Adaptive RAG

강사 장철원

Section 0

목차

❑ Adaptive RAG

Adaptive RAG

Section 1. Adaptive RAG

Section 1-1. Adaptive RAG의 개념

Adaptive RAG

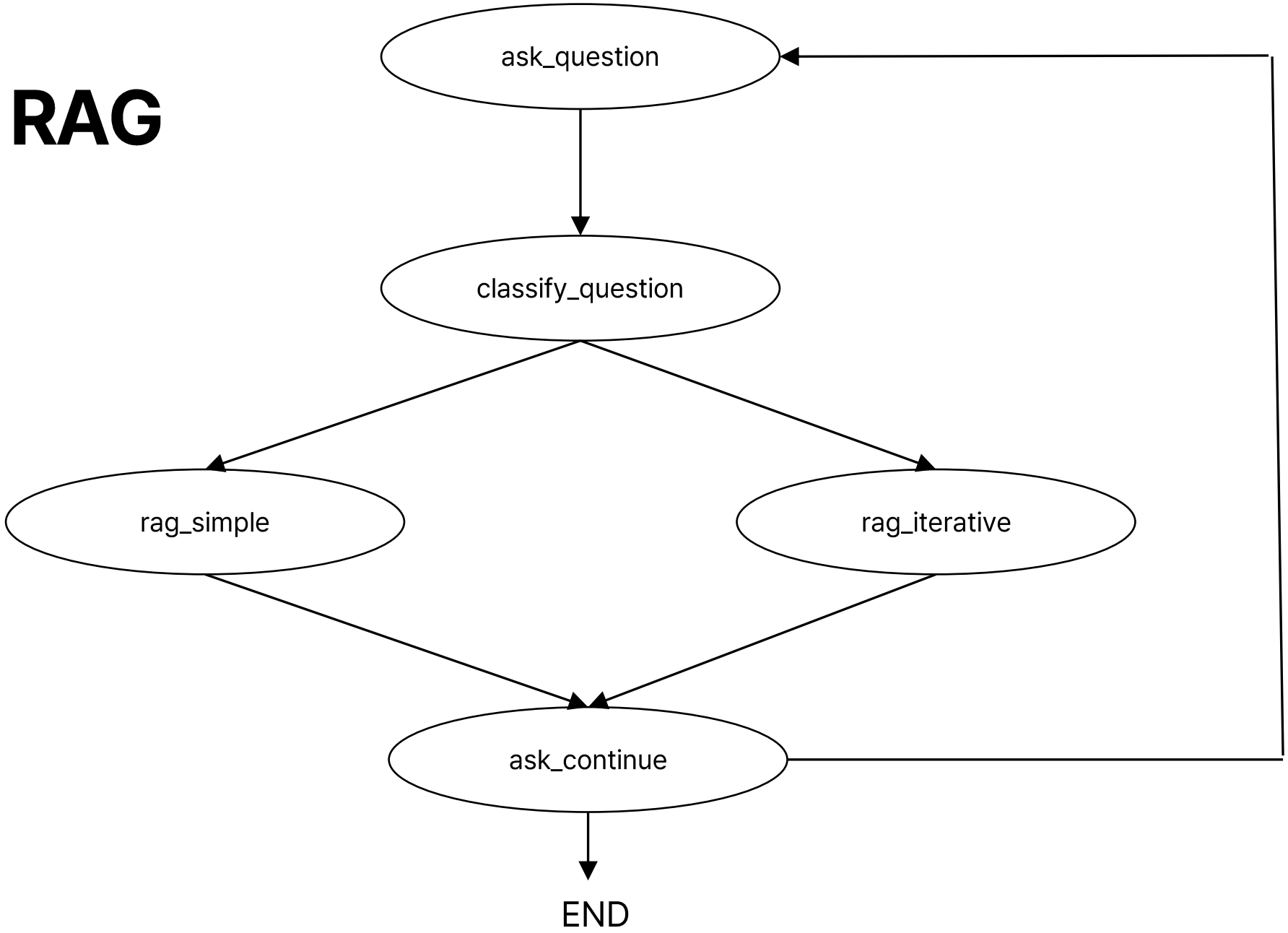
- 질문의 특성과 상황에 맞게 RAG 방식을 유동적으로 변경하는 방식
  - 고정된 방식으로 작동할 경우,,,
    - 질문은 간단한데 너무 많은 문서를 읽거나 질문이 복잡한데 문서를 적게보는 경우 생김
  - 질문에 맞게 똑똑하게 전략을 바꾸자

Adaptive RAG

Section 1. Adaptive RAG

Section 1-2. Adaptive RAG 실습

Adaptive RAG



Adaptive RAG

```python
import uuid
import pandas as pd
from langchain_chroma import Chroma
from langchain.schema import Document
from langgraph.graph import StateGraph, END
from langchain_huggingface import HuggingFaceEmbeddings, HuggingFacePipeline
from transformers import AutoTokenizer, AutoModelForQuestionAnswering, AutoModelForCausalLM, 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
```

```python
# 1. 문서 불러오기
df = pd.read_csv('./data/data.csv', encoding='utf-8')
```

```python
# 2. 문서 리스트로 변환
texts = df["text"].tolist()
docs = [Document(page_content=text) for text in texts]
```

```python
# 3. 임베딩 모델
embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2")
```

```python
# 4. Chroma에 저장
vectorstore = Chroma.from_documents(
    documents=docs,
    embedding=embedding_model,
    persist_directory="./chromaDB"
)
```

