



RAG Innovator Lab

Data & AI Team Lead
SungYong Pyo



■ RAG Innovator Lab의 목적

RAG 기반 AI 솔루션을 정형화하고
반복적인 리소스 투입 및 중복자원의 비효율성을 해결하는 과정을 함께 고민하는 과정

1. RAG Pipeline 을 어떻게 Automation 할 것인가?
2. 정확성 높고 신뢰성있는 답변을 위해서 고려해야 할 사항은 무엇인가?

■ RAG Innovator Lab 에서 진행되는 내용

RAG Pipeline 의 Automation

- Azure Storage Queue 의 사용 방법과 활용
- Logic Apps 의 사용 방법과 활용
- Function Apps 의 사용 방법과 활용

정확성 높은 답변을 만들기 위한 노력

- Multi Agent (AutoGen, LangGraph, Sementic Kernel)
- 이미지 분석을 위한 Document Intelligence + Azure OpenAI
- AI Search 를 활용한 Vector Indexing 구성
- Regular Model vs. Reasoning Model
- 비용 및 효율성에 대한 고려사항

RAG Innovator Lab 의 Demo

Demo

Multi Agent (AutoGen) Demo 환경

- Azure ML Studio
- Azure Storage Queue
- Azure Logic Apps
- Function Apps
- Document Intelligence /w Azure OpenAI
- AI Search

+ Multi Agent Workshop (Hands On Lab)

Multi Agent Workshop (Hands On Lab)

Introduction to Agent Framework

Hands-on Lab Part 1: Build a Multi-Agent Application

Step 1: 시나리오 이해 및 설계

Step 2: 배포

Step 3: Creative Writer 실행

Hands-on Lab Part 2: Build a Multi-Agent Using AutoGen

Step 1: Multi-Agent와 AutoGen

Step 2: AutoGen 실습 환경 설정

Step 3: Notebook 실행

Hands-on Lab Part 3: Build an Advanced Multi-Agent Using Magentic-One

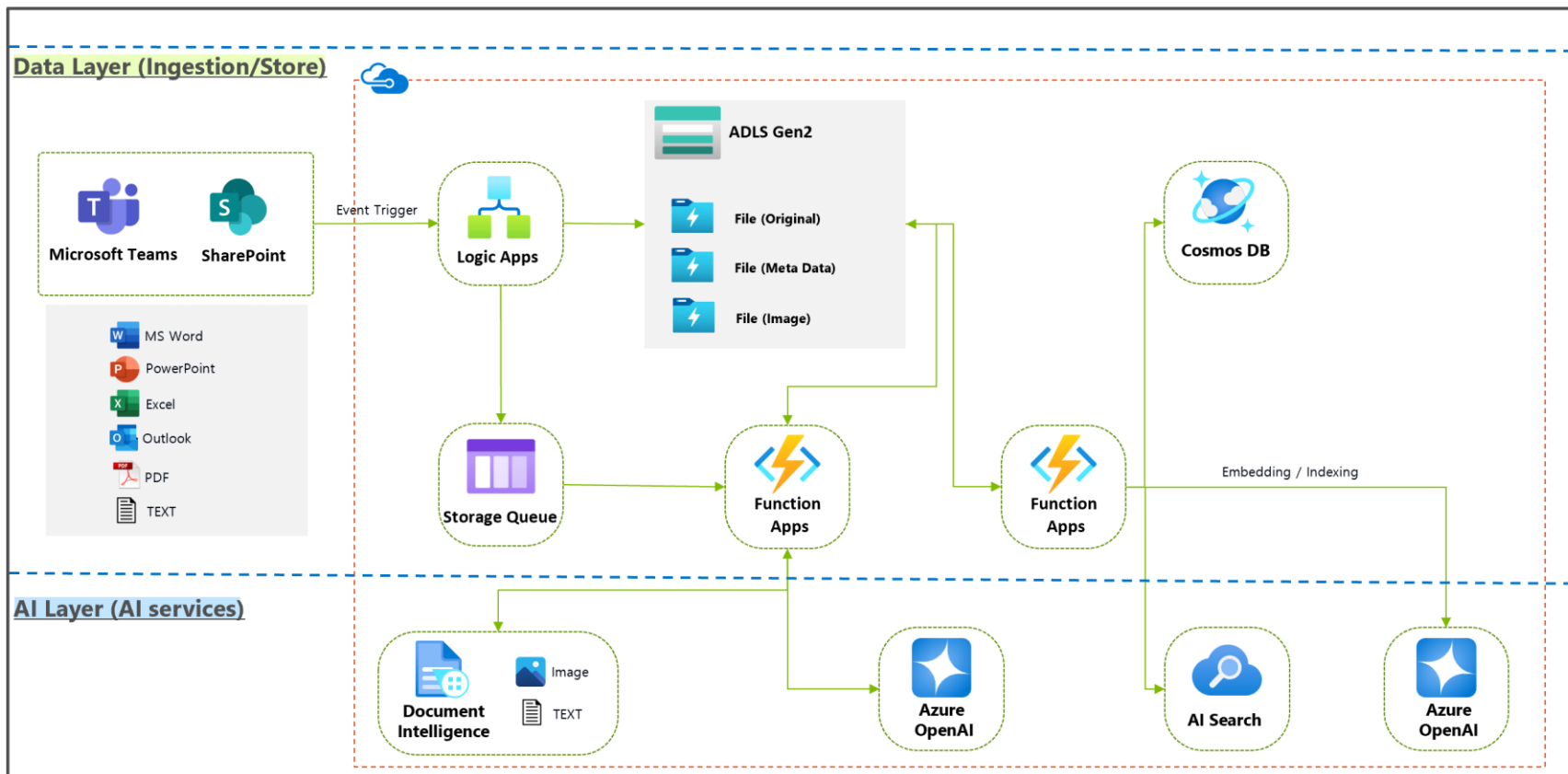
Lab 1: Coder와 ComputerTerminal을 이용한 Python 프로그램 실행

Lab 2: WebSurfer를 이용한 정보 검색

Lab 3: Coder와 ComputerTerminal을 이용한 게임 프로그램 개발

Lab 4: WebSurfer를 활용한 최신 정보 기반의 아이디어 구상

High Level Architecture (Scenario #1)



Session Overview

1

Overview

Agentic RAG &
RAG Automation

High Level
Architecture

2

Multi Agents

Non-Agentic vs.
Multi Agents

Autogen
Demo

3

RAG Automation

Azure Services
/w Demo

AI Services
/w Demo

4

Consideration

Reasoning Model

HoL Workshop
Introduction

CONTENTS

- I. Overview
- II. Multi Agents 소개
- III. RAG Automation
- IV. Consideration



I. Overview

1. RAG 가 필요한 이유
2. RAG Architecture
3. Non-Agentic RAG Pipeline 의 문제점
4. High Level Architecture

■ 할루시네이션 (Hallucination)?

정확하고 신뢰할 수 있는 정보를 제공함으로써 사용자의 AI 활용도를 극대화하려면?



할루시네이션이란?

- 인공지능 언어 모델이 사실이 아닌 정보를 생성하는 현상
- 실제 데이터에 기반하지 않은 허구의 내용 생성



주요원인

- 모델의 학습 데이터 한계
- 컨텍스트 부족으로 인한 잘못된 추론
- 복잡한 질문에 대한 부정확한 이해



예시

- 잘못된 역사적 사실 생성
- 존재하지 않는 인물이나 사건 언급
- 사실과 다른 과학적 정보 제공



해결 방법

- **RAG (Retrieval-Augmented Generation) 사용**
 - 외부 데이터베이스에서 관련 정보 검색
 - 정확한 정보 기반의 답변 생성
- 모델의 지속적 업데이트 및 검증



할루시네이션 방지의 필요성

- 인공지능 활용의 신뢰성 향상
- RAG 와 같은 기술을 통해 더 나은 성능 확보
- 정확하고 신뢰할 수 있는 정보를 제공함으로써 사용자의 신뢰를 확보

■ RAG (Retrieval Augmented Generation) 이 필요한 이유?

❖ 정확성 향상 [할루시네이션]

- 외부 데이터베이스 또는 사내 데이터베이스에서 관련 정보 검색을 통해 정확한 정보를 전달.
- 모델이 신뢰성 있는 답변을 제공할 수 있는 Reference 를 제공.

❖ 최신 정보 제공 [실시간 정보 업데이트]

- 항상 최신 정보 제공 (최신 데이터 검색 및 반영) 을 통해 정확하고 신속한 답변 제공
- Bing Search API 를 활용하여, 검색 결과 기능 추가

❖ 폭넓은 지식 활용

- 학습 데이터에 없는 정보도 검색을 하기 때문에, 도메인 특화된 고유한 정보를 활용
- 다양한 주제에 대한 답변 가능 (e.g. 고객지원, HR 인사 정보, 정보검색, 교육, 헬스케어 등)

❖ 문맥 이해 개선

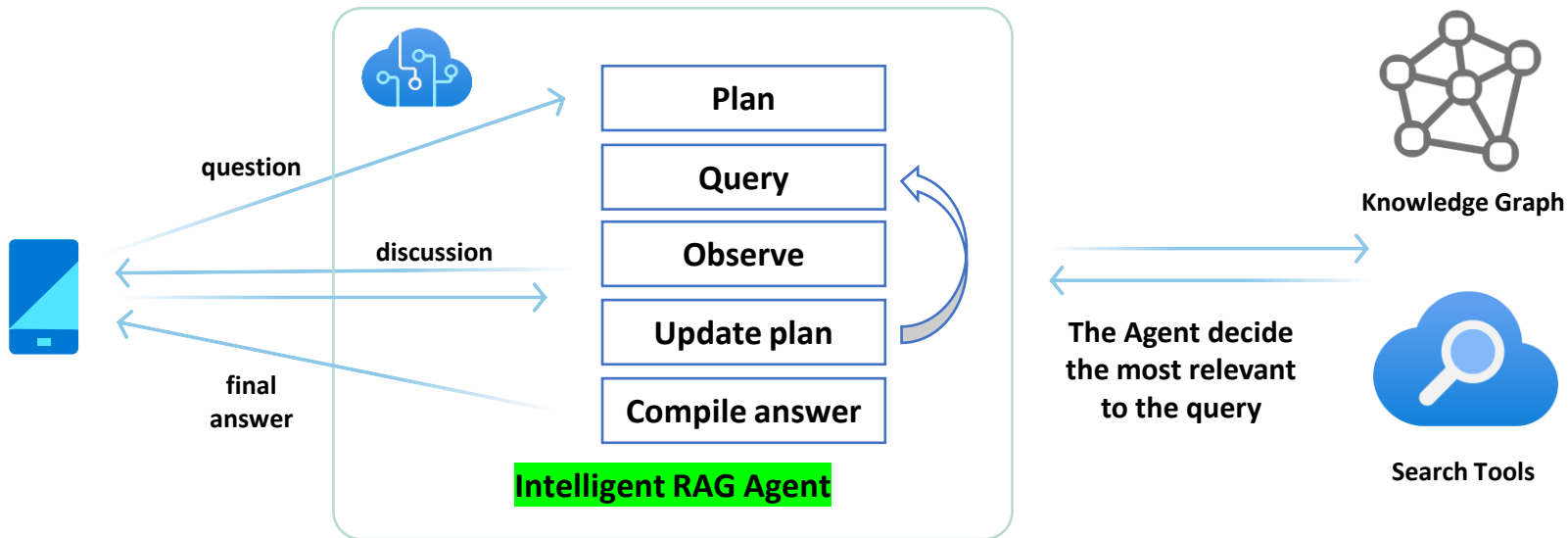
- 질문의 문맥을 이해하고 관련 정보 검색을 하여 사용자의 질문 의도를 정확히 파악
- 복잡한 질문에도 적절한 답변 제공할 수 있는 방안을 제시

