AI Frameworks / AutoGen

2025.12.23
peny.official

kakao

Compound AI Systems 구축을 위한 차세대 프로그래밍 패러다임

- 배경: 단일 모델 중심에서 여러 에이전트와 도구가 결합된 협업 시스템으로의 진화.
- 요약: 에이전틱 프로그래밍의 정의, 프레임워크 필요조건, 그리고 AutoGen의 핵심 기능 및 사례

Future of AI Application

- 생성형(Generative)에서 에이전(Agentic) 으로 진화:
  미래의 AI 애플리케이션은 단순히 콘텐츠를 생성하는 단계를 넘어,
  인간을 대신해 복잡한 작업을 독립적으로 실행
- 예시
  - 과학 에이전트: 자동 과학 발견 수행.
  - 웹 에이전트: 웹 탐색 및 업무 자동화.
  - 소프트웨어 에이전트: 스스로 소프트웨어 구축 및 오류 수정

Agentic Programming

AI 에이전트가 인간의 개입을 최소화하면서 독립적으로 복잡한 워크플로우를 실행하도록 설계하는 방식

1. 작업 효율성 및 결과물 품질 극대화: 반복과 협업을 통한 품질 향상, Divide & Conquer, 검증
2. 시스템 설계의 유연성 및 개발 편의성: 모듈화된 구조, Human-in-the-loop, 신속한 실험과 최적화

에이전트 프레임워크의 필요조건(Desiderata)

효과적인 에이전트 시스템 구축을 위해 프레임워크는 다음을 지원해야 함

1. 통합된 추상화: 모델, 도구, 인간을 하나의 AI 시스템 내에서 유기적으로 통합.
2. 유연한 오케스트레이션: 정적/동적 워크플로우, 중앙화/탈중앙화 등 다양한 상호작용 패턴 지원.
3. 설계 패턴 지원: 대화(Conversation), 계획 수립(Planning), 도구 사용, 멀티모달리티 등의 패턴 구현.
4. 확장성: 산업별 상이한 요구사항에 맞춰 커스텀 워크플로우 구성 가능

주요 에이전틱 AI 프레임워크 비교

1. AutoGen: 멀티 에이전트 '대화(Conversation)' 기반 프로그래밍을 지원하며 가장 포괄적이고 유연함.
2. LangGraph: 그래프 기반의 제어 흐름(Control Flow) 제공에 특화.
3. CrewAI: 고수준의 정적인 에이전트-태스크 워크플로우 중

AutoGen

AutoGen의 핵심: 대화형 프로그래밍

AutoGen은 에이전트형 AI를 위한 프로그래밍 프레임워크로, "대화 (Conversation)" 를 중심요소로 사용하여 AI 에이전트의 설계를 단순화

- Define Agents(에이전트 정의): AutoGen의 에이전트는 대화가 가능(Conversable)하며, LLM, 도구, 인간 또는 이들의 조합으로 자유롭게 맞춤 설정(Customizable)할 수 있음.
- 대화형 프로그래밍 (Conversation Programming): 에이전트 간의 대화 패턴(순차, 중첩, 그룹)을 정의하여 복잡한 로직 처리함.

