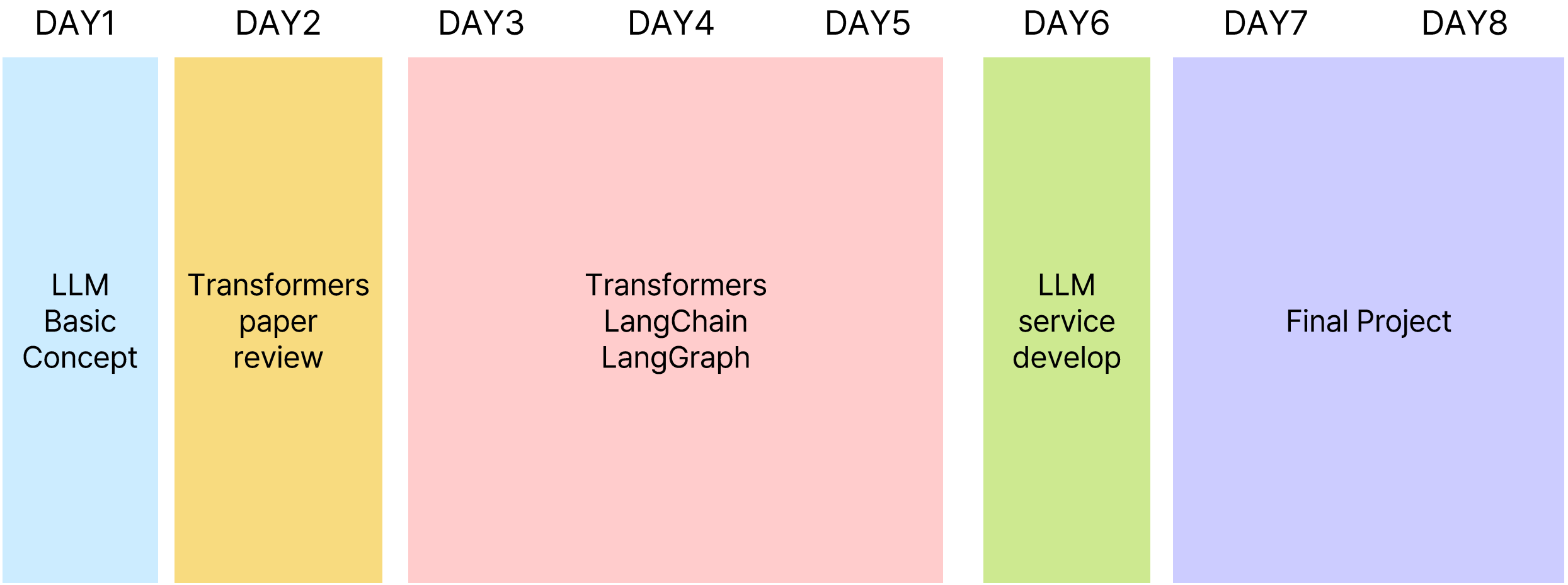
GEN AI 인텐시브 과정

강사 장철원

Section 0

코스소개



Section 0

목차

- ❑ LangChain 라이브러리 소개
- ❑ LangChain + RAG + Pandas
- ❑ LangChain + RAG + chromaDB
- ❑ LangChain + RAG + chromaDB + GEN

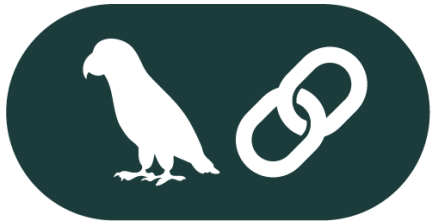
GEN AI 인텐시브 과정

Section 1. LangChain 실습

Section 1-1. LangChain 프레임워크 소개

LangChain

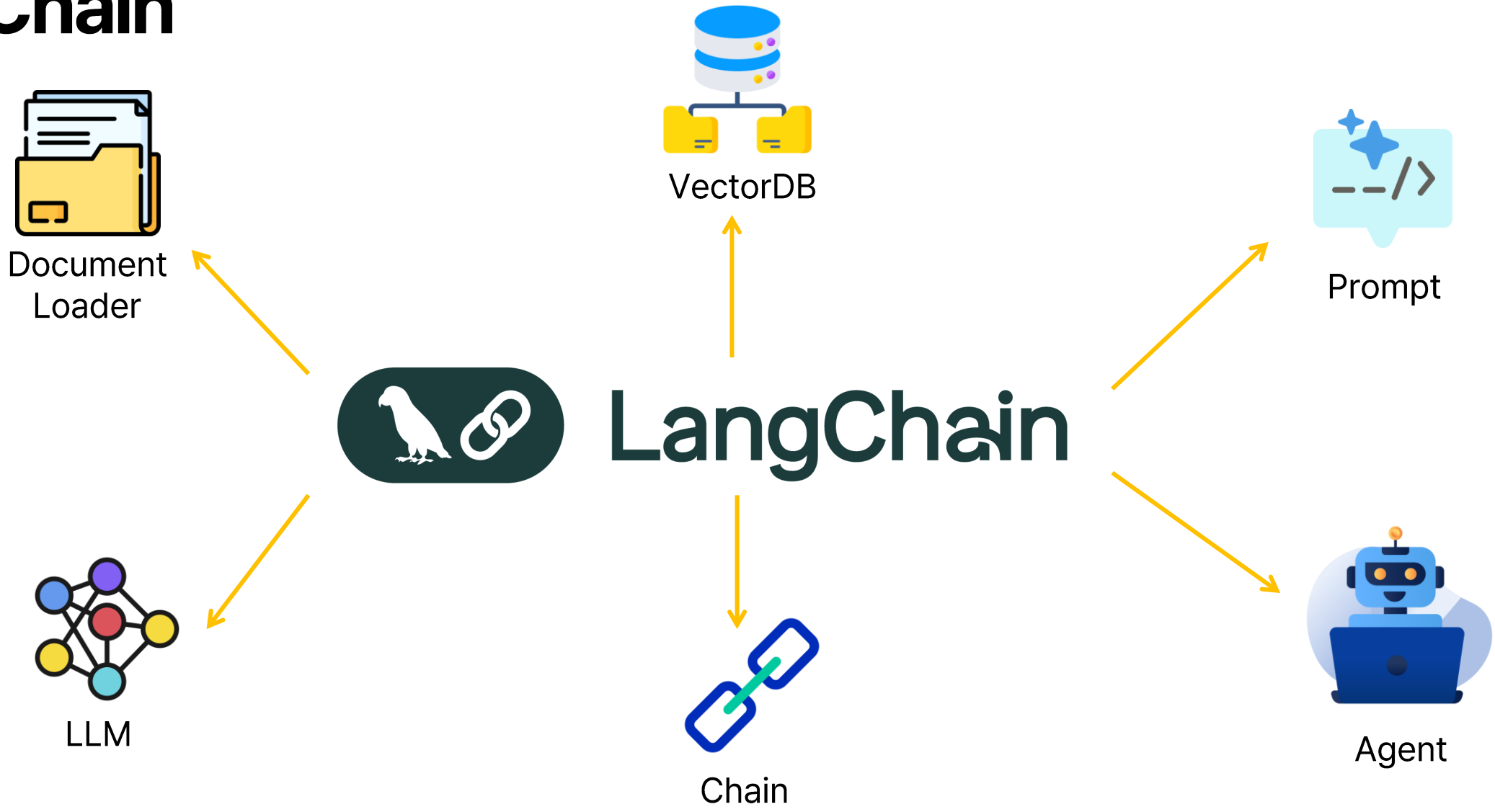
- LLM을 활용한 애플리케이션 개발을 도와주는 프레임워크
- LLM, 데이터 소스, Tools, API, 사용자 입력 등 다양한 요소들을 연결해주는 통합 도구 세트



