoning problems with LLMs. Each box signifies a distinct thought, functioning as a unified string of words that forms an incremental pathway to reasoning. Green boxes indicate ideas deemed promising by the LLM, while red boxes represent less promising concepts.

Agent system에서 다양한 Prompting(**reasoning**) 기법들
(https://arxiv.org/pdf/2308.10379)

Algorithm of Thoughs: 모든 knowlwdge graph를 방문하기에 유리하고,
상대적으로 적은 prompt 사용

2.3 ReAct (Reason + Act)

: complex task를 해결하도록 LLM에게 reasoning과 action을 결합하여 prompting 하는 것



※ QnA를 위한 ReAct 예

(1d) ReAct (Reason + Act)

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.
Act 1: Search[Apple Remote]
Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple … originally designed to control the Front Row media center program …

Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.
Act 2: Search[Front Row]
Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports',' Front Row (software)', …]

Thought 3: Front Row is not found. I need to search Front Row (software) .
Act 3: Search[Front Row (software)]
Obs 3: Front Row is a discontinued media center software …

Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.
Act 4: Finish[keyboard function keys]

(https://arxiv.org/pdf/2210.03629)

(2b) ReAct (Reason + Act)

Act 1: Think[First I need to find a pepper shaker...more likely to appear in cabinets (1-6), countertops (1-3), ...]
Act 2: Go to cabinet 1
Obs 2: On the cabinet 1, you see a vase 2.

(Here the agent go to cabinet 1, then cabinet 2, 3, then countertop 1 and 2)

Act 6: Go to countertop 3
Obs 6: On the countertop 3, you see a apple 1, a bread 1, ... a pepper shaker 1, and a vase 3.

Act 7: Take pepper shaker 1 from countertop 3
Obs 7: You pick up the pepper shaker 1 from the countertop 3.

Act 8: Think[Now I find a pepper shaker 1. Next, I need to put it in/on drawer 1.]
Act 9: Go to drawer 1
Obs 9: Drawer 1 is closed.

Act 10: Open drawer 1
Obs 10: You open Drawer 1 …

Act 11: Put pepper shaker 1 in/on drawer 1
Obs 11: You put pepper shaker 1 in/on the drawer 1.

