

GEN AI 인텐시브 과정

강사장철원

Section 0

코스소개

DAY1

DAY2

DAY3

DAY4

DAY5

DAY6

DAY7

DAY8

LLM
Basic
Concept

Transformers
paper
review

Transformers
LangChain
LangGraph

LLM
service
develop

Final Project

□ LangGraph 기초

□ LangGraph + LangChain

GEN AI 인텐시브 과정

Section 1. LangGraph

Section 1-1. LangGraph의 개념

LangGraph

- LangChain 팀이 만든 LLM 기반 에이전트 워크플로우 프레임워크

복잡한 LLM 시스템을 그래프 기반으로 정의할 수 있게 해주는 도구

if , LangGraph 가

LangGraph의 구성 요소

- 노드(Node) = 하나의 처리 단계(예, 질문 분석, 검색, 요약 등)
- 엣지(Edge) = 조건에 따라 다음 단계로 이동
- 상태(State) = LLM의 기억이나 진행 상황을 계속 유지

LangChain만으로 충분하지 않나요?

- LangChain은 대부분 단순 순차적 흐름 사용
 - 복잡한 조건 분기 / 반복 / 상태 추적 / 도구 선택 / 다단계 추론 구현이 어려움
 - Ex) "정보가 부족하니 검색 다시해봐 " 와 같은 반복 루프 처리 어려움
- 따라서 Graph 구조를 사용하는 LangGraph 필요

LangGraph의 특징

- stateful 에이전트 구성 가능(모든 노드 상태에 전달됨)
- 명시적 흐름 제어 가능(분기, 반복, 종료 조건 등 쉽게 구현 가능)
- LangChain과 호환 가능

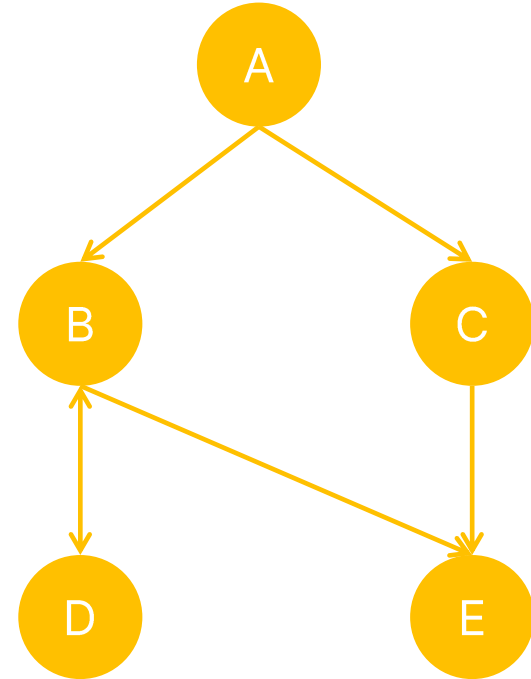
LangChain vs LangGraph



LangChain



LangGraph



transformers vs langchain vs langgraph

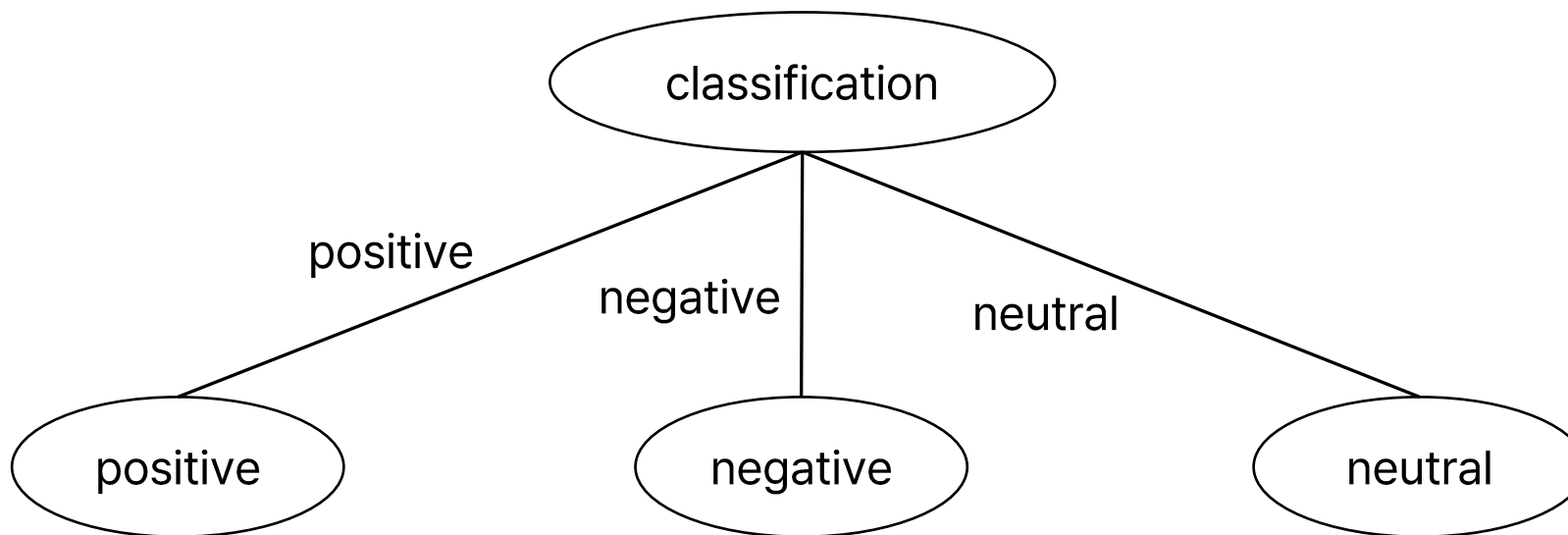
항목	transformers	langchain	langgraph
역할	LLM 모델 활용	LLM을 활용한 앱, 시스템 구축	복잡한 상태 추론/흐름 관리
개발 수준	낮은 수준(Low level)	높은 수준(High level)	Workflow-level
기능	모델 불러오기, 토크나이징, 추론	프롬프트 설계, 문서 검색, 멀티 스텝 추론, 메모리 등	분기/루프 포함 복잡한 Agent 구조 설계
비유	자동차 엔진	자동차	자율주행 자동차

GEN AI 인텐시브 과정

Section 1. LangGraph

Section 1-2. LangGraph 헬로 월드

실습 내용



라이브러리 불러오기

```
from langgraph.graph import StateGraph, END
```

상태 기반의 대화 흐름 구성에 사용
(상태는 딕셔너리 기반으로 표현)

그래프 흐름의 종료 지점을 나타낼 때 사용

노드 설정

1. 분류 노드

```
def classification(state):
    text = state.get("user_input", "")
    if any(word in text for word in ["좋아요", "네", "좋다", "응"]):
        label = "positive"
    elif any(word in text for word in ["싫어요", "아니요", "별로", "싫다"]):
        label = "negative"
    else:
        label = "neutral"
    return {**state, "label": label}
```

해당 key가 없을 때 빈 문자열 반환

2. 응답 노드

```
def positive_answer(state):
    return {**state, "response": "좋게 생각해주셔서 감사합니다!"}

def negative_answer(state):
    return {**state, "response": "무엇이 불편하셨나요?"}

def neutral_answer(state):
    return {**state, "response": "조금 더 자세히 말씀해 주세요."}
```

그래프 설계

4. 그래프 설계

graph = StateGraph(dict)

그래프에서 상태로 다룰 데이터 타입

graph.add_node("classification", classification)

