LangGraph

Fast campus
테디노트의 <RAG 비법노트>

LangGraph 탄생 배경

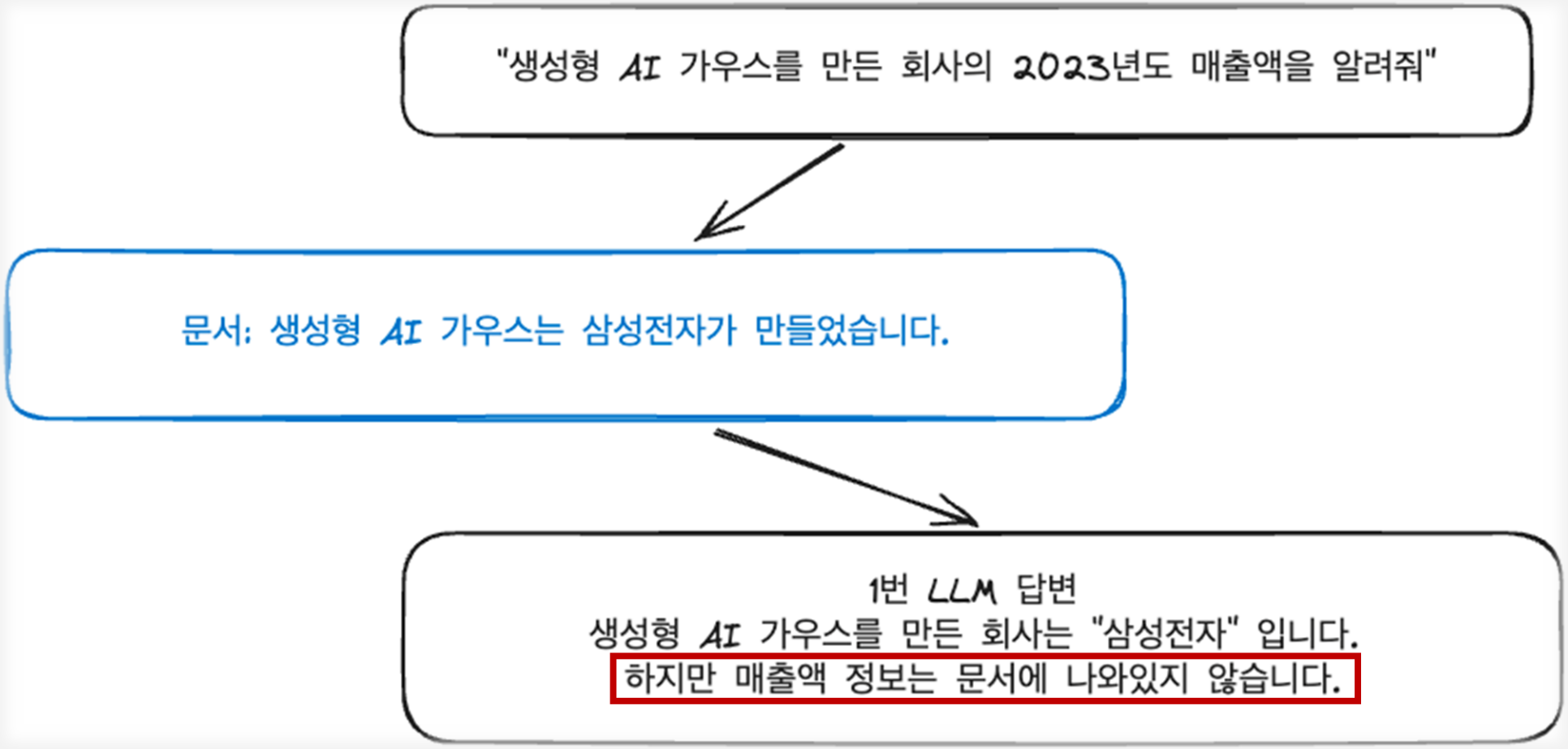
- RAG(Retrieval-Augmented Generation) 이라는 강력한 기능을 갖게 된 우리는 한 번쯤 다음의 **갈등**을 **마주**하게 됩니다.

1. LLM 이 생성한 답변이 **Hallucination** 이 아닐까?
2. RAG 를 적용하여 받은 답변이 **문서에는 없는 “사전지식”으로 답변**한 건 아닐까?
3. “문서 검색에서 원하는 내용이 없을 경우”

   > **“인터넷”** 혹은 **“논문”** 에서 **부족한 정보를 검색**하여 **지식을 보강**할 수 는 없을까?

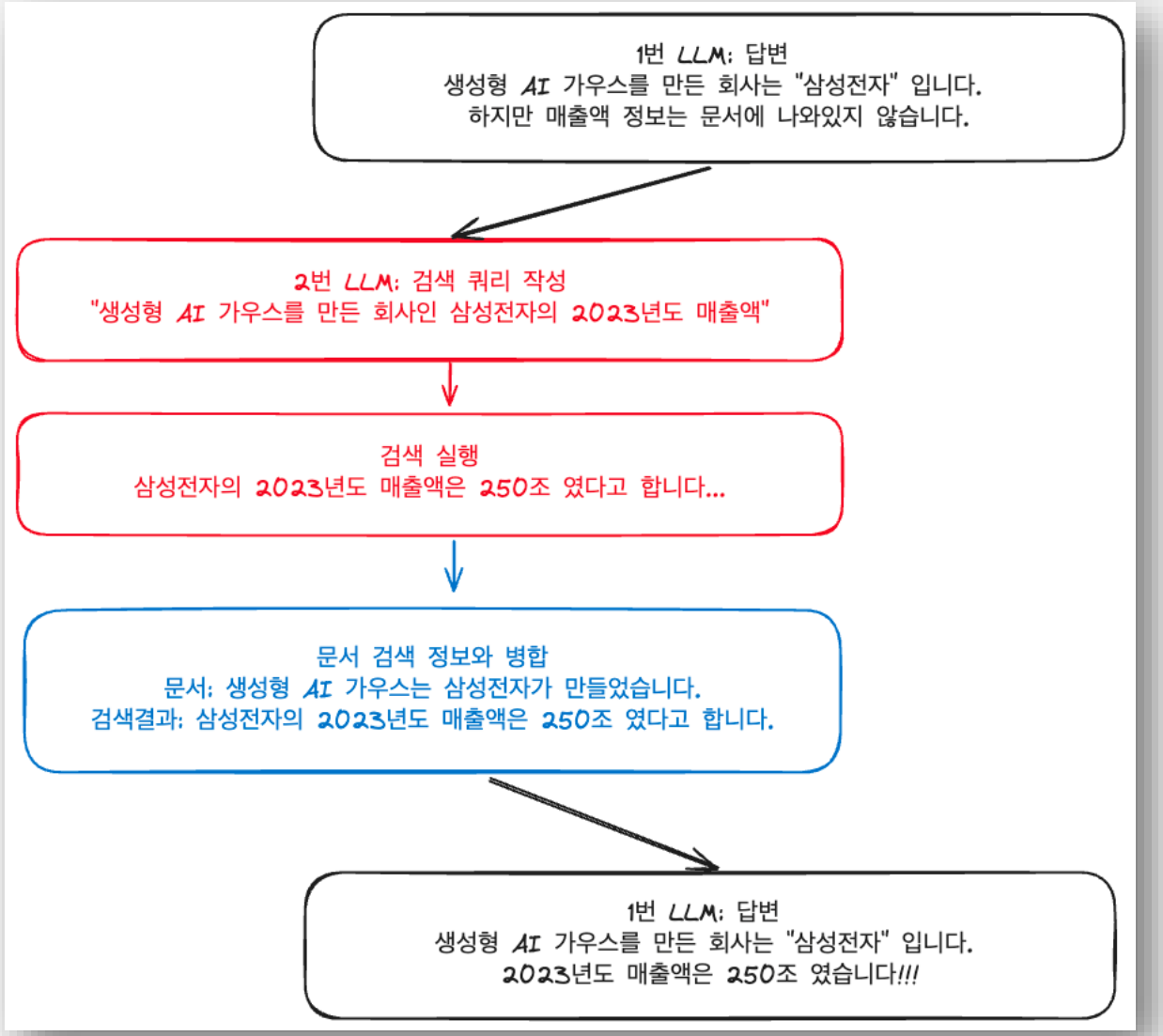
RAG 개발 단계의 고민 사례 (1)

- 일반 RAG 를 수행했을 때 흔히 볼 수 있는 상황 (문서 내 질문에 대한 답변이 존재하지 않는 경우)



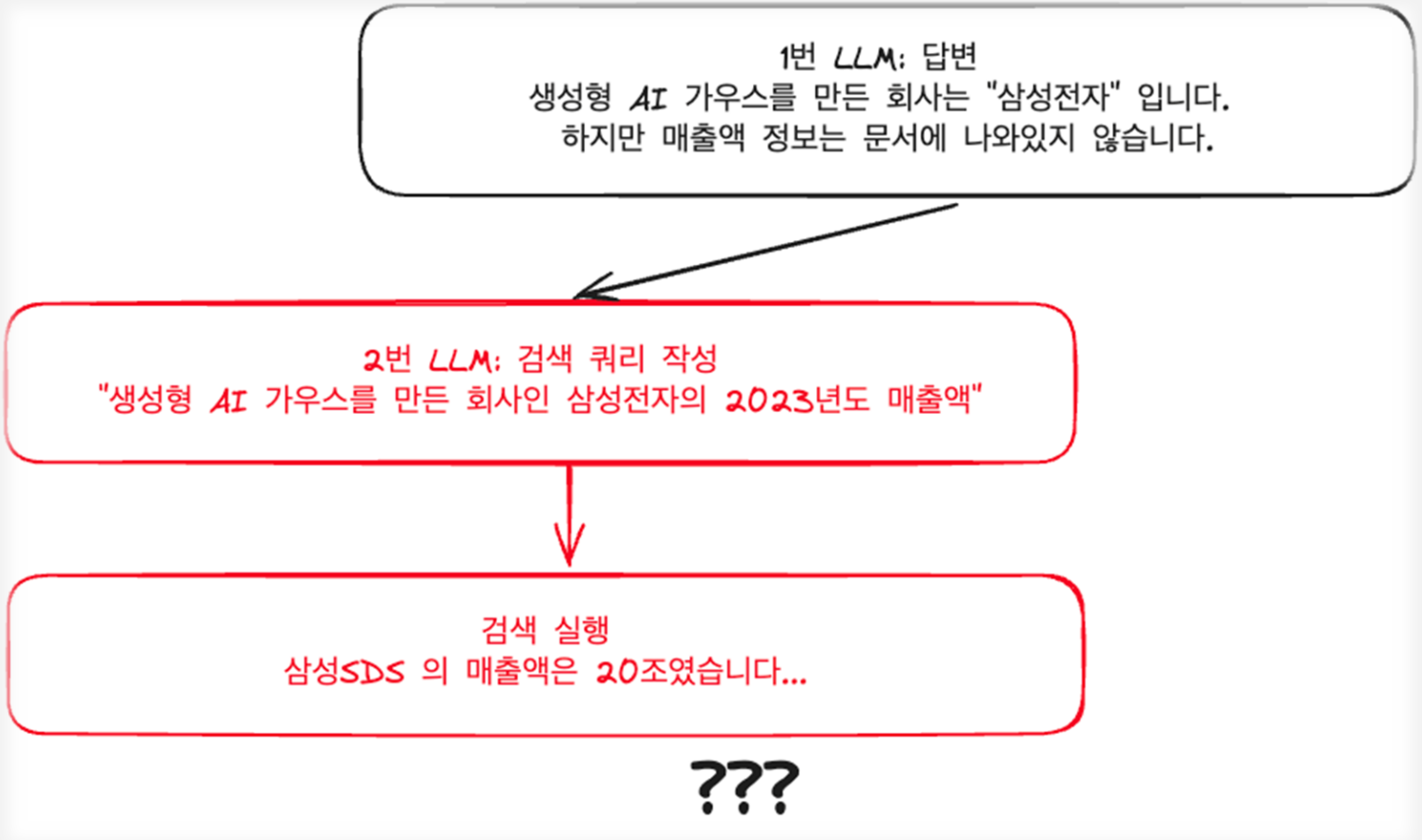
RAG 개발 단계의 고민 사례 (2)

- 그럼 부족한 **정보를 Web 검색하여 문서에 추가하는 로직을 추가**해 보자!



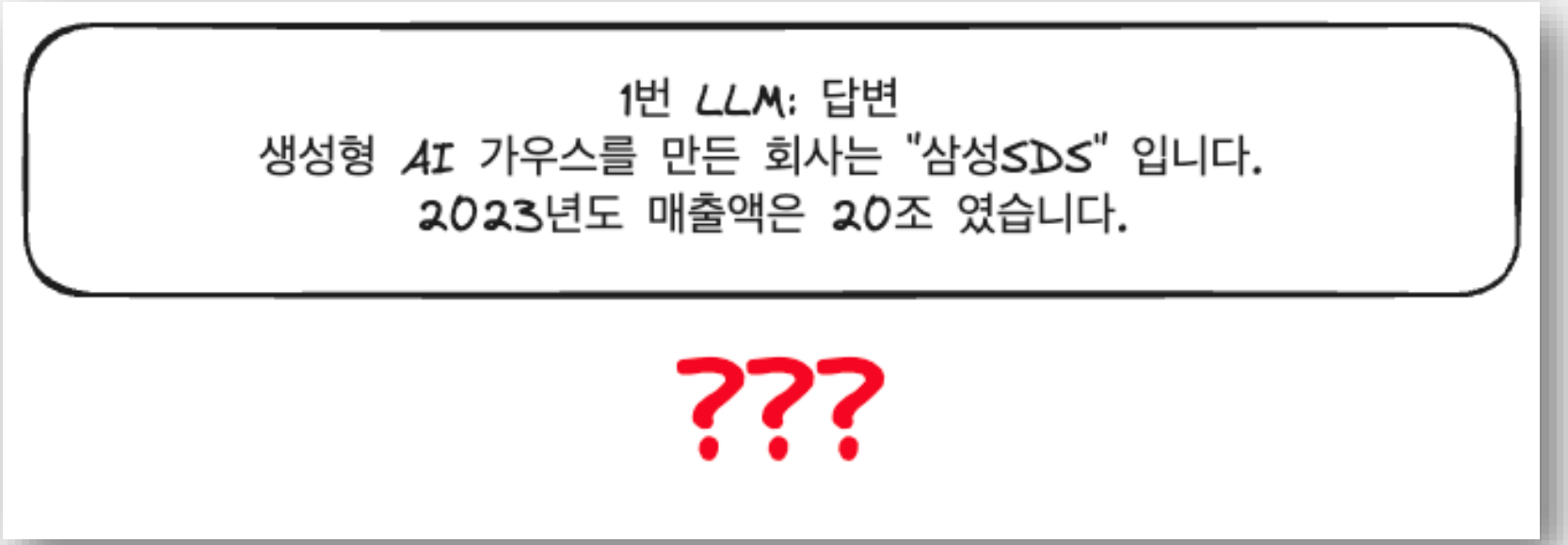
RAG 개발 단계의 고민 사례 (3)

- 만약 검색결과에 **잘못된 정보가 포함**되거나 혹은 **검색 결과에 없다면?**



RAG 개발 단계의 고민 사례 (4)

- 잘못된 검색결과가 결국 **Hallucination** 이 이어진다면?