※ Agent의 예 : HuggingGPT

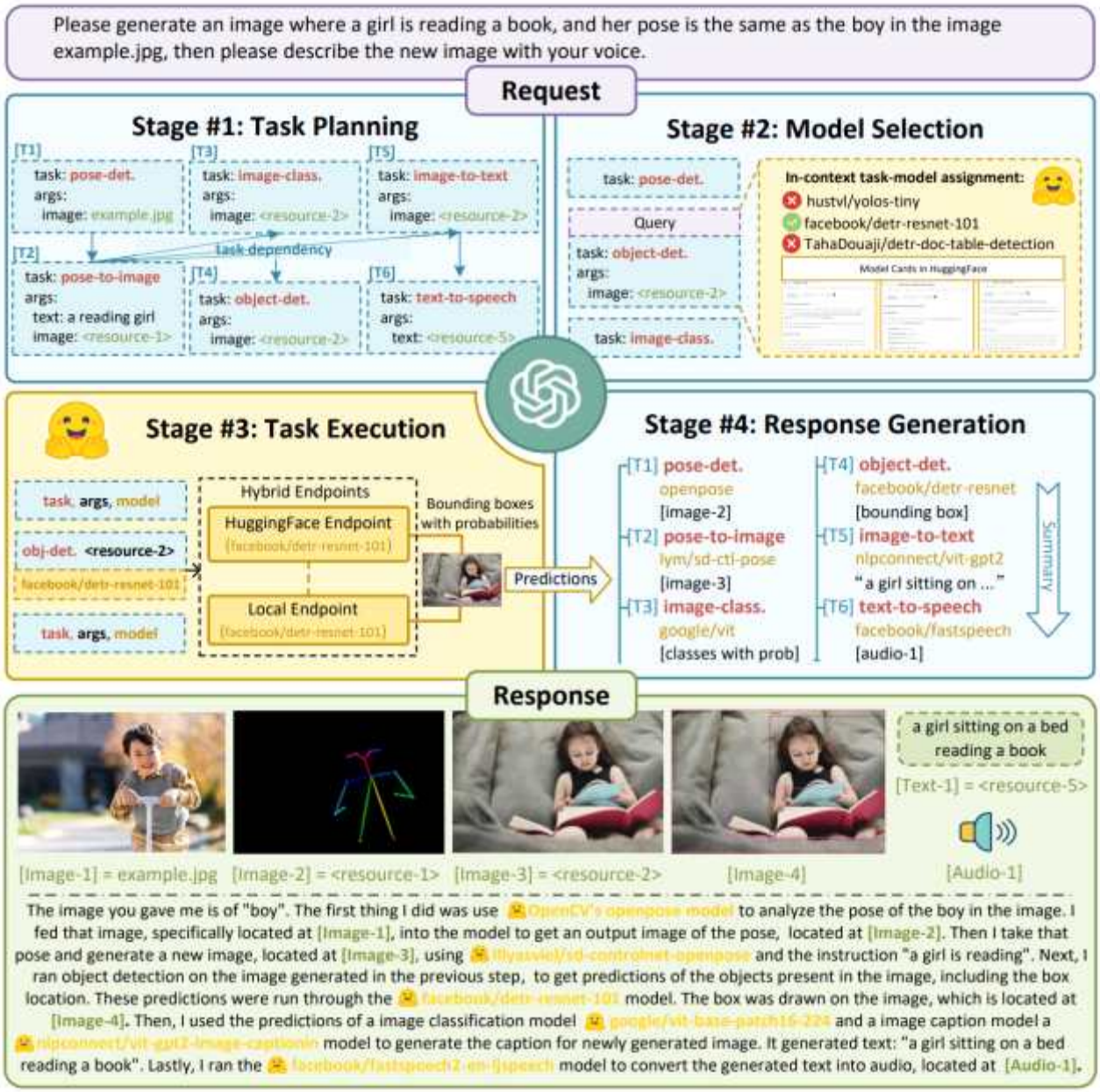


Figure 2: Overview of HuggingGPT. With an LLM (e.g., ChatGPT) as the core controller and the expert models as the executors, the workflow of HuggingGPT consists of four stages: 1) **Task planning**: LLM parses the user request into a task list and determines the execution order and resource dependencies among tasks; 2) **Model selection**: LLM assigns appropriate models to tasks based on the description of expert models on Hugging Face; 3) **Task execution**: Expert models on hybrid endpoints execute the assigned tasks; 4) **Response generation**: LLM integrates the inference results of experts and generates a summary of workflow logs to respond to the user.

HuggingGPT

(https://arxiv.org/pdf/2303.17580)

3. AI Agent의 특징

- 증가한 효율성 (**Efficiency**) : 일 처리 속도와 정확성이 인간보다 우수
- 개인 맞춤형 (**Personalization**) : 개인별 상황과 데이터에 맞춘 동작이 가능
- 적응성 (**Scalability**) : 요구되는 일의 양 증가에도 안정적으로 동작
- 항상성 (**Always On**) : 중단 없는 동작 24/7/365
- 비용 절감(**Reduced Costs**) : 인건비와 처리 비용의 동시 감소

==> 신뢰할 수 있고, 쉬지 않고, 나를 잘 아는, 많은 일도 잘하는 **무급 집사**
(https://medium.com/humansdotai/an-introduction-to-ai-agents-e8c4afd2ee8f)

==> 따라서, 보다 실제적인 AI 시스템은 LLM(LMM) + RAG + Agent 가 된다
현재로선, agent가 **AGI 구현에 있어 가장 적합한 방법론이자 도구**