노드 이름 *이 노드에서 실행될 함수*

graph.add_node("positive", positive_answer)

graph.add_node("negative", negative_answer)

graph.add_node("neutral", neutral_answer)

graph.set_entry_point("classification")

그래프 흐름에서 시작 노드 지정

<langgraph.graph.state.StateGraph at 0x1bb884b4210>

엣지 추가

```
def get_label(state):  
    return state.get("label", "")
```

분류 후 label에 따라 분기

```
graph.add_conditional_edges("classification", get_label, {  
    "positive": "positive",  
    "negative": "negative",  
    "neutral": "neutral"  
})
```

*"classification" 노드의 실행 결과 label이
"positive"이면 "positive" 노드로 이동
"negative"이면 "negative" 노드로 이동
"neutral"이면 "neutral" 노드로 이동*

<langgraph.graph.state.StateGraph at 0x1bb884b4210>

엣지 추가

각 노드 끝나면 종료

graph.add_edge("positive", END) "positive" 노드가 실행된 후에는 그래프 종료

graph.add_edge("negative", END) "negative" 노드가 실행된 후에는 그래프 종료

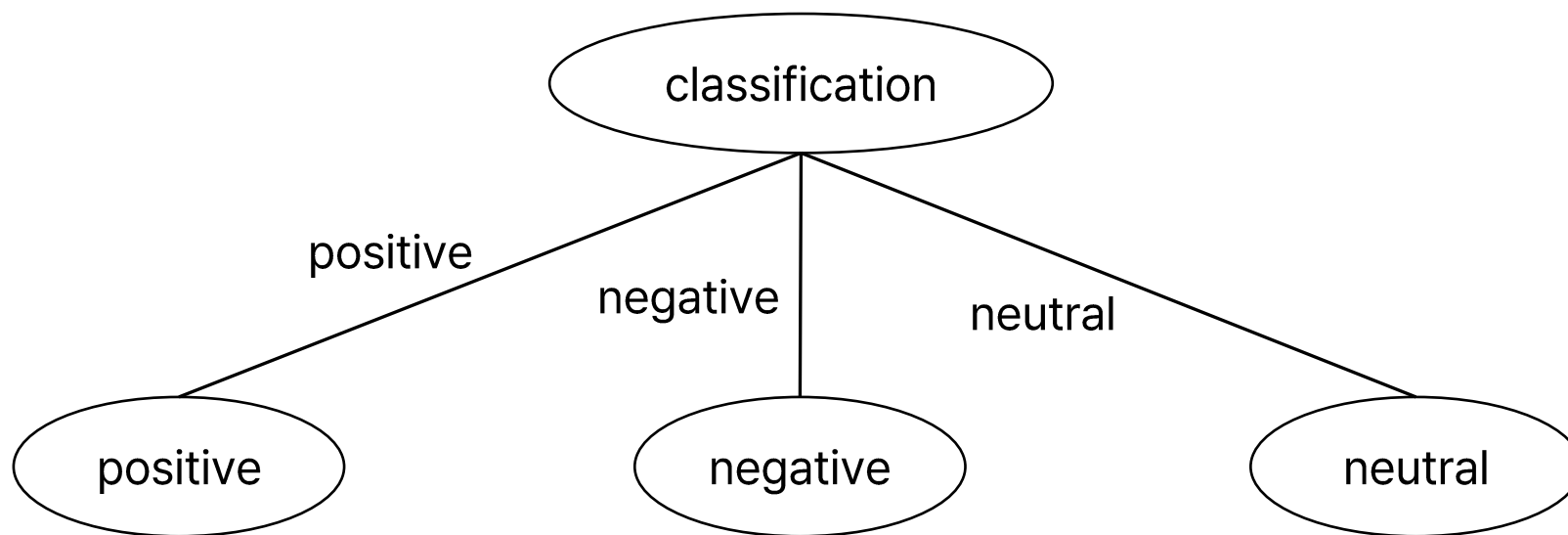
graph.add_edge("neutral", END) "neutral" 노드가 실행된 후에는 그래프 종료

<langgraph.graph.state.StateGraph at 0x1bb884b4210>

add_edge vs add_conditional_edges

`add_edge("A", "B")` → 무조건 A에서 B로 이동

`add_conditional_edges("A", 조건함수, 분기맵)` → 조건에 따라 다른 노드로 이동



그래프 실행

5. 실행기 구성

```
app = graph.compile()
```

 앞서 만든 그래프 설계를 컴파일해서 앱으로 생성

6. 테스트 실행 (한글 입력)

```
user_input = input("한글 입력: ")
```

한글 입력: 이 서비스 너무 좋아요.

```
final_state = app.invoke({"user_input": user_input})  
print("응답:", final_state.get("response", ""))
```

응답: 좋게 생각해 주셔서 감사합니다!

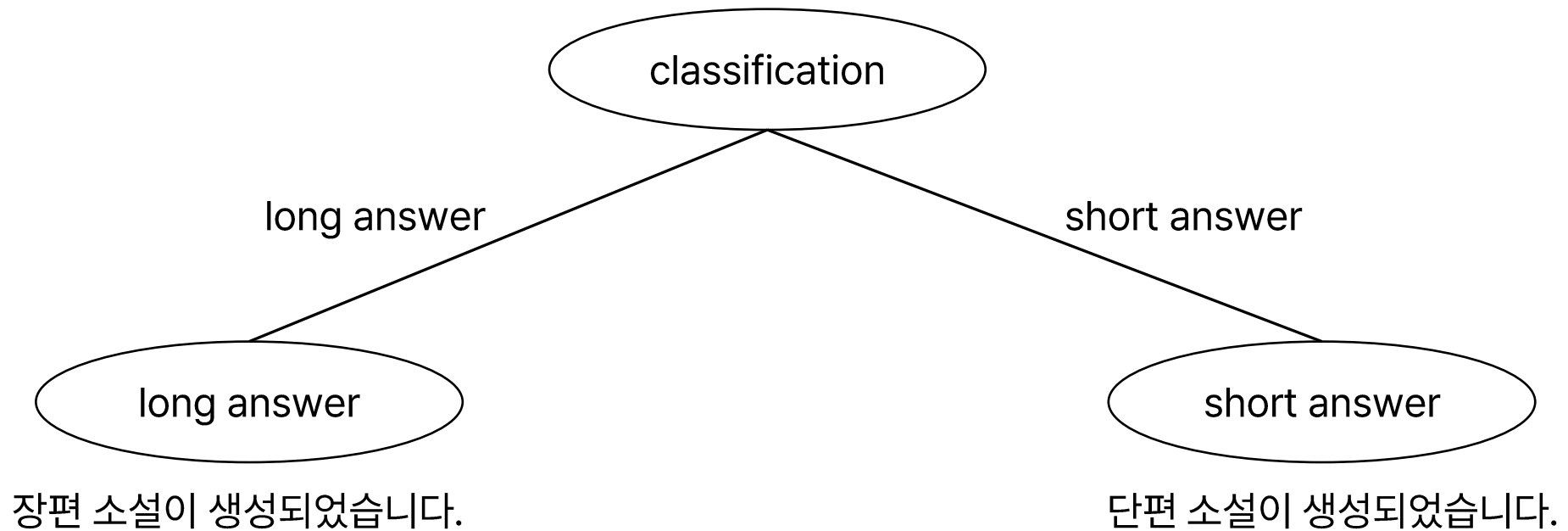
→ 앞서 함수 생성할 때 "user_input"을 사용했으므로

GEN AI 인텐시브 과정

Section 1. LangGraph

Section 1-3. LangGraph + LangChain

실습 내용



Section

LangGraph + LangChain

LangGraph + LangChain

```
[1]: from langgraph.graph import StateGraph, END
      from langchain_huggingface import HuggingFacePipeline
      from langchain_core.prompts import PromptTemplate
      from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
```

```
[2]: # 1. 한글 가능한 모델 설정
      model_id = "skt/kogpt2-base-v2"
      tokenizer = AutoTokenizer.from_pretrained(model_id)
      model = AutoModelForCausalLM.from_pretrained(model_id)
```

```
[3]: # 2. 파이프라인
      text_gen = pipeline(
          "text-generation",
          model=model,
          tokenizer=tokenizer,
          truncation=False,
          max_new_tokens=100,
          do_sample=True,
          return_full_text=True,
          temperature=0.7)

      llm = HuggingFacePipeline(pipeline=text_gen)
```