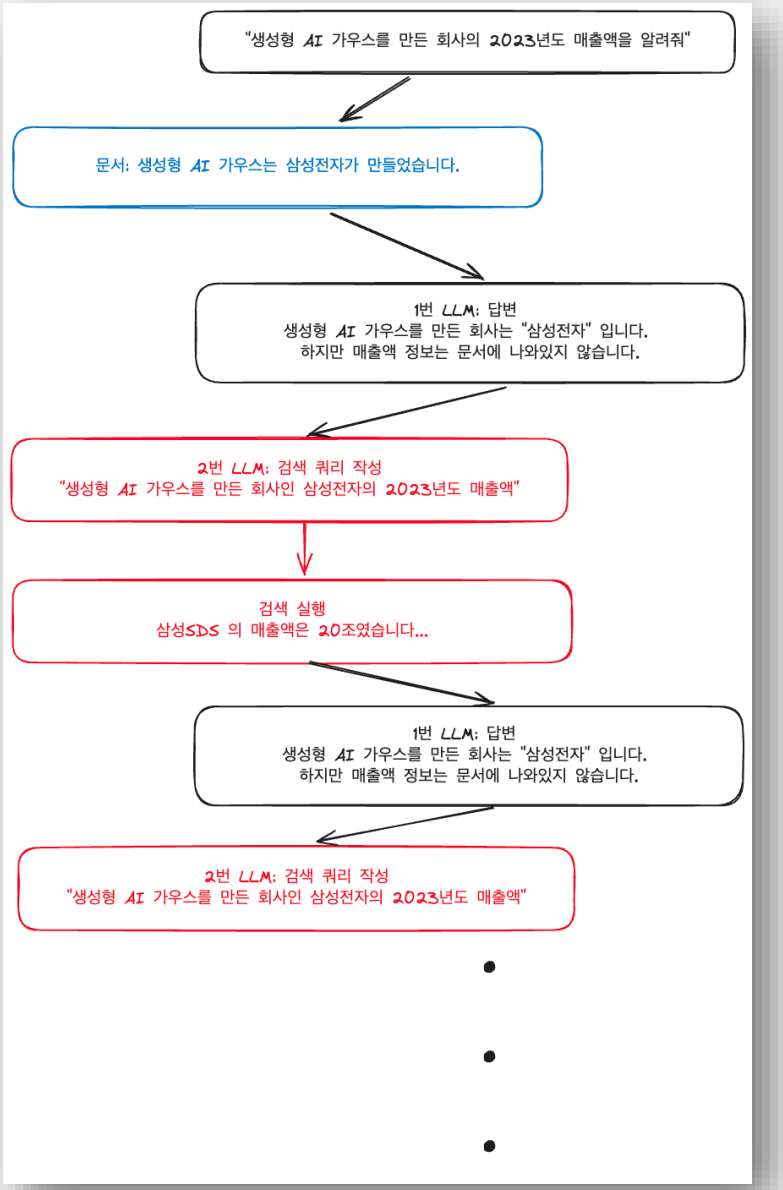


RAG 개발 단계의 고민 사례 (5)

- 검색이 제대로 나올때까지 반복하여 검색 해볼까? (반복문)
  - 무한히 제대로된 결과가 안 나올 경우 토큰 사용량이 폭증할텐데…
- Hallucination 을 방지하는 LLM 을 추가해야 하나?

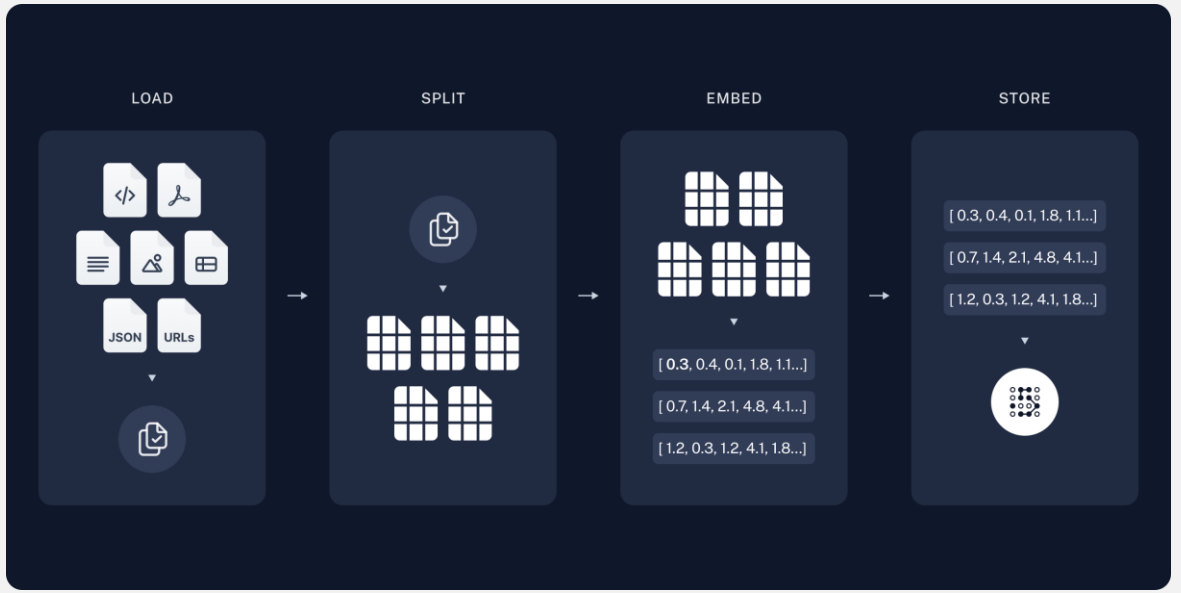
결국,

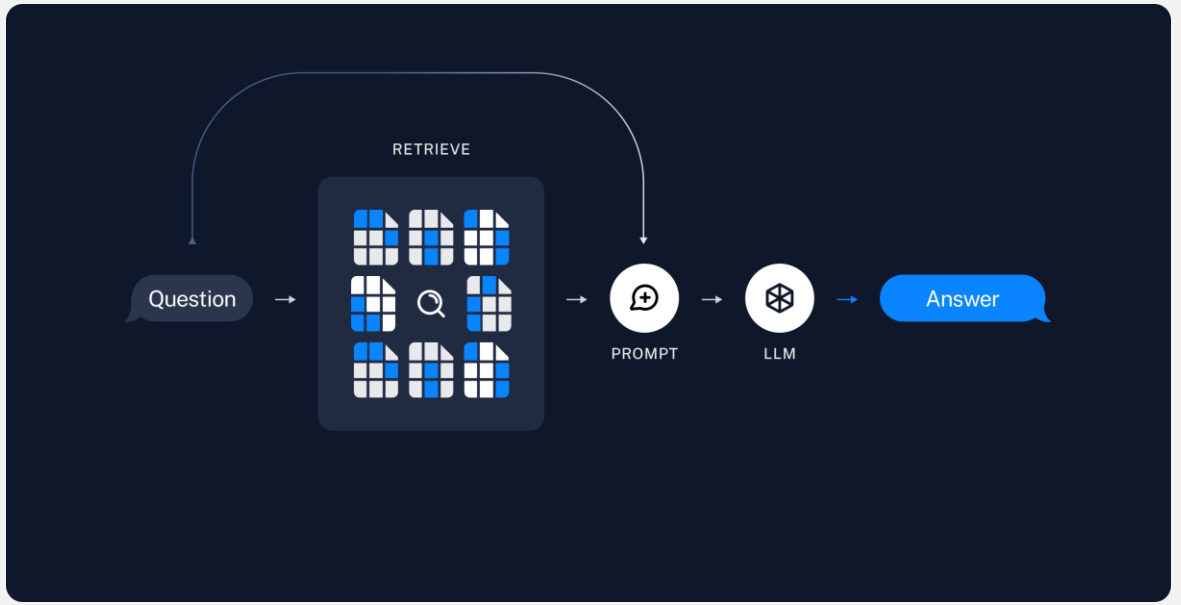
- 코드가 점점 길어지고 복잡해짐
- LLM 의 **일관되지 않은 답변**이 마치 나비효과로 이어져 **답변 품질저하**로 이어짐



Conventional RAG 문제점

- **사전에 정의**된 데이터 소싱(PDF, DB, Table 등) 자원
- **사전에 정의**된 Fixed Size Chunk
- **사전에 정의**된 Query 입력
- **사전에 정의**된 검색 방법
- **신뢰하기 어려운** LLM 혹은 Agent
- **고정된 프롬프트** 형식
- LLM의 답변 결과에 대한 **문서와의 관련성/신뢰성**



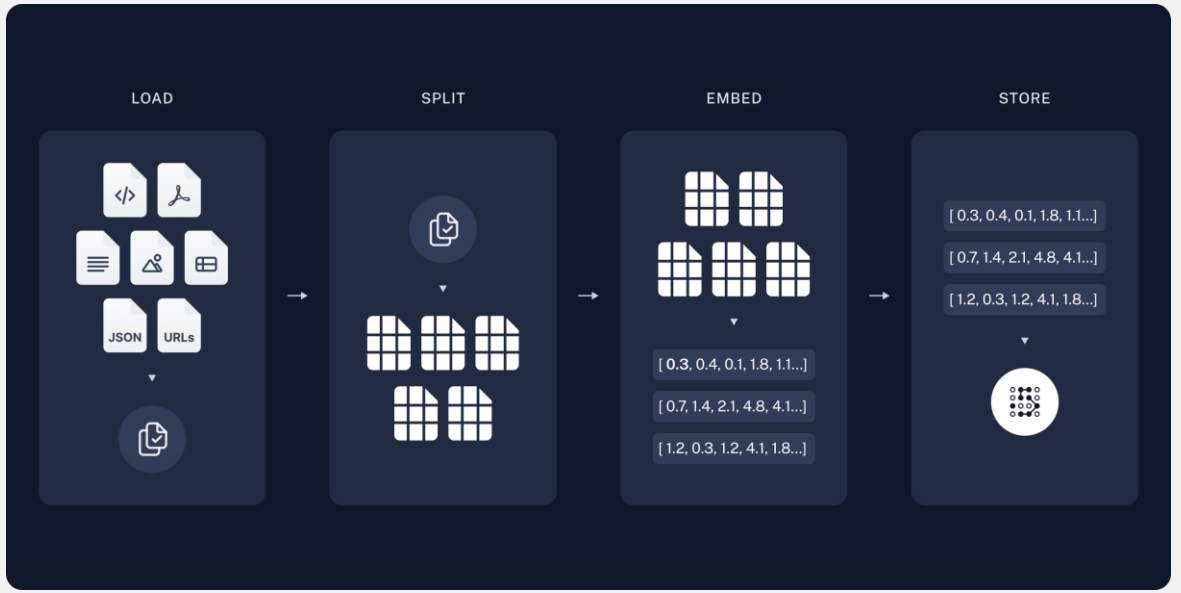


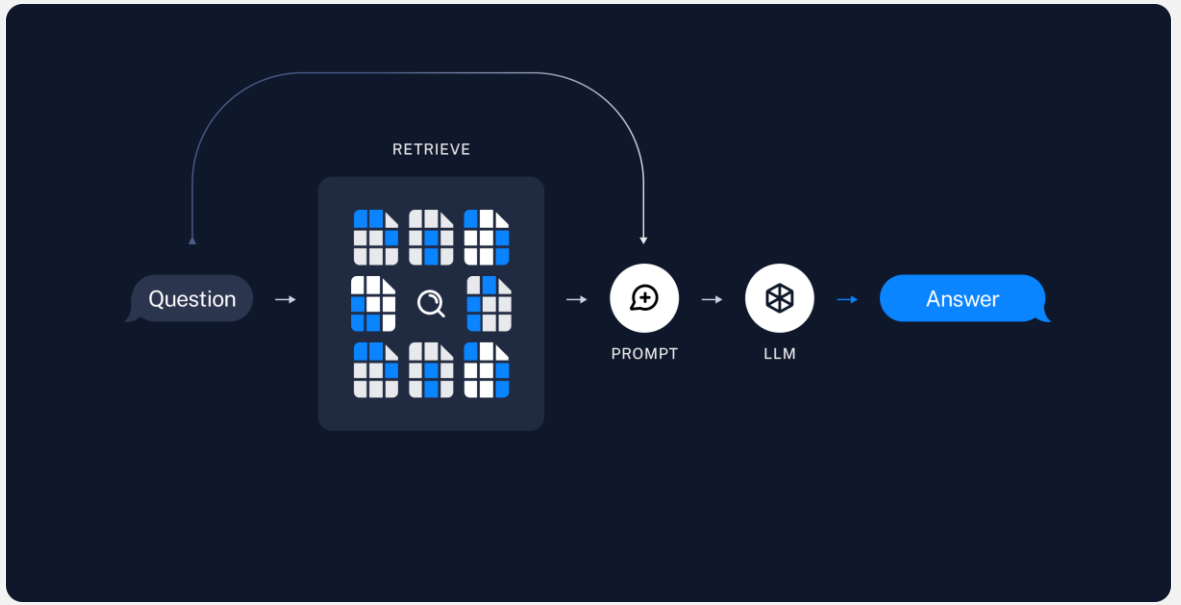
Conventional RAG 문제점

Document Loader(데이터로드) > Answer(답변)