AutoGen의 대화 패턴

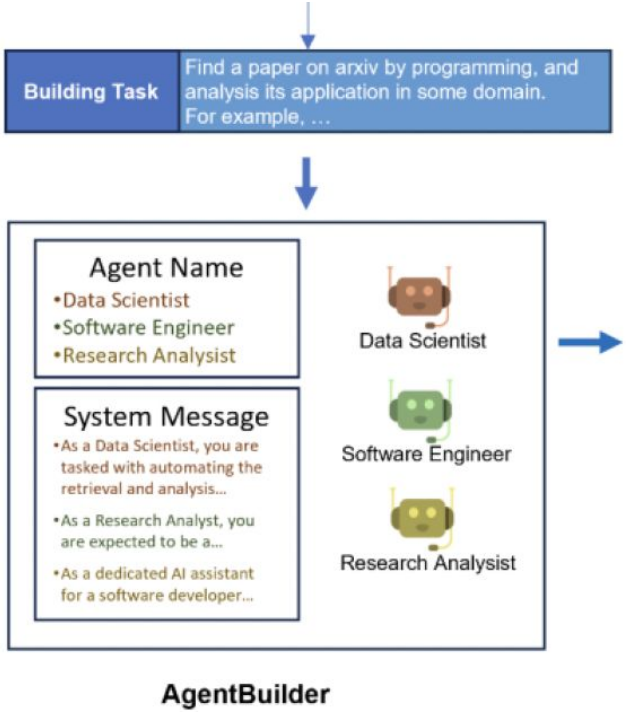
- 순차적 대화(Sequential Chat): 에이전트들이 정해진 순서대로 업무를 전달.
- 중첩 대화(Nested Chat): 한 에이전트가 내부적으로 다른 에이전트 팀과 대화하여 결과를 도출(외부에는 단일 에이전트처럼 보임).
- 그룹 대화(Group Chat): 관리자 에이전트가 상황에 맞춰 다음 발언자를 동적으로 선택.
- 상태 기반(StateFlow): 상태 머신(State Machine) 로직을 적용해 엄격한 전이 규칙 제어

AutoGen: AutoBuild

어떤 에이전트가 필요한지 일일이 설계하고 코딩할 필요 없이, 자동으로 멀티 에이전트 시스템을 구축(Build) 해주는 프레임워크

Adaptive Build: 작업을 여러 단계의 하위 작업(Subtask)으로 쪼개고, 각 단계마다 필요한 에이전트 팀을 동적으로 구성하거나 기존 에이전트 라이브러리에서 최적의 에이전트를 선택하여 투입

빌드 매니저(LLM): 사용자가 입력한 작업 요구사항을 분석하여 어떤 전문가들이 필요한지 결정. 이때 각 에이전트에게 이름과 함께 매우 상세한 "시스템 메시지(System Message)"를 생성해 주는데, 이 메시지가 바로 에이전트의 롤(페르소나)가 됨.



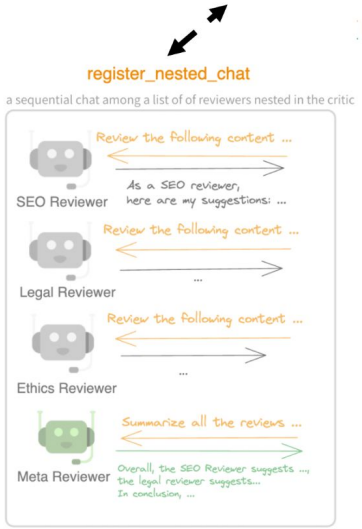
AutoGen 적용 사례

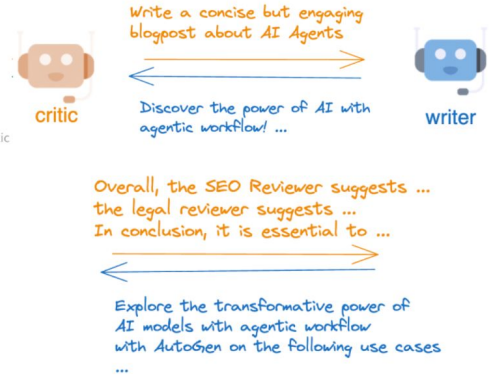
- 블로그 글쓰기: Writer(초안)와 Critic(비평) 간의 반복적인 리플렉션을 통해 품질 향상.

Blogpost Writing with Advanced Reflection

```
critic.register_nested_chats(
    review_chats,
    trigger=writer,
)

critic.initiate_chat(
    recipient=writer,
    message=task,
    max_turns=2,
    summary_method="last_msg"
)
```



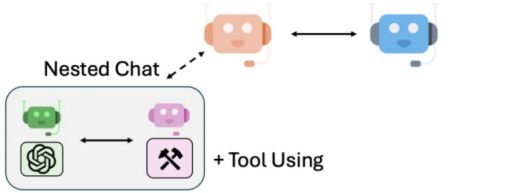


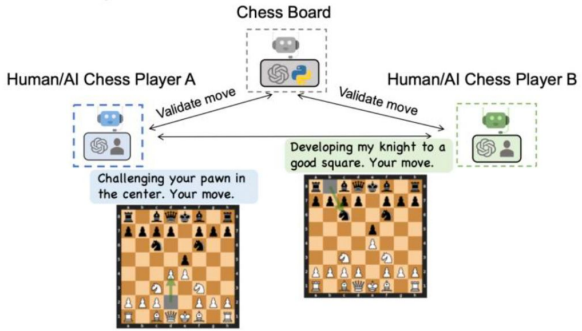
Nested Chat

AutoGen 적용 사례

- 대화형 체스: 체스판 도구(Tool)를 활용해 규칙을 엄격히 준수하며 게임 진행.