❖ 효율적인 자원 사용

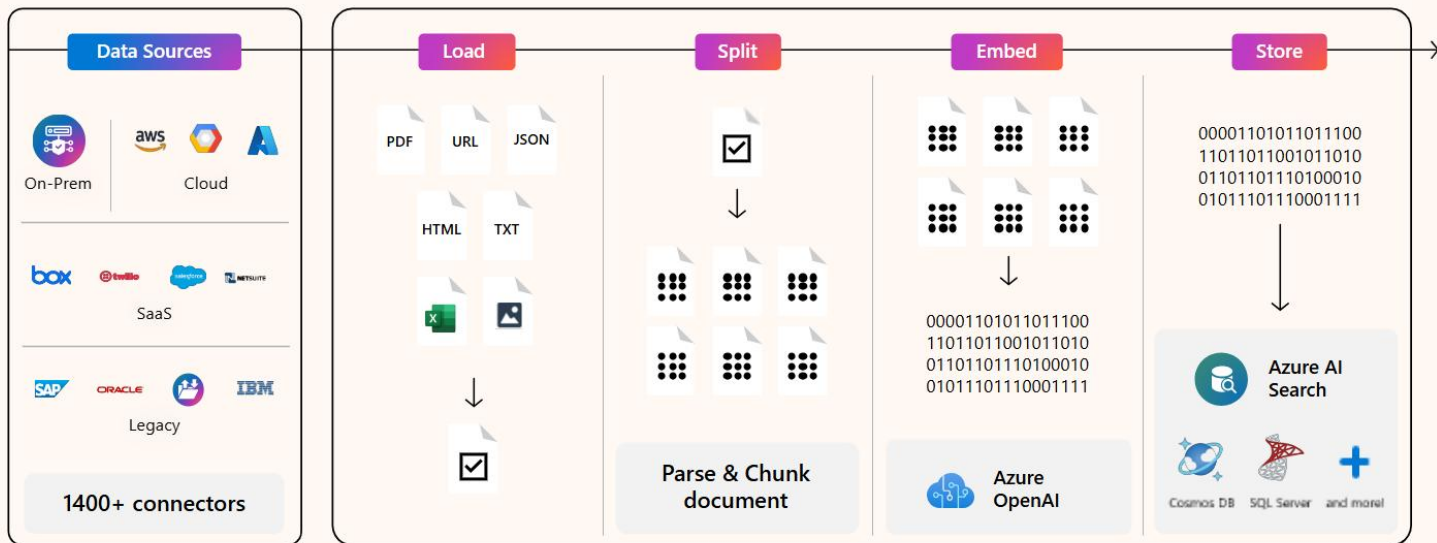
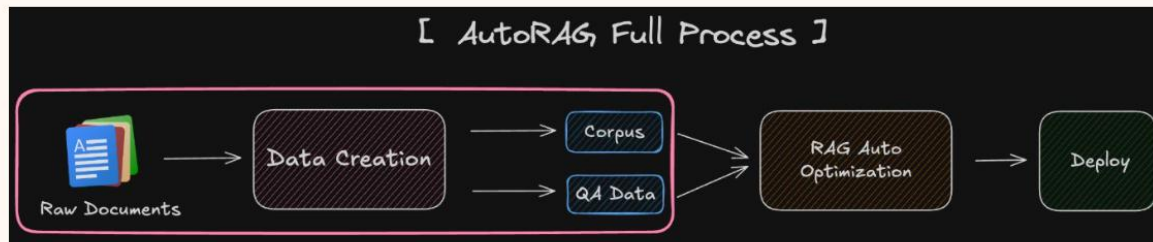
- 필요한 정보만 실시간 검색을 하고, 모델이 기본적으로 제공할 수 있는 정보를 분리하여 제공
- 모델의 크기와 복잡성 감소를 하게 되어, 비용절감 효과 기대

RAG (Retrieval Augmented Generation) Agent

사용자의 질문을 연구 과제로 변환하고, 사람의 개입을 통해 해당 도메인 범위 내에서 복잡한 질문에 대한 고품질 답변을 생성합니다.



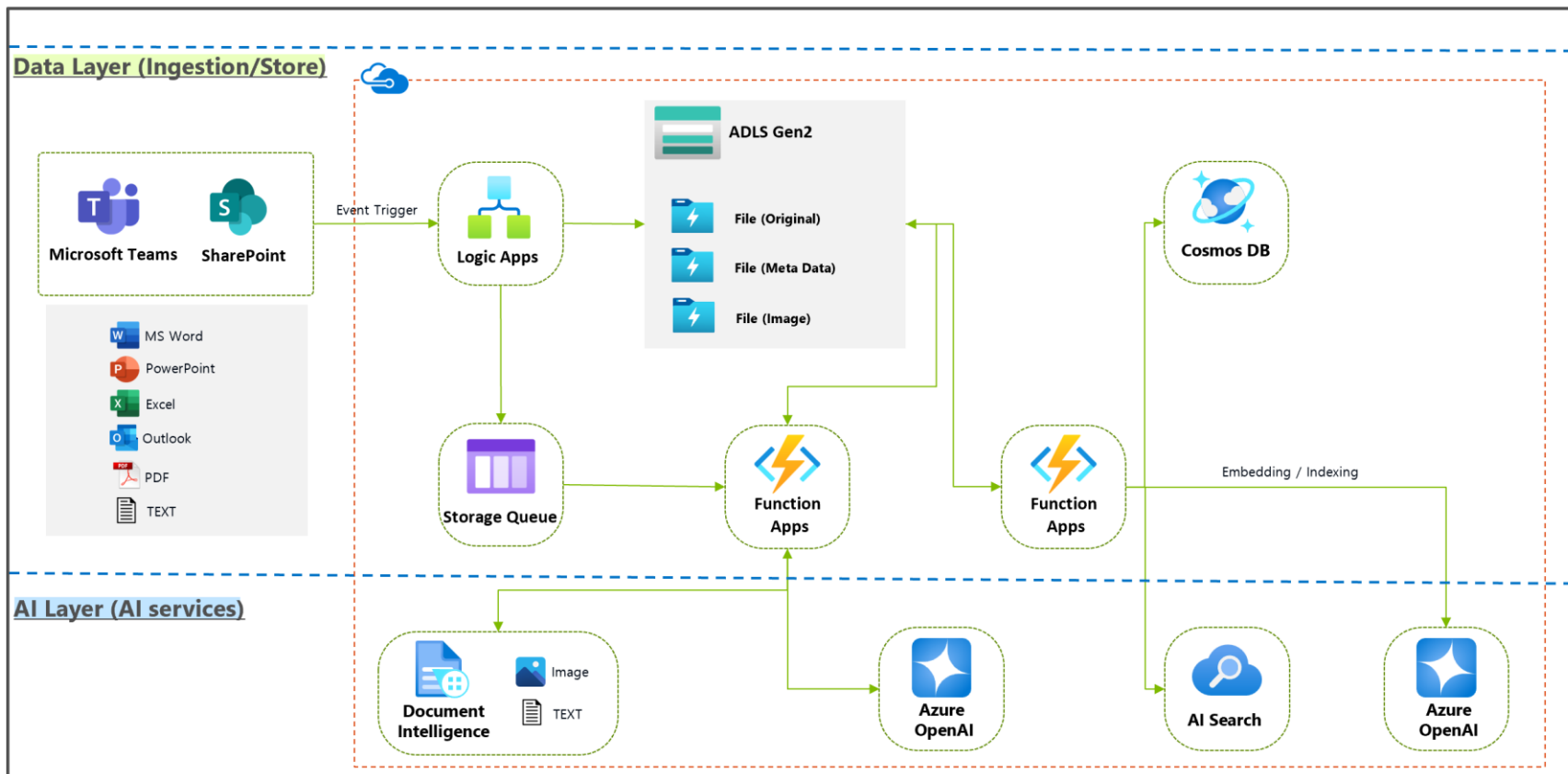
RAG Full Process



■ 기존 RAG 프로젝트의 문제점

RAG 기반 AI 솔루션을 정형화의 필요성
반복적인 리소스 투입 및 중복자원의 비효율성에 대하여...

High Level Architecture (Scenario #1)



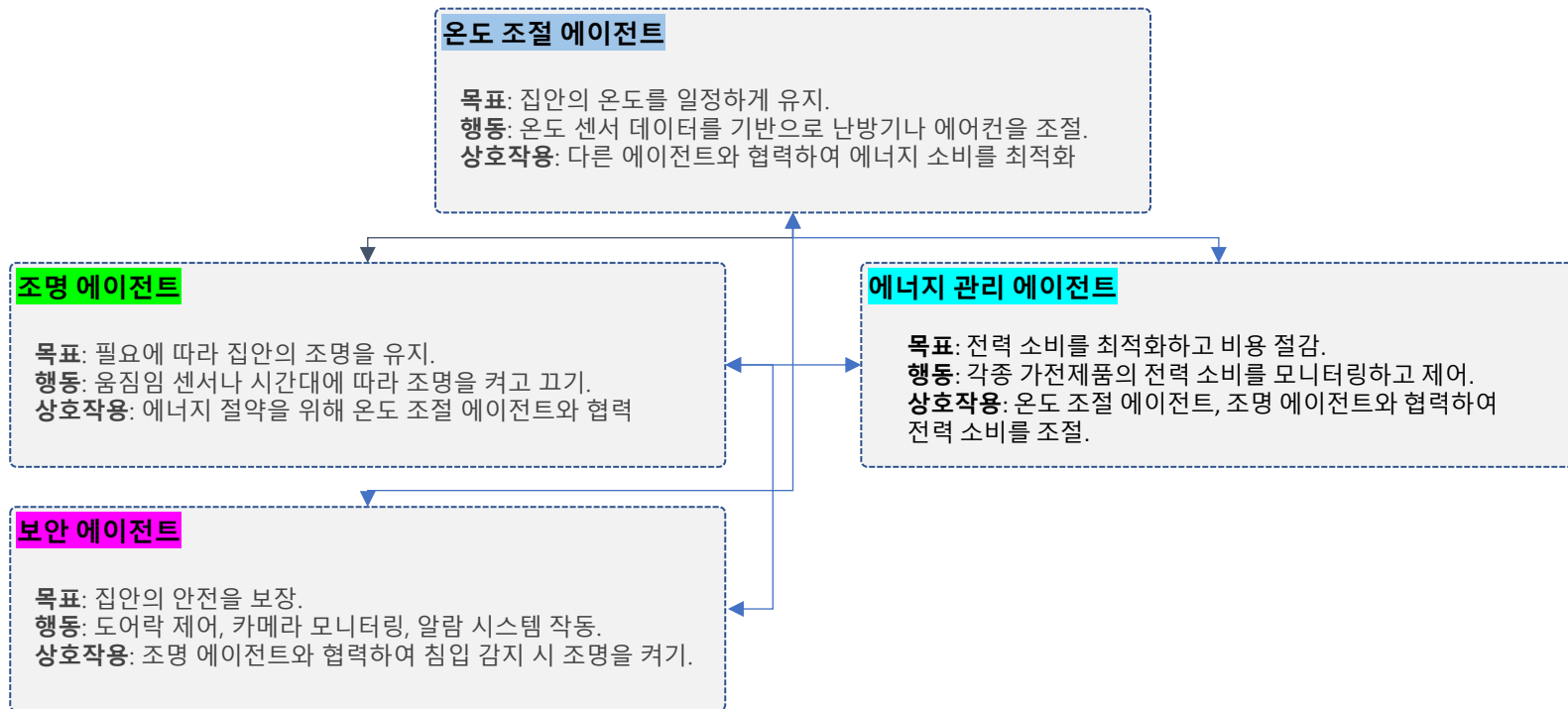


II. Multi Agents 소개

1. Agentic System 이란
2. Non-Agentic Workflow vs. Agentic Workflow
3. Agent Frameworks
4. AutoGen 에 대하여
5. Azure AI Services for Multi Agents
6. Demo