RAG 파이프라인이 단방향 구조이기 때문

- 모든 단계를 한 번에 다 잘해야 함
- 이전 단계로 되돌아가기 어려움
  - 이전 과정의 결과물을 수정하기 어려움

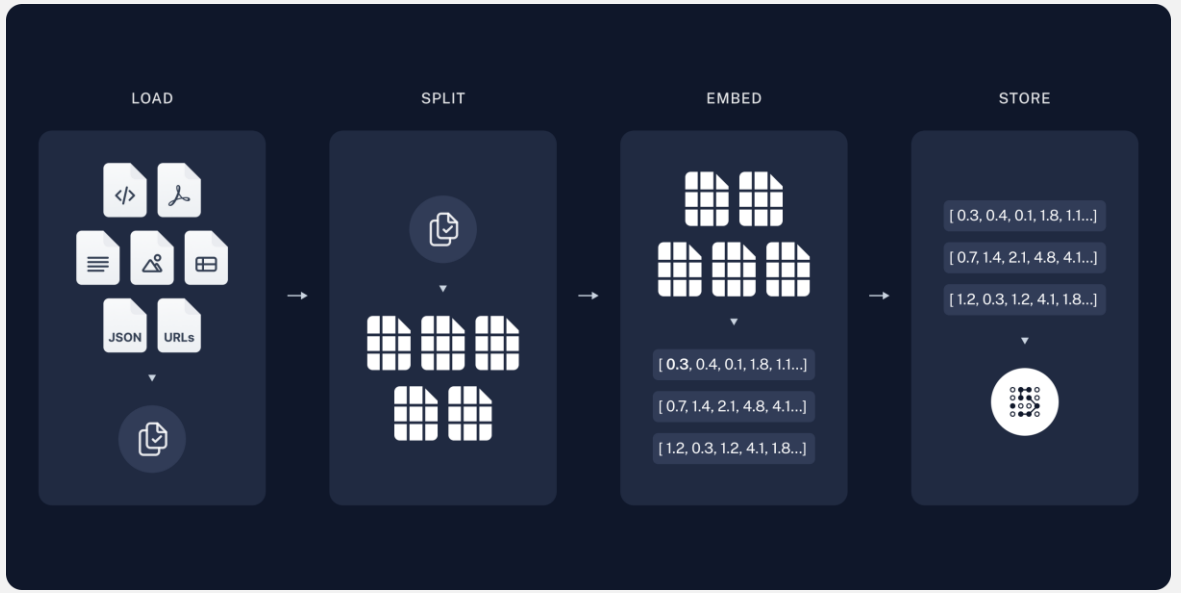


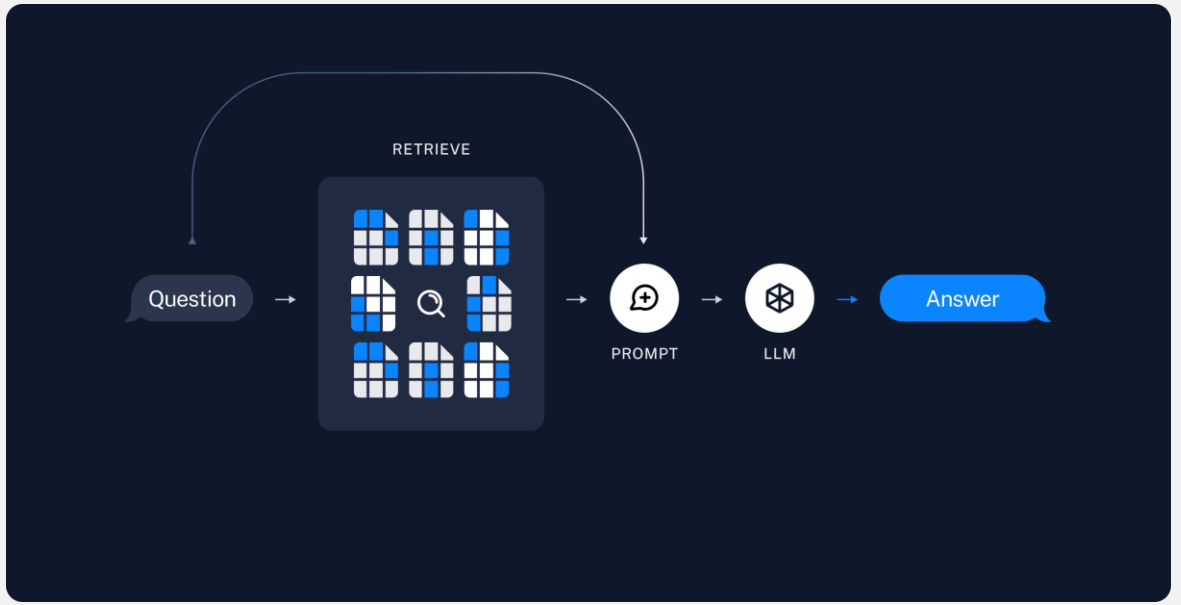


LangGraph 제안

- 각 세부과정을 **노드(Node)** 라고 정의
- 이전 노드 > 다음 노드: **엣지(Edge)** 연결
- **조건부 엣지** 를 통해 분기 처리

