생성형 AI 기반 RAG 개발자 과정

RAG Chain 구축
출처 인용 · 검증 · 평가

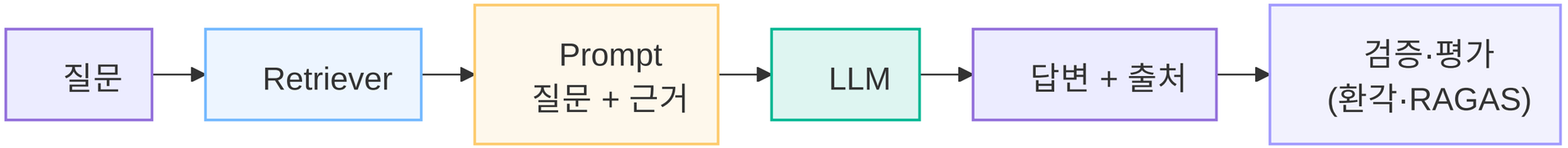
Session 16 / 33 — Day 2

검색 결과로 답을 만들고 → 출처 달고 → 검증·평가한다

박수현 · (사)한국정보공학기술사회 · 2026

LLM 이론 · LangChain · ChromaDB CRUD · Agent — 4일 핸즈온

Retriever | Prompt | LLM, 그리고 신뢰 장치



체인 골격 (LCEL)

```
chain = (
  {"context": retriever,
   "question": RunnablePassthrough()}
  | prompt | llm | StrOutputParser()
)
```

오늘 더하는 신뢰 장치

- **출처 인용** (Citation)
- **환각 검증** (Hallucination)
- **정량 평가** (RAGAS)

🔑 검색 결과를 프롬프트에 결합해 답을 생성하는 골격 위에 **신뢰성 3종**을 더한다.

PART B

PART B
표준 RAG 체인 — Day 2의 종착점

> 부품을 LCEL 한 줄로 잇고 환각을 막는다 — 약 15분

질문 → Retriever → context 결합 → Prompt → LLM → 답변+출처

① format_docs 로 context 결합

```
def format_docs(docs):
    return "\n\n".join(
        d.page_content for d in docs)
```

② 환각 가드 프롬프트

```
prompt = ChatPromptTemplate.from_messages([
    ("system",
     "아래 문서만 참고해서 답하세요. "
     "문서에 없으면 '모르겠습니다'라고 "
     "답하세요.\n\n문서:\n{context}"),
    ("user", "{question}"),
])
```

③ LCEL 체인 (표준형)

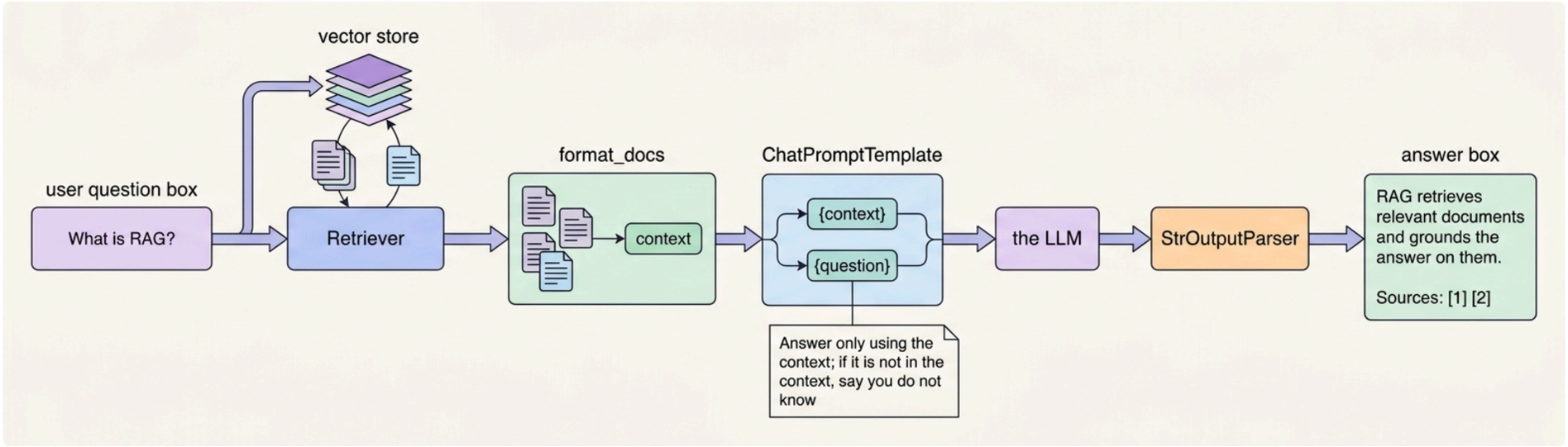
```
chain = (
    RunnablePassthrough.assign(
        context=lambda x: format_docs(
            retriever.invoke(x["question"])))
    | prompt
    | llm                  # temperature=0
    | StrOutputParser()
)

print(chain.invoke(
    {"question": "NVMe 와 SATA 차이?"}))
```