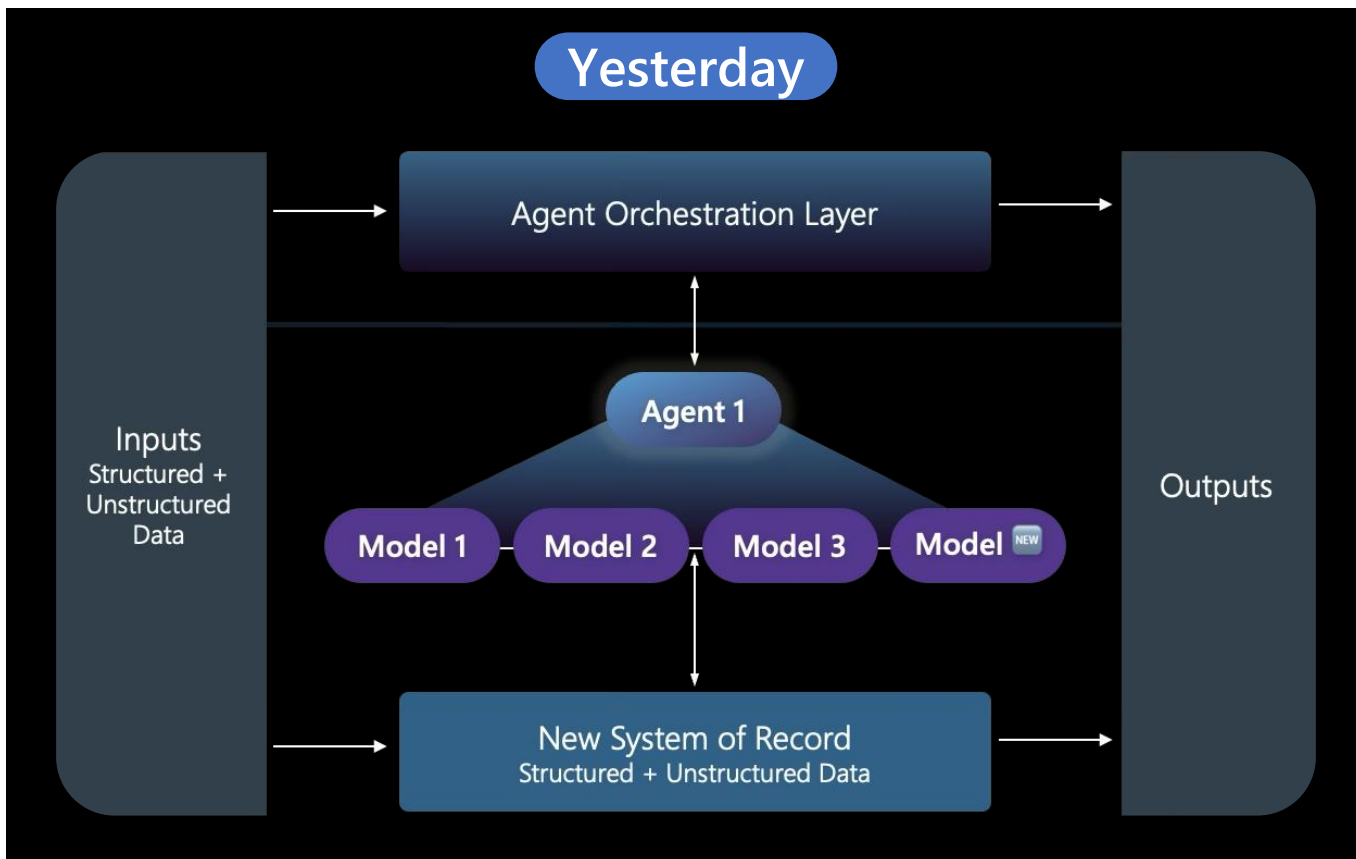
Multi-Agent Hide and Seek

Multi Agents 예시 (스마트 홈 시스템)

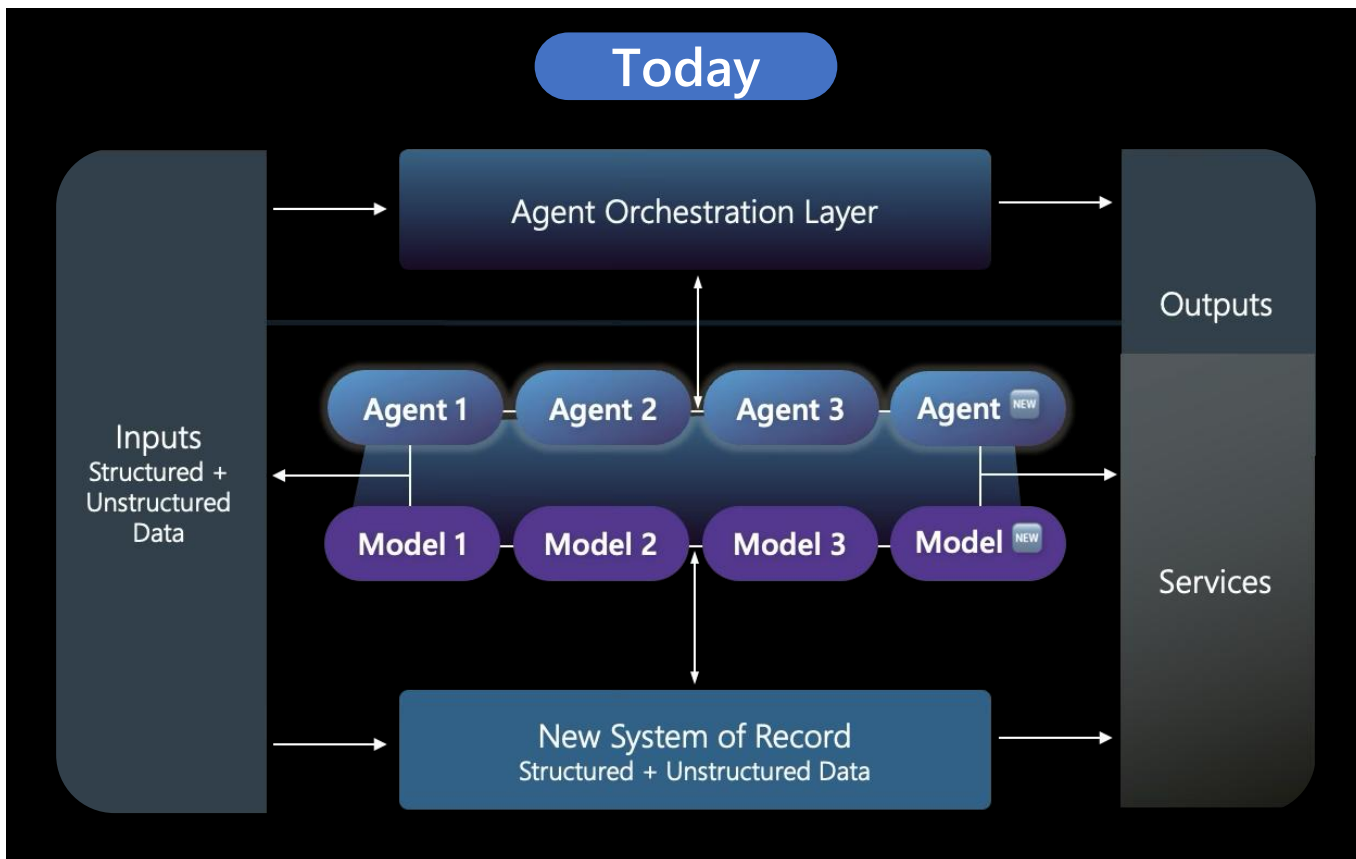


- 각 에이전트는 독립적으로 작동하면서도, 특정 상황에서 협력하여 시스템 전체의 효율성과 편의성을 높이는 역할을 합니다.
- Multi-Agent Systems는 복잡한 문제를 여러 개의 작은 문제로 나누어 해결할 수 있게 해주며, 각각의 에이전트가 분산된 방식으로 작동하면서 전체 시스템의 목표를 달성하도록 합니다.

Agent Evolution



Agent Evolution



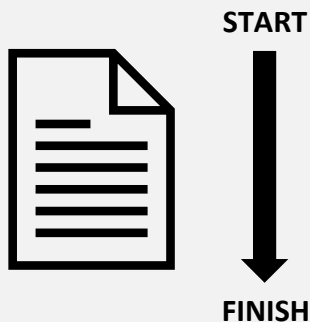
Non-Agentic Workflow vs. Agentic Workflow

Non-Agentic Workflow

- Zero-Shot prompting
- One-Shot prompting
- Few-Shot Prompting

一筆揮之 (일필휘지)

단숨에 흥취 있고 힘차게 글씨를 써 내려가다.