새로운 토큰 생성 갯수

템플릿까지 출력할 것인지?

```
[4]: # 3. 프롬프트 및 체인 생성
      prompt = PromptTemplate.from_template("아주 먼 옛날 {user_input}이 살고 있었습니다. 그러던 어느날 ")
      chain = prompt | llm
```

LangGraph + LangChain

```
result1 = chain.invoke({"user_input": "토끼"})  
print(result1)
```

아주 먼 옛날 토끼이 살고 있었습니다. 그러던 어느날 똥똥한 토끼가 와서 토끼에게 물었습니다." 토끼는 고개를 갸우뚱했습니다.
"그랬구나. 나는 토끼를 도울 수 있는 방법을 찾았습니다." 토끼는 이렇게 대답했습니다.
"이 방법은 아주 간단하고 좋습니다." 토끼는 정말 용감했습니다.
"나는 토끼에게 물었습니다." 토끼는 이렇게 말했습니다.
"물고기가 토끼에게 물었을 때, 나는, 토끼가 물었을 때 나는 놀라서 울었지. 그런데 그 물음에 나는

```
len(result1)
```

246

```
result2 = chain.invoke({"user_input": "고양이"})  
print(result2)
```

아주 먼 옛날 고양이가 살고 있었습니다. 그러던 어느날 댁이 찾아왔습니다. 고양이의 주인이며 그 집 아주머니는 바로 댁의 부인이었습니다. 그런데 이 고양이는 아주머니의 집 근처에도 잘 보이지 않았습니다. 물론 이 고양이는 이 집에서 멀리 떨어진 한적한 시골마을에 살고 있는 것 같았습니다. 그런데 이 고양이가 이 작은 시골마을에 사는 것을 알게 된 순간, 댁은 아주머니에게 그 일을 말기고 싶었습니다."
"그렇습니다. 그런데 그 고양이의 주인은 다름 아닌 아주머니였습니다. 그 고양이는

```
len(result2)
```

272

LangGraph + LangChain

3. 판단 노드

```
def classification(state):
    user_input = state.get("user_input", "")
    result = chain.invoke({"user_input": user_input}).strip()

    if len(result) > 250:
        label = "long_answer"
    else:
        label = "short_answer"

    return {**state, "label": label}
```

4. 응답 노드

```
def long_answer(state):
    return {**state, "response": "장편 소설이 생성되었습니다."}

def short_answer(state):
    return {**state, "response": "단편 소설이 생성되었습니다."}
```

```
def get_label(state):
    return state.get("label", "")
```


LangGraph + LangChain

5. LangGraph 구성

```
graph = StateGraph(dict)
```

```
graph.add_node("classification", classification)
```

```
graph.add_node("long_answer", long_answer)
```

```
graph.add_node("short_answer", short_answer)
```

```
graph.set_entry_point("classification")
```

```
graph.add_conditional_edges("classification", get_label, {
    "long_answer": "long_answer",
    "short_answer": "short_answer"
})
```

```
graph.add_edge("long_answer", END)
```

```
graph.add_edge("short_answer", END)
```

```
<langgraph.graph.state.StateGraph at 0x226ff287350>
```

6. 실행

```
app = graph.compile()
```

```
user_input = input("이야기를 만들고 싶은 동물 입력: ")
```

```
final_state = app.invoke({"user_input": user_input})
```

```
print("응답:", final_state.get("response", ""))
```

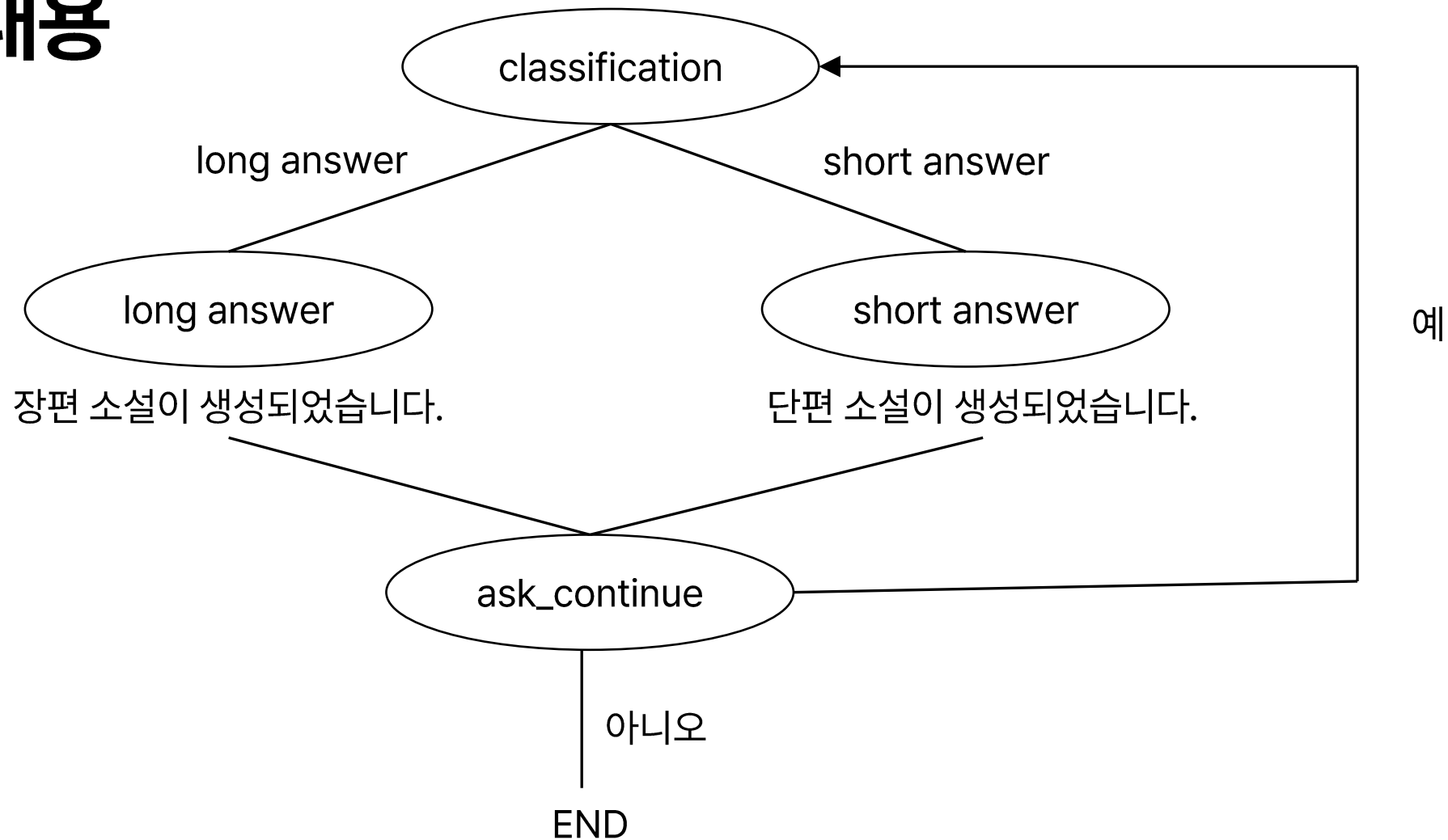
이야기를 만들고 싶은 동물 입력: 거북이
응답: 장편 소설이 생성되었습니다.