- `assign` 이 `question` 보존 + `context` 추가
- 문서에 없는 질문 → "모르겠습니다"

🔑 환각 방지의 핵심 = **"문서만 참고, 없으면 모른다"는 프롬프트 지시 +** `temperature=0`. 모르는 것을 모른다고 답하는 것이 신뢰성의 첫 조건.

질문 → Retriever → context → Prompt → LLM → 답변



RAG 체인 · 질문→Retriever→context 결합→ChatPromptTemplate({context},{question})→LLM→답변+출처 · '문서에 없으면 모른다' 환각 가드

PART C

PART C
Citation + Hallucination 검증

> 출처 인용이 신뢰성의 기반 — 약 15분

답변에 출처를 다는 스타일 (1·2)

패턴 1 — Inline citation (각 문장 끝)

```
RAG 는 검색을 통해 LLM 응답을 보강하는 패턴입니다 [doc1 p.3].
첫 단계는 임베딩으로 문서를 벡터로 변환하는 것입니다 [doc1 p.5][doc2 p.12].
```

패턴 2 — Footnote / Reference

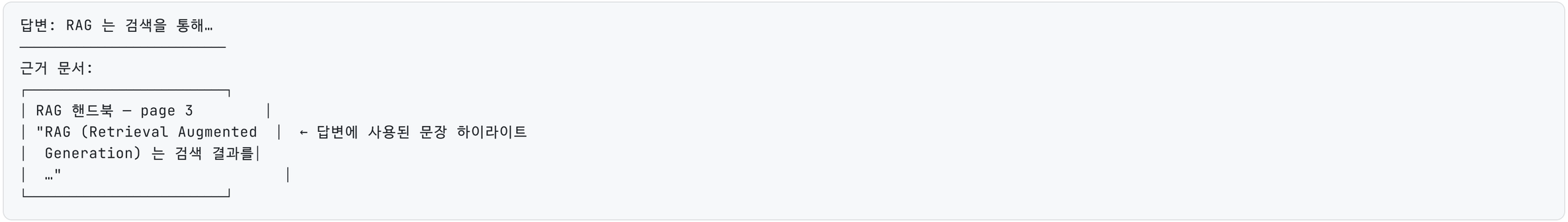
```
RAG 는 검색을 통해 LLM 응답을 보강하는 패턴입니다.
첫 단계는 임베딩으로 문서를 벡터로 변환하는 것입니다.

참고:
[1] RAG 핸드북, page 3
[2] RAG 핸드북, page 5
[3] 벡터DB 입문, page 12
```

> 🎯 **본 코스 권장**: 패턴 1 (inline) — 한 문장씩 책임 분리, 사용자 검증 쉬움.

답변에 출처를 다는 스타일 (3) — UI 하이라이트

패턴 3 — Source highlighting (UI)



> 🎯 답변에 실제로 쓰인 문장을 원문에서 강조 — 사용자가 근거를 한눈에 대조.

LLM 이 인용 마커를 직접 생성하도록

프롬프트 + 도큐먼트 라벨링

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate

CITATION_PROMPT = ChatPromptTemplate.from_messages([
    ("system", """다음 근거 문서들로만 답변하세요.
각 문장 끝에 사용한 문서의 인용 마커 `[doc N]`
을 반드시 붙이세요. 근거에 없으면 "모릅니다".