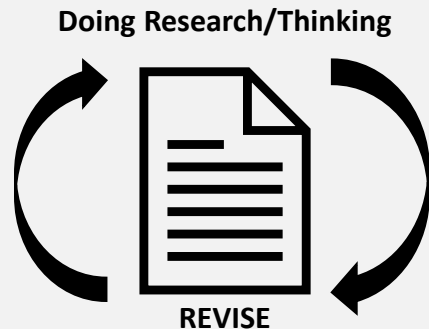


Agentic Workflow – iterative process

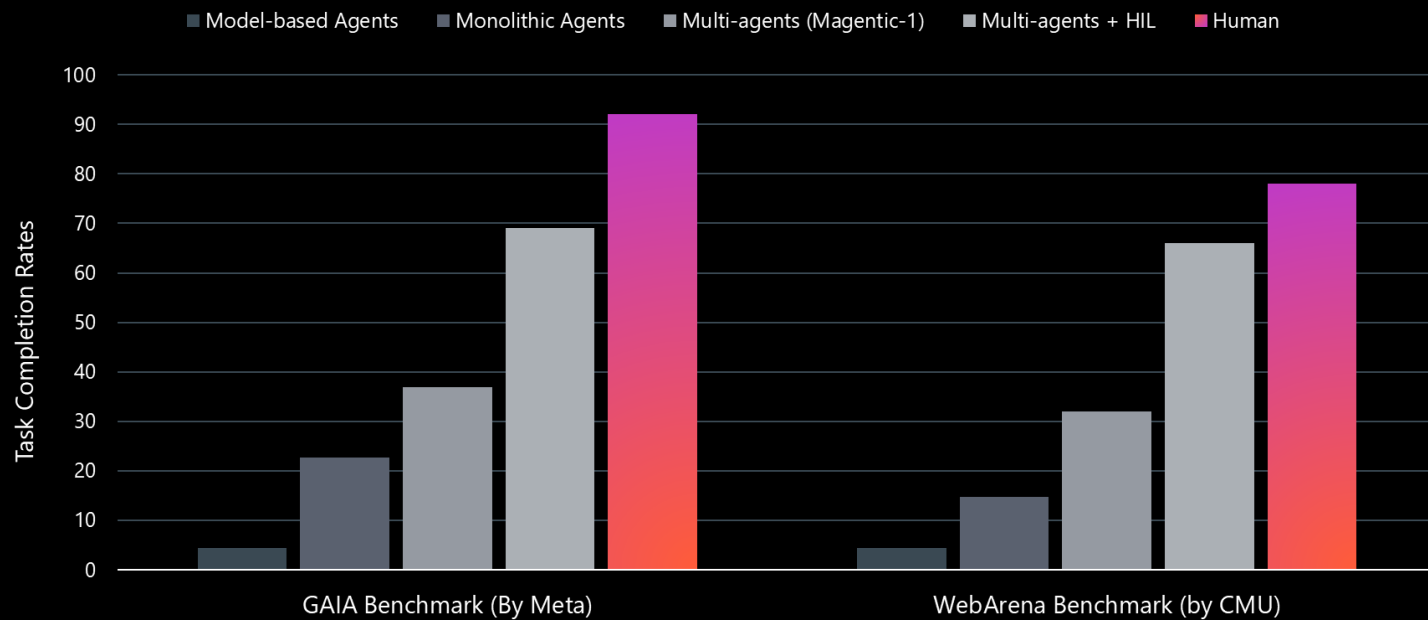
- Doing a web search
- Iterate and revise
- Multi-perspective

切磋琢磨 (절차탁마)

옥이나 돌 따위를 갈고 닦아서 빛을 낸다.

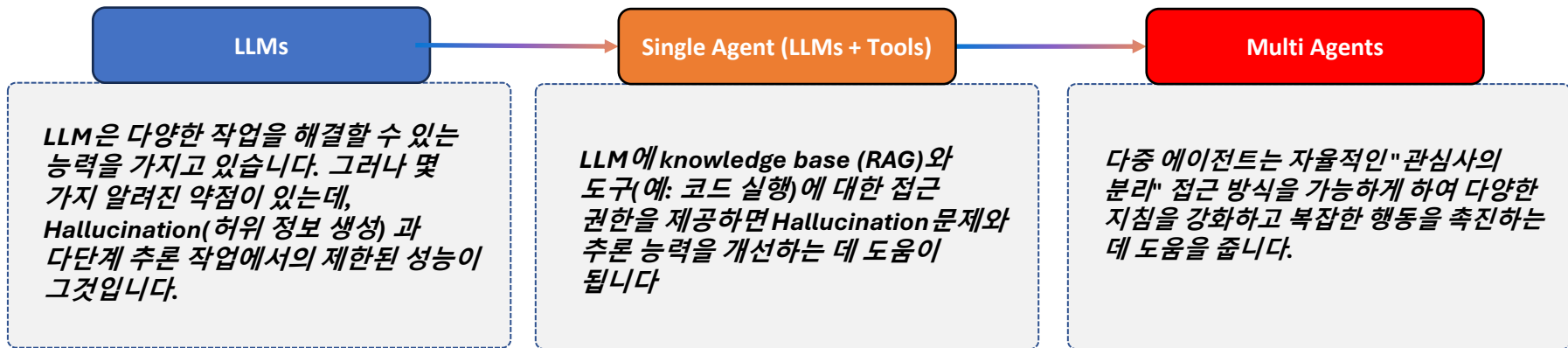


Agents are becoming more capable



From Single LLMs to Multi Agents

Task Complexity



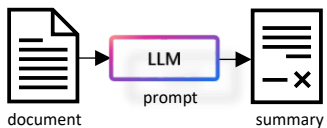
Example Tasks

“에펠타워의 높이는 어떻게 되지?”

“MSFT의 YTD 주식 가격을 그래프로 표시해줘”

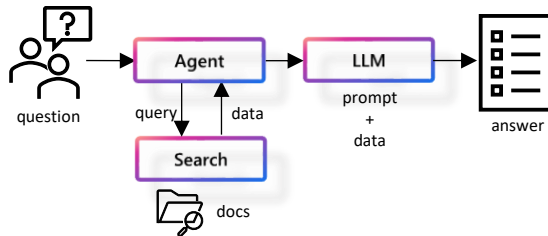
“사용자가 주식 가격을 확인하고, 주식을 구매하며, 세금을 신고할 수 있도록 돕는 모바일 애플리케이션을 개발하세요.”

Evolution of LLM based solutions



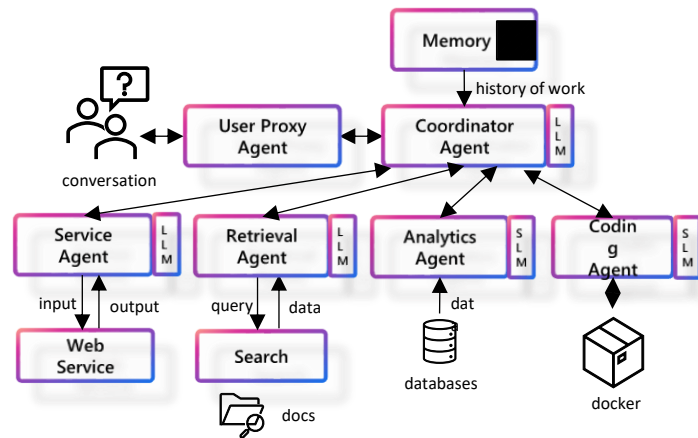
No Agent
매우 좁은 단일 작업

예: log 를 json 으로 변환해줘



Single Agent
매우 명확하게 정의된 반복 작업

예: 복잡한 질문에 대한 답변을 근거와 함께 제공하기



Multi-agent Systems
다양한 기술이 필요한 광범위하고 복잡한 사용 사례

예: 지난 분기 동안의 미국 판매에서 상위2개의 최신 트렌드를 활용하여 메일링 리스트 사용자 기반을 확대할 수 있는 2개의 Instagram 마케팅 캠페인을 제안하고, 각 캠페인의 영향을 예측하기.

VALUE

■ Difference between Single-Agent and Multi-Agents

Single-Agent System	Multi-Agent System
n = 1 agent	n > 1 agents
Less complex tasks	Complex tasks
Synchronous	Capable of asynchronous
Architectural focus on internal processes	Architecture focus on inter-agent interactions

- 증가된 전문성, 견고성 및 모듈성
- 다양한 페르소나가 소통하고, 협력하며, 맥락을 추가하고 **병렬**로 작업할 수 있음
- 내부 구조에 초점을 맞춘 설계 예: 에이전트 간 **Collaboration** 패턴 및 **Orchestration** 패턴

Agentic Frameworks

AutoGen

MemGPT

*GraphRAG

*Promptflow

Semantic Kernel

Taskweaver



AgentVerse

Assistants API

AutoGPT + P

BabyAGI

CrewAI

DyLAN

LangChain/LangGraph

LATS

MetaGPT

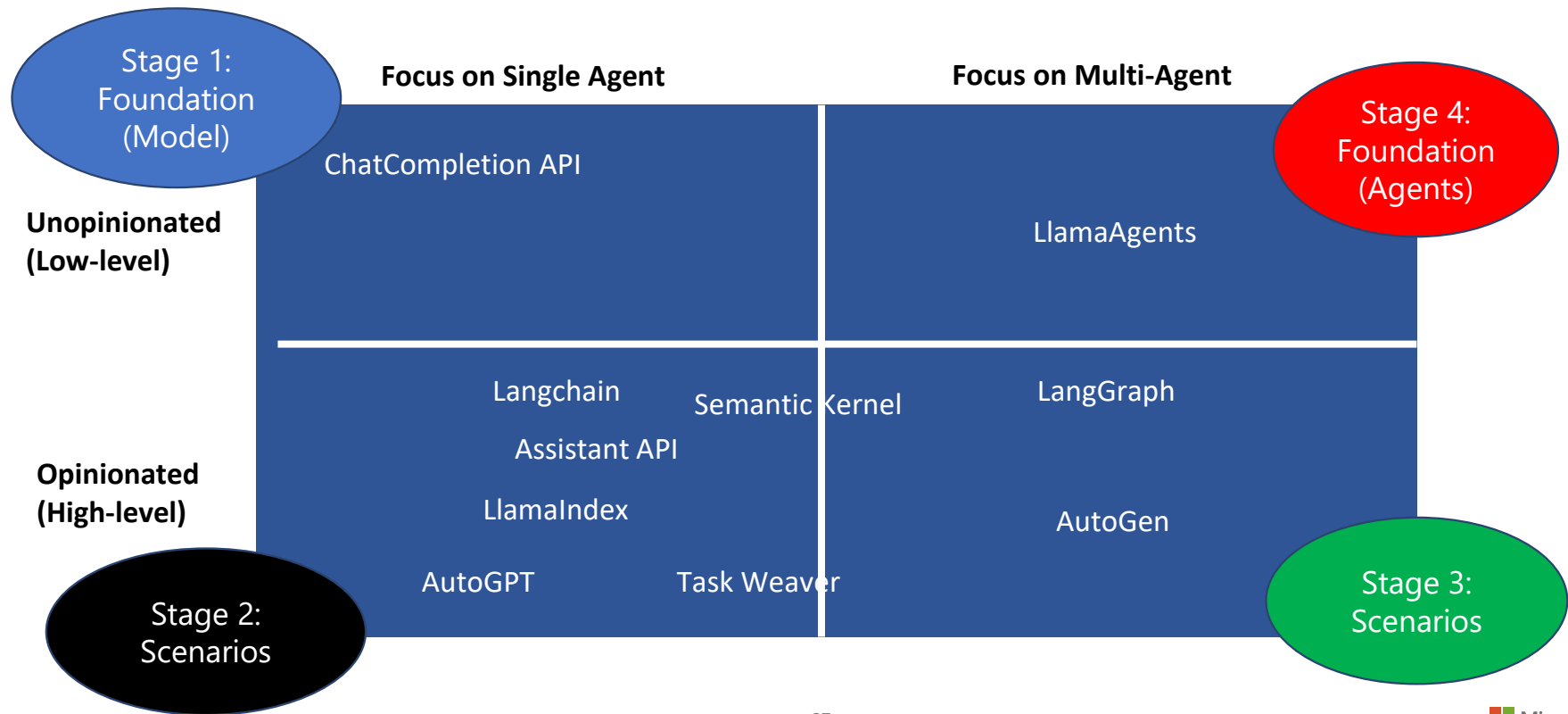
ReAct

RAISE

Reflexion

분산 시스템과 마찬가지로, 단일 아키텍처가 모든 에이전트 시나리오에 적합하지는 않습니다.

Agent Frameworks (AutoGen, LangGraph)



Microsoft Agent Frameworks

1 Single-agent



Managed agent
micro-services



2 Multi-agent

AutoGen과 Semantic Kernel을 사용하여
이들을 함께 오케스트레이션 하세요.



State-of-the-art
research SDK



Production-ready
and stable SDK

AutoGen 에 대하여.