GEN AI 인텐시브 과정

Section 1. LangGraph

Section 1-4. LangGraph + LangChain + multi-turn

실습 내용



라이브러리 불러오기 & 모델 설정

```
from langgraph.graph import StateGraph, END
from langchain_huggingface import HuggingFacePipeline
from langchain_core.prompts import PromptTemplate
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
```

1. 모델 설정

```
model_id = "skt/kogpt2-base-v2"
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(model_id)
```

파이프라인 & 프롬프트 & 체인 생성

2. 파이프라인 생성

```
text_gen = pipeline(  
    "text-generation",  
    model=model,  
    tokenizer=tokenizer,  
    truncation=False,  
    max_new_tokens=100,  
    do_sample=True,  
    return_full_text=True,  
    temperature=0.7  
)
```

```
llm = HuggingFacePipeline(pipeline=text_gen)
```

3. 프롬프트 및 체인

```
prompt = PromptTemplate.from_template("아주 먼 옛날 {user_input}이 살고 있었습니다. 그러던 어느날 ")  
story_chain = prompt | llm
```

이야기 생성 및 분류 노드 생성

이야기 생성 및 분류

```
def classification(state):
    user_input = state.get("user_input", "")
    story = story_chain.invoke({"user_input": user_input}).strip()
    label = "long_answer" if len(story) > 250 else "short_answer"
    return {**state, "label": label, "story": story}
```

4. 응답 노드

```
def long_answer(state):
    res = "장편 소설이 생성되었습니다."
    print(res)
    return {**state, "response": res}
```

```
def short_answer(state):
    res = "단편 소설이 생성되었습니다."
    print(res)
    return {**state, "response": res}
```

Section

LangGraph + LangChain + multi-turn

추가 질문 노드

5. 사용자 질문 여부

```
def ask_continue(state):
    reply = input("\n추가 질문하시겠습니까? (예/아니오)\n> ").strip()
    if reply in ["예", "네", "yes", "Yes"]:
        state["continue"] = True
        question = input("추가 질문을 입력하세요:\n> ").strip()
        state["user_input"] = question
    else:
        state["continue"] = False
    return state
```

6. 분기: 계속할지 판단

```
def should_continue(state):
    if state.get("continue"):
        answer = "replay"
    else:
        answer = "__end__"
    return answer
```

7. 이전 스토리 기반 응답

```
def replay_story(state):
    question = state.get("user_input", "")
    if "보여줘" in question:
        print("\n이전에 생성된 이야기:\n", state.get("story", "[이야기가 없습니다]"))
        return {**state, "response": "이전 이야기 다시 보여줌"}
    else:
        # 새로운 동물 입력일 경우 새로운 이야기 생성
        return classification(state)
```

Section

LangGraph + LangChain + multi-turn

그래프 생성

```
def get_label(state):  
    return state.get("label", "")
```

9. LangGraph 구성

```
graph = StateGraph(dict)  
graph.add_node("classification", classification)  
graph.add_node("long_answer", long_answer)  
graph.add_node("short_answer", short_answer)  
graph.add_node("ask_continue", ask_continue)  
graph.add_node("replay", replay_story)  
  
graph.set_entry_point("classification")  
  
graph.add_conditional_edges("classification", get_label, {  
    "long_answer": "long_answer",  
    "short_answer": "short_answer"  
})  
  
graph.add_edge("long_answer", "ask_continue")  
graph.add_edge("short_answer", "ask_continue")  
  
graph.add_conditional_edges("ask_continue", should_continue, {  
    "replay": "replay",  
    "__end__": END  
})  
  
graph.add_edge("replay", "ask_continue")
```


Section

LangGraph + LangChain + multi-turn

실행

10. 실행

```
app = graph.compile()

print("동물 이름을 입력해 이야기를 시작해보세요!")
user_input = input("동물 이름 입력: ")
final_state = app.invoke({"user_input": user_input})
print("\n대화 종료")
```

동물 이름을 입력해 이야기를 시작해보세요!

동물 이름 입력: 고양이

첫 번째 입력

장편 소설이 생성되었습니다.

추가 질문하시겠습니까? (예/아니오)

> 예

두 번째 입력

추가 질문을 입력하세요:

> 이전 이야기 보여줘

세 번째 입력

이전에 생성된 이야기:

아주 먼 옛날 고양이이 살고 있었습니다. 그러던 어느날 뭔가 생각났습니다. 고양이를 데리고 집으로 들어갔더니 고양이를 데리고 산책을 하고 있었습니다. 저는 고양이를 주울 수 있는 방법을 찾아냈습니다. 고양이가 집을 떠나기 전에 이미 고양이를 데리고 간 뒤였습니다. 주인이 고양이를 데리고 집으로 돌아와 보니 주인은 고양이를 데리고 집에 들어갔습니다. 주인도 고양이를 안고 집으로 들어가 보았습니다. 주인도 고양이를 데리고 집으로 가 보았습니다. 고양이를 데리고 집에 들어온 고양이는 주인이 고양이를 안아 주려고 안간힘을 썼지만 아무도 오지 않았

추가 질문하시겠습니까? (예/아니오)