LangChain

LLM을 실제 애플리케이션으로 만들기 위해 사용하는 도구

LangChain


VectorDB
Document Loader
Prompt
LangChain
LLM
Chain
Agent

LangChain의 구성 요소

구성 요소	설명
LLM	작업을 수행할 LLM 모델, 본 실습에서는 transformer 라이브러리로 불러와서 사용 예정
PromptTemplate	다양한 변수 기반 프롬프트 템플릿 생성 도구
Memory	이전 대화 맥락을 기억해서 대화 지속성 부여
Chains	LLM 과 프롬프트를 연결하는 작업 흐름
Tools	계산기, 웹 검색, 데이터베이스 조회 등 LLM이 쓸 수 있는 외부도구
Agents	상황에 따라 도구를 선택하고 반복해서 추론하는 지능적인 LLM 구조
Document Loader	PDF, 웹페이지, CSV 등 다양한 외부 문서 불러오기
Vector Store / Retriever	문서 검색 기반 QA RAG 개발시 사용(FAISS, Chroma 등과 연동)

transformers vs LangChain

비교 항목	transformers	langchain
역할	LLM 모델 활용	LLM을 활용한 앱, 시스템 구축
개발 수준	낮은 수준(Low level)	높은 수준(High level)
기능	모델 불러오기, 토크나이징, 추론	프롬프트 설계, 문서 검색, 멀티 스텝 추론, 메모리 등
비유	자동차 엔진	자동차

GEN AI 인텐시브 과정

Section 1. LangChain 실습

Section 1-2. LangChain 헬로 월드 (text-generation)

실습 내용

라이브러리 & 모델 불러오기

```
from langchain_huggingface import HuggingFacePipeline
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
```

트랜스포머의 pipeline을 랭체인 용으로 래핑

```
# 토크나이저 & 모델 불러오기
model_id = "skt/kogpt2-base-v2"
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(model_id)
```

GPT계열과 같이 문장을 생성하는 모델 불러올때 사용

모델 정보

- 제작자: SK텔레콤
- 베이스: GPT2
- 용도: 문장 생성

Section

LangChain 헬로 월드(text-generation)

파이프라인 생성

https://huggingface.co/docs/transformers/main_classes/pipelines

위 페이지 참고

```
# 파이프라인 생성
text_gen = pipeline(
    task="text-generation",
    model=model,
    tokenizer=tokenizer,
    max_length=100,
    truncation=True,
    do_sample=True,
    temperature=0.7,
    device=-1
)
```

```
Device set to use cpu
```

수행할 작업 종류 지정

사용할 모델

토크나이저 설정

함께 사용

-1: CPU, 0: 0번째 GPU

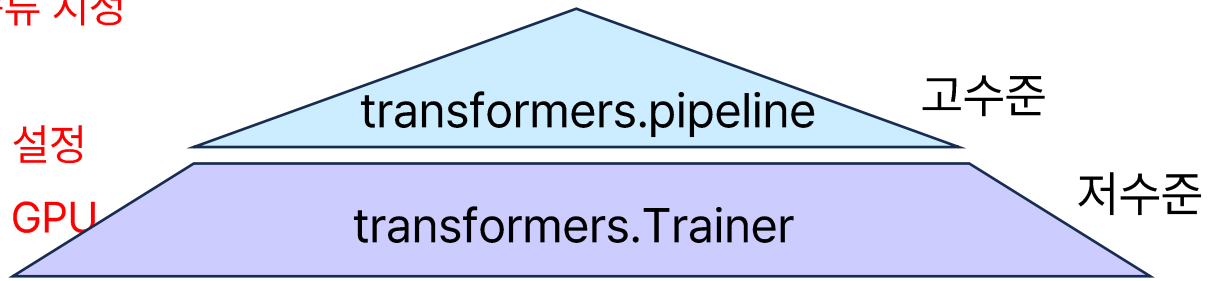
길면 자르겠다.

함께 사용

텍스트 생성 시 무작위 샘플링 허용 여부, False로 설정하면 항상 확률이 가장 높은 단어만 선택

보수적으로 절대 틀리면 안되는 경우, False

샘플링할 때 무작위성을 얼마나 줄 것인가
0~0.3: 보수적, 확률 높은 단어만 선택
0.7: 기본, 적당한 창의성
1 이상 : 무작위성이 강해지며, 창의적인 문장 생성



Section

LangChain 헬로 월드(text-generation)