Customizable and conversable agents :

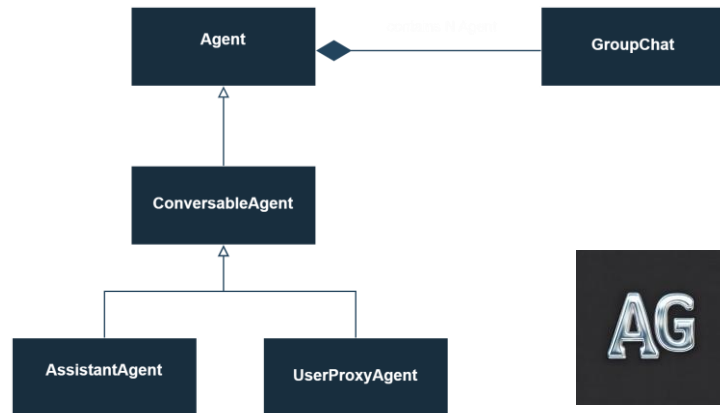
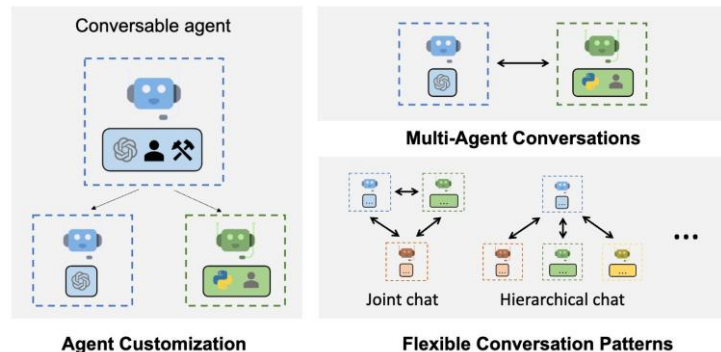
AutoGen은 LLM(대규모 언어 모델), 인간 입력, 도구 또는 이들의 조합을 활용할 수 있는 범용적인 에이전트 설계를 사용합니다.

Conversation programming:

특정 역량과 역할을 가진 대화 가능한 에이전트 세트를 정의합니다.

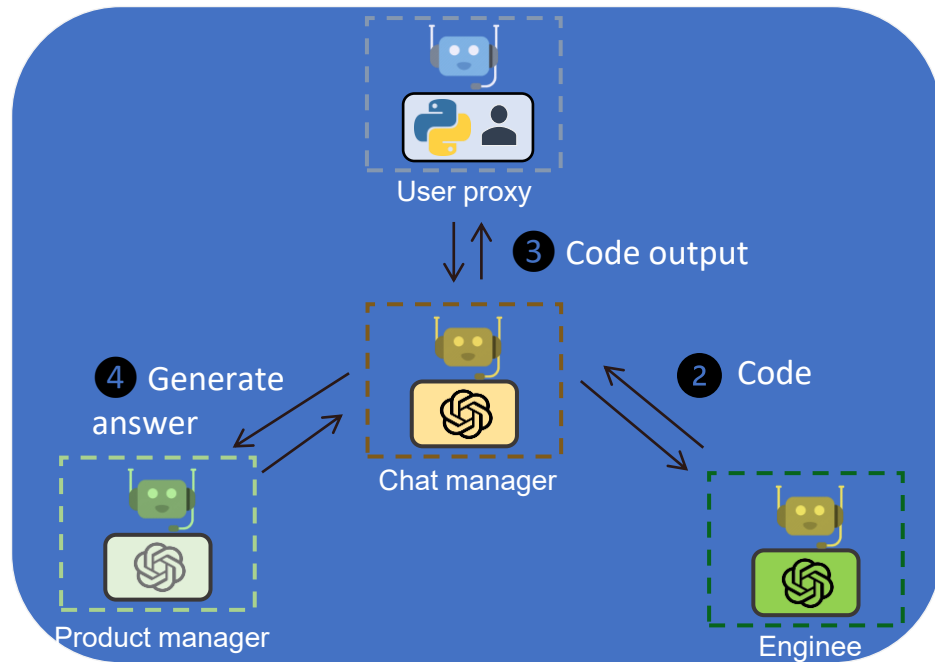
대화 중심의 계산 및 제어를 통해 에이전트 간 상호 작용 동작을 프로그래밍합니다.

[AutoGen | AutoGen
\(microsoft.github.io\)](https://microsoft.github.io/autogen/)



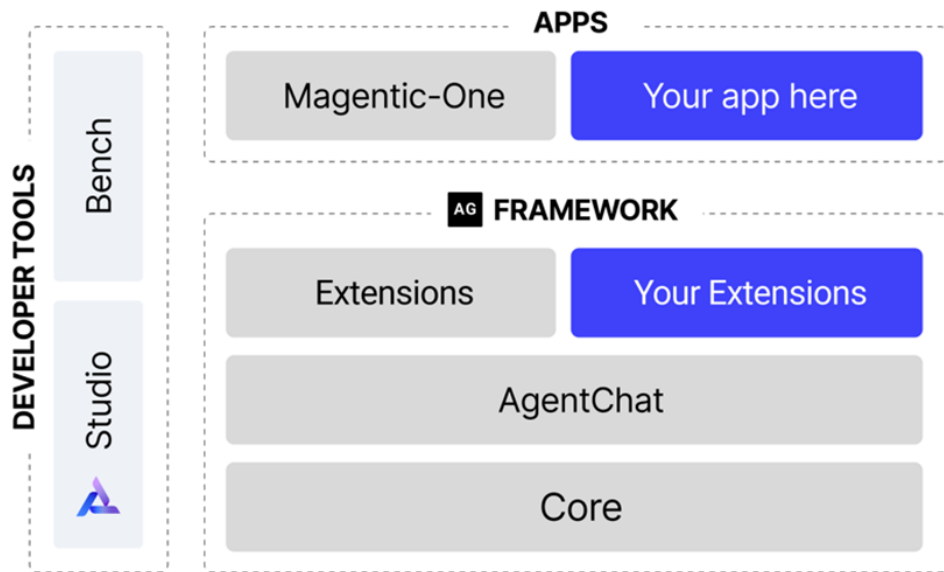
AutoGen

```
1 import autogen
2
3 # Construct Agents
4 user_proxy = autogen.UserProxyAgent(
5     name="User_proxy",
6     system_message="A human admin.",
7     code_execution_config={
8         "last_n_messages": 2,
9         "work_dir": "groupchat",
10        "use_docker": False,
11    },
12    human_input_mode="TERMINATE",
13 )
14 coder = autogen.AssistantAgent(
15     name="Coder",
16     llm_config=llm_config,
17 )
18 pm = autogen.AssistantAgent(
19     name="Product_manager",
20     system_message="Creative in software product ideas.",
21     llm_config=llm_config,
22 )
23
24 # Group chat manager
25 groupchat = autogen.GroupChat(agents=[user_proxy, coder, pm], messages=[], max_round=12)
26 manager = autogen.GroupChatManager(groupchat=groupchat, llm_config=llm_config)
```



[AutoGen Groupchat Notebook](#)

AutoGen Structure





[AutoGen](#)

- **Magentic-One:** 웹 및 파일 기반 작업을 위한 콘솔 기반 멀티 에이전트 어시스턴트. AgentChat을 기반으로 구축됨.
- **Autogen Studio:** 코드를 작성하지 않고 에이전트를 프로토타이핑하고 관리할 수 있는 애플리케이션. AgentChat을 기반으로 구축됨.
- **AgentChat:** 대화형 단일 및 멀티 에이전트 애플리케이션을 구축하기 위한 프로그래밍 프레임워크. Core를 기반으로 구축됨.
- **Core:** 확장 가능한 멀티 에이전트 AI 시스템을 구축하기 위한 이벤트 기반 프로그래밍 프레임워크.
- **Extensions:** 외부 서비스나 기타 라이브러리와 인터페이스하는 Core 및 AgentChat 구성 요소의 구현체.


Azure AI Services – Full Packages

Azure AI Agent Service




Azure OpenAI Assistants API

-  File Search
-  Code Interpreter

Model catalog







 **Azure OpenAI Service**
(GPT-4o, GPT-4o mini)

Models-as-a-Service




-  Llama 3.1-405B-Instruct
-  Mistral Large
-  Cohere-Command-R-Plus

Extensive ecosystem of tools

Knowledge

-  Microsoft Fabric
-  SharePoint
-  Bing Search
-  Azure AI Search
-  Licensed data
-  Files (local or Azure Blob)

Actions

-  Azure Logic Apps
-  OpenAPI 3.0 Specified Tools
-  Azure Functions

Built-In Enterprise Readiness

BYO-file storage | BYO-search index | BYO-thread storage | BYO-virtual network | OBO Authorization Support | Enhanced Observability