근거 문서:
{context}
"""),
    ("user", "{question}"),
])

llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)

def format_with_ids(docs):
    """[doc 1] [doc 2] 형식으로 라벨링."""
    return "\n\n".join(
        f"[doc {i+1}] (출처: "
        f"{d.metadata.get('source','?')} "
        f"p.{d.metadata.get('page','?')})\n"
        f"{d.page_content}"
        for i, d in enumerate(docs)
    )
```

RAG 체인 + 후처리

```
def rag_with_citation(question: str):
    docs = retriever.invoke(question)
    context = format_with_ids(docs)
    answer = (CITATION_PROMPT | llm).invoke(
        {"context": context, "question": question})

    # 후처리 — [doc N] 마커를 실제 source 로 치환
    import re
    def expand_citation(m):
        idx = int(m.group(1)) - 1
        d = docs[idx]
        return (f"[{d.metadata.get('source','?')}"
                f" p.{d.metadata.get('page','?')}]")

    final = re.sub(r"\[doc (\d+)\]",
                   expand_citation, answer.content)
    return final, docs

answer, sources = rag_with_citation(
    "RAG 의 핵심 단계는?")
print(answer)
# → "RAG 는 색인 → 검색 → 답변 3단계
#    로 진행됩니다 [rag.md p.3]."
```

출처를 인용해도 RAG 가 사실과 다른 답을 내는 4가지 원인

원인	증상	빈도
출처에 답이 없음	검색은 성공했으나 해당 chunk 에 정답이 없는데도 LLM 이 생성	흔함
여러 출처 정보 혼합	doc1 의 사실과 doc2 의 사실을 잘못 결합	흔함
시점 불일치	2020년 자료와 2024년 자료가 충돌, 최신 판단 실패	중간
모델 일반 지식 누수	출처에 없는 내용을 학습된 일반 지식으로 답변	흔함 (특히 유명 주제)

검증 패턴 3가지

1. **Self-check** — LLM 에게 "답변이 출처에 포함되는가?" 다시 물음
2. **NLI 모델** — Natural Language Inference 로 entailment 검증
3. **Citation density** — 답변 문장 수 vs 인용 수 (1:1 이상 권장)

Self-check 한 줄 프롬프트

```
verify_prompt = """다음 답변이 주어진 근거 문서에서 도출 가능한가?
가능하면 "YES", 불가능하면 "NO" 와 함께 어떤 부분이 근거에 없는지 1문장.

근거: {context}
답변: {answer}
"""
```

Natural Language Inference 모델로 사실 검증

```
# pip install transformers
from transformers import pipeline

nli = pipeline("text-classification", model="cross-encoder/nli-deberta-v3-base")

def check_entailment(premise: str, hypothesis: str) → dict:
    """premise (근거) → hypothesis (답변 문장) 가 entailed 인지."""
    result = nli({"text": premise, "text_pair": hypothesis})
    return result[0]   # {'label': 'ENTAILMENT'/'CONTRADICTION'/'NEUTRAL', 'score': 0.xx}

answer_sentences = answer.split(". ")
context_full = "\n".join(d.page_content for d in source_docs)

for sent in answer_sentences:
    if not sent.strip():
        continue
    result = check_entailment(premise=context_full, hypothesis=sent)
```

출력 예

```
✅ [ENTAILMENT 0.92] RAG 는 색인 → 검색 → 답변 3단계로 진행됩니다
⚠️ [NEUTRAL 0.54]    매주 1회 재색인이 권장됩니다              ← 출처에 없는 정보
✅ [ENTAILMENT 0.88] 임베딩 모델 교체 시 재색인이 필요합니다
```

> 🎯 **운영 패턴**: NEUTRAL 또는 CONTRADICTION 점수 30% 넘으면 답변을 "확신할 수 없음" 으로 강등하거나 사용자에게 경고.

PART D

PART D
RAG 평가 — 5대 메트릭

> 숫자로 측정해야 개선 가능 — 약 15분

Retrieval 측정 vs Generation 측정

메트릭	측정 대상	의미
Recall@K	Retrieval	정답 chunk 가 top-K 안에 들어왔는가
MRR (Mean Reciprocal Rank)	Retrieval	정답이 K개 중 몇 번째 — 1/rank 평균
Context Precision	Retrieval+LLM	top-K 중 실제로 답변에 쓰인 비율
Faithfulness	Generation	답변이 출처에 충실한가 (hallucination 없음)
Answer Relevancy	Generation	답변이 질문에 실제로 답하는가

측정 위치 한눈에