> 아니오

네 번째 입력

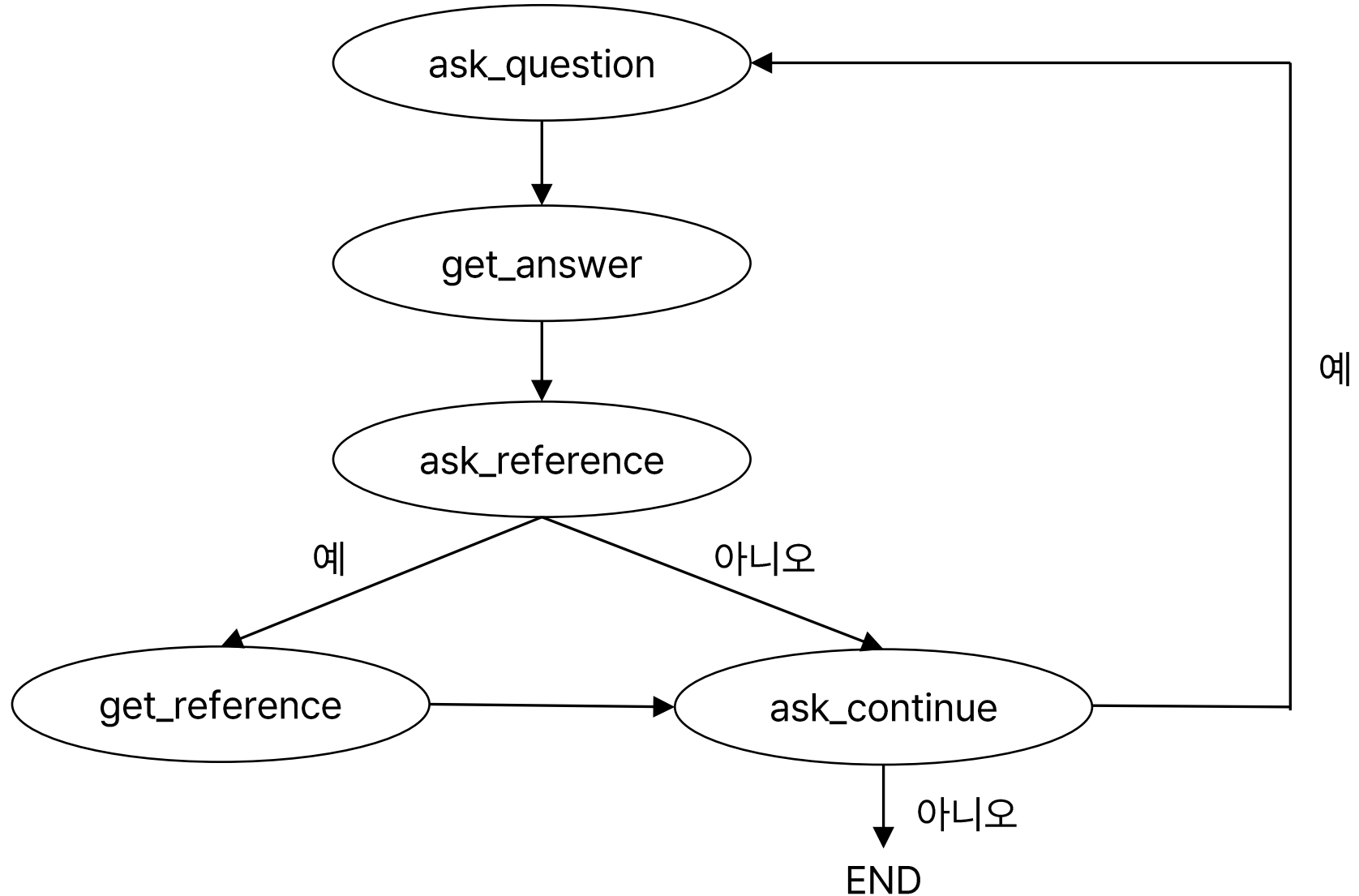
대화 종료

GEN AI 인텐시브 과정

Section 1. LangGraph

Section 1-4. LangGraph + LangChain + multi-turn + Memory + RAG(1)

실습 내용 RetrievalQA 사용



라이브러리 & 데이터 불러오기

```
import pandas as pd
from langchain_chroma import Chroma
from langchain.schema import Document
from langgraph.graph import StateGraph, END
from langchain.chains import RetrievalQA
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_huggingface import HuggingFacePipeline
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
```

1. 문서 불러오기

```
df = pd.read_csv('./data/data.csv', encoding='utf-8')
df.head(10)
```

text

0 Etching 공정 전에는 반드시 세정 공정이 완료되어야 합니다.

1 PM 장비의 필터는 주 1회 정기적으로 교체해야 합니다.

Section

LangGraph + LangChain + multi-turn + Memory + RAG(1)

pandas -> chromaDB

2. 문서 리스트로 변환

```
texts = df["text"].tolist()
docs = [Document(page_content=text) for text in texts]
```

3. 임베딩 모델

```
embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2")
```

4. Chroma에 저장

```
vectorstore = Chroma.from_documents(
    documents=docs,
    embedding=embedding_model,
    persist_directory="./chromaDB"
)
```

모델 & 체인 생성

RetrievalQA 체인을 사용하기 위해서는
"text-generation 모델을 사용해야함."

RetrievalQA 체인 생성

- RAG 기반 질문 응답 시스템 구축시 사용
- 검색 -> 생성 과정을 합친 체인

1. 사용자 질문을 받는다(query)
2. retrieve를 통해 관련 문서 찾기
3. 찾은 문서 + 질문을 LLM에 넘김
4. LLM이 답변 생성

5. LLM 준비

```
model_id = "skt/kogpt2-base-v2"
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(model_id)
```

```
text_gen = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    truncation=False,
    max_new_tokens=100,
    do_sample=True,
    return_full_text=True,
    temperature=0.7
)
```

```
llm = HuggingFacePipeline(pipeline=text_gen)
```

6. RAG QA 체인

```
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=vectorstore.as_retriever(),
    return_source_documents=True
)
```

노드 설정

7. 노드 설정

```
def ask_question(state):
    question = input("질문을 입력해주세요: ")
    result = qa_chain.invoke({"query": question})
    res = {
        "question": question,
        "answer": result["result"],
        "source_docs": result["source_documents"]
    }
    return res

def get_answer(state):
    print("\n답변:", state["answer"])
    return state

def ask_reference(state):
    reply = input("\n참고문서를 보시겠습니까? (예/아니오): ").strip()
    return {**state, "see_reference": reply}
```

RetrievalQA 체인의 입력키 이름이 "query"

RetrievalQA의 출력값은 딕셔너리이며, "result"에 정답이 포함되어 있음

RetrievalQA의 "source_documents"에 참고문서 포함

노드 설정

```
def get_reference(state):  
    print("\n[참고 문서]")  
    for i, doc in enumerate(state["source_docs"], 1):  
        print(f"{i}. {doc.page_content[:200]}...")  
    return state  
  
def ask_continue(state):  
    reply = input("\n계속하시겠습니까? (예/아니오): ").strip()  
    return {**state, "continue": reply}  
  
def reference_or_not(state):  
    return "get_reference" if state.get("see_reference") == "예" else "ask_continue"  
  
def continue_or_not(state):  
    return "ask_question" if state.get("continue") == "예" else END
```


Section

LangGraph + LangChain + multi-turn + Memory + RAG(1)

그래프 설계

8. 그래프 구성

```
graph = StateGraph(dict)
```

노드 추가

```
graph.add_node("ask_question", ask_question)
```

```
graph.add_node("get_answer", get_answer)
```

```
graph.add_node("ask_reference", ask_reference)
```

```
graph.add_node("get_reference", get_reference)
```

```
graph.add_node("ask_continue", ask_continue)
```

시작점 설정

```
graph.set_entry_point("ask_question")
```

엣지 설정

```
graph.add_edge("ask_question", "get_answer")
```

```
graph.add_edge("get_answer", "ask_reference")
```

```
graph.add_conditional_edges("ask_reference", reference_or_not, {
    "get_reference": "get_reference",
    "ask_continue": "ask_continue"
})
```

```
graph.add_edge("get_reference", "ask_continue")
```

```
graph.add_conditional_edges("ask_continue", continue_or_not, {
    "ask_question": "ask_question",
    END: END
})
```

<langgraph.graph.state.StateGraph at 0x21fd303f650>

LangGraph + LangChain + multi-turn + Memory + RAG(1)

실행

```
app = graph.compile()
app.invoke({})
```

질문을 입력해주세요:

MES 공정 로그 보관 기간은?

첫 번째 입력

답변: Use the following pieces of context to answer the question at the end. If you don't know the answer, just say that you don't know. Don't make up an answer.

MES 시스템에 기록된 공정 로그는 6개월간 보관됩니다.

공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.

소자 특성 분석 결과는 전자문서 시스템에 등록합니다.

이온 주입 공정 후 열처리 시간은 최소 30분 이상 확보합니다.

Question: MES 공정 로그 보관 기간은?