수행 가능 작업 https://huggingface.co/docs/transformers/main_classes/pipelines

Parameters

- **task** (`str`) — The task defining which pipeline will be returned. Currently accepted tasks are:
  - `"audio-classification"`: will return a AudioClassificationPipeline.
  - `"automatic-speech-recognition"`: will return a AutomaticSpeechRecognitionPipeline.
  - `"depth-estimation"`: will return a DepthEstimationPipeline.
  - `"document-question-answering"`: will return a DocumentQuestionAnsweringPipeline.
  - `"feature-extraction"`: will return a FeatureExtractionPipeline.
  - `"fill-mask"`: will return a FillMaskPipeline:.
  - `"image-classification"`: will return a ImageClassificationPipeline.
  - `"image-feature-extraction"`: will return an ImageFeatureExtractionPipeline.
  - `"image-segmentation"`: will return a ImageSegmentationPipeline.
  - `"image-text-to-text"`: will return a ImageTextToTextPipeline.
  - `"image-to-image"`: will return a ImageToImagePipeline.
  - `"image-to-text"`: will return a ImageToTextPipeline.
  - `"mask-generation"`: will return a MaskGenerationPipeline.
  - `"object-detection"`: will return a ObjectDetectionPipeline.
  - `"question-answering"`: will return a QuestionAnsweringPipeline.
  - `"summarization"`: will return a SummarizationPipeline.
  - `"table-question-answering"`: will return a TableQuestionAnsweringPipeline.
  - `"text2text-generation"`: will return a Text2TextGenerationPipeline.
  - `"text-classification"` (alias `"sentiment-analysis"` available): will return a TextClassificationPipeline.
  - `"text-generation"`: will return a TextGenerationPipeline:.
  - `"text-to-audio"` (alias `"text-to-speech"` available): will return a TextToAudioPipeline:.
  - `"token-classification"` (alias `"ner"` available): will return a TokenClassificationPipeline.
  - `"translation"`: will return a TranslationPipeline.
  - `"translation_xx_to_yy"`: will return a TranslationPipeline.
  - `"video-classification"`: will return a VideoClassificationPipeline.
  - `"visual-question-answering"`: will return a VisualQuestionAnsweringPipeline.
  - `"zero-shot-classification"`: will return a ZeroShotClassificationPipeline.
  - `"zero-shot-image-classification"`: will return a ZeroShotImageClassificationPipeline.
  - `"zero-shot-audio-classification"`: will return a ZeroShotAudioClassificationPipeline.
  - `"zero-shot-object-detection"`: will return a ZeroShotObjectDetectionPipeline.

LangChain

```
# LangChain 연결
llm = HuggingFacePipeline(pipeline=text_gen)
```

트랜스포머의 pipeline을 랭체인 용으로 래핑

```
# 프롬프트 예시
prompt = "산 속에 토끼 한 마리가 살고 있었습니다. 그러던 어느 날 "

# LangChain으로 실행
response = llm.invoke(prompt)
```

```
print("생성된 문장:", response)
```

```
생성된 문장: 산 속에 토끼 한 마리가 살고 있었습니다. 그러던 어느 날 한 남자가 나타나 토끼 한 마리를 잡으러 갑니다. 그는 토끼의 얼굴을 살펴보았습니다. 토끼는 마치 한 사람 같았습니다. 그러자 그는 토끼의 얼굴과 팔, 가슴을 보고
```

GEN AI 인텐시브 과정

Section 1. LangChain 실습

Section 1-3. LangChain 헬로 월드 (text-generation) - 간단한 버전

실습 내용

Section

LangChain 헬로 월드(text-generation)

실습

[1]:
```
from langchain_huggingface import HuggingFacePipeline
from transformers import pipeline
```

[2]:
```
# 모델명 설정
model_id = "skt/kogpt2-base-v2"
```

[3]:
```
# 파이프라인 생성
text_gen = pipeline(
    task="text-generation",
    model=model_id,
    max_length=100,
    truncation=True,
    do_sample=True,
    temperature=0.7,
    device=-1
)
```

```
Device set to use cpu
```

[4]:
```
# LangChain 연결
llm = HuggingFacePipeline(pipeline=text_gen)
```

[5]:
```
# 프롬프트 예시
prompt = "산 속에 토끼 한 마리가 살고 있었습니다. 그러던 어느 날 "

# LangChain으로 실행
response = llm.invoke(prompt)
```

[6]:
```
print("생성된 문장:", response)
```

```
생성된 문장: 산 속에 토끼 한 마리가 살고 있었습니다. 그러던 어느 날 뚱뚱한 토끼가 나타나 토끼의 집에 찾아왔습니
집 문을 두드렸지요. 그 때 토끼는 그 자리에서 뚱뚱한 토끼에게 말했습니다. 그러자 뚱뚱한 토끼는 그 자리에서 바로
달려왔습니다. 그 순간 뚱뚱한 토끼는 뚱뚱한 토끼의 집
```

GEN AI 인텐시브 과정

Section 1. LangChain 실습

Section 1-4. LangChain 헬로 월드 (text-generation) - langchain 라이브러리만 사용

Section

LangChain 헬로 월드(text-generation)

langchain만 사용

[1]:
```
from langchain_huggingface import HuggingFacePipeline
```

[2]:
```
# 모델명 설정
model_id = "skt/kogpt2-base-v2"
model_opt = {
    "do_sample": True,
    "temperature": 0.7,
}
pipeline_opt = {
    "max_length": 100,
    "truncation": True
}
```

[3]:
```
# 파이프라인 생성
llm = HuggingFacePipeline.from_model_id(
    task = "text-generation",
    model_id = model_id,
    model_kwargs = model_opt,
    pipeline_kwargs = pipeline_opt,
    device = -1
)
```

```
Device set to use cpu
```

[4]:
```
# 프롬프트 예시
prompt = "산 속에 토끼 한 마리가 살고 있었습니다. 그러던 어느 날 "
```

[5]:
```
# 실행
response = llm.invoke(prompt)
print("생성된 문장:", response)
```

```
생성된 문장: 산 속에 토끼 한 마리가 살고 있었습니다. 그러던 어느 날 덫에 걸려 있는 토끼가 토끼를 물었습니다.
러자 덫이 갑자기 뚝 끊겼습니다. 그러자 토끼는 그 자리에서 도망쳤습니다. 그러자 덫을 물었을 때 꼼짝도 하지 않
었습니다.
그런데 덫이 뚝 끊겼을 때, 그 덫의 밑바닥 아래에
```

GEN AI 인텐시브 과정

Section 1. LangChain 실습

Section 1-5. LangChain 헬로 월드 (question-answering)

Section

LangChain 헬로 월드(question-answering)

실습 내용

```
from transformers import AutoTokenizer, AutoModelForQuestionAnswering, pipeline
from langchain_huggingface import HuggingFacePipeline
```

```
# 모델 및 토크나이저 불러오기
model_id = "monologg/koelectra-base-v3-finetuned-korquad"
tokenizer = AutoTokenizer.from_pretrained(model_id)
tokenizer_opt = {
        "max_length": 512,
        "truncation": True
}
model = AutoModelForQuestionAnswering.from_pretrained(model_id)
```