Azure Services for Automation



Azure Logic Apps

1,400개 이상의 커넥터로
광범위한 자동화
워크플로 구축



OpenAPI 3.0

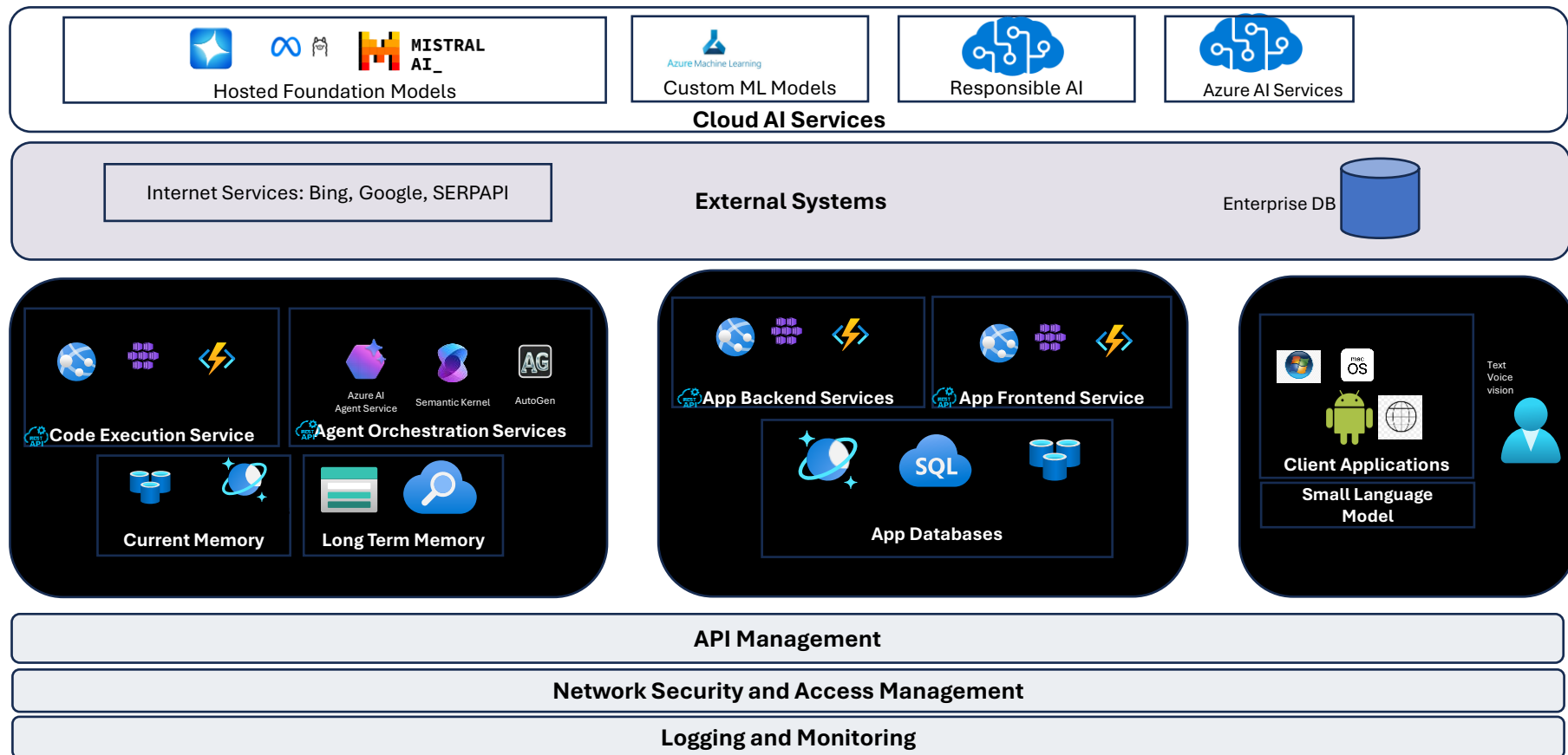
OpenAPI 3.0 사양을
활용하여 표준화된
사용자 정의 도구 도입



Azure Functions

Azure Functions를 사용해
상태 stateless 또는
stateful 코드 기반 액션
구현

System Architecture: Component View



AutoGen Demo

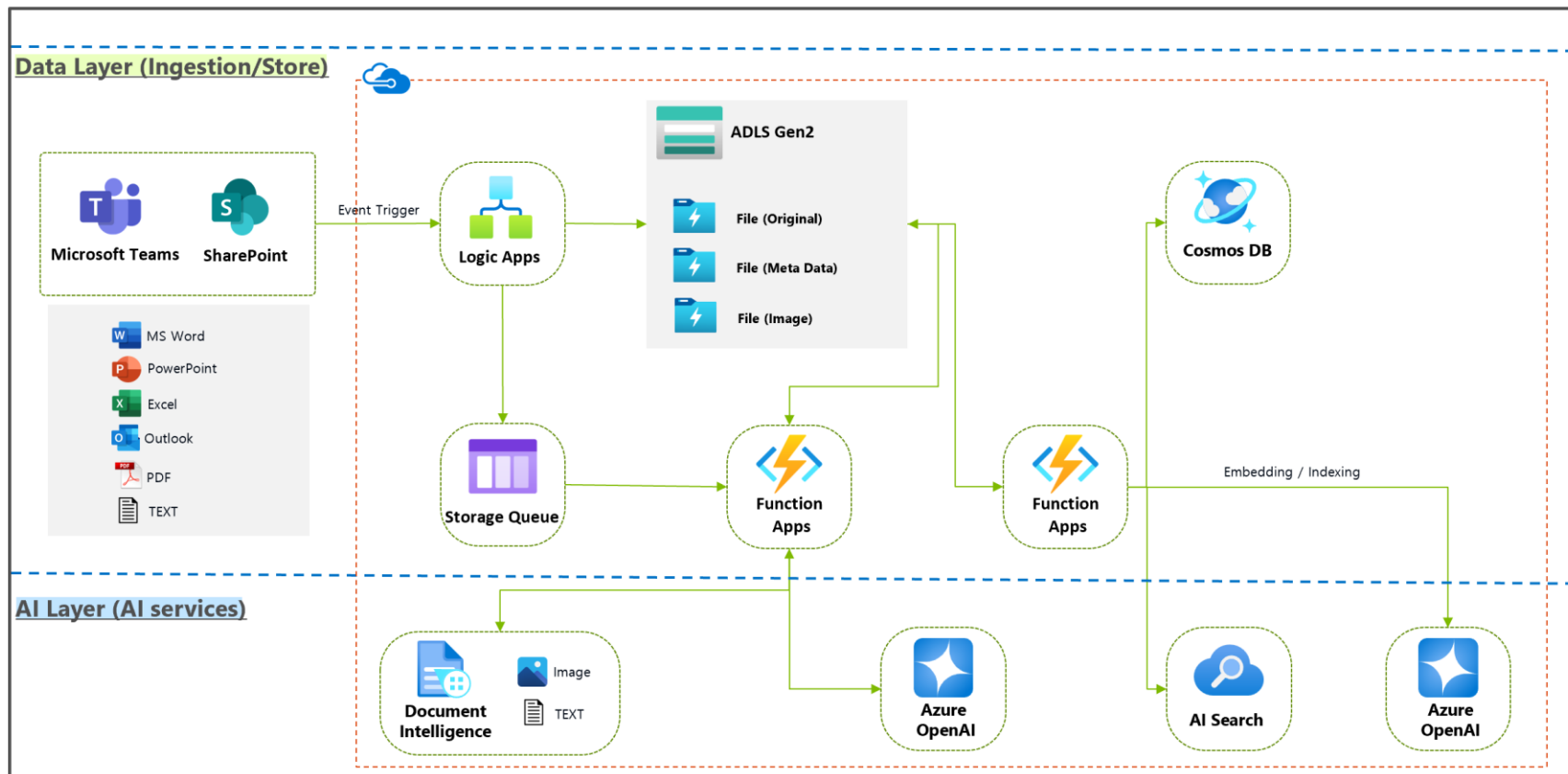
**ConversableAgent()
Roles and Conversation
Initiate_chat()
GroupChat()**



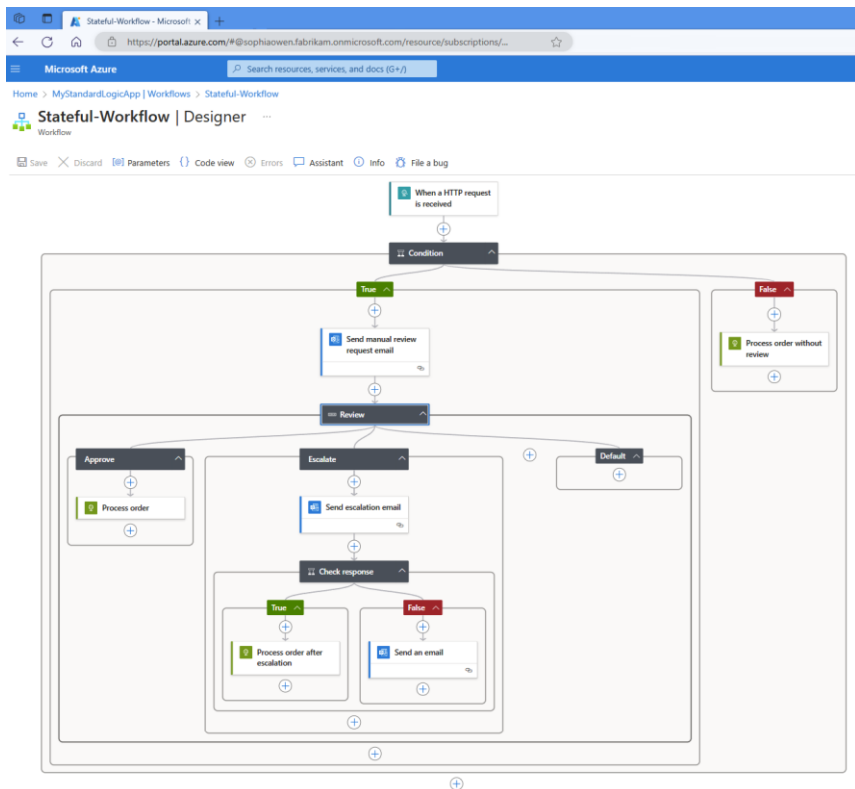
III. RAG Automation

1. High Level Architecture
2. Logic Apps
3. Function Apps
4. Document Intelligence /w Azure AI Services [Demo]
5. AI Search
6. Reflection

High Level Architecture (Scenario #1)



Logic Apps



1.워크플로우 자동화: 반복적인 업무와 비즈니스 프로세스를 자동화하여 생산성을 향상시킵니다.

2.통합 서비스: 다양한 클라우드 및 온프레미스 서비스와 애플리케이션을 통합할 수 있습니다. 예를 들어, Office 365, Dynamics 365, Salesforce, SharePoint, SQL Server 등과 통합할 수 있습니다.

3.트리거 및 액션: 특정 이벤트가 발생했을 때(트리거) 실행되는 일련의 작업(액션)을 정의할 수 있습니다. 예를 들어, 이메일이 수신되면 데이터를 처리하고 응답을 보내는 워크플로우를 설정할 수 있습니다.

4.커넥터: 미리 정의된 수백 개의 커넥터를 사용하여 다양한 서비스와 애플리케이션을 손쉽게 연결할 수 있습니다.

5.조건부 논리: 조건부 논리를 적용하여 복잡한 분기와 루프를 처리할 수 있습니다.

6.모니터링 및 관리: Logic Apps의 실행 상태를 모니터링하고, 문제 발생 시 디버깅하고 관리할 수 있는 도구를 제공합니다.

7.확장성: 클라우드 기반이므로 필요한 리소스를 유연하게 확장할 수 있습니다

Azure Data Lake Storage Gen2 (Event Trigger 방식)

Event-Condition-Action framework

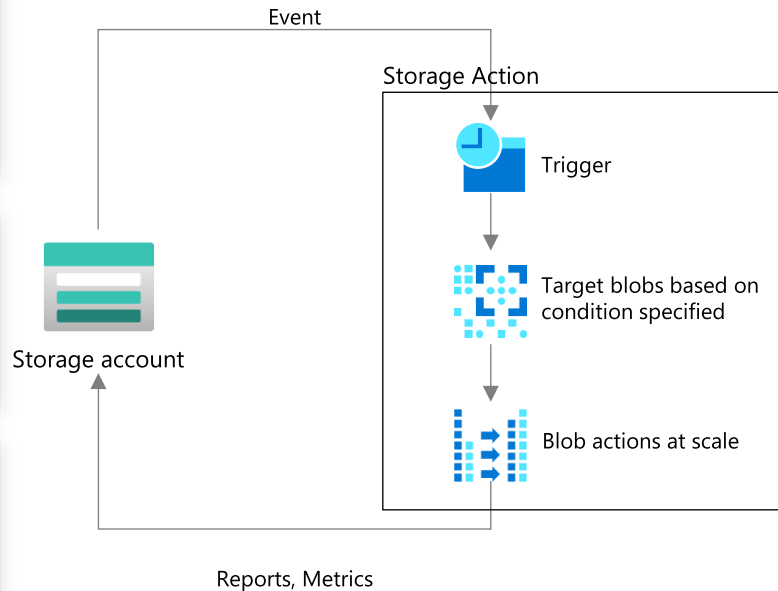
- 스케줄 기반 및 온디맨드 실행
- Blob 속성을 기반으로 한 조건부 처리
- Blob에 대한 네이티브 Blob 작업을 액션으로 사용