[illegible]

실행

참고문서를 보시겠습니까? (예/아니오):

예

두 번째 입력

[참고 문서]

1. MES 시스템에 기록된 공정 로그는 6개월간 보관됩니다....
2. 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다....
3. 소자 특성 분석 결과는 전자문서 시스템에 등록합니다....
4. 이온 주입 공정 후 열처리 시간은 최소 30분 이상 확보합니다....

Section

LangGraph + LangChain + multi-turn + Memory + RAG(1)

실행

계속하시겠습니까? (예/아니오):

예

세 번째 입력

질문을 입력해주세요:

클린룸 입장시 주의 사항 알려줘

네 번째 입력

답변: Use the following pieces of context to answer the question at the end. If you don't know the answer, just say you don't know. Do not make up an answer.

공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.

장비 재가동 시 Warm-up 절차를 반드시 이행합니다.

Etching 공정 전에는 반드시 세정 공정이 완료되어야 합니다.

클린룸 입장 전에는 반드시 정전기 방지복을 착용해야 합니다.

Question: 클린룸 입장시 주의 사항 알려줘

Helpful Answer: 클린룸 입장시 주의 사항

Attitude: 클린룸 입장시 주의 사항

Schedule: 클린룸 입장시 주의 사항

Salvation: 클린룸 입장시 주의 사항

Attitude: 클린룸 입장시 주의 사항

There's prepared by the corporate of the question at the end of the question from the end of the

Section

LangGraph + LangChain + multi-turn + Memory + RAG(1)

실행

참고문서를 보시겠습니까? (예/아니오): 다섯 번째 입력

[참고 문서]

1. 공정 이탈 발생 시 **Shift** 리더에게 즉시 보고합니다....
2. 장비 재가동 시 **Warm-up** 절차를 반드시 이행합니다....
3. **Etching** 공정 전에는 반드시 세정 공정이 완료되어야 합니다....
4. 클린룸 입장 전에는 반드시 정전기 방지복을 착용해야 합니다....

계속하시겠습니까? (예/아니오): 여섯 번째 입력

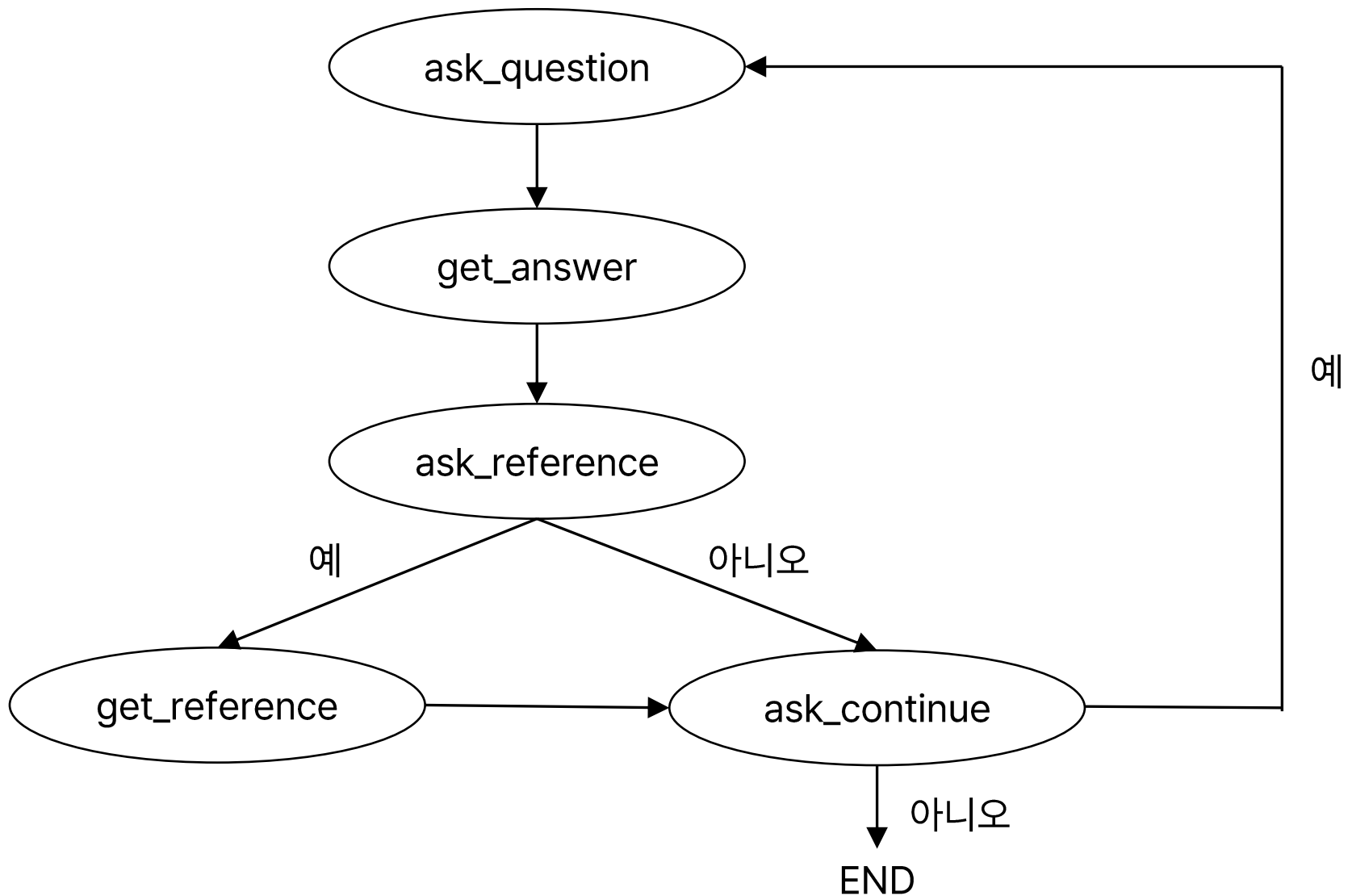
```
{'question': '클린룸 입장시 주의 사항 알려줘',  
 'answer': "Use the following pieces of context to answer the question at the end. If you don't know th  
to make up an answer.\n\n공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.\n\n장비 재가동 시 Warm-up 절차를  
공정이 완료되어야 합니다.\n\n클린룸 입장 전에는 반드시 정전기 방지복을 착용해야 합니다.\n\nQuestion: 클린룸 입  
주의 사항\nAttitude: 클린룸 입장시 주의 사항\nSchedule: 클린룸 입장시 주의 사항\nSalvation: 클린룸 입장시 주:  
repared by the corporate of the question at the end of the question from the end of the ",  
 'source_docs': [Document(metadata={}, page_content='공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.'),  
 Document(metadata={}, page_content='장비 재가동 시 Warm-up 절차를 반드시 이행합니다.'),  
 Document(metadata={}, page_content='Etching 공정 전에는 반드시 세정 공정이 완료되어야 합니다.'),  
 Document(metadata={}, page_content='클린룸 입장 전에는 반드시 정전기 방지복을 착용해야 합니다.')],  
 'see_reference': '예',  
 'continue': '아니오'}
```

GEN AI 인텐시브 과정

Section 1. LangGraph

Section 1-4. LangGraph + LangChain + multi-turn + Memory + RAG(2)

실습 내용 RetrievalQA 미사용



Section

LangGraph + LangChain + multi-turn + Memory + RAG(2)

라이브러리 & 데이터 불러오기

```
from langchain_chroma import Chroma
from langgraph.graph import StateGraph, END
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_huggingface import HuggingFacePipeline
from transformers import AutoTokenizer, AutoModelForQuestionAnswering, pipeline
```