토크나이저 설정

```
# QA 파이프라인
qa_pipeline = pipeline(
    task="question-answering",
    model = model,
    tokenizer = tokenizer,
    tokenizer_kwargs = tokenizer_opt,
    device = -1
)
```

앞서 사용한 HuggingPacePipeline은 text-generation용으로 개발되어 QA용으로는 사용하지 않음

```
Device set to use cpu
```

```
# 입력
question = "세종대왕은 어떤 업적을 남겼나요?"
context = (
    "새종대왕은 조선의 네 번째 왕으로, 한글을 창제하고 과학, 음악, 농업 등 다양한 분야에서 업적을 남겼습니다."
    "그는 집현전을 설치하고 학문을 장려하였으며, 측우기와 해시계 등의 과학 기구 개발을 지원했습니다."
)
```

```
response = qa_pipeline(question=question, context=context)
print("답변: ", response["answer"])
```

```
답변:  과학, 음악, 농업 등
```

Section

LangChain 헬로 월드(question-answering)

간단한 버전

[1]:
```
from transformers import pipeline
from langchain_huggingface import HuggingFacePipeline
```

[2]:
```
# 모델 및 토크나이저 불러오기
model_id = "monologg/koelectra-base-v3-finetuned-korq
tokenizer_opt = {
        "max_length": 512,
        "truncation": True
}
```

model 에 따라, langchain 도 transformer 가 필요하다. - > koelectra는 필요, kogpt2는 불 필요.

[3]:
```
# QA 파이프라인
qa_pipeline = pipeline(
    task="question-answering",
    model = model_id,
    tokenizer_kwargs = tokenizer_opt,
    device = -1
)
```

koelectra와 같은 QA 전용 모델은
transformers 라이브러리를 사용하는게 좋음

```
Device set to use cpu
```

[4]:
```
# 입력
question = "세종대왕은 어떤 업적을 남겼나요?"
context = (
    "새종대왕은 조선의 네 번째 왕으로, 한글을 창제하고 과학, 음악, 농업 등 다양한 분야에서 업적을 남겼습니다."
    "그는 집현전을 설치하고 학문을 장려하였으며, 측우기와 해시계 등의 과학 기구 개발을 지원했습니다."
)
```

[5]:
```
response = qa_pipeline(question=question, context=context)
print("답변: ", response["answer"])
```

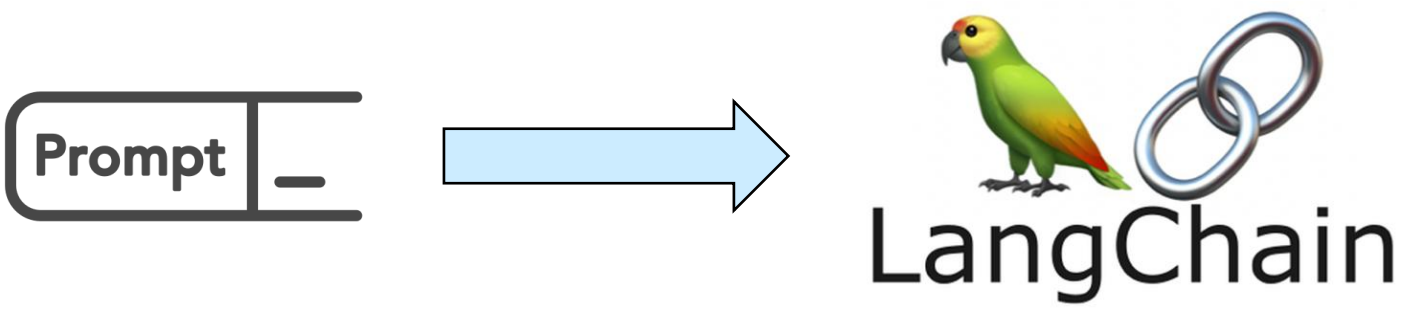
```
답변:  과학, 음악, 농업 등
```

GEN AI 인텐시브 과정

Section 1. LangChain 실습

Section 1-6. 프롬프트 사용

실습 내용



Section

프롬프트사용

LangChain

```
from langchain_huggingface import HuggingFacePipeline
from langchain_core.prompts import PromptTemplate
```

LangChain에서 프롬프트 템플릿 형식으로 만들때 사용

```
# 토크나이저 & 모델 불러오기
model_id = "skt/kogpt2-base-v2"

model_opt = {
    "do_sample": True,
    "temperature": 0.7,
}

pipeline_opt = {
    "max_length": 100,
    "truncation": True
}
```

```
# 파이프라인 생성
llm = HuggingFacePipeline.from_model_id(
    task = "text-generation",
    model_id = model_id,
    model_kwargs = model_opt,
    pipeline_kwargs = pipeline_opt,
    device=-1,
)
```

```
Device set to use cpu
```

Section

프롬프트사용

LangChain

```
template = "산 속에 {animal} 한 마리가 살고 있었습니다. 그러던 어느 날 "
prompt = PromptTemplate.from_template(template)
```

```
# LangChain으로 실행
chain = prompt | llm
```

gpt 질문을 형식없이 하기때문에, prompt chain 으로 보강

```
response = chain.invoke({"animal": "토끼"})
print(response)
```

```
산 속에 토끼 한 마리가 살고 있었습니다. 그러던 어느 날 녀석이 토끼를 찾아왔습니다. 토끼는 '꿈꾸는 토끼'였습니다. 그러자 녀석은 뭔가 이는 토끼를 보고 깜짝 놀라서 물었습니다. 녀석은 '꿈꾸는 토끼'였습니다. 녀석은 토끼를 보고 깜짝 놀라서 물었습니다. 녀석은 '꿈꾸는 토끼'였비웃고 말았습니다. '
```

GEN AI 인텐시브 과정

Section 1. LangChain 실습

Section 1-5. LangChain + RAG + from pandas

실습 내용

Section

LangChain + RAG + from pandas

LangChain

[1]:
```python
import pandas as pd
from sentence_transformers import SentenceTransformer, util
from transformers import AutoTokenizer, AutoModelForQuestionAnswering, AutoModelForCausalLM, pipeline
```

[2]:
```python
# 1. 모델 설정
embedding_id = "sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2"
qa_model_id = "monologg/koelectra-base-v3-finetuned-korquad"
tokenizer_opt = {
        "max_length": 512,
        "truncation": True
}
```

[3]:
```python
# 2. 문서 불러오기
df = pd.read_csv('./data/data.csv', encoding='utf-8')
df.head(10)
```

[3]:

	text
0	Etching 공정 전에는 반드시 세정 공정이 완료되어야 합니다.
1	PM 장비의 필터는 주 1회 정기적으로 교체해야 합니다.
2	웨이퍼 투입 전 챔버 내부의 온도 안정화가 필요합니다.
3	불량률이 2%를 초과할 경우 원인 분석 보고서를 제출해야 합니다.