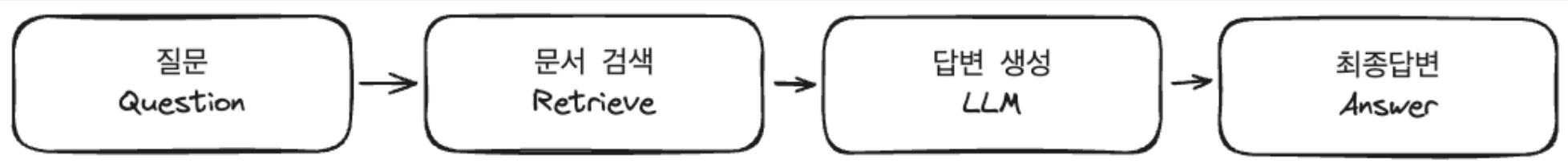
RAG 파이프라인을 보다 **유연하게** 설계

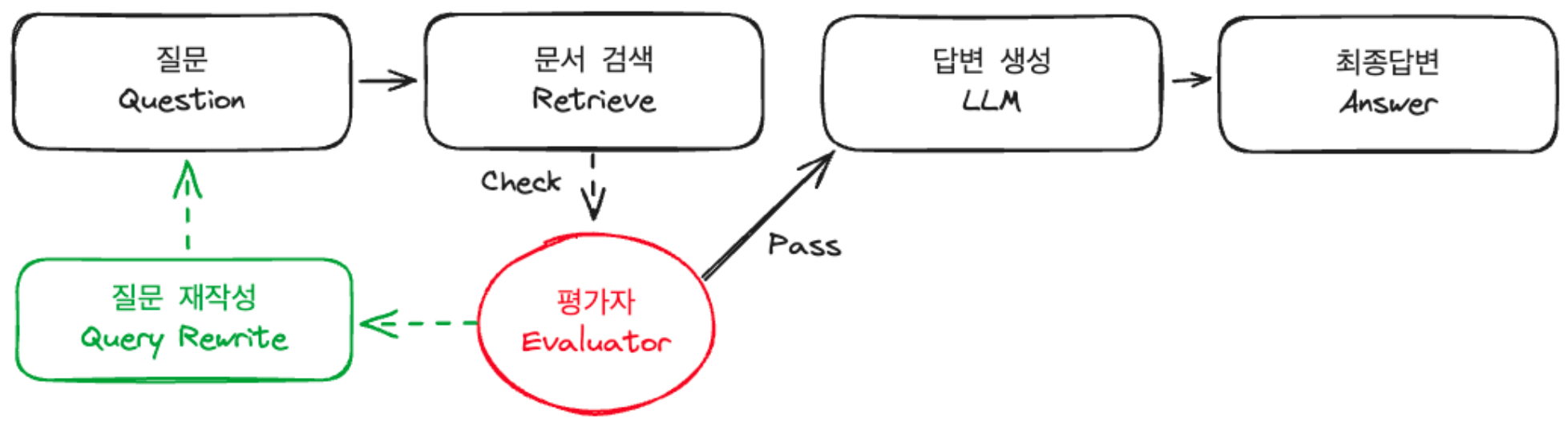




LangGraph 제안

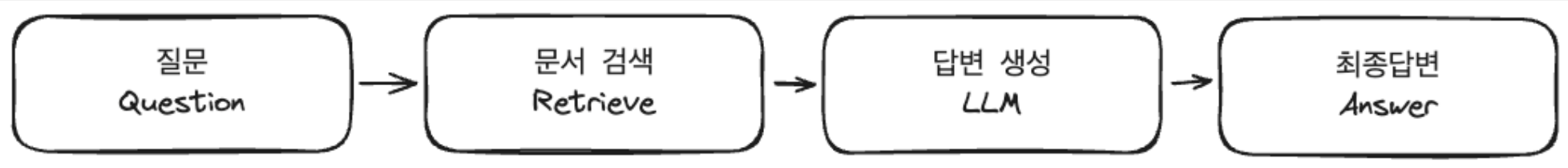
- 평가자 & Query Transform 추가

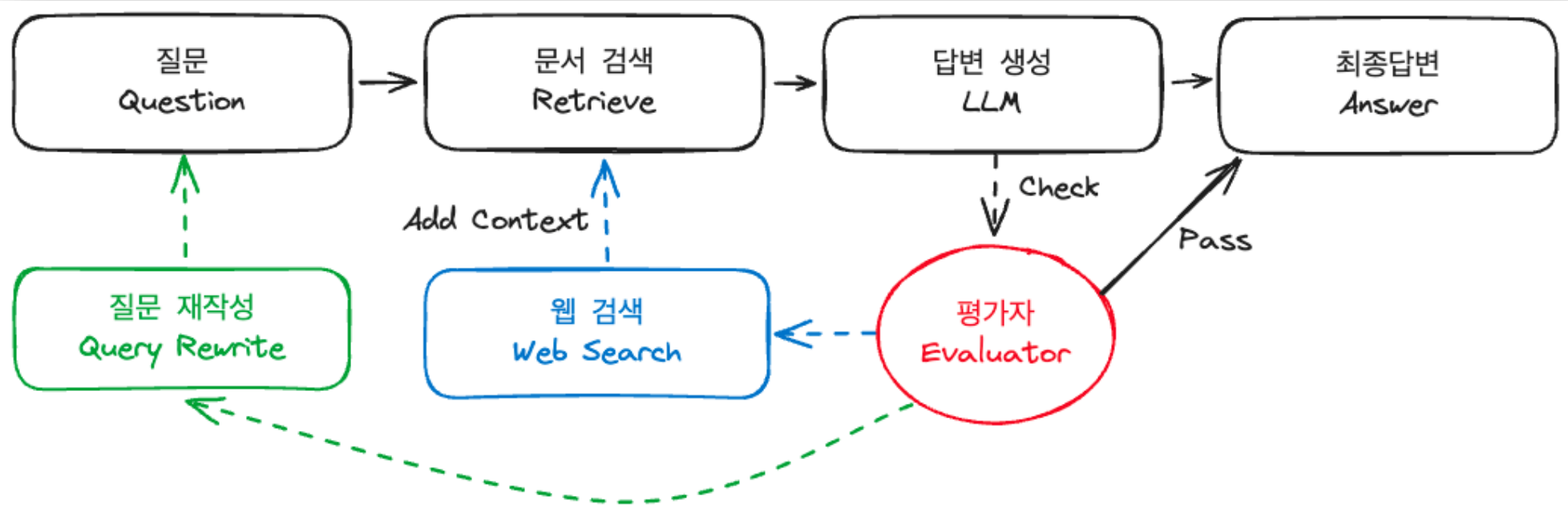




LangGraph 제안

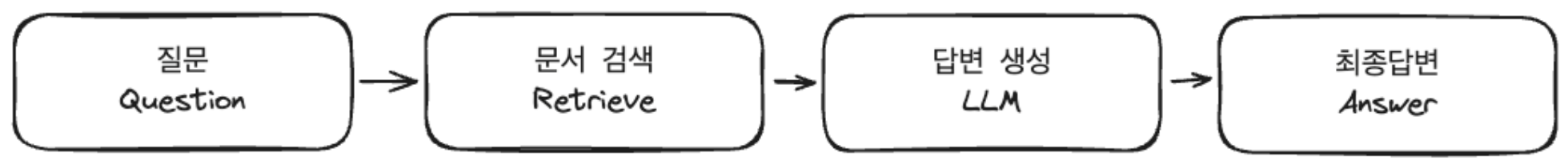
- 추가 검색기를 통하여 문맥(context) 보강

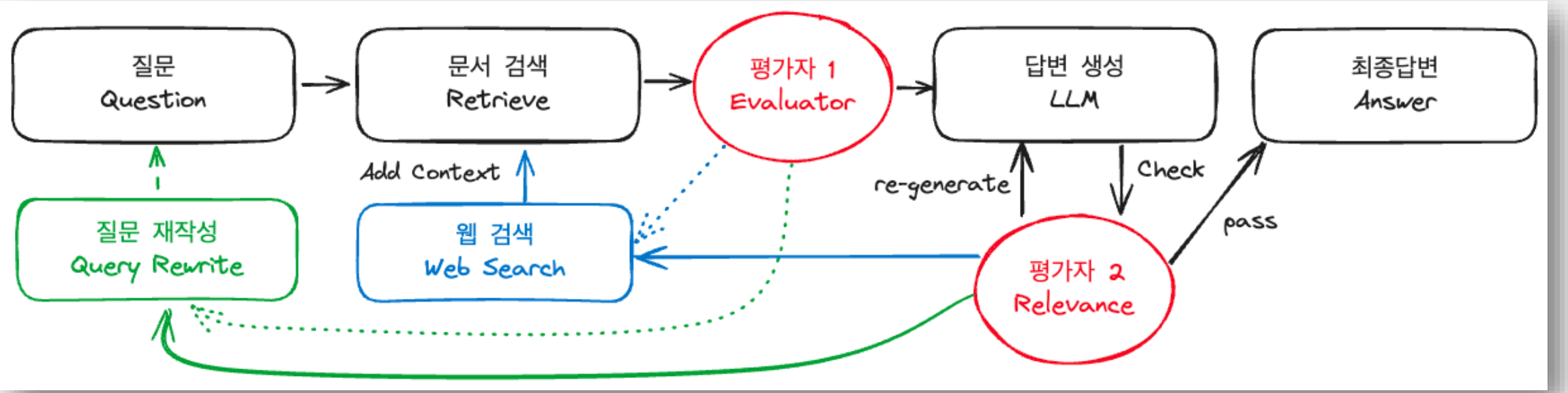




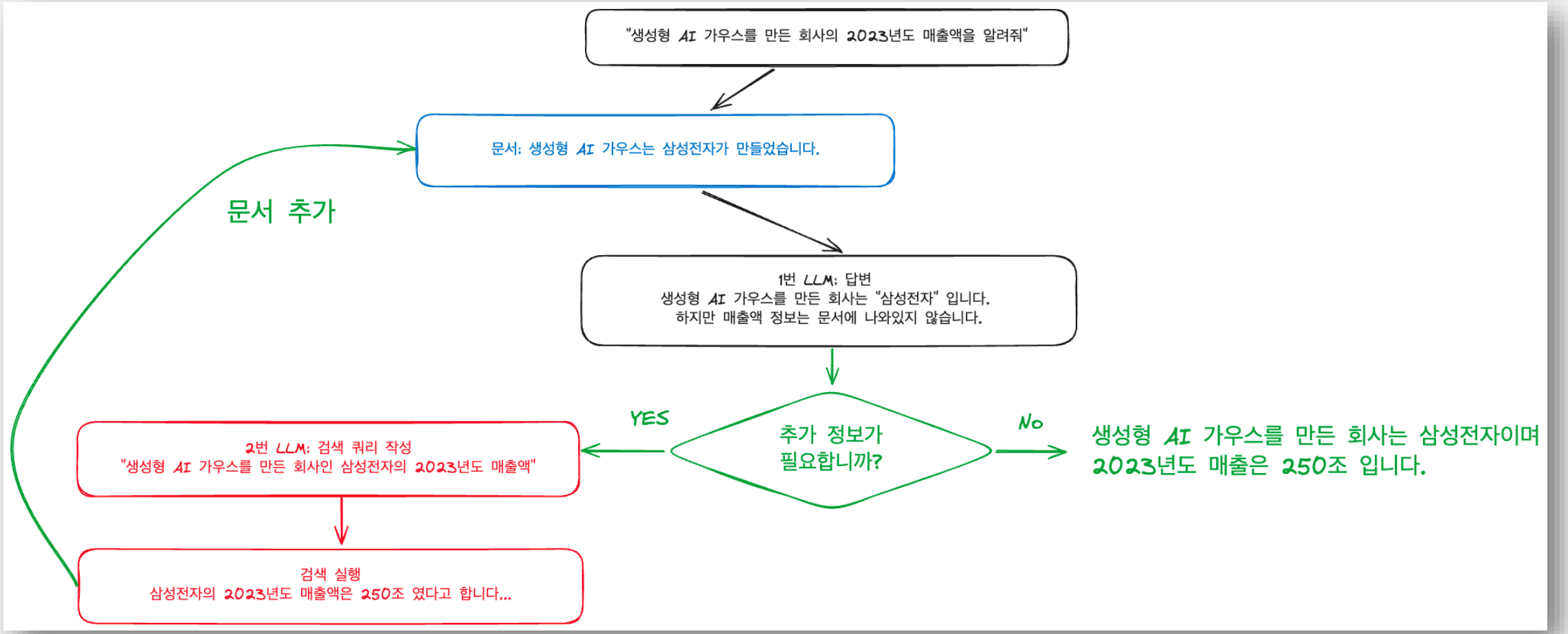
LangGraph 제안

- 문서-답변 간 관련성 여부를 판단하는 평가자2 를 추가하여 검증



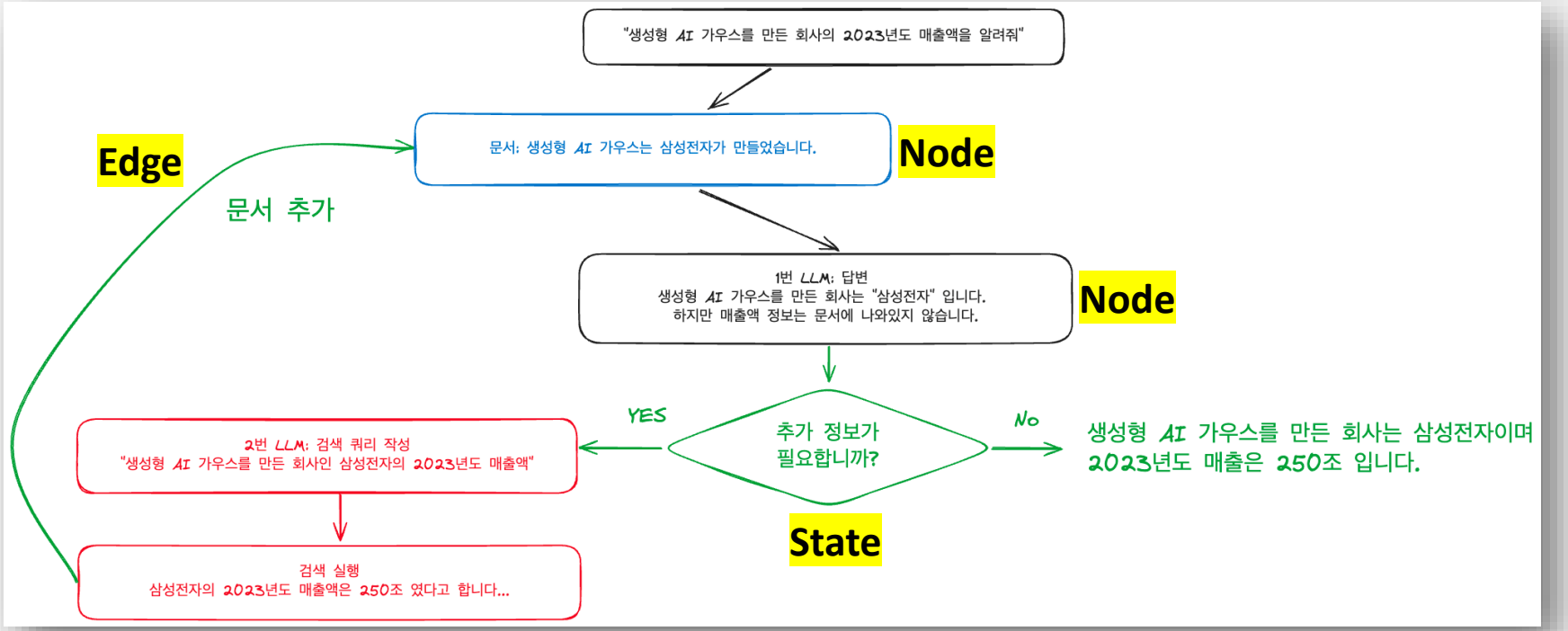


LangGraph 로 구현한 예시



LangGraph

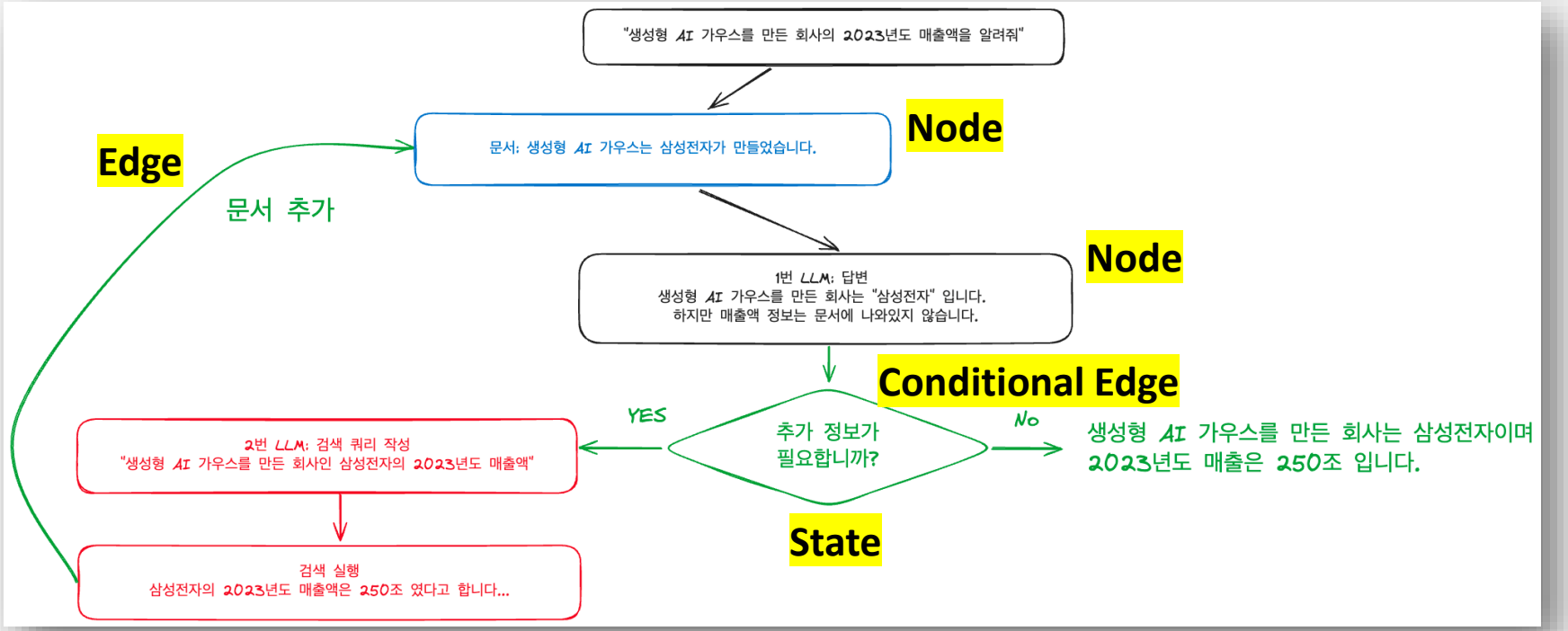
- **Node(노드), Edge(엣지), State(상태관리)** 를 통해 LLM 을 활용한 워크플로우에 **순환(Cycle) 연산 기능**을 추가하여 손쉽게 흐름을 제어.
- RAG 파이프라인의 **세부 단계별 흐름제어**가 가능
- **Conditional Edge**: **조건부** (if, elif, else 와 같은..) 흐름 제어
- **Human-in-the-loop**: 필요시 **중간 개입**하여 다음 단계를 결정
- **Checkpointer**: 과거 실행 과정에 대한 **“수정” & “리플레이”** 기능



LangGraph

주요 용어

- Node(노드): 어떤 작업(task) 을 수행할지 정의
- Edge(엣지): 다음으로 실행할 동작 정의
- State(상태): 현재의 상태 값을 저장 및 전달하는데 활용
- Conditional Edge(조건부 엣지): 조건에 따라 분기 처리



상태(State)

상태(State)

노드와 노드 간에 정보를 전달할 때 상태(State) 객체에 담아 전달합니다.

- **TypedDict**: 일반 파이썬 **dict** 에 **타입힌팅**을 추가한 개념이지만, 쉽게 **dictionary**로 생각하셔도 좋습니다.
- 모든 값을 다 채우지 않아도 됩니다.
- 새로운 노드에서 값을 덮어쓰기(Overwrite) 방식으로 채웁니다.
- Reducer (add_messages 혹은 operator.add): 자동으로 list 에 메시지를 추가해 주는 기능

```python
from typing import Annotated, TypedDict
from langgraph.graph.message import add_messages


# GraphState 상태를 저장하는 용도로 사용합니다.
class GraphState(TypedDict):
    question: Annotated[list, add_messages]  # 질문(Query Rewrite 누적)
    context: Annotated[str, "Context"]  # 문서의 검색 결과
    answer: Annotated[str, "Answer"]  # 답변
    messages: Annotated[list, add_messages]  # 메시지
    relevance: Annotated[str, "Relevance"]  # 관련성
```

상태(State)

Reducer

- Reducer (add_messages 혹은 operator.add): 자동으로 list 에 메시지를 추가해
  - ` left` (Messages): 기본 메시지 리스트
  - ` right` (Messages): 병합할 메시지 리스트 또는 단일 메시지

```python
from typing import Annotated, TypedDict
from langgraph.graph.message import add_messages


# GraphState 상태를 저장하는 용도로 사용합니다.
class GraphState(TypedDict):
    question: Annotated[list, add_messages]  # 질문(Query Rewrite 누적)
    context: Annotated[str, "Context"]  # 문서의 검색 결과
    answer: Annotated[str, "Answer"]  # 답변
    messages: Annotated[list, add_messages]  # 메시지
    relevance: Annotated[str, "Relevance"]  # 관련성
```

상태(State)

Reducer

- Reducer (add_messages 혹은 operator.add): 자동으로 list 에 메시지를 추가해
  - ` left` (Messages): 기본 메시지 리스트
  - ` right` (Messages): 병합할 메시지 리스트 또는 단일 메시지

```python
from langchain_core.messages import AIMessage, HumanMessage
from langgraph.graph import add_messages

# 기본 사용 예시
msgs1 = [HumanMessage(content="안녕하세요?", id="1")]
msgs2 = [AIMessage(content="반갑습니다~", id="2")]

result1 = add_messages(msgs1, msgs2)
print(result1)
```

✓ 0.3s

```
[HumanMessage(content='안녕하세요?', additional_kwargs={}, response_metadata={}, id='1'),
AIMessage(content='반갑습니다~', additional_kwargs={}, response_metadata={}, id='2')]
```

상태(State)

- 노드 별 상태 값의 변화

Key	Value
time	**1**
name	
llm	**GPT**

노드1

Key	Value
time	**2**
name	**테디**
llm	GPT

노드2

Key	Value
time	**3**
name	테디
llm	GPT

노드3

Key	Value
time	**4**
name	**셜리**
llm	GPT

노드4

- 각 노드에서 새롭게 업데이트 하는 값은 **기존 Key 값을 덮어쓰는** 방식
- 노드에서 **필요한 상태 값을 조회하여 동작에 활용**할 수 있음
- **노드4** 에서 **노드1**에서 반영된 llm 값이 **그대로 상태 전달**되어 조회가 가능

상태(State) - RAG 사례

- 노드 별 상태 값의 변화

1 Question

Key	Value
context	
question	**질문1**
answer	
score	

노드1

2 Retrieve

Key	Value
context	**문서1**
question	질문1
answer	
score	

노드2

3 Answer

Key	Value
context	문서1
question	질문1
answer	**답변1**
score	

노드3

4 Evaluate

Key	Value
context	문서1
question	질문1
answer	답변1
score	**BAD**

노드4

- 노드4에서 “문서” 에 대하여 답변 관련성 점수를 부여
- score 가 “BAD” 인 경우 (선택할 수 있는 행동 3가지)
  - 노드 1: 질문을 재작성 요청
  - 노드 2: 문서를 다시 검색 / 검색을 통한 정보 보완
  - 노드 3: 답변을 재작성 요청

상태(State) - RAG 사례

- 노드 1: **질문을 재작성** 요청

5 Question

Key	Value
context	문서1
question	**질문2**
answer	답변1
score	BAD1

6 Retrieve

Key	Value
context	**문서2**
question	질문2
answer	답변1
score	BAD1

7 Answer

Key	Value
context	문서2
question	질문2
answer	**답변2**
score	BAD1

8 Evaluate

Key	Value
context	문서2
question	질문2
answer	답변2
score	**GOOD**

노드1 **노드2** **노드3** **노드4**

Query Transform: 질문 재작성

상태(State) - RAG 사례

- 노드 2: **문서 검색을** 재요청

1 Question

Key	Value
context	
question	질문1
answer	
score	

노드1

5 Retrieve

Key	Value
context	**문서2**
question	질문1
answer	답변1
score	BAD

노드2

6 Answer

Key	Value
context	문서2
question	질문1
answer	**답변2**
score	BAD

노드3

6 Evaluate

Key	Value
context	문서2
question	질문1
answer	답변2
score	**GOOD**

노드4

Context Retrieval 조정
(Chunk, 다른 검색기, web search, ...)