Section

Adaptive RAG 실습

Adaptive RAG

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

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

qa_pipeline = pipeline(
    "question-answering",
    model=qa_model,
    tokenizer=qa_tokenizer,
    device=-1)

qa_llm = HuggingFacePipeline(pipeline=qa_pipeline)
```

```
Device set to use cpu
```

```
# 7. generation 모델
rewrite_model_id = "skt/kogpt2-base-v2"
rewrite_tokenizer = AutoTokenizer.from_pretrained(rewrite_model_id)
rewrite_model = AutoModelForCausalLM.from_pretrained(rewrite_model_id)
rewrite_tokenizer.model_max_length = 1024

rewrite_pipeline = pipeline(
    "text-generation",
    model=rewrite_model,
    tokenizer=rewrite_tokenizer,
    max_new_tokens=64,
)
rewrite_llm = HuggingFacePipeline(pipeline=rewrite_pipeline)
```

```
Device set to use cpu
```

Adaptive RAG

```python
# 8. 세션 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]
```

Section

Adaptive RAG 실습

노드 정의(1)

```python
# 9. 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()
    chat_history.add_user_message(question)
    return {"question": question, "history": chat_history}

def classify_question(state):
    question = state["question"]
    # 단순 규칙 기반 분기: 질문 길이와 키워드
    if len(question) < 15:
        strategy = "simple"
    elif any(kw in question for kw in ["어떻게", "왜", "조건", "경우"]):
        strategy = "iterative"
    else:
        strategy = "default"
    print(f"[분석 결과] 선택된 전략: {strategy}")
    return {**state, "strategy": strategy, "attempt": 1}

def rag_simple(state):
    docs = retriever.invoke(state["question"])
    context = "\n".join([doc.page_content for doc in docs])
    result = qa_pipeline(question=state["question"], context=context)
    print(f"\n[단순 RAG] 질문: {state['question']}\n답변: {result['answer']}")
    state["history"].add_ai_message(result["answer"])
    return {**state, "answer": result["answer"]}
```

노드 정의(2)

```python
def rag_iterative(state):
    if state["attempt"] > 2:
        print("\n[반복 RAG] 최대 재질문 횟수 도달. 종료합니다.")
        return state

    docs = retriever.invoke(state["question"])
    context = "\n".join([doc.page_content for doc in docs])
    result = qa_pipeline(question=state["question"], context=context)

    answer = result["answer"]
    if len(answer.strip()) < 10:
        print("\n[반복 RAG] 불충분한 답변, 재질문 시도 중...")
        prompt = f"다음 질문을 더 구체적으로 바꿔주세요: {state['question']}"
        followup = rewrite_pipeline(prompt)[0]['generated_text']
        print(f"[재질문] {followup.strip()}")
        return {**state, "question": followup.strip(), "attempt": state["attempt"] + 1}

    print(f"\n[반복 RAG] 질문: {state['question']}\n답변: {answer}")
    state["history"].add_ai_message(answer)
    return {**state, "answer": answer}
```

노드 정의(3)

```python
def route_strategy(state):
    strategy = state.get("strategy")
    if strategy == "simple":
        return "rag_simple"
    elif strategy == "iterative":
        return "rag_iterative"
    return "rag_simple"

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

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

LangGraph 구성

```python
# 10. LangGraph 구성
graph = StateGraph(dict)
graph.add_node("ask_question", ask_question)
graph.add_node("classify_question", classify_question)
graph.add_node("rag_simple", rag_simple)
graph.add_node("rag_iterative", rag_iterative)
graph.add_node("ask_continue", ask_continue)

graph.set_entry_point("ask_question")
graph.add_edge("ask_question", "classify_question")
graph.add_conditional_edges("classify_question", route_strategy, {
    "rag_simple": "rag_simple",
    "rag_iterative": "rag_iterative"
})
graph.add_edge("rag_simple", "ask_continue")
graph.add_edge("rag_iterative", "ask_continue")
graph.add_conditional_edges("ask_continue", should_continue, {
    "ask_question": "ask_question",
    END: END
})
```

```
<langgraph.graph.state.StateGraph at 0x2a4db474110>
```

Section

Adaptive RAG 실습

Adaptive RAG

```python
# 11. 실행
session_id = str(uuid.uuid4())
config = {"configurable": {"session_id": session_id}}
app = graph.compile()
app.invoke({}, config=config)
```

```
질문을 입력해주세요: MES 공정 로그 보관 기간은?
[분석 결과] 선택된 전략: default

[단순 RAG] 질문: MES 공정 로그 보관 기간은?
답변: 6개월간

계속하시겠습니까? (예/아니오): 예
질문을 입력해주세요: 해당 기간이 지나면 어떻게 되니?
[분석 결과] 선택된 전략: iterative

[반복 RAG] 불충분한 답변, 재질문 시도 중...
[재질문] 다음 질문을 더 구체적으로 바꿔주세요: 해당 기간이 지나면 어떻게 되니?"
"그건 정말 궁금하네요,"
"내가 왜 그런걸 물어보는 거지?"
"내 대답은 틀렸다고 생각해요."
"이럴 수도 있다는 뜻이죠."
"정확히 그건 아니겠죠?"
"그런데 당신은 그 사람 기억능력이 좋은가요?"
"

계속하시겠습니까? (예/아니오): 아니오
{'question': '다음 질문을 더 구체적으로 바꿔주세요: 해당 기간이 지나면 어떻게 되니?'
생각해요."\n"이럴 수도 있다는 뜻이죠."\n"정확히 그건 아니겠죠?"\n"그런데 당신은 그 
 'history': InMemoryChatMessageHistory(messages=[HumanMessage(content='MES 공정
(content='6개월간', additional_kwargs={}, response_metadata={}), HumanMessage(c
data={})]),
 'strategy': 'iterative',
 'attempt': 2,
 'continue': False}
```

감사합니다.

Q & A