LangChain

```python
# 3. 리스트로 변환
documents = df['text'].tolist()
```

```python
# 4. 임베딩
embedding_model = SentenceTransformer(embedding_id)
doc_embeddings = embedding_model.encode(documents, convert_to_tensor=True)
```

```python
# 5. KoELECTRA QA 모델 로딩
qa_tokenizer = AutoTokenizer.from_pretrained(qa_model_id)
qa_model = AutoModelForQuestionAnswering.from_pretrained(qa_model_id)
```

LangChain

```
# 6. 파이프라인 생성
text_gen = pipeline(
    "question-answering",
    model=qa_model,
    tokenizer=qa_tokenizer,
    tokenizer_kwargs = tokenizer_opt,
    max_length=512,
    truncation=True,
    do_sample=False,
    device=-1
)
```

QA 작업을 할 예정이므로 “question-answering” 이라고 입력

샘플링 금지(최대한 정확도 중심으로)

```
Device set to use cpu
```

LangChain

```python
# 6. 질문 입력 + 문서 검색
question = "클린룸 입장 시 주의사항 알려줄래?"
question_embedding = embedding_model.encode(question, convert_to_tensor=True)
cos_sim = util.cos_sim(question_embedding, doc_embeddings)[0]
best_idx = cos_sim.argmax().item()
best_context = documents[best_idx]
print("참고 문서: ", best_context)
```

```
참고 문서:  클린룸 입장 전에는 반드시 정전기 방지복을 착용해야 합니다.
```

```python
# 7. 결과 확인
result = text_gen(question=question, context=best_context)
print("정답:", result["answer"])
```

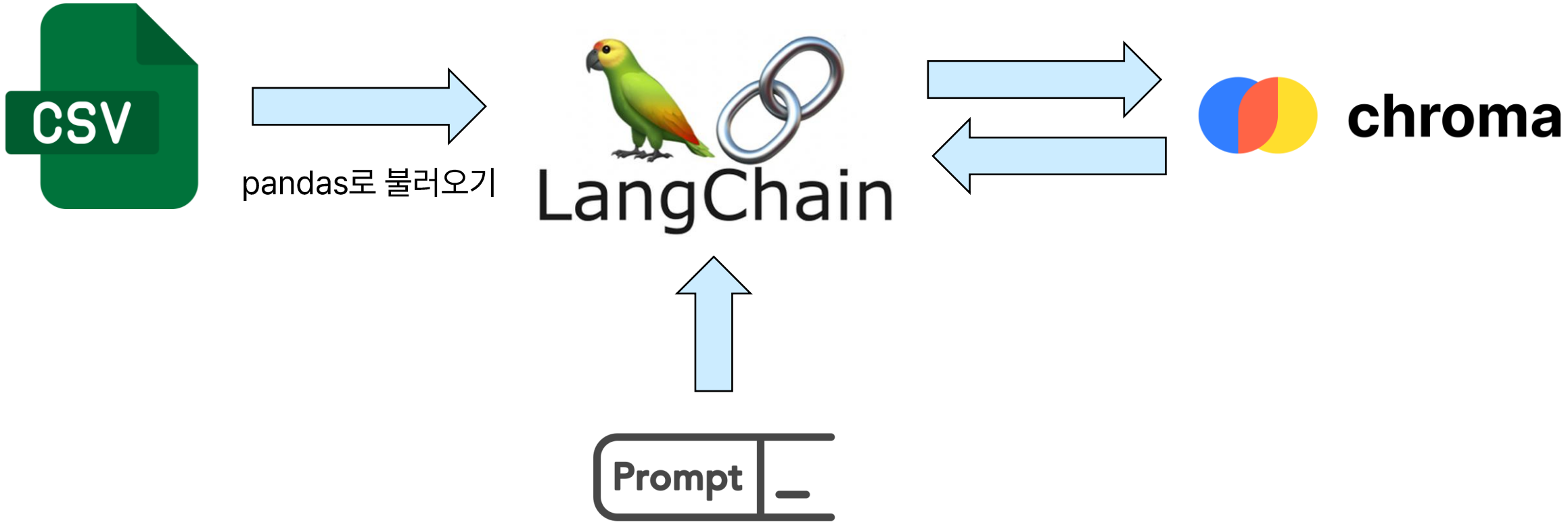
```
정답: 정전기 방지복을
```

GEN AI 인텐시브 과정

Section 1. LangChain 실습

Section 1-6. LangChain + RAG + from chromaDB

실습 내용



LangChain

```
import pandas as pd
from langchain_chroma import Chroma
from langchain.schema import Document
from langchain_huggingface import HuggingFaceEmbeddings
from transformers import AutoTokenizer, AutoModelForQuestionAnswering, pipeline
```

```
# 1. 설정
embedding_id = "sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2"
qa_model_id = "monologg/koelectra-base-v3-finetuned-korquad"
tokenizer_opt = {
        "max_length": 512,
        "truncation": True
}
```

LangChain

```python
# 2. 데이터 로딩
df = pd.read_csv("./data/data.csv", encoding="utf-8")
```