상태(State) - RAG 사례

- 노드 3: **답변을 재생성** 요청

1 Question

Key	Value
context	
question	질문
answer	
score	

노드1

2 Retrieve

Key	Value
context	문서1
question	질문1
answer	
score	

노드2

5 Answer

Key	Value
context	문서1
question	질문1
answer	**답변2**
score	BAD

노드3

6 Evaluate

Key	Value
context	문서1
question	질문1
answer	답변2
score	**GOOD**

노드4

프롬프트 조정 &
다른 LLM 사용
GPT, Claude, Local

노드(Node) & 엣지(Edge)

노드(Node)

- 함수로 정의
- 입력인자: 상태(State) 객체
- 반환(return)
  - 대부분 상태(State) 객체
  - Conditional Edge 의 경우 다를 수 있음

```python
# 문서에서 검색하여 관련성 있는 문서를 찾습니다.
def retrieve_document(state: GraphState) -> GraphState:
    # Question 에 대한 문서 검색을 retriever 로 수행합니다.
    retrieved_docs = pdf_retriever.invoke(state["question"])
    # 검색된 문서를 context 키에 저장합니다.
    return GraphState(context=format_docs(retrieved_docs))
```

입력

Key	Value
context	
question	질문1
answer	
score	

동작을 위한 코드 구현

- Python 코드
- DB 조회
- API 호출
- …

출력

Key	Value
context	문서1
question	질문1
answer	
score	

Graph 생성 후 노드 추가

- 이전에 정의한 함수를 **Graph 에 추가**
- add_node(“노드이름”, 함수)

```python
from langgraph.graph import END, StateGraph
from langgraph.checkpoint.memory import MemorySaver

# langgraph.graph에서 StateGraph와 END를 가져옵니다.
workflow = StateGraph(GraphState)

# 노드들을 정의합니다.
workflow.add_node("retrieve", retrieve_document)  # 에이전트 노드를 추가합니다.
workflow.add_node("llm_answer", llm_answer)  # 정보 검색 노드를 추가합니다.
```

노드

```python
# 문서에서 검색하여 관련성 있는 문서를 찾습니다.
def retrieve_document(state: GraphState) -> GraphState:
    # Question 에 대한 문서 검색을 retriever 로 수행합니다.
    retrieved_docs = pdf_retriever.invoke(state["question"])
    # 검색된 문서를 context 키에 저장합니다.
    return GraphState(context=format_docs(retrieved_docs))
```

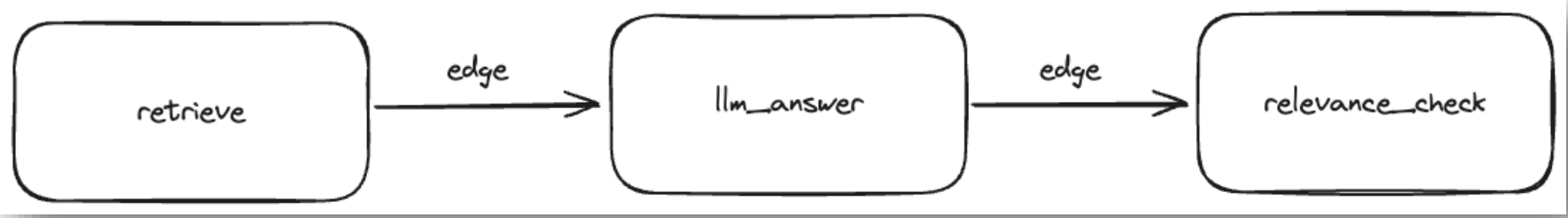
노드

```python
# Chain을 사용하여 답변을 생성합니다.
def llm_answer(state: GraphState) -> GraphState:
    return GraphState(
        answer=pdf_chain.invoke({"question": state["question"], "context": state["context"]}))
```

엣지(Edge)

- 노드에서 노드간의 연결
- add_edge(“노드이름”, “노드이름”)
  - from > to

```
# 각 노드들을 연결합니다.
workflow.add_edge("retrieve", "llm_answer")  # 검색 -> 답변
workflow.add_edge("llm_answer", "relevance_check")  # 답변 -> 관련성 체크
```



조건부 엣지(Conditional Edge)

- 노드에 조건부 엣지를 추가하여 분기를 수행할 수 있습니다.
- add_conditional_edges("노드이름", **조건부 판단 함수**, **dict 로 다음 단계 결정**)

흐름

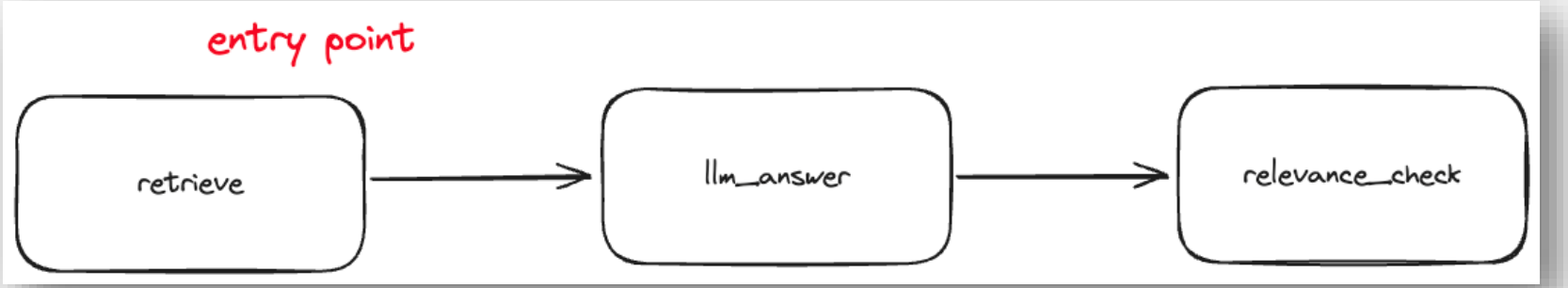
- **"relevance_check"** 노드에서 나온 결과를 **is_relevant 함수**에 입력
- 반환된 값은 **"grounded", "notGrounded", "notSure"** 중 하나
  - value 에 해당하는 값이 END 면 Graph 실행 종료
  - "llm_answer" 와 같이 노드이름이면 해당 노드로 연결

```
# 조건부 엣지를 추가합니다.
workflow.add_conditional_edges(
    "relevance_check",  # 관련성 체크 노드에서 나온 결과를 is_relevant 함수에 전달합니다.
    is_relevant,
    {
        "grounded": END,  # 관련성이 있으면 종료합니다.
        "notGrounded": "llm_answer",  # 관련성이 없으면 다시 답변을 생성합니다.
        "notSure": "llm_answer",  # 관련성 체크 결과가 모호하다면 다시 답변을 생성합니다.
    },
)
```

시작점 지정

- set_entry_point(“노드이름”)
- 지정한 시작점부터 Graph 가 시작

```
# 시작점을 설정합니다.
workflow.set_entry_point("retrieve")
```



그래프 생성 및 시각화

체크포인터(memory)

- Checkpointer: 각 노드간 실행결과를 추적하기 위한 메모리(대화에 대한 기록과 유사 개념)
- 체크포인터를 활용하여 특정 시점(Snapshot) 으로 되돌리기 기능도 가능
- compile(checkpointer=memory) 지정하여 그래프 생성

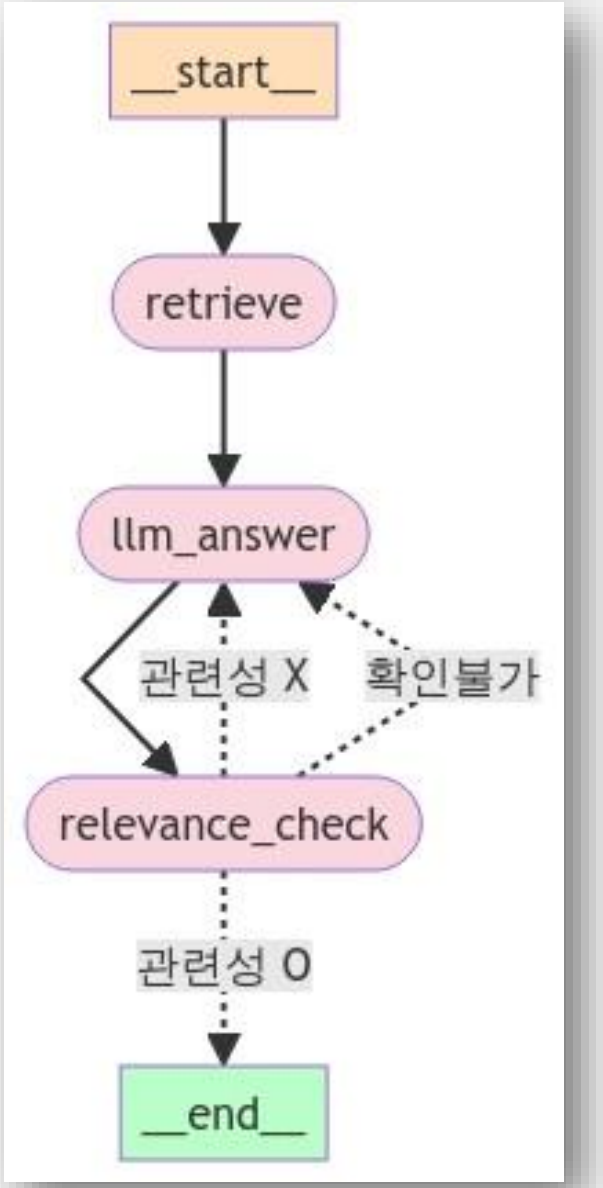
```
# 기록을 위한 메모리 저장소를 설정합니다.
memory = MemorySaver()

# 그래프를 컴파일합니다.
app = workflow.compile(checkpointer=memory)
```

그래프 시각화

- get_graph(xray=True).draw_mermaid_png()
  - 생성한 그래프 시각화

```
display(
    Image(app.get_graph(xray=True).draw_mermaid_png())
)  # 실행 가능한 객체의 그래프를 mermaid 형식의 PNG로 그려서 표시합니다.
```



실행 및 결과확인

그래프 실행

- RunnableConfig
  - recursion_limit: 최대 노드 실행 개수를 지정합니다. (13인 경우: 총 13개의 노드까지 실행합니다)
  - thread_id: 그래프 실행 아이디를 기록하고, 추후 추적하기 위한 목적으로 활용합니다.
- 상태(State) 로 시작합니다.
  - 여기서 “question” 에 질문만 입력하고 상태를 첫 번째 노드에게 전달 합니다.
- invoke(상태, config) 전달하여 실행합니다.

```python
from langchain_core.runnables import RunnableConfig

# recursion_limit: 최대 반복 횟수, thread_id: 실행 ID (구분용)
config = RunnableConfig(recursion_limit=13, configurable={"thread_id": "SELF-RAG"})

# GraphState 객체를 활
inputs = GraphState(question="삼성전자가 개발한 생성형 AI의 이름은?")
output = app.invoke(inputs, config=config)

# 출력 결과를 확인합니다.
print("Question: \t", output["question"])
print("Answer: \t", output["answer"])
print("Relevance: \t", output["relevance"])
```

(function) question: Any

결과 확인

- 출력된 결과로 최종 확인합니다.
- 출력 결과 역시 상태(State) 에 담겨 있습니다.

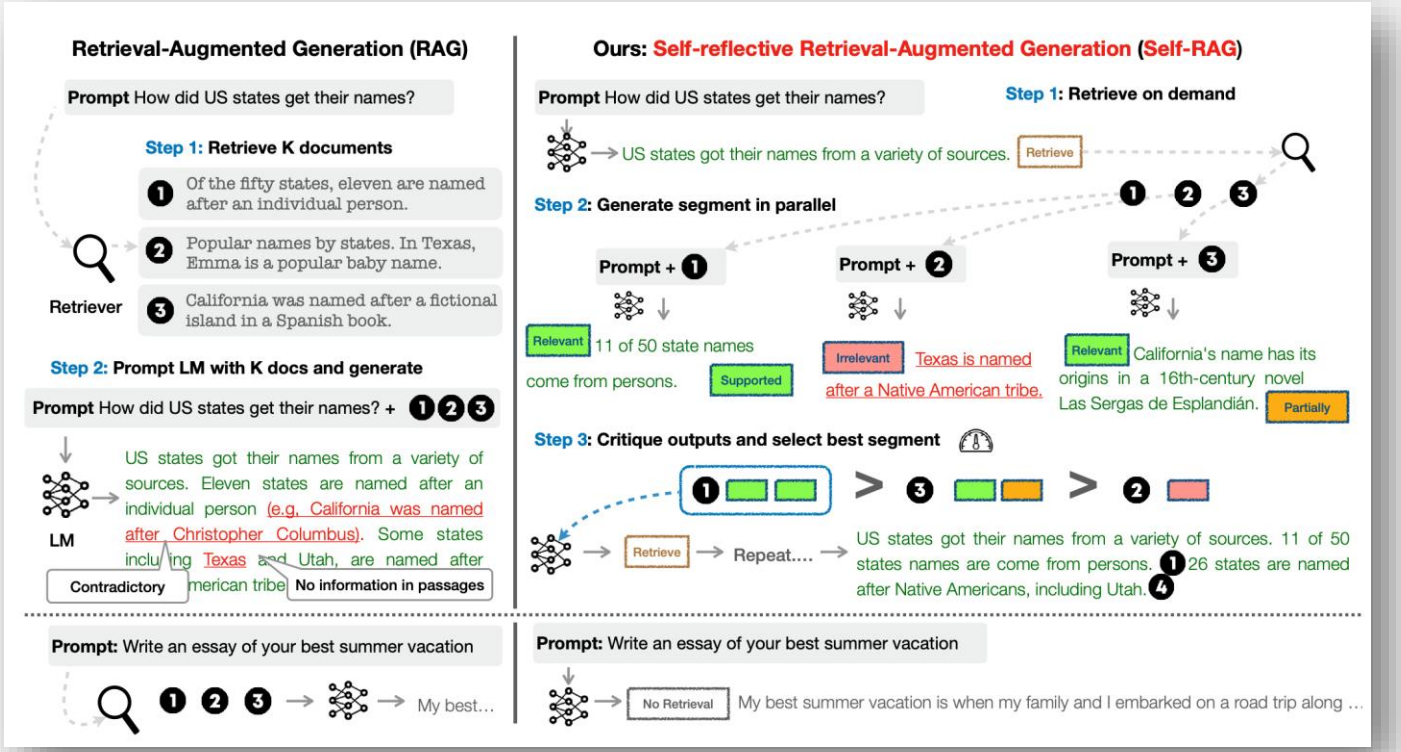
```python
# 출력 결과를 확인합니다.
print("Question: \t", output["question"])
print("Answer: \t", output["answer"])
print("Relevance: \t", output["relevance"])
```

```
Question:       삼성전자가 개발한 생성형 AI의 이름은?
Answer:         삼성전자가 개발한 생성형 AI의 이름은 '삼성 가우스'입니다. (출처: data/SPRI_AI_Brief_2023년12월호_F.pdf, 페이지 13)
Relevance:      grounded
```