1. 임베딩 모델

```
embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2")
```

2. 저장된 ChromaDB 불러오기

```
retriever = Chroma(
    persist_directory="./chromaDB",
    embedding_function=embedding_model
).as_retriever(search_kwargs={"k": 3})
```


모델 & 파이프라인 설정

3. KoELECTRA QA 모델 로딩

```
model_id = "monologg/koelectra-base-v3-finetuned-korquad"  
qa_tokenizer = AutoTokenizer.from_pretrained(model_id)  
qa_model = AutoModelForQuestionAnswering.from_pretrained(model_id)
```

4. 파이프라인 & LangChain 래핑

```
text_gen = pipeline(  
    "question-answering",  
    model=qa_model,  
    tokenizer=qa_tokenizer,  
    device=-1,  
    max_length=512,  
    do_sample=False,  
    temperature=0.1,  
    truncation=True  
)
```

노드 설정

5. 노드 설정

```
def ask_question(state):
    question = input("질문을 입력해주세요: ").strip()
    docs = retriever.invoke(question)
    top_docs = docs[:3]
    best_contexts = [doc.page_content for doc in top_docs]
    combined_context = "\n".join(best_contexts)
    result = text_gen(question=question, context=combined_context)
    res = {
        "question": question,
        "answer": result["answer"],
        "context": best_contexts
    }
    return res

def get_answer(state):
    print(f"\n질문: {state['question']}")
    print(f"답변: {state['answer']}")
    return state
```

노드 설정

```
def ask_reference(state):
    user_input = input("\n 참고 문서를 보시겠습니까? (예/아니오): ").strip()
    state["show_reference"] = user_input.startswith("예")
    return state

def get_reference(state):
    if state.get("show_reference"):
        print("\n 참고 문서:\n", state["context"])
    return state

def ask_continue(state):
    user_input = input("\n 계속하시겠습니까? (예/아니오): ").strip()
    state["continue"] = user_input.startswith("예")
    return state

# 분기 함수
def should_show_reference(state):
    return "get_reference" if state.get("show_reference") else "ask_continue"

def should_continue(state):
    return "ask_question" if state.get("continue") else END
```

그래프 구성

6. 그래프 구성

```
graph = StateGraph(dict)
```

노드 추가

```
graph.add_node("ask_question", ask_question)
```

```
graph.add_node("get_answer", get_answer)
```

```
graph.add_node("ask_reference", ask_reference)
```

```
graph.add_node("get_reference", get_reference)
```

```
graph.add_node("ask_continue", ask_continue)
```

시작점 설정

```
graph.set_entry_point("ask_question")
```

엣지 설정

```
graph.add_edge("ask_question", "get_answer")
```

```
graph.add_edge("get_answer", "ask_reference")
```

```
graph.add_conditional_edges("ask_reference", should_show_reference, {
    "get_reference": "get_reference",
    "ask_continue": "ask_continue"
})
```

```
graph.add_edge("get_reference", "ask_continue")
```

```
graph.add_conditional_edges("ask_continue", should_continue, {
    "ask_question": "ask_question",
    END: END
})
```

Section

LangGraph + LangChain + multi-turn + Memory + RAG(2)

7. 실행

```
app = graph.compile()  
app.invoke({})
```

질문을 입력해주세요: MES 공정 로그 보관 기간은? 첫 번째 입력

질문: MES 공정 로그 보관 기간은?

답변: 6개월간

참고 문서를 보시겠습니까? (예/아니오): 예 두 번째 입력

참고 문서:

['MES 시스템에 기록된 공정 로그는 6개월간 보관됩니다.', '공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.', '소자 특성 분석 결과는 전자문서 시스템에 등록합니다.']

계속하시겠습니까? (예/아니오): 예 세 번째 입력

질문을 입력해주세요: 클린룸 입장 시 주의사항 알려줘 네 번째 입력

질문: 클린룸 입장 시 주의사항 알려줘

답변: 반드시

참고 문서를 보시겠습니까? (예/아니오): 예 다섯 번째 입력

참고 문서:

['공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.', '장비 재가동 시 Warm-up 절차를 반드시 이행합니다.', 'Etching 공정 전에는 반드시 세정 공정이 완료되어야 합니다.']

계속하시겠습니까? (예/아니오): 아니오 여섯 번째 입력

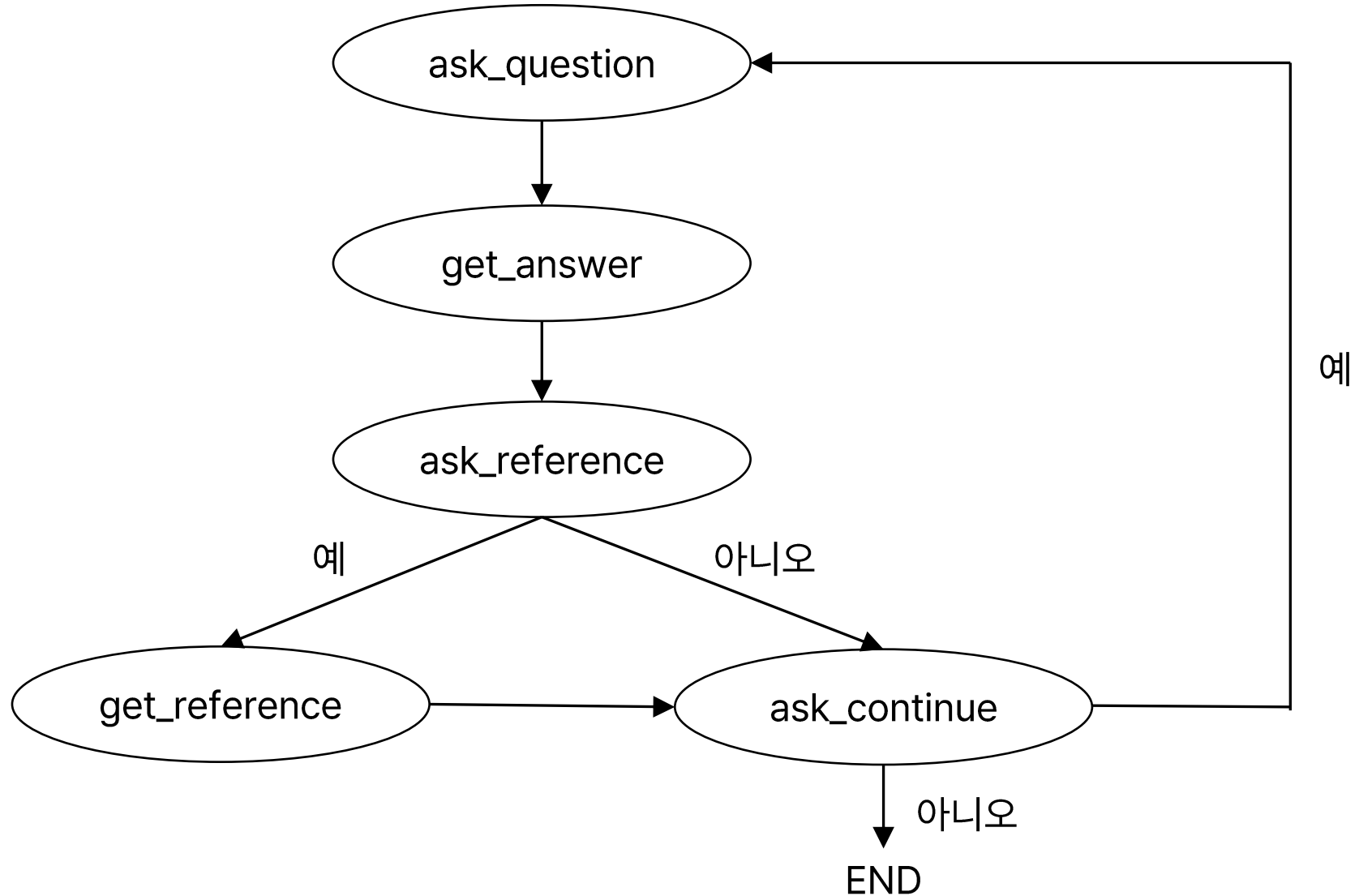
```
{  
  'question': '클린룸 입장 시 주의사항 알려줘',  
  'answer': '반드시',  
  'context': ['공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.',  
              '장비 재가동 시 Warm-up 절차를 반드시 이행합니다.',  
              'Etching 공정 전에는 반드시 세정 공정이 완료되어야 합니다.'],  
  'show_reference': True,  
  'continue': False  
}
```