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

```
[Document(metadata={}, page_content='Etching 공정 전에는 반드시 세정 공정이 완료되어야 합니다.'),
 Document(metadata={}, page_content='PM 장비의 필터는 주 1회 정기적으로 교체해야 합니다.'),
 Document(metadata={}, page_content='웨이퍼 투입 전 챔버 내부의 온도 안정화가 필요합니다.'),
 Document(metadata={}, page_content='불량률이 2%를 초과할 경우 원인 분석 보고서를 제출해야 합니다.'),
 Document(metadata={}, page_content='클린룸 입장 전에는 반드시 정전기 방지복을 착용해야 합니다.'),
 Document(metadata={}, page_content='포토 공정 시 PR 코팅 두께는 1.5μm 이상 유지해야 합니다.'),
 Document(metadata={}, page_content='검사 장비는 매일 초기화 후 기능 점검을 실시합니다.'),
 Document(metadata={}, page_content='주간 생산 계획은 매주 월요일 오전 9시에 확정됩니다.'),
 Document(metadata={}, page_content='X선 검사 장비는 비정상 신호 발생 시 즉시 사용 중지합니다.'),
 Document(metadata={}, page_content='로더 설비의 진공 펌프는 월 1회 이상 윤활 상태를 점검합니다.'),
 Document(metadata={}, page_content='공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.'),
 Document(metadata={}, page_content='동일 Lot 내에서 불량이 5개 이상 발생하면 전수 검사 실시합니다,
 Document(metadata={}, page_content='물류 이송 로봇은 경로 장애 감지 시 자동 우회합니다.'),
 Document(metadata={}, page_content='이온 주입 공정 후 열처리 시간은 최소 30분 이상 확보합니다.'),
 Document(metadata={}, page_content='클린룸 청소는 일 2회, 장비 청소는 주 1회 이상 실시합니다.'),
 Document(metadata={}, page_content='신입 엔지니어는 공정별 교육을 모두 이수한 후 장비 조작이 가능합
 Document(metadata={}, page_content='공정 레시피 변경 시 Change History 문서 작성을 필수로 합니다.
 Document(metadata={}, page_content='수율 개선안은 월간 품질 회의에서 발표되어야 합니다.'),
```

LangChain

```python
# 4. 임베딩 모델
embedding_model = HuggingFaceEmbeddings(model_name=embedding_id)
```

```python
# 5. Chroma에 저장
save_vectordb = Chroma.from_documents(
    documents=docs,
    embedding=embedding_model,
    persist_directory="./chromaDB1"
)
```

```python
# 6. 저장된 Chroma 불러오기
load_vectordb = Chroma(
    persist_directory="./chromaDB1",
    embedding_function=embedding_model
)
```

LangChain

```
# 7. 질문 정의 + 문서 검색
question = "클린룸 입장 시 주의사항 알려줄래?"
retriever = load_vectordb.as_retriever(search_kwargs={"k": 3})
relevant_docs = retriever.invoke(question)
best_context = relevant_docs[0].page_content
print("참고 문서:", best_context)
```

→ 질문과 가장 유사한 top 3개 문장 선택 설정

```
참고 문서: 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.
```

```
relevant_docs
```

```
[Document(id='8eca18ef-0ec4-4d78-9997-32cd3c0f3558', metadata={}, page_content='공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.'),
 Document(id='ce7b02b0-9c11-4019-a3fe-fe8964c7e1d4', metadata={}, page_content='클린룸 입장 전에는 반드시 정전기 방지복을 착용해야 합니다.'),
 Document(id='84819b1f-c843-4414-90fa-df2b823f425e', metadata={}, page_content='장비 재가동 시 Warm-up 절차를 반드시 이행합니다.')]
```

LangChain

```python
# 8. KoELECTRA QA 모델 로딩
qa_tokenizer = AutoTokenizer.from_pretrained(qa_model_id)
qa_model = AutoModelForQuestionAnswering.from_pretrained(qa_model_id)
```

```python
# 9. 파이프라인
text_gen = pipeline(
    task="question-answering",
    model=qa_model,
    tokenizer=qa_tokenizer,
    max_length=512,
    truncation=True,
    do_sample=False,
    device=-1
)
```

```
Device set to use cpu
```

```python
# 10. QA 수행
result = text_gen(question=question, context=best_context)
print("정답:", result["answer"])
```

```
정답: 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다
```

GEN AI 인텐시브 과정

Section 1. LangChain 실습

Section 1-7. LangChain + RAG + from chromaDB(랭체인 위주)

Section

LangChain + RAG + from chromaDB

실습 내용

```python
import pandas as pd
from transformers import pipeline
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_chroma import Chroma
from langchain.schema import Document
from langchain_huggingface import HuggingFacePipeline
```

```python
# 1. 모델 설정
embedding_id = "sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2"
qa_model_id = "monologg/koelectra-base-v3-finetuned-korquad"
tokenizer_opt = {
    "max_length": 512,
    "truncation": True
}
```

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

```python
# 3. LangChain Document 객체로 변환
texts = df['text'].tolist()
documents = [Document(page_content=text) for text in texts]
```

```python
# 4. LangChain 임베딩
embedding_model = HuggingFaceEmbeddings(model_name=embedding_id)
```

Section

LangChain + RAG + from chromaDB

실습 내용

```
# 5. ChromaDB
save_vectordb = Chroma.from_documents(
    documents = documents,
    embedding = embedding_model,
    persist_directory = "./chroma_db"
)

load_vectordb = Chroma(
    persist_directory="./chroma_db",
    embedding_function=embedding_model
)
```

```
# 5. KoELECTRA QA 파이프라인
qa_pipeline = pipeline(
    task="question-answering",
    model=qa_model_id,
    tokenizer_kwargs=tokenizer_opt,
    device=-1
)
```

```
Device set to use cpu
```

실습 내용

```
# 6. 질문 입력
question = "클린룸 입장 시 주의사항 알려줄래?"
```

```
# 7. LangChain retriever로 관련 문서 검색
retriever = load_vectordb.as_retriever(search_kwargs={"k": 1})
relevant_docs = retriever.invoke(question)
best_context = relevant_docs[0].page_content
print("참고 문서:", best_context)
```

```
참고 문서: 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.
```

```
# 8. QA 실행
result = qa_pipeline(question=question, context=best_context)
print("정답:", result["answer"])
```