Self-RAG

Self-RAG

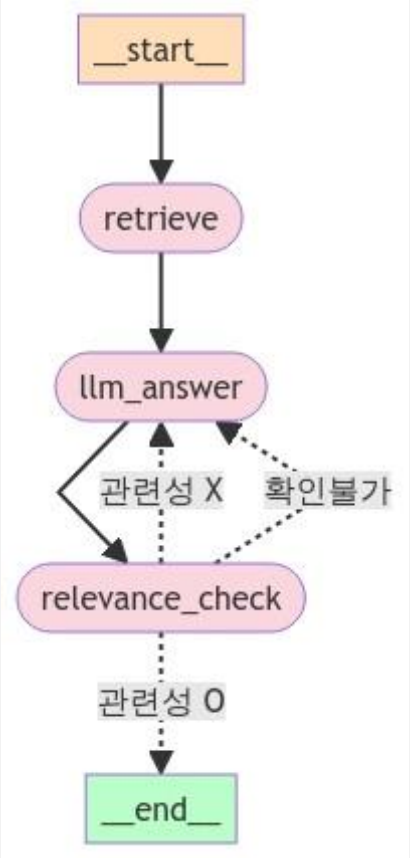
- https://arxiv.org/pdf/2310.11511
- 배경
  - **Fixed Size Retrieval**: 좋든 싫은 정해진 크기만큼 검색하여 가져오기 때문에, **검색에 노이즈**가 있을 수 있음
  - 무분별하게 주입되는 검색으로 인하여 **문서내 다른 정보를 참고**하거나, **제대로 된 답변이 나오지 않는 경우**가 있음
  - 검색된 정보의 정확성을 신뢰할 수 없음
- 제안
  - 선택적 Retrieval 을 도입 (필요한 만큼만 Retrieve)
  - Retrieval 로부터 답변을 도출
  - **도출된 답변**과 **Retieval Passage 간 관련성** 체크

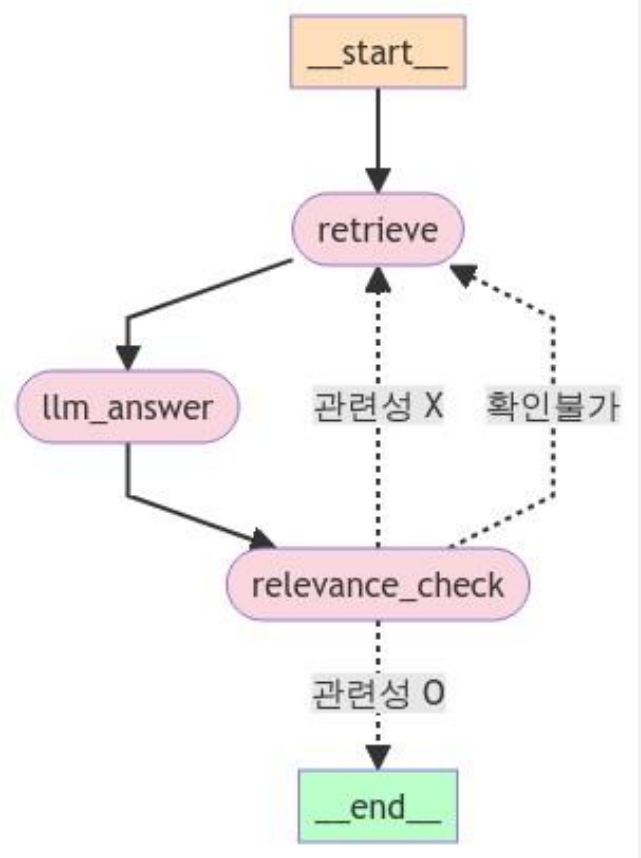


Self-RAG: 관련성 체크 흐름

참고 코드: 02-langgraph-groundcheck.ipynb

- 코드 한 줄로 흐름 제어
  - relevance_check > llm_answer
  - relevance_check > retrieve



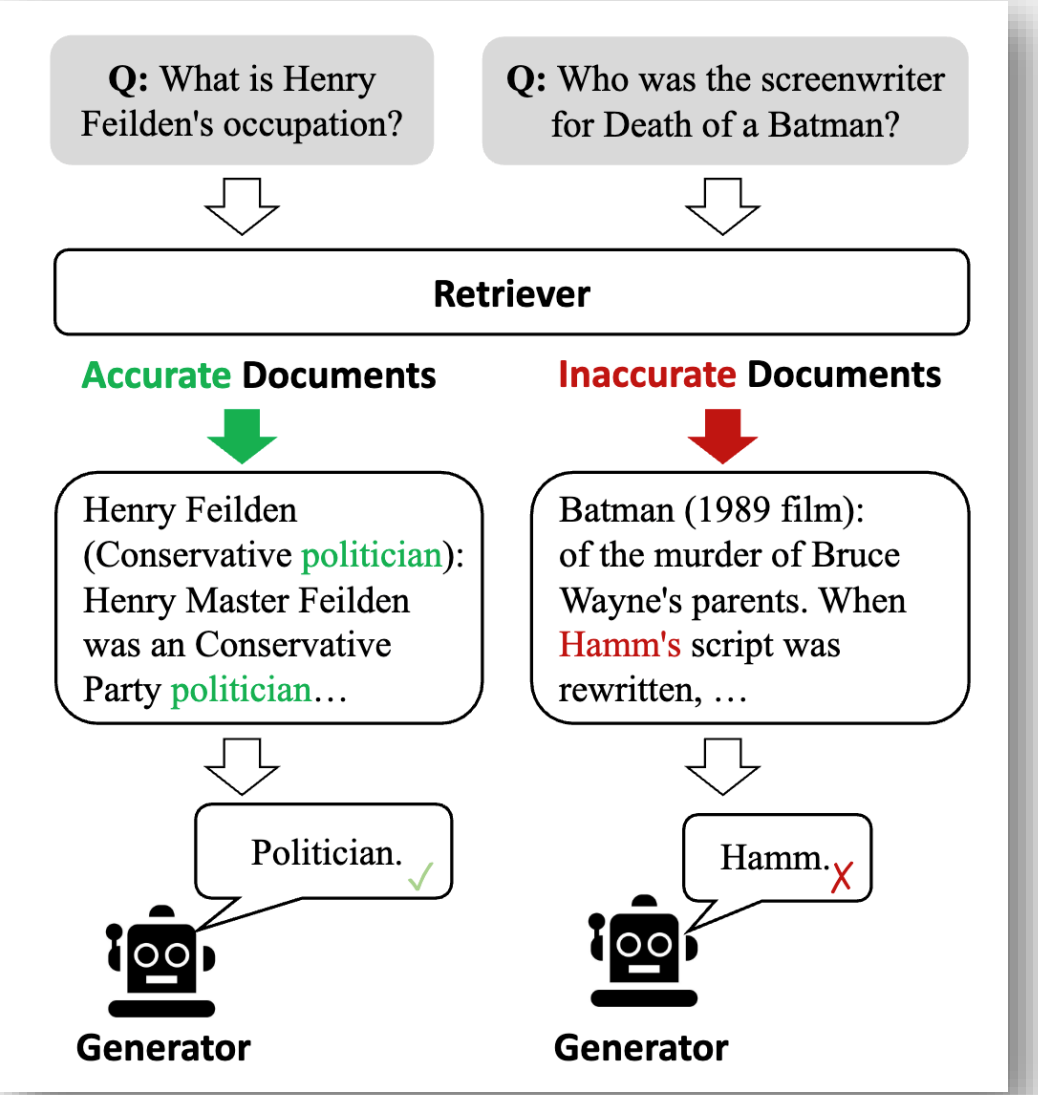


Corrective-RAG

Corrective RAG

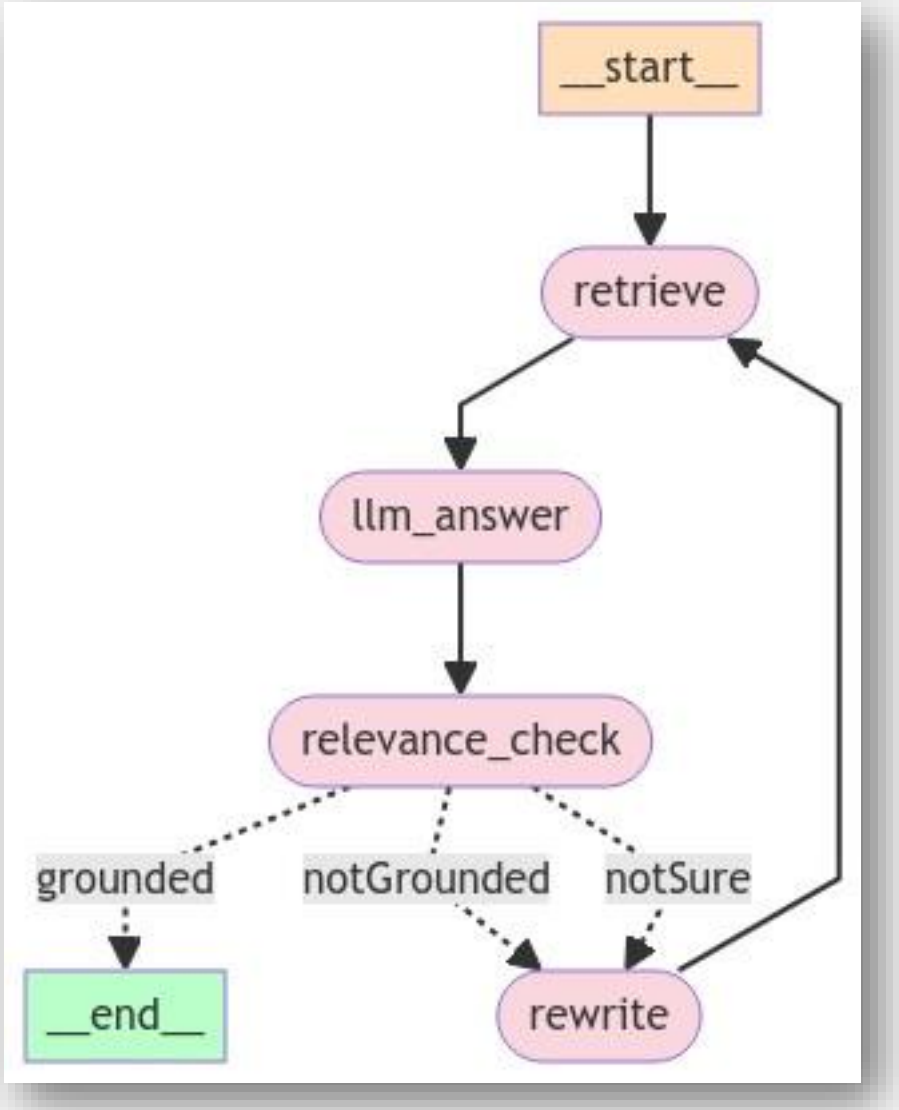
https://arxiv.org/pdf/2401.15884

- 배경
  - RAG의 답변 결과는 검색된 문서의 관련성에 크게 의존적
  - 따라서, 검색이 잘못된 경우 답변에 대한 품질 우려가 됨
- 제안
  - 사용자 입력 쿼리(query) 에 대하여 검색된 문서의 품질을 평가
  - 한마디로 검색된 결과가 사용자 입력 쿼리와 관련성이 높도록 쿼리를 수정(Corrective)



Corrective RAG

- 이전의 Relevance Check 를 수행 후 결과에 따라 쿼리를 조정(Corrective)
- 재작성된 쿼리로 다시 문서 검색 수행

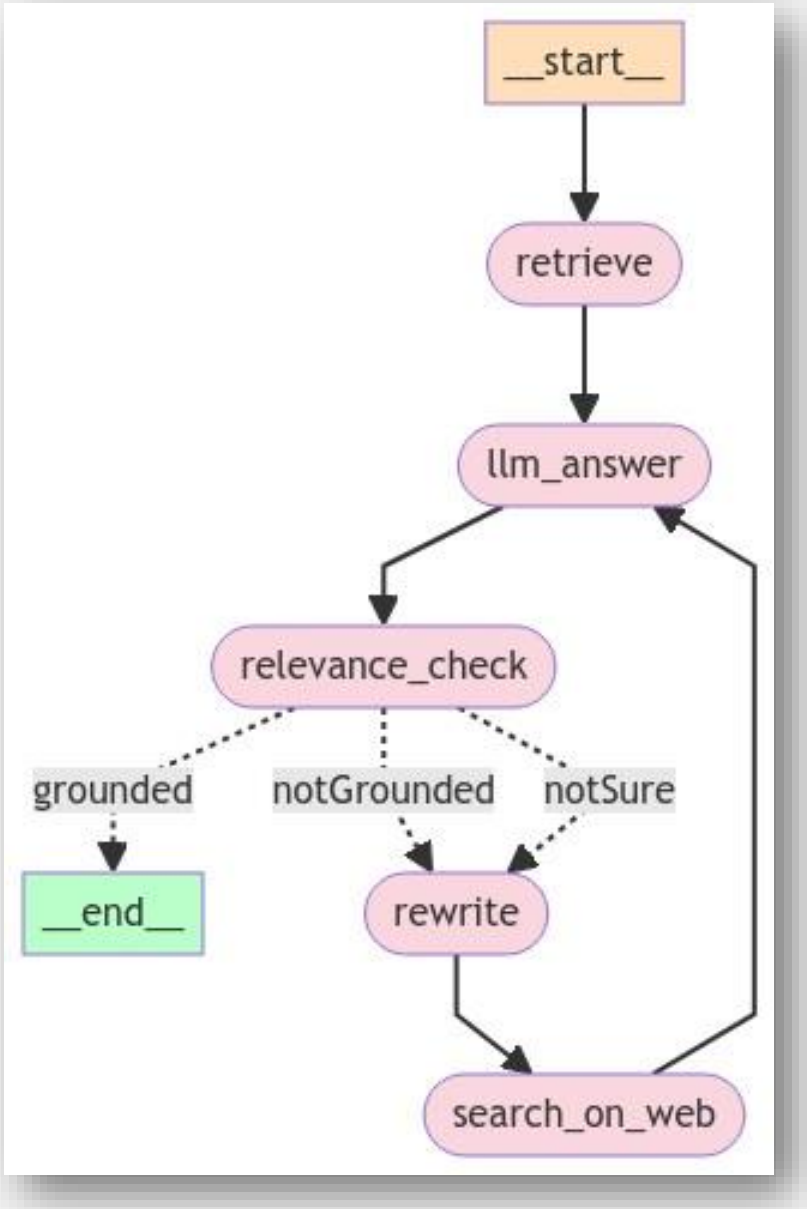


Corrective-RAG

Retrieve and Search

- **전통적인 RAG** 를 수행
- 검색된 문서에 답변에 필요한 **정보가 부족한 경우**

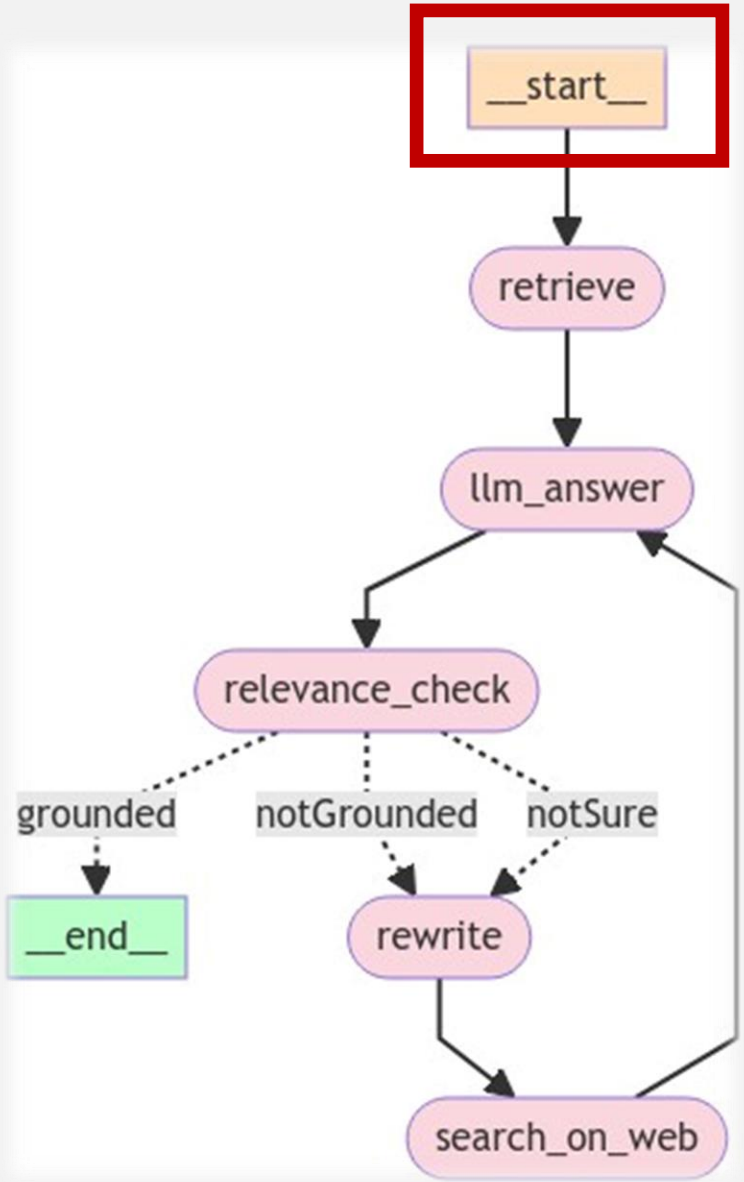
  “웹 검색”을 위한 쿼리 재작성
- “웹 검색” 으로 **보충된 정보로 답변 도출** 시도
- **새로운 답변**으로 관련성 체크 후 **재조정/종료** 진행



Retrieve and Search

- __start__: 질문 입력

"생성형 AI 가우스를 만든 회사의 2023년도 매출액은 얼마인가요?"