```
질의 → [Retrieval] → top-K → [Generation] → 답변
        ↑ Recall@K·MRR          ↑ Faithfulness·Answer Relevancy
        Context Precision 은 두 단계 사이
```

평가 데이터셋

- **Golden set** = (질의, 정답 chunk IDs, 정답 답변) 50~200건
- 자체 구축: LLM 으로 "이 문서에서 생성 가능한 질문 5개" 추출 후 사람 검수
- 공개 데이터: BEIR, MS MARCO, RAGAS-Wiki

Retrieval 측정

Recall@K

$$ \text{Recall@K} = \frac{|{\text{정답}} \cap {\text{top-K}}|}{|{\text{정답}}|} $$

- 질의 한 개당 0 또는 1 (정답이 K개 중 있으면 1)
- 여러 정답이 있으면 비율
- **K=5 가 95% 이상** 이면 retrieval 양호

MRR (Mean Reciprocal Rank)

$$ \text{MRR} = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank}_q} $$

- 정답 chunk 의 **순위의 역수** 평균
- 1위면 1, 2위면 0.5, 10위면 0.1
- 0.7 이상이면 좋음

직접 측정 코드

```
golden = [
    {"query": "RAG 의 본질은?",
     "answer_chunk_ids": ["c_001", "c_042"]},
    # ... 100개
]

def evaluate_retrieval(retriever, golden, k=5):
    recalls, rrs = [], []
    for item in golden:
        results = retriever.invoke(item["query"])
        retrieved_ids = [d.metadata["chunk_id"]
                         for d in results[:k]]
        golden_set = set(item["answer_chunk_ids"])

        # Recall@K
        hit = len(set(retrieved_ids) & golden_set) \
              / len(golden_set)
        recalls.append(hit)

        # MRR — 첫 정답의 역수
        for rank, rid in enumerate(retrieved_ids, 1):
            if rid in golden_set:
                rrs.append(1 / rank); break
        else:
            rrs.append(0)
    return (sum(recalls)/len(recalls),
            sum(rrs)/len(rrs))

r, m = evaluate_retrieval(retriever, golden)
print(f"Recall@5 = {r:.2f}, MRR = {m:.2f}")
```

Generation 측정 — LLM-as-Judge

Faithfulness — 답변이 출처에 충실한가 (0~1)

```
주어진 답변 = 5개 문장
각 문장이 context 에 entailed 되는가?
→ entailed 비율 = faithfulness 점수
```

```
def faithfulness(answer, context, llm) → float:
    """LLM 으로 답변 문장 각각 검증."""
    prompt = f"""답변의 각 문장이 context 에서
도출 가능한지 yes/no:

Context: {context}
답변: {answer}

각 문장마다 줄 단위로 "yes" 또는 "no" 만 출력.
"""
    result = llm.invoke(prompt).content
    decisions = [l.strip().lower()
                 for l in result.split("\n") if l.strip()]
    if not decisions:
        return 0.0
    return decisions.count("yes") / len(decisions)
```

Answer Relevancy — 질문 ↔ 답변 적합도

```
def answer_relevancy(question, answer,
                     llm, embeddings) → float:
    """답변에서 LLM 이 역추론한 질문 N개를
    원 질문과 임베딩 유사도로 비교."""
    reverse_prompt = f"""다음 답변이 주어졌을 때,
가능한 사용자 질문 3개를 만드세요:

답변: {answer}

질문 3개를 한 줄씩 출력.
"""
    generated_questions = llm.invoke(
        reverse_prompt).content.split("\n")
    q_emb = embeddings.embed_query(question)
    sims = [
        cosine_sim(q_emb, embeddings.embed_query(gq))
        for gq in generated_questions if gq.strip()
    ]
    return sum(sims) / len(sims) if sims else 0.0
```

> 🎯 두 메트릭 모두 **LLM-as-Judge** — 자동화 가능하지만 LLM 호출 비용 발생.

`pip install ragas` — 5대 메트릭 한 번에

평가 데이터 + 실행