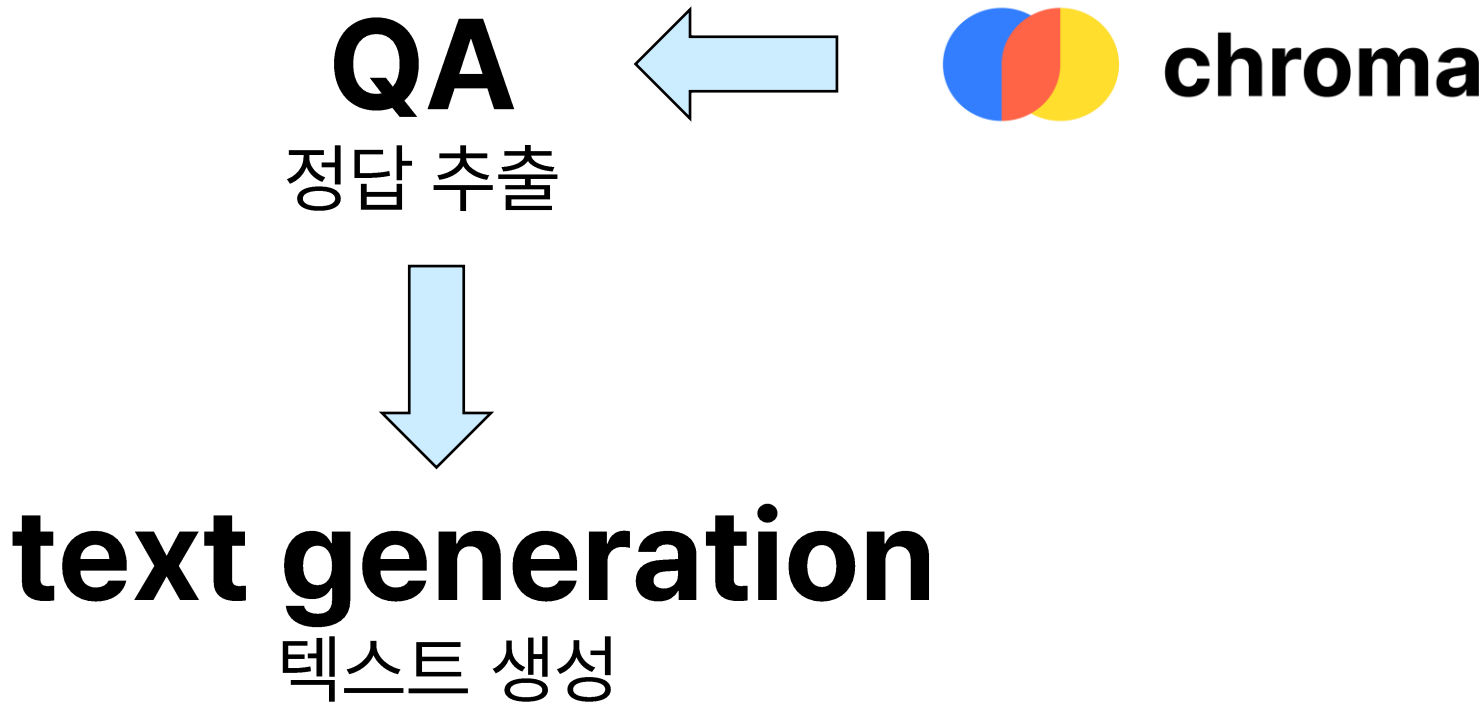
```
정답: 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다
```

GEN AI 인텐시브 과정

Section 1. LangChain 실습

Section 1-7. LangChain + RAG + from chromaDB + GEN

이번 실습에서는 RAG로 생성된 답변을 토대로 문장 생성



Section

LangChain + RAG + from chromaDB + GEN

LangChain

```
import pandas as pd
from langchain_chroma import Chroma
from langchain_huggingface import HuggingFaceEmbeddings
from transformers import AutoTokenizer, AutoModelForQuestionAnswering, AutoModelForCausalLM, pipeline
from langchain_core.prompts import PromptTemplate
from langchain_huggingface import HuggingFacePipeline
```

```
# 1. 모델 설정
embedding_id = "sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2"
qa_model_id = "monologg/koelectra-base-v3-finetuned-korquad"
qa_tokenizer_opt = {
    "max_length": 512,
    "truncation": True
}

gen_model_id = "EleutherAI/polyglot-ko-1.3b"
```

Section

LangChain + RAG + from chromaDB + GEN

LangChain

```python
# 1. 임베딩 모델
embedding_model = HuggingFaceEmbeddings(model_name=embedding_id)
```

```python
# 2. 저장된 ChromaDB 불러오기
load_vectordb = Chroma(
    persist_directory="./chromaDB1",
    embedding_function=embedding_model
)
```

이미 chromaDB에 데이터가 존재한다고 가정

```python
# 3. 질문 정의 + 문서 검색
question = "클린룸 입장 시 주의사항 알려줄래?"
retriever = load_vectordb.as_retriever(search_kwargs={"k": 3})
relevant_docs = retriever.invoke(question)
best_context = relevant_docs[0].page_content
print("참고 문서:", best_context)
```

```
참고 문서: 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.
```

Section

LangChain + RAG + from chromaDB + GEN

LangChain

```
# 4. KoELECTRA QA 모델 로딩
qa_tokenizer = AutoTokenizer.from_pretrained(qa_model_id)
qa_model = AutoModelForQuestionAnswering.from_pretrained(qa_model_id)
```

```
# 5. 파이프라인 & LangChain 래핑
text_gen = pipeline(
    task="question-answering",
    model=qa_model,
    tokenizer=qa_tokenizer,
    tokenizer_kwargs=qa_tokenizer_opt,
    max_length=512,
    truncation=True,
    do_sample=False,
    device=-1,
)
```

```
Device set to use cpu
```

```
# 6. QA 수행
result = text_gen(question=question, context=best_context)
print("정답:", result["answer"])
```

```
정답: 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다
```

Section

LangChain + RAG + from chromaDB + GEN

LangChain

```
# 7. 자연어 생성용 모델 로딩 (Polyglot-ko)
gen_tokenizer = AutoTokenizer.from_pretrained(gen_model_id)
gen_model = AutoModelForCausalLM.from_pretrained(gen_model_id)
```

Loading checkpoint shards: 100% 3/3 [00:03<00:00, 1.04s/it]

```
# 8. 자연어 생성 파이프라인 생성
gen_pipeline = pipeline(
    task="text-generation",
    model=gen_model,
    tokenizer=gen_tokenizer,
    max_length=200,
    truncation=True,
    do_sample=True,
    temperature=0.1,
    device=-1,
)
gen_llm = HuggingFacePipeline(pipeline=gen_pipeline)
```

```
Device set to use cpu
```