4. AI Agent의 이용 사례

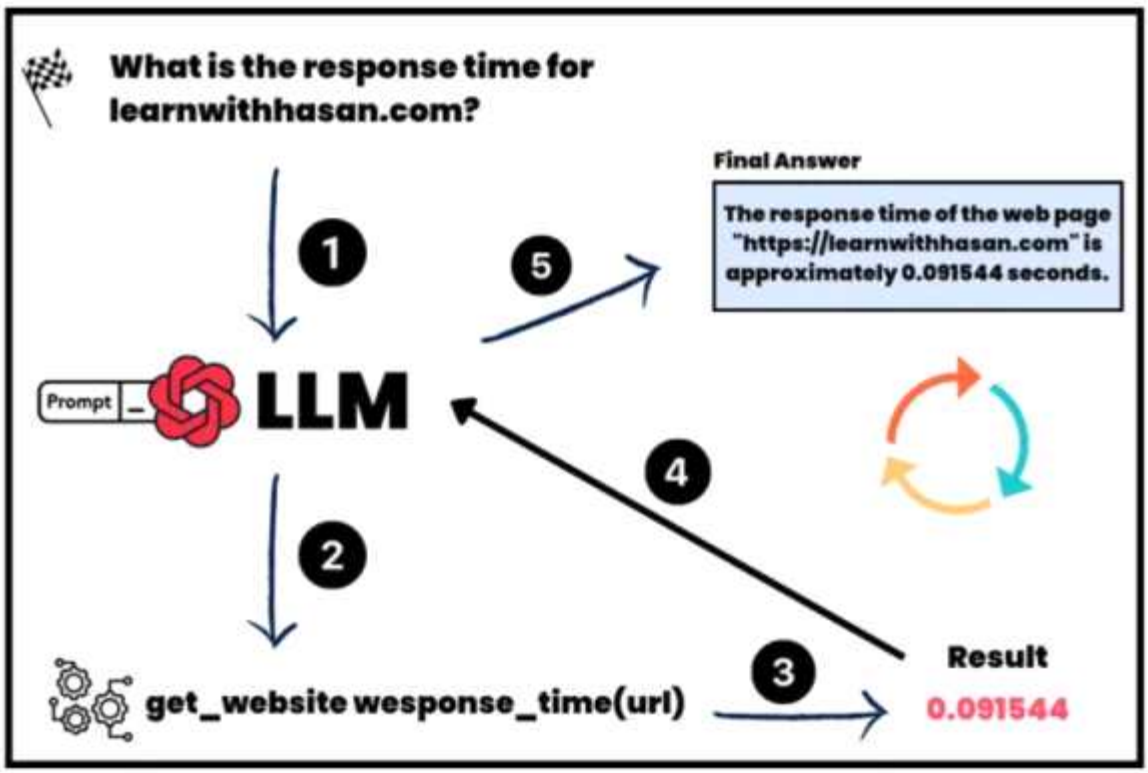
- 금융 : Trading, 리스크관리, 사기 탐지, 시장 데이터 분석
- 에너지 : 전력 생산과 분배/공급의 자동화
- 수송 : 자율주행, 교통흐름 제어, 물류 제어
- 건강 : 진료기록에 기반한 진단과 처방, 개인별 처방, 병원 자원의 효율적 할당
- 고객 서비스 : 챗봇기반 중단없는 고객 서비스
- 게임 : 현실감 있는 도전적 상황 생성으로 게임 몰입감 증대
- 스마트 빌딩 : 에너비 소비 최적화, 난방, 조명 시스템의 최적화/자동화
- 로보틱스 : 로봇 제어 및 작업 자동화/효율화
- 자연언어 처리 : 언어번역, 챗봇을 통화 사용자 상호작용 개선
- 사이버 보안 : 침입탐지, 악성S/W분석, 네트웍보안강화를 통한 보안성 향상
- 환경 모니터링 : 자연 보호, 기후 위기 모니터링 등의 생존 유지를 지원
- 소셜 미디어 : 트렌드와 패턴 데이터 기반 SNS 데이터 분석, 개인별 추천을 통한 UX 풍부화

5. AI Agent의 문제점

- 비싼 Planning 비용 : LLM으로의 반복적인 Prompting에 의한 **token의 과다 사용** 및 **시간 지연**
- LLM의 **non-deterministic**한 성질: sql문장과 같이 명확한(predictable) 출력 원할 때 어려울 수 있음

6. Agent의 구현 Soure의 예

(https://hasanaboulhasan.medium.com/
how-to-create-autonomous-ai-agents-from-scratch-56950bb31f7e)