.






```python
from autogen import register_function

for caller in [player_white, player_black]:
    register_function(
        get_legal_moves,
        caller=caller,
        executor=board_proxy,
        name="get_legal_moves",
        description="Get legal moves.",
    )

    register_function(
        make_move,
        caller=caller,
        executor=board_proxy,
        name="make_move",
        description="Call this tool to make a move.",
    )
```

AutoGen 적용 사례

```python
def state_transition(last_speaker, groupchat):
    messages = groupchat.messages

    if last_speaker is initializer:
        # init -> retrieve
        return coder
    elif last_speaker is coder:
        # retrieve: action 1 -> action 2
        return executor
    elif last_speaker is executor:
        if messages[-1]["content"] == "exitcode: 1":
            # retrieve --(execution failed)--> retrieve
            return coder
        else:
            # retrieve --(execution success)--> research
            return scientist
    elif last_speaker == "Scientist":
        # research -> end
        return None


groupchat = autogen.GroupChat(
    agents=[initializer, coder, executor, scientist],
    messages=[],
    max_round=20,
    speaker_selection_method=state_transition,
)
```

- 그룹 대화를 통한 복잡한 작업 기획: 관리자(Manager) 에이전트는 전체 대화 흐름을 모니터링하며, 현재 상황에서 다음에 발언할 가장 적절한 에이전트를 선택하여 작업의 진행 상황에 따라 동적으로 워크플로우를 결정
- 상태 기반 워크플로우: 작업 수행 과정을 "상태 머신(State Machine)"으로 개념화하여, LLM이 미리 정의된 상태 전이(state transfer) 규칙에 따라 움직이게 함.

AutoGen 적용 사례2

- SCIAgent: AutoGen 프레임워크를 기반 과학 연구용 멀티 에이전트 리즈닝 에이전트로, 과학적 발견을 자동화하기 위해 지식 그래프와 멀티 에이전트 시스템을 결합하여 가설 수립 및 검증을 수행
- Agent-E: AutoGen 프레임워크를 기반으로 구축된 혁신적인 자율형 웹 에이전트, 계층적 에이전트 팀을 구성하여 복잡한 웹 환경에서 항공권 예약이나 클리닉 양식 작성 등의 작업을 자동화

Multimodal Assistant:

LlamaIndex

2025.12.23
peny.official

kakao

(기업) 데이터 기반의 컨텍스트 증강 LLM 애플리케이션

- 목표: 어떤 작업(입력)이든 수행하여 최적의 결과(출력)를 제공하는 인터페이스 구축.
- 지원 범위: 프로토타입에서 운영(Production) 단계까지의 전 과정 지원.
- 핵심 요소: 고품질 데이터 처리, 멀티모달 RAG, 에이전트적 추론, 확장 가능한 배포

Retrieve (검색)

기본 RAG(Naive RAG)의 한계와 도전

"Garbage In = Garbage Out" 원칙

- 단순 텍스트 분할 및 고정된 검색 방식은 복잡한 문서 구조를 무시함.
- 낮은 쿼리 이해도 및 계획 능력 부재로 인해 할루시네이션(환각) 유발.
- 단순 검색만으로는 복잡한 태스크 활용에 한계가 있음

High-Quality Multi-Modal RAG: LlamaParse

표, 차트, 이미지, 불규칙한 레이아웃이 포함된 복잡한 PDF를
AI 기반으로 파싱하여 데이터 무결성을 유지.