Serverless

- 완전 관리형 인프라
- 몇 분 만에 배포 – 복잡한 소프트웨어나 인프라가 필요 없음
- 스토리지에 따라 자동 확장

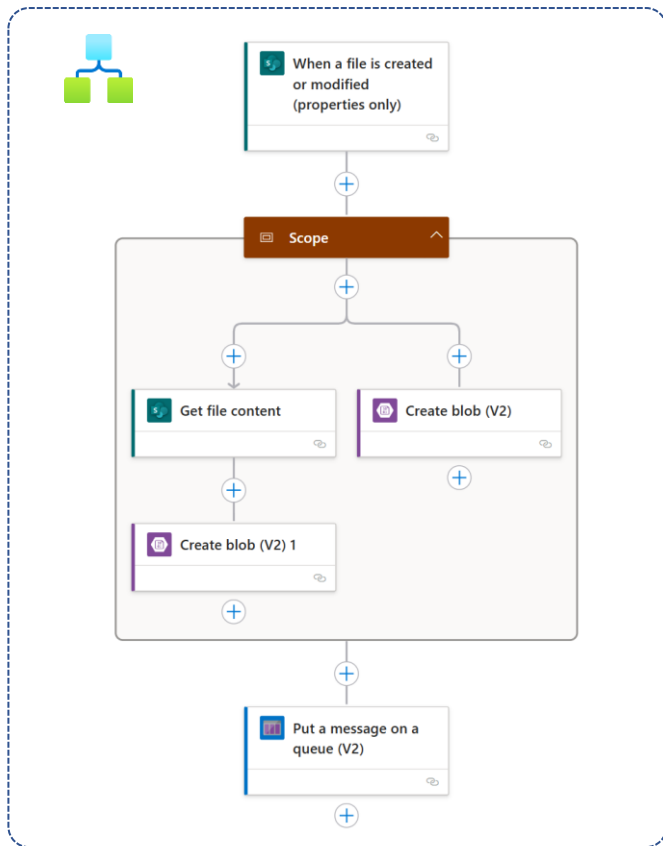
No-code with simplified management

- 코드 없이 클릭만으로 작업 구성
- 여러 스토리지 계정에 작업을 손쉽게 적용
- 집계 메트릭 및 세부 분석을 통해 스토리지 전반의 작업 실행 모니터링



Logic Apps

Logic Apps 를 활용하여, Teams (Sharepoint)의 업로드 된 파일을 원본과 메타정보로 분리하여 ADLS Gen2에 업로드



Add a trigger

sharepoint

Runtime: Select a runtime | Action type: Triggers

☒ Group by Connector

SharePoint

- When a file is classified by a Microsoft Syntax model [Trigger]
- When a file is created (properties only) [Trigger]
- When a file is created or modified (properties only) [Trigger]

Sharepoint 파일 변경에 대해서 Event Trigger 방식으로 시작

- Sharepoint 채널 단위로 설정 가능한 Logic Apps 트리거
- 기존 Data 공통 인프라 방식 활용

파일 기반 처리, 메타 데이터 관리를 통해 데이터의 일관성 보장

- 파일 고유 아이디 기반 처리로, 파일 변경(CRUID)에 대한 효율적 관리 가능
- 메타 데이터 별도 Json 파일 저장 (경로, 작성자, 작성일, 변경일, 썸네일 정보 등)

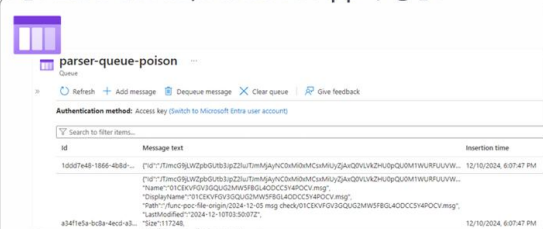
Azure Storage Queue 에 작업 메시지 생성

- Parsing 작업 트리거 목적 Queue에 메시지 인입

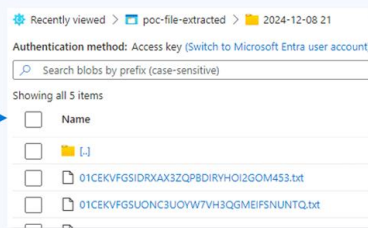
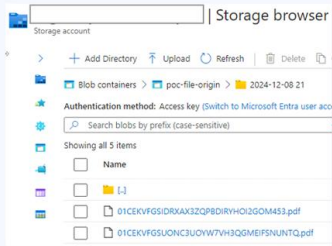
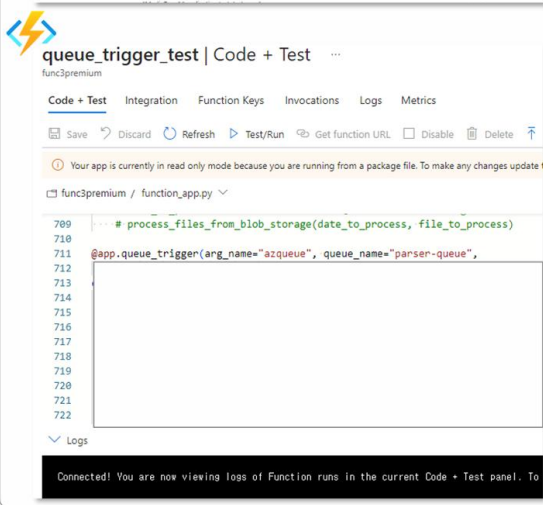
Function Apps + Storage Queue

Function App 을 활용하여, Azure Storage Queue 및 Azure Storage 를 활용한 사례
[ADLS Gen2 내 다양한 형식의 문서를 자동으로 파싱]

【 Azure Queue, Function App 구성 】



```
{ "id": "JTJmcG9jLW...",  
  "Name": "01CEKVFGSJPC5YDEQEVDLA4DFNSDCHKJZW.msg",  
  "DisplayName": "01...",  
  "Path": "/poc-file-origin/2024-12-10/01CEKVFGS...DCHKJZW.msg",  
  "LastModified": "2024-12-10T03:50:07Z",  
  "Size": 117248, ...  
}
```



Queue 기반 비동기 처리 구성을 통한 데이터 처리 효율성 향상

- 작업 대기열 관리로 처리 지연 최소화 및 시스템 부하 관리

Function App Scale out 기능으로 데이터 처리 성능 강화

- 필요 시점에 자동 확장을 통한 작업 병목 현상 해소

Function Apps + AI Search

AI Search 및 Function App 를 활용하여, 벡터 인덱싱 자동화 구현

【파일 형식별 파싱 Flow】



poc-index-241208

Save X Discard Refresh Create demo app Edit JSON Delete Encryption

Documents 1,298 Total storage 40.74 MB Vector index size 7.68 MB Max storage 160 GB

Search explorer Fields COES Scoring profiles Semantic configurations Vector profiles

+ Add field + Add subfield Delete Autocomplete settings

Search field names

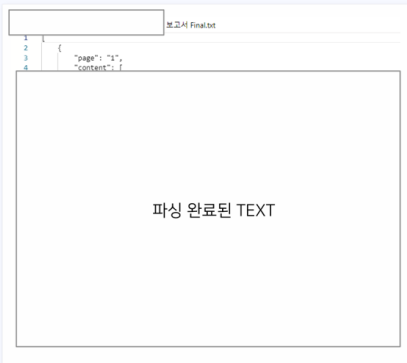
Field name	Type	Retrievable	Filterable	Sortable	Facetable	Searchable
page	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
content	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
file	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
filename	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
thumbnail	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
link	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
modified	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
contentVector	SingleCollection	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
id	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>



```
def initialize_search_index():  
    """  
    AI Search 인덱스 생성 or 업데이트  
    """  
    print("Initializing search index...")  
    # 인덱스 스키마 정의  
    fields = [  
        SearchableField(name="page", type=SearchFieldDataType.  
            SearchableField(name="content", type=SearchFieldDat  
            SearchableField(name="file", type=SearchFieldDat
```



Indexer-queue



```
{  
  "page": "1",  
  "content": "  
  \"file\": \"010  
  \"filename\":  
  \"thumbnail\":  
  \"link\": \"htt  
  \"modified\":  
  \"contentVector\": [  
    0.024675159,  
    0.005590924,  
    0.0024244583,  
    0.02233372,  
    0.020992834,  
    -0.017718386,  
    -0.00897739,  
    0.011071176,  
  ]  
}
```

검색기와 UI를 고려한 인덱스 설계로 검색 최적화

- 검색 필터링을 위한 파일 아이디, 문서 출처 표현을 위한 파일명, 페이지, 원본 파일 링크 등 포함
- 파싱 완료 데이터와 메타 정보를 결합하여 데이터 인덱싱

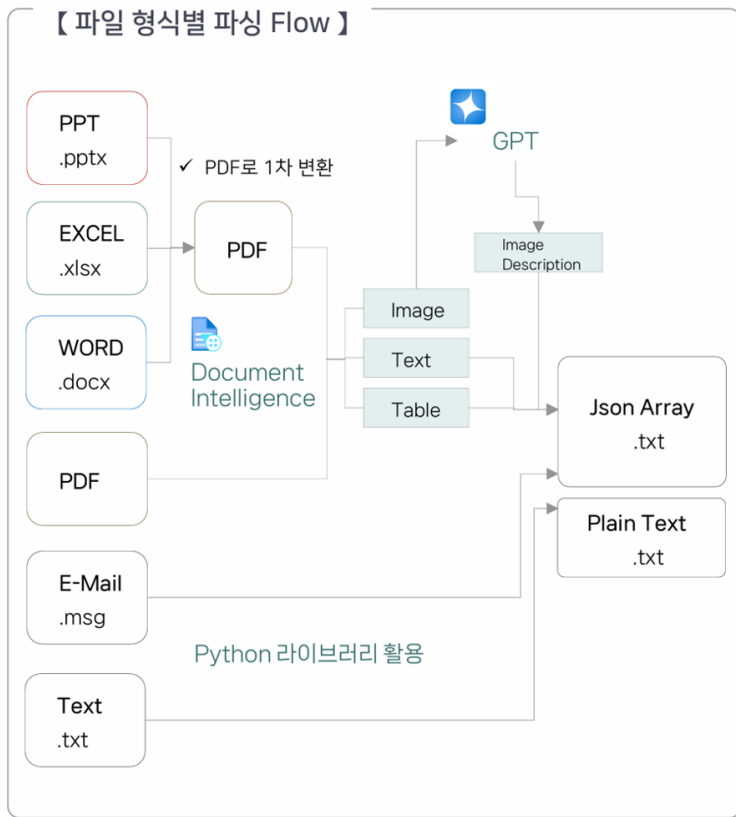
범용 기준 Chunking, Embedding 기법 적용

- 페이지 단위 Chunking, text-embedding-3 모델 사용
- 추후 embedding 모델 변경 및 전문가 튜닝을 위한 파라미터화 필요

Queue Trigger 기반 Function App 수행으로 처리 자동화

- 파싱 완료 파일 인입 즉시 vector db에 인덱싱

Function Apps + Document Intelligence



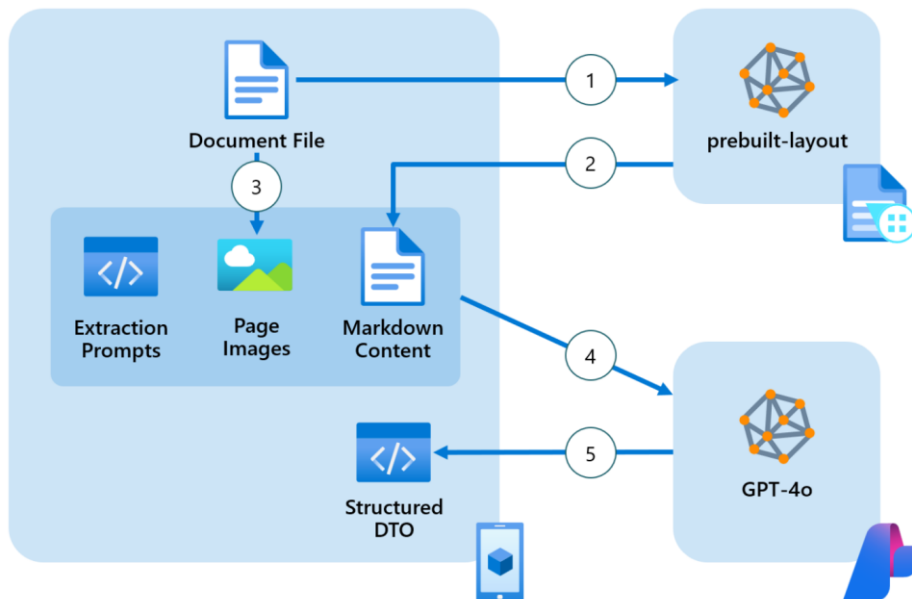
파일 형식별 파싱 로직 및 이미지 처리 로직 구현

- 확장자에 따라 최적의 파싱 방법 적용
- Document Intelligence 반환 요소 별 (Paragraph, Table, Figure) 맞춤형 처리
- Figure 요소는 Base64 인코딩 후, GPT-4o 모델을 활용해 설명을 생성하고, 이를 내용에 포함.