```
from datasets import Dataset
from ragas import evaluate
from ragas.metrics import (
    context_precision, context_recall,
    faithfulness, answer_relevancy,
    answer_correctness,
)

# 평가 데이터 — (질의, RAG 답변, context, 정답)
data = {
    "question":     [i["query"]      for i in golden],
    "answer":       [i["rag_answer"] for i in golden],
    "contexts":     [i["retrieved"]  for i in golden],
    "ground_truth": [i["true_answer"] for i in golden],
}
ds = Dataset.from_dict(data)

result = evaluate(
    dataset=ds,
    metrics=[
        context_precision,
        context_recall,
        faithfulness,
        answer_relevancy,
        answer_correctness,
    ],
)
print(result)
```

결과 + 메트릭 진단

```
# {
#   'context_precision':  0.82,
#   'context_recall':     0.78,
#   'faithfulness':       0.91,
#   'answer_relevancy':   0.85,
#   'answer_correctness': 0.74,
# }
```

메트릭이 알려주는 진단

낮은 메트릭	의심 원인	대응
context_recall ↓	청킹 잘못 / 임베딩 약함	chunk_size 조정, 모델 교체
context_precision ↓	top-K 너무 큼	reranker 추가, K 축소
faithfulness ↓	LLM 이 hallucinate	프롬프트 강화, temperature ↓
answer_relevancy ↓	프롬프트 부적합	시스템 프롬프트 다듬기

PART E

PART E
미니 실습

> 검색 + reranker + 평가 — 약 30분

실습 과제 ① (20분) — 측정 준비

1단계 — Golden set 구축 (10분)

- 색인해 둔 문서로 **질문 10개** 작성
- 각 질문마다 답이 들어있는 chunk_id 1~3개 표시
- LLM 에게 "이 chunk 로 만들 수 있는 질문 3개" 자동 생성도 OK

2단계 — 3가지 retrieval 비교 (10분)

```
# A. Similarity (baseline)
retriever_sim = vectorstore.as_retriever(
    search_kwargs={"k": 5})

# B. MMR
retriever_mmr = vectorstore.as_retriever(
    search_type="mmr",
    search_kwargs={"k": 5, "fetch_k": 20,
                   "lambda_mult": 0.5},
)

# C. BM25 + Vector hybrid
bm25 = BM25Retriever.from_documents(all_chunks)
bm25.k = 5
retriever_hybrid = EnsembleRetriever(
    retrievers=[bm25, retriever_sim],
    weights=[0.4, 0.6])

# 각각 Recall@5 측정
for name, r in [("sim", retriever_sim),
                ("mmr", retriever_mmr),
                ("hybrid", retriever_hybrid)]:
    recall, mrr = evaluate_retrieval(r, golden)
    print(f"{name}: Recall@5={recall:.2f}"
          f"  MRR={mrr:.2f}")
```

실습 과제 ② (10분) — 답변 검증 + 보고

3단계 — Citation 답변 생성 (5분)

- 위 슬라이드의 `rag_with_citation()` 적용
- 질문 3개에 출처 인용 답변 확인

4단계 — Faithfulness 측정 (5분)

- RAGAS 또는 직접 self-check 프롬프트
- 점수 0.7 미만 답변 1개 찾아 어디가 hallucination 인지 분석

📤 결과 보고

- Sim / MMR / Hybrid 중 대상 도메인에 가장 적합한 것 + 한 줄 이유
- Faithfulness 가 낮았던 답변 사례 1건

평가 기준

- Recall@5 0.7 이상 = 양호
- Faithfulness 0.85 이상 = 운영 가능
- 두 메트릭이 모두 낮으면 → chunk_size 또는 system prompt 부터 재검토

핵심 6가지

1. ✅ RAG Chain 골격 = **Retriever → format_docs(context) → Prompt → LLM** (LCEL)
2. ✅ **환각 가드** — "문서에 없으면 모른다" 프롬프트 + `temperature=0` 으로 RAG 파이프라인 완성
3. ✅ 자동 인용 = **프롬프트 지시 + 후처리**
4. ✅ **Hallucination** 검증 — NLI 로 근거-답변 일치 확인
5. ✅ 평가 5대 지표 — Recall@K·MRR·Faithfulness·Answer Relevancy·Context Precision
6. ✅ **RAGAS** 로 RAG 품질을 정량 측정