Retrieve and Search

- **retrieve: PDF 문서에 대한 검색 수행**
- **"생성형 AI 가우스를 만든 회사의 2023년도 매출액은 얼마인가요?"**

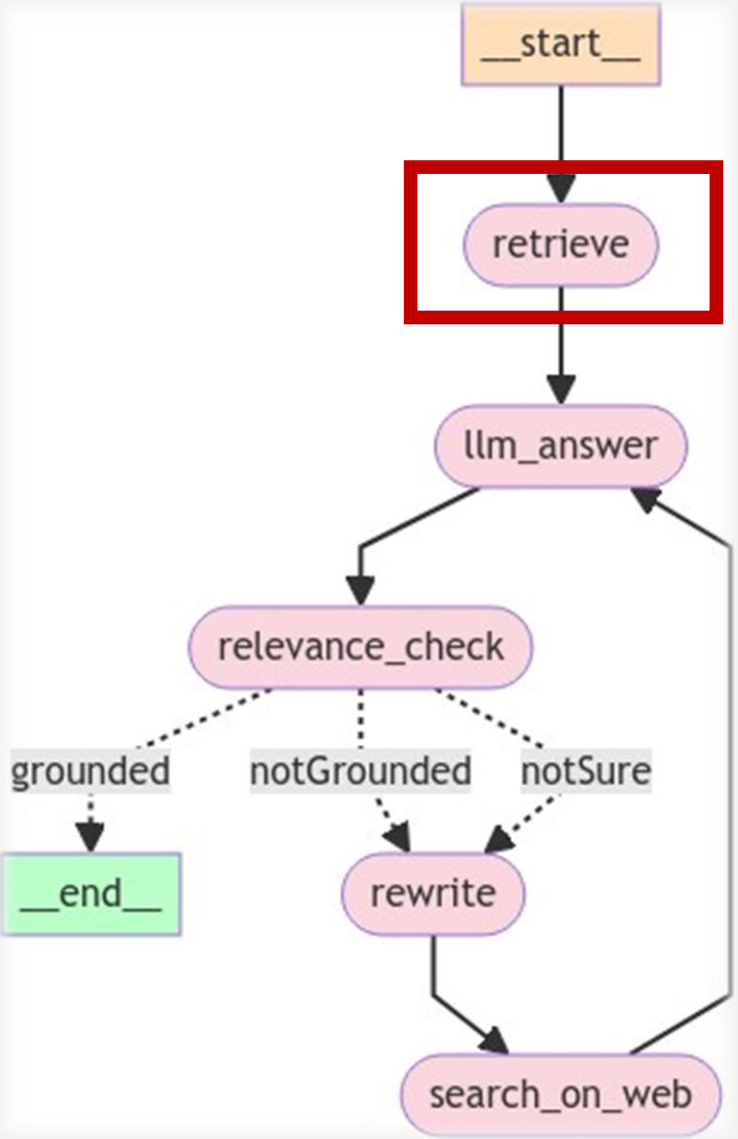
삼성전자, 자체 개발 생성 AI '삼성 가우스' 공개

KEY Contents

- 삼성전자가 온디바이스에서 작동 가능하며 언어, 코드, 이미지의 3개 모델로 구성된 자체 개발 생성 AI 모델 '삼성 가우스'를 공개
- 삼성전자는 삼성 가우스를 다양한 제품에 단계적으로 탑재할 계획으로, 온디바이스 작동이 가능한 삼성 가우스는 외부로 사용자 정보가 유출될 위험이 없다는 장점을 보유

◎ 언어, 코드, 이미지의 3개 모델로 구성된 삼성 가우스, 온디바이스 작동 지원

- 삼성전자가 2023년 11월 8일 열린 '삼성 AI 포럼 2023' 행사에서 자체 개발한 생성 AI 모델 '삼성 가우스'를 최초 공개
  - 정규분포 이론을 정립한 천재 수학자 가우스(Gauss)의 이름을 본뜬 삼성 가우스는 다양한 상황에 최적화된 크기의 모델 선택이 가능
  - 삼성 가우스는 라이선스나 개인정보를 침해하지 않는 안전한 데이터를 통해 학습되었으며, 온디바이스에서 작동하도록 설계되어 외부로 사용자의 정보가 유출되지 않는 장점을 보유
  - 삼성전자는 삼성 가우스를 활용한 온디바이스 AI 기술도 소개했으며, 생성 AI 모델을 다양한 제품에 단계적으로 탑재할 계획



Retrieve and Search

- **llm_answer: 문서 기반 답변 도출 (GPT)**
- “생성형 AI 가우스를 만든 회사의 **2023년도 매출액**은 얼마인가요?”

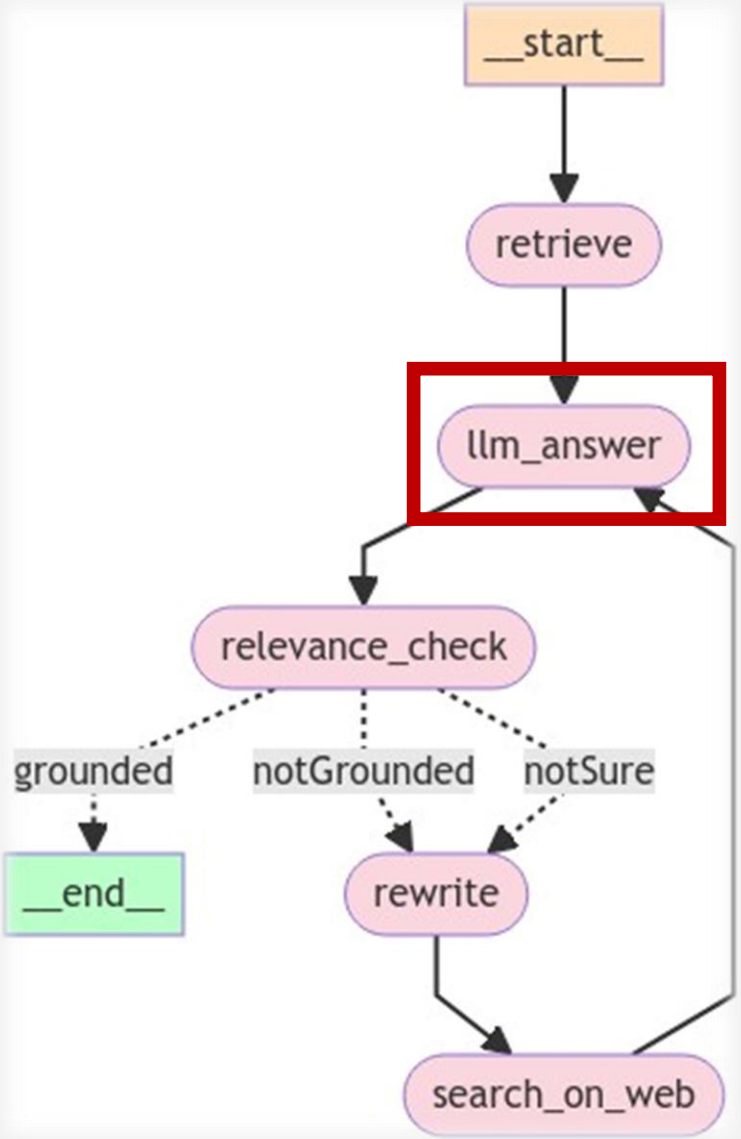
답변

생성형 AI **'가우스'**를 만든 회사는 **삼성전자**입니다.

그러나 제공된 문맥에서는 **삼성전자의 2023년도 매출액에 대한**

정보는 언급되어 있지 않습니다.

따라서 삼성전자의 2023년도 매출액에 대한 정보를 제공할 수 없습니다.



Retrieve and Search

- **relevance_check: 관련성/유효성 체크**
- **“생성형 AI 가우스를 만든 회사의 2023년도 매출액은 얼마인가요?”**

답변

생성형 AI **'가우스'**를 만든 회사는 **삼성전자**입니다.

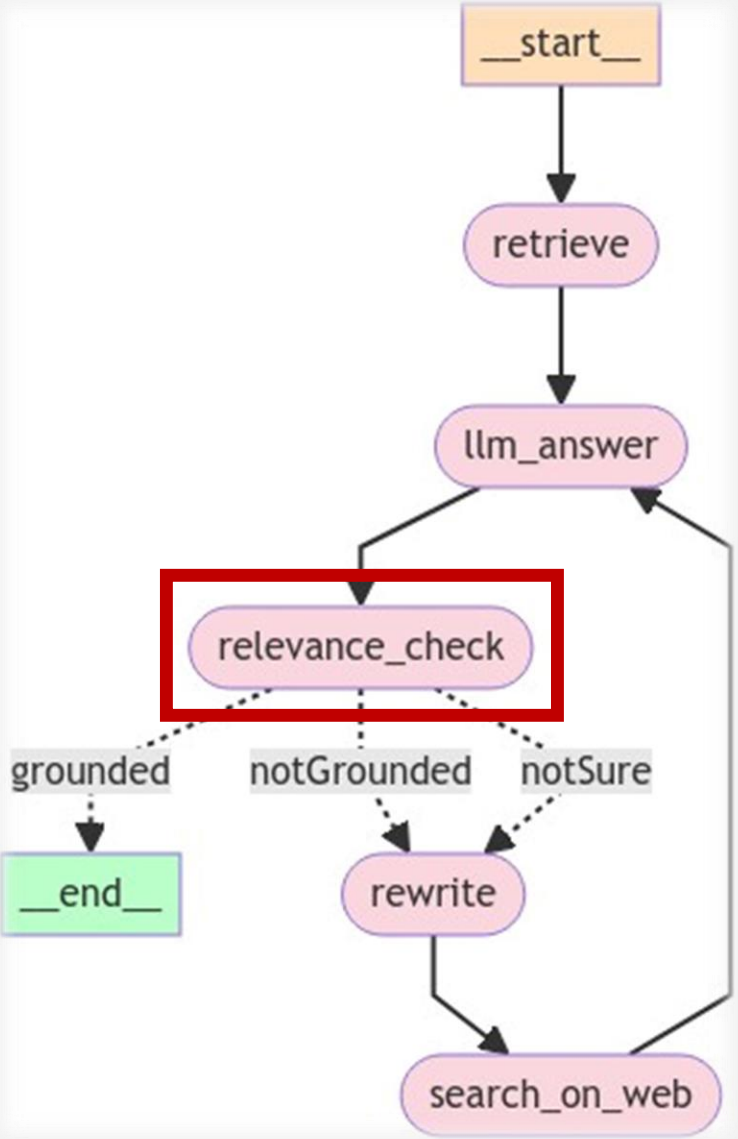
그러나 제공된 문맥에서는 **삼성전자의 2023년도 매출액에 대한**

정보는 언급되어 있지 않습니다.

따라서 삼성전자의 2023년도 매출액에 대한 정보를 제공할 수 없습니다.

이전 답변의 관련성/유효성 체크

“notGrounded”



Retrieve and Search

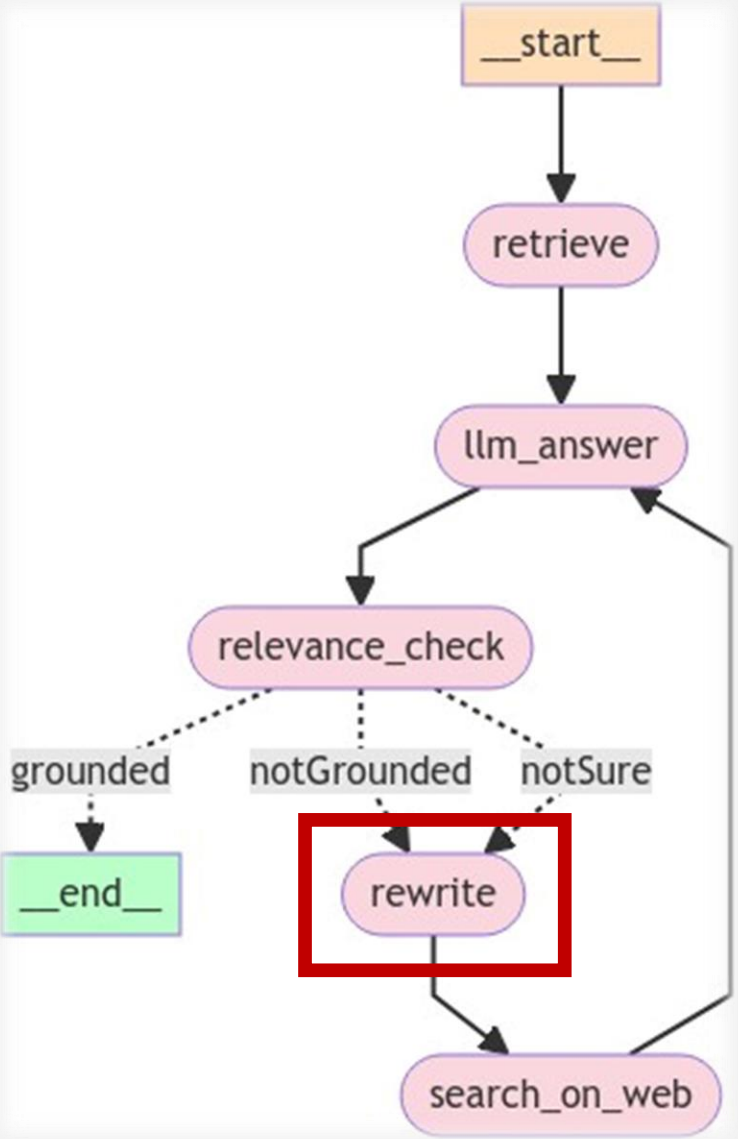
- **rewrite: 추가정보 검색을 위한 쿼리 재작성**
- **“생성형 AI 가우스를 만든 회사의 2023년도 매출액은 얼마인가요?”**

rewrite: 쿼리 재작성

“Rewrite the question to get **additional information to get the answer**”

결과

“삼성전자의 2023년도 매출액은 얼마인가요?”