LangGraph

Section 1. LangGraph

Section 1-5. LangGraph + LangChain + multi-turn + Memory + RAG(3)

실습 내용 RetrievalQA 미사용



Section

LangGraph + LangChain + multi-turn + Memory + RAG(3)

라이브러리 & 데이터 불러오기

```
import uuid
from langchain_chroma import Chroma
from langgraph.graph import StateGraph, END
from langchain_huggingface import HuggingFaceEmbeddings, HuggingFacePipeline
from transformers import AutoTokenizer, AutoModelForQuestionAnswering, pipeline
from langchain_core.chat_history import InMemoryChatMessageHistory
from langchain_core.messages import get_buffer_string
from langchain_core.runnables import RunnableConfig
from langchain_core.chat_history import BaseChatMessageHistory
```

1. 임베딩 모델

```
embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2")
```

2. 저장된 ChromaDB 불러오기

```
retriever = Chroma(
    persist_directory="./chromaDB",
    embedding_function=embedding_model
).as_retriever(search_kwargs={"k": 3})
```


Section

LangGraph + LangChain + multi-turn + Memory + RAG(3)

모델 로딩 & 파이프라인

3. KoELECTRA QA 모델 로딩

```
model_id = "monologg/koelectra-base-v3-finetuned-korquad"
qa_tokenizer = AutoTokenizer.from_pretrained(model_id)
qa_model = AutoModelForQuestionAnswering.from_pretrained(model_id)
```

4. 파이프라인 & LangChain 래핑

```
text_gen = pipeline(
    "question-answering",
    model=qa_model,
    tokenizer=qa_tokenizer,
    device=-1,
    max_length=512,
    do_sample=False,
    temperature=0.1,
    truncation=True
)
llm = HuggingFacePipeline(pipeline=text_gen)
```

Device set to use cpu

히스토리 관리

```
# 5. 히스토리 관리
```

```
chats_by_session_id = {}
```

```
def get_chat_history(session_id: str) -> BaseChatMessageHistory:
    if session_id not in chats_by_session_id:
        chats_by_session_id[session_id] = InMemoryChatMessageHistory()
    return chats_by_session_id[session_id]
```

노드 생성

6. 노드 정의

```
def ask_question(state, config: RunnableConfig):
    session_id = config["configurable"]["session_id"]
    chat_history = get_chat_history(session_id)

    question = input("질문을 입력해주세요: ").strip()
    docs = retriever.invoke(question)
    top_docs = docs[:3]
    best_contexts = [doc.page_content for doc in top_docs]
    combined_context = "\n".join(best_contexts)

    history_text = get_buffer_string(chat_history.messages)
    full_context = f"{history_text}\n\n{combined_context}" if history_text else combined_context

    result = text_gen(question=question, context=full_context)

    chat_history.add_user_message(question)
    chat_history.add_ai_message(result["answer"])

    return {
        "question": question,
        "answer": result["answer"],
        "context": best_contexts
    }
```

노드 생성

```
def get_answer(state):
    print(f"\n질문: {state['question']}")
    print(f"답변: {state['answer']}")
    return state

def ask_reference(state):
    user_input = input("\n참고 문서를 보시겠습니까? (예/아니오): ").strip()
    state["show_reference"] = user_input.startswith("예")
    return state

def get_reference(state):
    if state.get("show_reference"):
        print("\n참고 문서:\n")
        for i, ctx in enumerate(state["context"], 1):
            print(f"[{i}] {ctx}\n")
    return state

def ask_continue(state):
    user_input = input("\n계속하시겠습니까? (예/아니오): ").strip()
    state["continue"] = user_input.startswith("예")
    return state

def should_show_reference(state):
    return "get_reference" if state.get("show_reference") else "ask_continue"

def should_continue(state):
    return "ask_question" if state.get("continue") else END
```

그래프 생성

```
# 7. LangGraph 구성
graph = StateGraph(dict)
graph.add_node("ask_question", ask_question)
graph.add_node("get_answer", get_answer)
graph.add_node("ask_reference", ask_reference)
graph.add_node("get_reference", get_reference)
graph.add_node("ask_continue", ask_continue)

graph.set_entry_point("ask_question")
graph.add_edge("ask_question", "get_answer")
graph.add_edge("get_answer", "ask_reference")
graph.add_conditional_edges("ask_reference", should_show_reference, {
    "get_reference": "get_reference",
    "ask_continue": "ask_continue"
})
graph.add_edge("get_reference", "ask_continue")
graph.add_conditional_edges("ask_continue", should_continue, {
    "ask_question": "ask_question",
    END: END
})
```

<langgraph.graph.state.StateGraph at 0x792e5334fd10>

Section

LangGraph + LangChain + multi-turn + Memory + RAG(3)

실행

```
# 8. 실행
session_id = str(uuid.uuid4()) # 고유 세션 ID
config = {"configurable": {"session_id": session_id}}

app = graph.compile()
app.invoke({}, config=config)
```

질문을 입력해주세요: 첫 번째 입력

질문: MES 공정 로그 보관 기간은?

답변: 6개월간

참고 문서를 보시겠습니까? (예/아니오): 두 번째 입력

참고 문서:

[1] MES 시스템에 기록된 공정 로그는 6개월간 보관됩니다.

[2] 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.

[3] 소자 특성 분석 결과는 전자문서 시스템에 등록합니다.

실행

계속하시겠습니까? (예/아니오):

예

세 번째 입력

질문을 입력해주세요:

그럼 MES 공정 보관 기간이 지나면 사용할 수 없니?

네 번째 입력

질문: 그럼 MES 공정 보관 기간이 지나면 사용할 수 없니?

답변: 6개월간

참고 문서를 보시겠습니까? (예/아니오):

예

다섯 번째 입력

참고 문서:

[1] 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.

[2] MES 시스템에 기록된 공정 로그는 6개월간 보관됩니다.

[3] Etching 공정 전에는 반드시 세정 공정이 완료되어야 합니다.

계속하시겠습니까? (예/아니오):

아니오

여섯 번째 입력

```
[12]: {'question': '그럼 MES 공정 보관 기간이 지나면 사용할 수 없니?',  
      'answer': '6개월간',  
      'context': ['공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.',  
                  'MES 시스템에 기록된 공정 로그는 6개월간 보관됩니다.',  
                  'Etching 공정 전에는 반드시 세정 공정이 완료되어야 합니다.'],  
      'show_reference': True,  
      'continue': False}
```

감사합니다.

Q & A