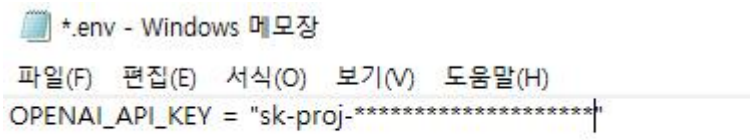
7. Agent 실습

7.1 Source Code (Agent with a Single Function)

```
# pip install python-dot-env openai
fromopenai importOpenAI
import os
fromdotenv importload_dotenv
import re,json,webbrowser
from datetime import datetime
# Load environment variables
load_dotenv()
# Create an instance of the OpenAI class
openai_client =OpenAI(api_key=os.getenv("OPENAI_API_KEY"))
#================================================================
def generate_text_with_conversation(messages,model ="gpt-4o"):
    response =openai_client.chat.completions.create(
        model=model,
        messages=messages
        )
    return response.choices[0].message.content
def get_response_time(url):
    time1 =datetime.now()
    webbrowser.open(url)
    time2 =datetime.now()

    return time2-time1

def extend_search_new(text,span):
    start,end =span
    nest_count =1  # Starts with 1 since we know '{' is at the start position
    for i in range(end,len(text)):
        if text[i]=='{':
            nest_count +=1
        elif text[i]=='}':
            nest_count -=1
            if nest_count ==0:
                return text[start:i+1]
    return text[start:end]

def extract_json(text_response):
    pattern =r'\{.*?\}'
    matches =re.finditer(pattern,text_response,re.DOTALL)
    json_objects =[]
    for match in matches:
        json_str =extend_search_new(text_response,match.span())
        try:
            json_obj =json.loads(json_str)
            json_objects.append(json_obj)
        except json.JSONDecodeError:
            continue
    return json_objects if json_objects else None
#================================================================
system_prompt ="""
You run in a loop of Thought, Action, PAUSE, Action_Response.
At the end of the loop you output an Answer.
Use Thought to understand the question you have been asked.
Use Action to run one of the actions available to you - then return PAUSE.
Action_Response will be the result of running those actions.
Your available actions are:
```

```
get_response_time:
e.g. get_response_time: naver.com
Returns the response time of a website
Example session:
Question: what is the response time for naver.com?

Thought: I should check the response time for the web page first.

Action:
{
"function_name": "get_response_time",
"function_parms": {
"url": "naver.com"
}
}
PAUSE
You will be called again with this:
Action_Response: 00:00:00023
You then output:
Answer: The response time for naver.com is 00:00:0.005 seconds.

"""
user_prompt ="What is the response time for naver.com?"
messages =[
    {"role":"system","content":system_prompt},
    {"role":"user","content":user_prompt},
]
available_actions ={
    "get_response_time":get_response_time
}
turn_count =1
max_turns =5
while turn_count <max_turns:
    print (f"\nLoop: {turn_count}")
    print("----------------------")
    turn_count +=1
    response =generate_text_with_conversation(messages,model="gpt-4o")
    print("*** response : \n",response)
    json_function =extract_json(response)
    if json_function:
            function_name =json_function[0]['function_name']
            function_parms =json_function[0]['function_parms']
            if function_name not in available_actions:
                raise Exception(f"Unknown action: {function_name}: {function_parms}")
            print(f" *** running {function_name}{function_parms}")
            action_function =available_actions[function_name]
            #call the function
            result =action_function(**function_parms)
            function_result_message =f"\nAction_Response: {result}"

            messages.append({"role":"user","content":function_result_message})
            print(function_result_message)
    else:
         break
```

7.2 실행결과




```
(rag) D:\myprojects\nlp\agent>python agent.py

Loop: 1
----------------------
*** response :
 Thought: I should check the response time for the web page first.

Action:
{
    "function_name": "get_response_time",
    "function_parms": {
        "url": "naver.com"
    }
}
 *** running get_response_time {'url': 'naver.com'}

Action_Response: 0:00:00.368017

Loop: 2
----------------------
*** response :
 Answer: The response time for naver.com is 0:00:00.368 seconds.

(rag) D:\myprojects\nlp\agent>
```

※ [Practical Consideration]

Agent를 잘 만들기 위해서는 다음 요소가 잘 고려되어야 한다

1.

※ [Questions]

1.

※ [실습 문제]

2.