High-Quality Multi-Modal RAG: Advanced Indexing

텍스트, 표, 이미지 등 이질적인 데이터를 계층적으로 인덱싱하고,
각 요소의 요약본을 만들어 참조

- 요약본 기반 참조(Summary-based References): 표/이미지에서 텍스트 요약본을 추출하여 벡터 DB에 인덱싱.
- 노드(Node) 구조: 요약본 노드가 실제 원본 데이터(Source Chunk)를 가리키는 포인터 역할 수행.
- 재귀적 검색(Recursive Retrieval): 요약 노드 검색 후 관련 원본 데이터를 추적하여 정밀하게 추출

Multi-Modal RAG Pipeline

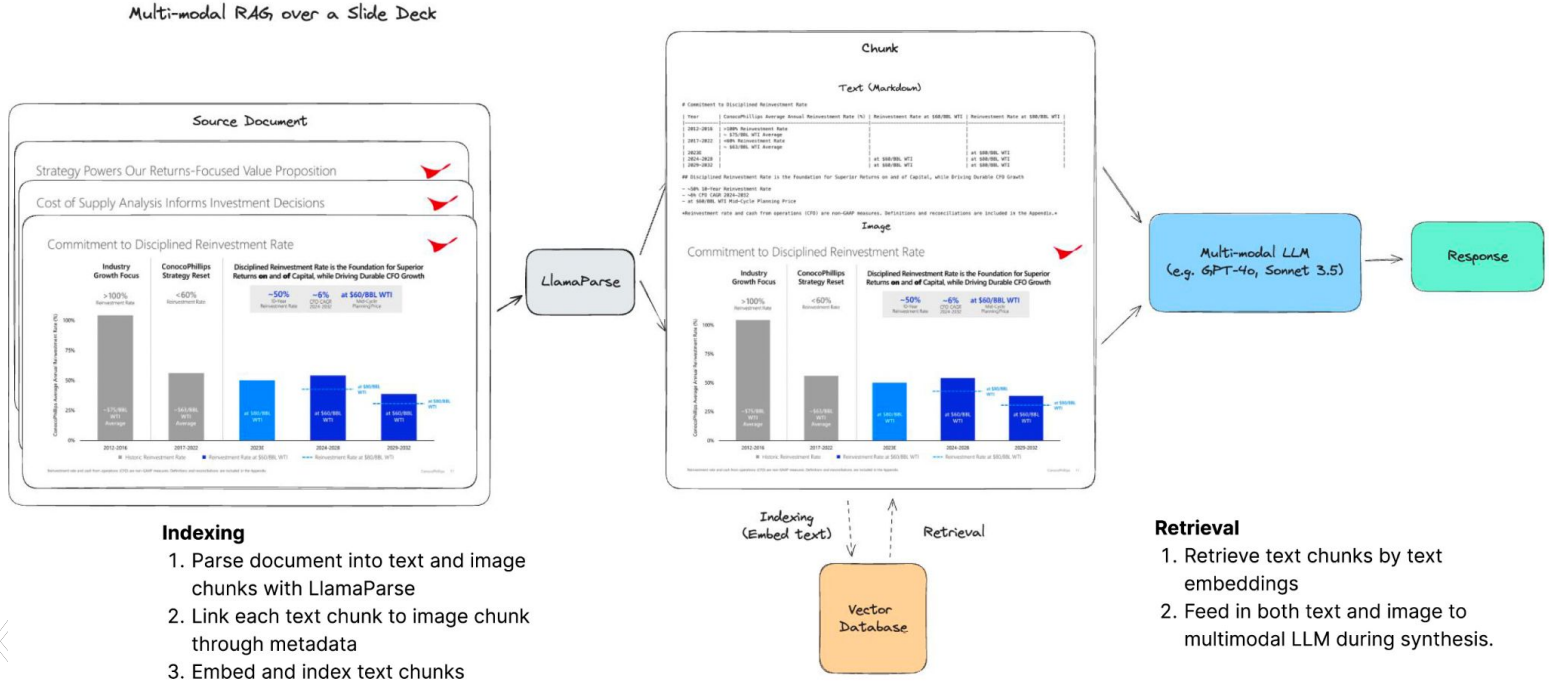
문서를 텍스트와 이미지 청크로 파싱한 뒤 메타데이터로 연결.
생성 시 멀티모달 LLM에 텍스트와 이미지를 동시에 입력하여 확장된 쿼리 결과