Section

LangChain + RAG + from chromaDB + GEN

LangChain

```python
# 9. 프롬프트 템플릿 정의
prompt = PromptTemplate.from_template("""
너는 똑똑한 제조 전문가야. 아래의 질문과 정답을 바탕으로 자연스럽고 친절하게 응답해줘.

질문: {question}
정답: {answer}

자연어 응답:
""")
```

```python
# 10. 체이닝
chain = prompt | gen_llm
```

```python
# 11. 실행
final_response = chain.invoke({
    "question": question,
    "answer": result["answer"]
})

print("\n 자연어 응답:")
print(final_response)
```

```
 자연어 응답:

너는 똑똑한 제조 전문가야. 아래의 질문과 정답을 바탕으로 자연스럽고 친절하게 응답해줘.

질문: 클린룸 입장 시 주의사항 알려줄래?
정답: 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다

자연어 응답:
정답: 공정 이탈 발생 시 Shift 리더에게 즉시 보고합니다.
```

GEN AI 인텐시브 과정

Section 1. LangChain 실습

Section 1-8. LangChain + Memory

Section

LangChain + Memory

Memory

[1]:
```python
from langchain.memory import ChatMessageHistory
from langchain_core.prompts import PromptTemplate
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
from langchain_huggingface import HuggingFacePipeline
from uuid import uuid4
```

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

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

Memory

[4]:
```python
# 3. 프롬프트 설정: 히스토리 포함
prompt = PromptTemplate.from_template(
    "지금까지의 이야기:\n{history}\n\n이제 '{animal}'에 대한 다음 이야기를 써주세요:\n"
)
```

[5]:
```python
# 4. 세션별 히스토리 관리
chat_histories = {}

def get_history(session_id):
    if session_id not in chat_histories:
        chat_histories[session_id] = ChatMessageHistory()
    return chat_histories[session_id]
```

LangChain에서 대화형 애플리케이션의
메시지를 저장하고 관리하기 위한 클래스

- 사용자의 입력과 AI의 응답을 저장한 후,
나중에 다시 사용할 수 있게 해줌.

Section

LangChain + Memory

Memory

[10]:

```python
# 5. 대화 루프
def run():
    session_id = str(uuid4())  # 새로운 세션 ID
    print("동물 이름을 입력해 이야기를 시작하세요.")

    while True:
        animal = input("\n동물 이름 또는 질문 입력 (종료하려면 '종료'): ").strip()
        if animal in ["종료", "exit", "quit"]:
            print("대화를 종료합니다.")
            break

        # 히스토리 가져오기 및 포맷팅
        history = get_history(session_id)

        if history.messages:
            history_text = "\n".join(
                f"{msg.type.upper()}: {msg.content}" for msg in history.messages
            )
            prompt_input = prompt.format(history=history_text, animal=animal)
        else:
            prompt_input = f"'아주 먼 옛날 {animal} 한 마리가 살고 있었습니다. 그러던 어느날'"

        # 응답 생성
        response = llm.invoke(prompt_input)

        # 출력
        print("\n생성된 이야기:")
        print(response)

        # 히스토리에 기록
        history.add_user_message(animal) 사용자 메시지를 추가
        history.add_ai_message(response)  AI 메시지를 추가
```

Memory

```
[11]: run()
```

```
동물 이름을 입력해 이야기를 시작하세요.

동물 이름 또는 질문 입력 (종료하려면 '종료'): 토끼
```

첫 번째 입력

```
생성된 이야기:
 소녀는 숲 속 풀숲을 지나다 산책을 하고 있었습니다.
토끼는 풀숲을 걷고 있었습니다.
토끼는 풀숲을 건너다 보니 숲 속 풀숲이 생각났습니다.
토끼는 풀숲을 걸으면서 숲 속 풀숲을 보았습니다.
토끼는 풀숲을 걷다가 풀숲이 보여서 숲 속 풀숲이 보이지 않자 다시 걸었습니다.
풀숲을 걸으면서 풀숲에 있는 토끼를 보고서는 '토끼는 풀을 뜯고 있구나. 나는 풀을

동물 이름 또는 질문 입력 (종료하려면 '종료'): 거북이
```

두 번째 입력

```
생성된 이야기:
토끼는 풀숲을 걸으면서 토끼가 좋아하는 토끼를 떠올렸습니다.
토끼는 토끼에게 '거북이' 라는 이름을 지어주었습니다.
토끼는 거북이에게 '거북이'라는 이름을 지어주었습니다.
토끼는 '거북이'라고 해서 거북이에게 '거북이'라는 이름을 지어주었습니다.
토끼는 '거북이'라는 이름을 지어주었습니다.
토끼는 '거북이'라고 해서 '거북이'라고 했습니다.
토끼는 '토끼'라는 이름을 지어주는 데 성공했습니다.
토끼는

동물 이름 또는 질문 입력 (종료하려면 '종료'): 종료
```

세 번째 입력

```
대화를 종료합니다.
```

Memory

[15]:
```
session_id = str(uuid4())
print(session_id)
```

```
2ab566db-852c-4aff-8f04-12650774a7de
```

세션ID 생성

[16]:
```
history = get_history(session_id)
print(history)
```

history 초기 생성시 데이터 없음

[18]:
```
vars(history)
```

```
{'messages': []}
```

history 초기 생성시 데이터 없음을 확인

Memory

[21]:
```
animal = '토끼'
response = "아주 먼 옛날 토끼한 마리가 살고 있었습니다."
```

[22]:
```
history.add_user_message(animal)
history.add_ai_message(response)
```

[23]:
```
history
```

[23]:
```
InMemoryChatMessageHistory(messages=[HumanMessage(content='토끼', additional_kwargs={}, response_metadata={}), AIMessag
e(content='아주 먼 옛날 토끼한 마리가 살고 있었습니다.', additional_kwargs={}, response_metadata={})])
```

[24]:
```
vars(history)
```

[24]:
```
{'messages': [HumanMessage(content='토끼', additional_kwargs={}, response_metadata={}),
  AIMessage(content='아주 먼 옛날 토끼한 마리가 살고 있었습니다.', additional_kwargs={}, response_metadata={})]}
```

일반적인 파이썬 딕셔너리 구조가 아닌 LangChain 라이브러리에서 사용하는 특수 자료구조

Memory

[27]:
```
for msg in history.messages:
    print(msg.type.upper())
    print(msg.content)
```

```
HUMAN
토끼
AI
아주 먼 옛날 토끼한 마리가 살고 있었습니다.
```

[28]:
```
history_text = "\n".join(
    f"{msg.type.upper()}: {msg.content}" for msg in history.messages
)
```

데이터 합치기

[29]:
```
history_text
```

[29]:
```
'HUMAN: 토끼\nAI: 아주 먼 옛날 토끼한 마리가 살고 있었습니다.'
```

감사합니다.

Q & A