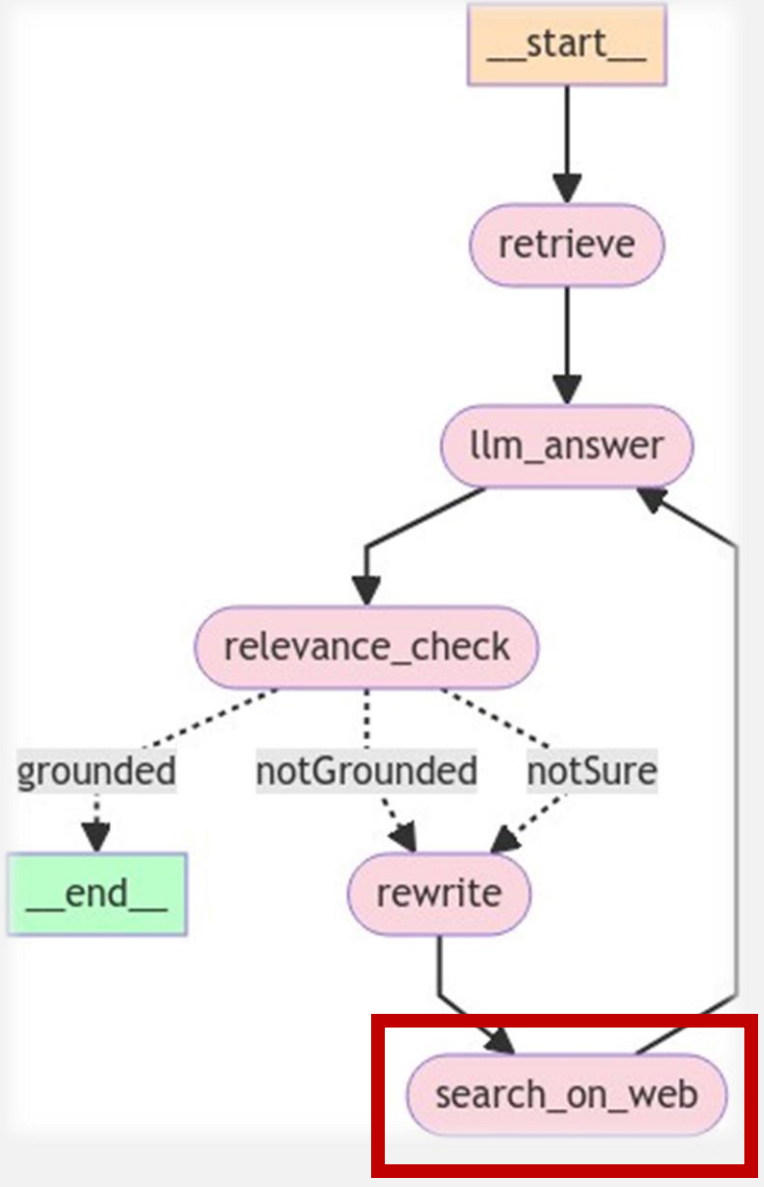


Retrieve and Search

- **search_on_web: 재작성된 쿼리로 검색 수행**
- **"삼성전자의 2023년도 매출액은 얼마인가요?"**

검색 결과

```
'웹 알림 동의 (크롬브라우저만 가능)\n'
'속보\n'
'삼성전자 2023년 매출 258조·영업이익 6.5조\n'
'4분기 잠정 실적...매출 67조·영업익 2.8조\n'
'서울 서초구 삼성전자 서초사옥에서 삼성 깃발이 바람에 휘날리고 있다. 이 기사와 관련된 기사\n'
'당신이 관심 있을만한 이슈\n'
'경제 많이 본 뉴스\n'
'LIVE ISSUE\n'
'댓글0\n'
'댓글 신고\n'
```



Retrieve and Search

- **llm_answer: 문서 기반 답변 도출 (GPT)**

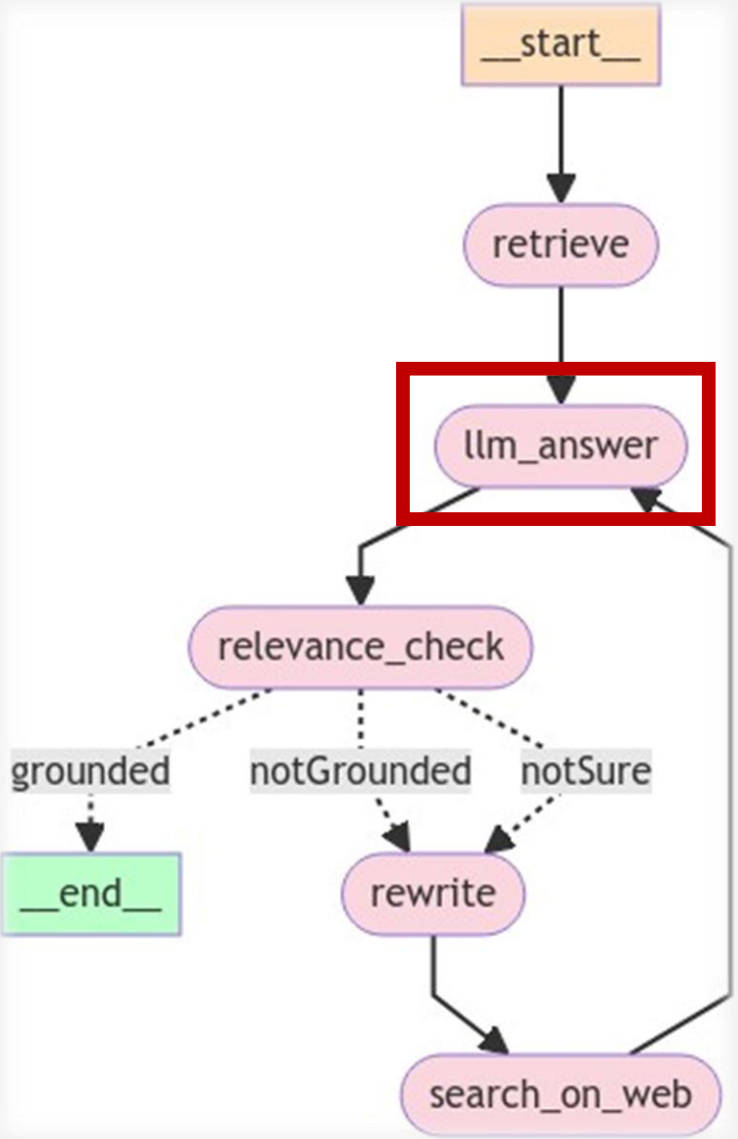
"삼성전자의 2023년도 매출액은 얼마인가요?"

```
'웹 알림 동의 (크롬브라우저만 가능)\n'
'속보\n'
'삼성전자 2023년 매출 258조·영업이익 6.5조\n'
'4분기 잠정 실적...매출 67조·영업익 2.8조\n'
'서울 서초구 삼성전자 서초사옥에서 삼성 깃발이 바람에 휘날리고 있다. 이 기사와 관련된 기사\n'
'당신이 관심 있을만한 이슈\n'
'경제 많이 본 뉴스\n'
'LIVE ISSUE\n'
'댓글0\n'
'댓글 신고\n'
```

답변

삼성전자의 **2023년도 매출액은 258.94조원**입니다.

(출처: https://news.samsung.com/kr/삼성전자-2023년-4분기-실적-발표)



Retrieve and Search

- **relevance_check: 관련성/유효성 체크**

“삼성전자의 2023년도 매출액은 얼마인가요?”

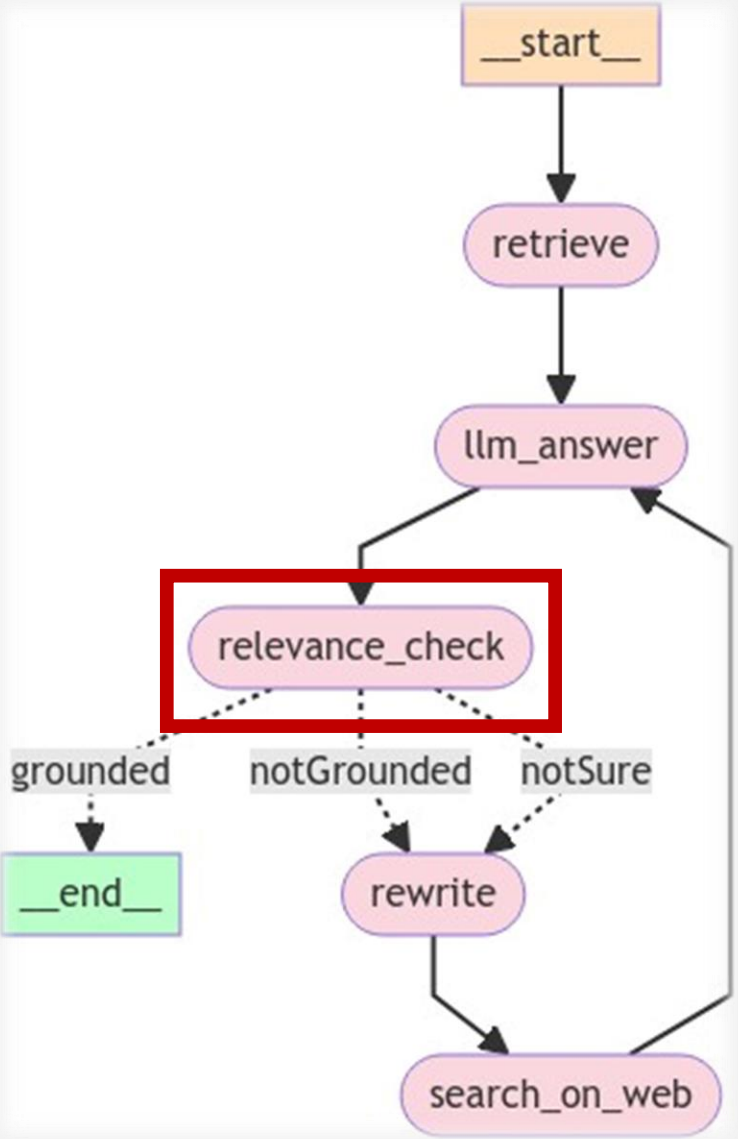
답변

삼성전자의 2023년도 매출액은 258.94조원입니다.

(출처: ' 'https://news.samsung.com/kr/삼성전자-2023년-4분기-실적-발표)

이전 답변의 관련성/유효성 체크

“grounded”



Retrieve and Search

- **__end__: 종료**

질문

“생성형 AI 가우스를 만든 회사의 2023년도 매출액은 얼마인가요?”

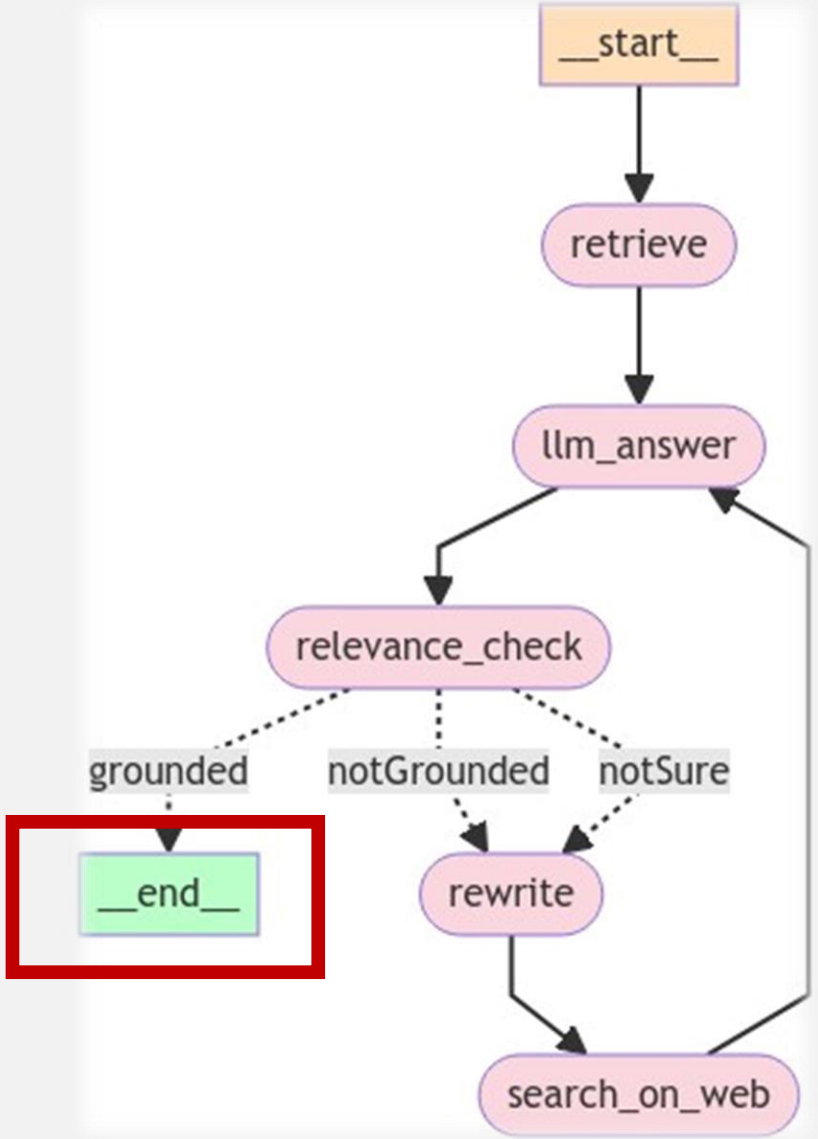
답변1

삼성전자의 2023년도 매출액은 258.94조원입니다.

답변2

생성형 AI “가우스” 를 만든 회사는 삼성전자이며,

2023년도 매출액은 258.94조원입니다



Retrieve and Search

- 랭스미스 추적
  - https://smith.langchain.com/public/ce7b4ea8-df34-4bb6-850e-afc8d12e75e6/r