- 메타데이터 연결(Metadata Linking): 파싱 시 텍스트 청크와 이미지 청크를 메타데이터로 상호 연결.
- 검색 및 합성(Retrieval & Synthesis): 텍스트 임베딩으로 검색 후, 텍스트와 이미지를 멀티모달 LLM에 동시 입력

Multi-Modal RAG Pipeline

Multimodal RAG Pipeline

A true multimodal RAG pipeline stores both text and image chunks for use within a multi-modal LLM



Indexing

1. Parse document into text and image chunks with LlamaParse
2. Link each text chunk to image chunk through metadata
3. Embed and index text chunks

Retrieval

1. Retrieve text chunks by text embeddings
2. Feed in both text and image to multimodal LLM during synthesis.

Report Generation

단순한 챗봇 응답형태를 넘어 텍스트와 이미지가 섞인 형태의 종합 보고서를 자동 생성합니다. 이는 단순 답변보다 시간 절약 및 역량 강화 측면에서 훨씬 높은 투자 대비 효과(ROI)를 제공

Agentic Reasoning over Complex Input

(복잡한 입력에 대한 에이전트적 추론)

단순한 검색을 넘어 LLM이 스스로 판단하고 계획하여
어려운 과제를 해결하는 핵심 능력

Agentic Reasoning over Complex Input: Agentic RAG

복잡한 작업 해결을 위한 에이전트적 추론

- 핵심: 모든 데이터 인터페이스를 '도구(Tool)'로 간주하고 에이전트가 스스로 선택
  - 기본적인 RAG: 질문 → 검색 → 답변
  - 에이전트 RAG: 검색뿐만 아니라 SQL 쿼리 실행, 웹 검색, API 호출 등 다양한 도구 중 현재 질문을 해결하는 데 가장 적합한 것을 스스로 선택
- 추론 루프: 순차적(Sequential), DAG, 트리 구조 등을 활용해 고난도 과제 수행

Agentic Reasoning over Complex Input: Flows

추론의 방식과 자율성

- Constrained (제약된 흐름): 라우터 기반으로 인간이 흐름을 정의하여 신뢰성이 높지만 표현력은 제한적.
- Unconstrained (제약 없는 흐름): React, LLM Compiler 등을 사용하여 에이전트가 스스로 계획을 세우며 표현력이 높지만, 무한 루프 위험과 높은 비용이 발생할 수 있음.

Agentic Reasoning over Complex Input: Orchestration

LlamaIndex는 에이전트 오케스트레이션 프레임워크가 갖춰야 할 필수 속성으로 다음 6가지를 정의합니다.

- 이벤트 기반 (Event-Driven): 이벤트 입출력을 통해복잡한 비동기 로직을 효과적으로 관리.
- 코드 우선 (Code-first): 오케스트레이션 로직을 파이썬 코드로 직접 작성합니다. 이는 읽기 쉽고 확장성이 뛰어나며, 개발자에게 친숙한 환경을 제공합니다.
- 구성 가능성 (Composable): 세분화된 작은 워크플로우들을 결합하여 더 높은 수준의 복잡한 시스템을 구축할 수 있습니다.
- 유연성 (Flexible): LLM 호출뿐만 아니라 일반 파이썬 코드를 사용하여 로직을 자유롭게 작성할 수 있습니다,.
- 디버깅 및 관찰 가능성 (Debuggable & Observable): 시스템의 상태를 관찰하고 각 단계를 추적할 수 있어, 추론 과정에서 발생하는 오류를 쉽게 찾아낼 수 있습니다.
- 프로덕션 배포 용이성: 주피터 노트북에서 작성한 코드를 실제 운영 환경의

운영 환경에서의 에이전트 배포

Production 앱을 위한 5대 필수 요구사항

1. 캡슐화: 각 에이전트 기능을 독립 서비스로 관리.
2. 표준화 통신: 에이전트 간 및 클라이언트와의 표준 인터페이스 확보.
3. 확장성: 사용자 수 및 에이전트 수 증가에 대한 대응 능력.
4. Human-in-the-loop: 에이전트의 판단 지원 및 승인을 위한 인간 개입 서비스.
5. 관찰 가능성: 개발자를 위한 디버깅 및 모니터링 도구

END