인덱싱 최적화를 위한 JSON Array 구조 활용

- 현업 과제 진행 경험을 바탕으로 한 전문가의 의견 반영
- 페이지별 'page', 'content', 'filename' 항목으로 Json 구성 후, 여러 페이지를 배열로 통합

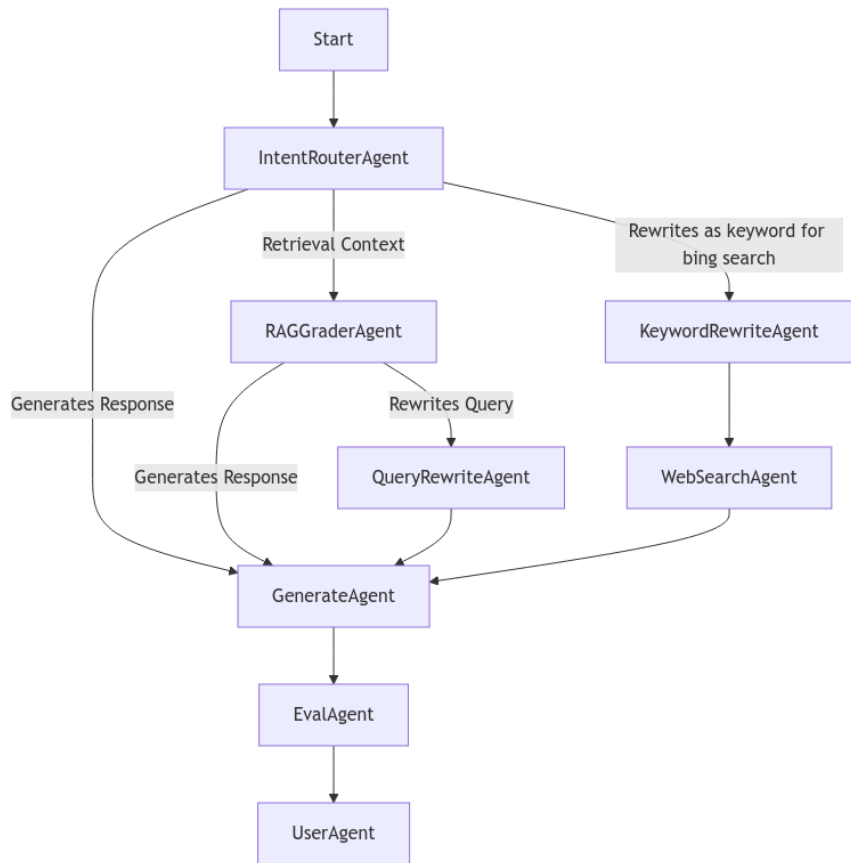
Document Intelligence /w Azure AI Services



[azure-ai-document-processing-samples/samples/extraction/vision-based/comprehensive.ipynb](https://github.com/Azure-Samples/azure-ai-document-processing-samples/tree/main/samples/extraction/vision-based/comprehensive.ipynb) at main · Azure-Samples/azure-ai-document-processing-samples · GitHub

Image Processing

Reflection – Agentic Design Pattern



1. 자기 모니터링(Self-Monitoring):

에이전트는 자신의 상태와 행동을 지속적으로 모니터링합니다. 예를 들어, 현재의 목표, 진행 상황, 사용된 자원 등을 추적합니다.

2. 피드백 수집(Feedback Collection):

에이전트는 자신의 행동 결과와 성과에 대한 피드백을 수집합니다. 외부 환경이나 다른 에이전트로부터 받은 반응을 기반으로 평가합니다.

3. 자기 평가(Self-Evaluation):

수집된 피드백을 분석하여 자신의 행동과 성과를 평가합니다. 어떤 행동이 효과적이었는지, 어떤 부분이 개선이 필요한지 판단합니다.

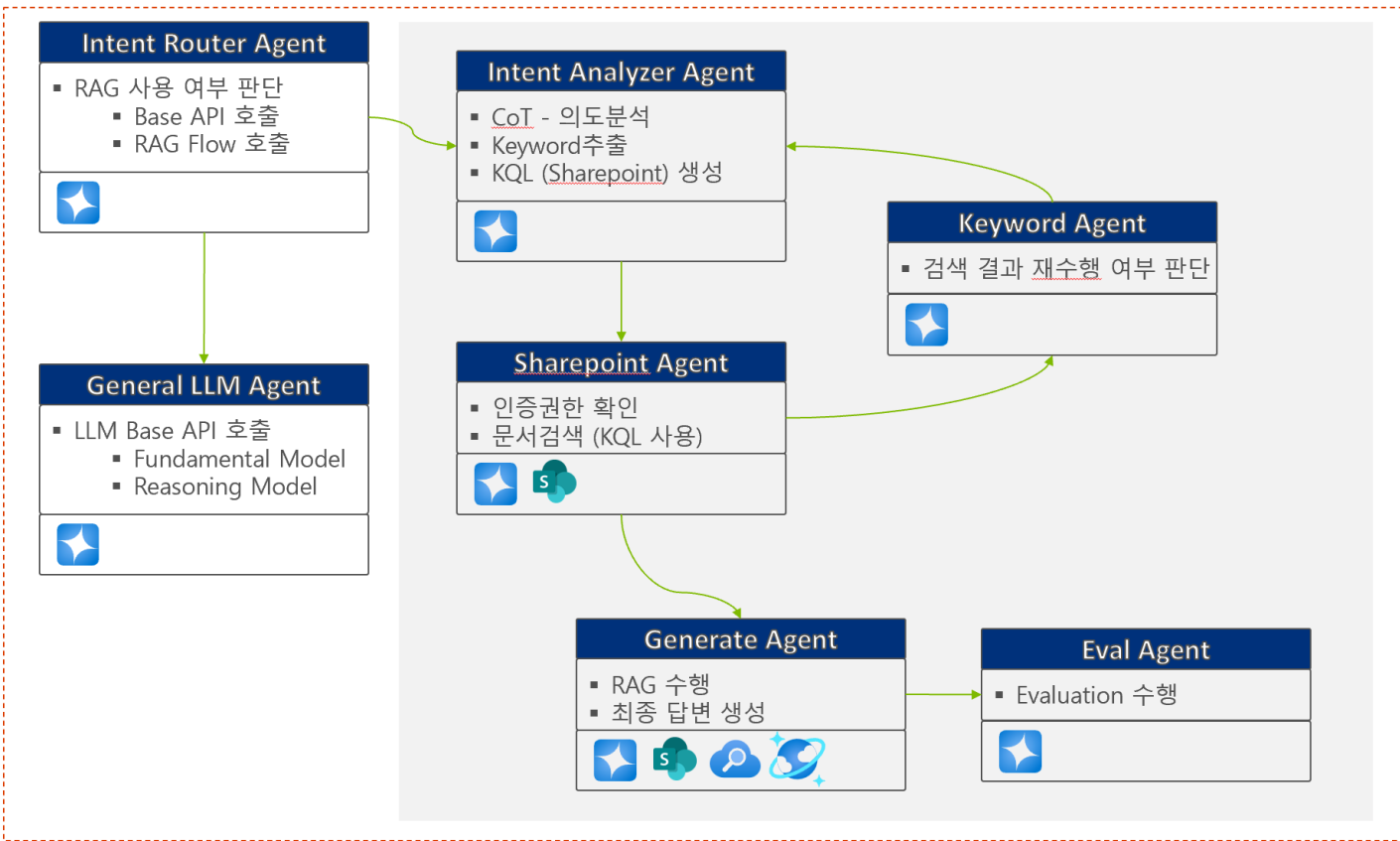
4. 학습 및 적응(Learning and Adaptation):

자기 평가 결과를 바탕으로 학습 알고리즘을 통해 새로운 전략을 개발하거나 기존 전략을 수정합니다. 환경 변화에 적응하기 위해 행동 방식을 조정합니다.

5. 행동 수정(Behavior Modification):

학습된 내용을 바탕으로 에이전트의 행동을 수정하고 개선된 계획을 실행합니다. 이를 통해 반복적인 피드백 루프를 형성하여 지속적으로 성능을 향상시킵니다.

Reflection – Agentic Route



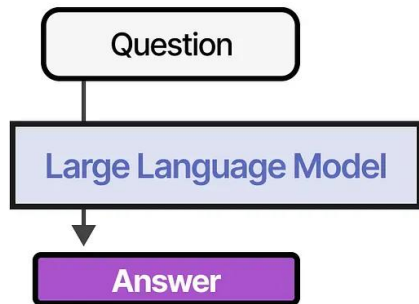


IV. Consideration

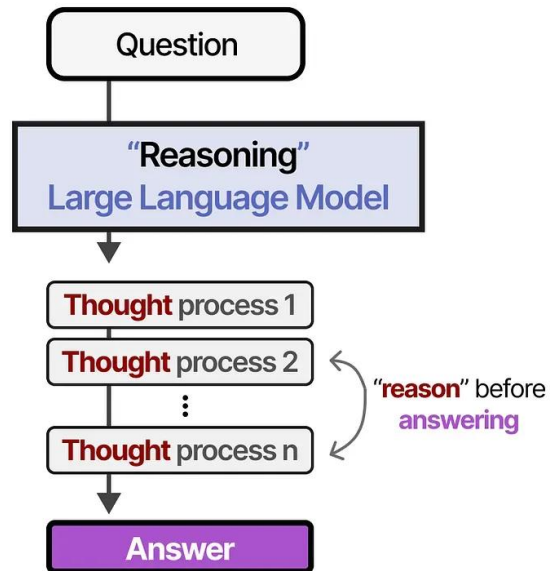
1. Reasoning LLMs
2. Lab Instruction

■ LLM 모델의 변화 - Reasoning LLMs

“Regular” LLMs



“Reasoning” LLMs



HoL Workshop Instruction

<https://github.com/trentkim-ms-ats/multi-agent-workshop>

Q&A