생성형 AI 기반 RAG 개발자 과정

Prompt Engineering + 미니앱 6종

Session 10 / 33 — Day 2

모델은 동일, 결과는 2~3배 — LCEL 로 즉시 구현

박수현 · (사)한국정보공학기술사회 · 2026

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

출처: Day 2 Prompt Engineering

이번 시간을 마치면

🛠️ LCEL 미니앱 6종

- 이메일 · 상품명 · 번역
- 요약 · 블로그 글감 · 코드리뷰
- `prompt | model | parser` 패턴 반복 숙달

🎯 프롬프트 엔지니어링

- 좋은 프롬프트 **6원칙**
- **Zero-shot vs Few-shot**
- **Chain of Thought** (CoT)
- 좋은 **System Prompt** 작성법

> 🎯 목표: 문법을 실제 결과물로 — 미니앱 6개를 구현하며 프롬프트 설계 감각을 체득한다.

PART F

PART F
실전 미니앱 6종

> 이메일 · 상품명 · 번역 · 요약 · 블로그 · 코드리뷰 — 약 30분

2.prompts/4.6_emailgeneration_chat.py 기반

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

llm = ChatOpenAI(model="gpt-4o-mini", temperature=0.5)

prompt = ChatPromptTemplate.from_messages([
    ("system", """당신은 한국어 비즈니스 이메일 작가입니다.
요청 사항을 기반으로 다음 형식으로 작성하세요:

제목: ...
받는 사람: ...
본문: (3~5단락, 격식체)
서명: ...
"""),
    ("user", """- 목적: {purpose}
- 받는 사람: {recipient}
- 핵심 메시지: {message}
- 톤: {tone}"""),
])
```

체인 실행

```
chain = prompt | llm | StrOutputParser()

email = chain.invoke({
    "purpose":   "프로젝트 일정 지연 양해 요청",
    "recipient": "팀장님",
    "message":   "외부 API 응답 지연으로 2일 지연 예상, 대안 마련 중",
    "tone":      "정중하고 책임감 있는",
})
print(email)
```

> 💡 **확장**: 톤·길이·언어를 변수화하면 동일 체인으로 영문 이메일·격식체·비격식 톤을 모두 처리할 수 있다.

스타일별 후보 병렬 생성 — RunnableParallel

```
from pydantic import BaseModel, Field
from typing import Literal

class ProductName(BaseModel):
    name: str = Field(description="상품명")
    style: Literal["기술적", "감성적", "유머"] = Field(description="네이밍 스타일")
    reason: str = Field(description="작명 이유 한 문장")

llm_struct = ChatOpenAI(model="gpt-4o-mini", temperature=0.9)\
    .with_structured_output(ProductName)

prompt = ChatPromptTemplate.from_template(
    "{product} 의 상품명을 {style} 스타일로 1개 제안."
)

# 3가지 스타일 병렬
chain = (
    RunnableParallel({
        s: prompt.partial(style=s) | llm_struct
        for s in ["기술적", "감성적", "유머"]
    })
)

results = chain.invoke({"product": "친환경 텀블러"})
for style, p in results.items():
    print(f"[{style}] {p.name} — {p.reason}")

# [기술적] EcoCarry — 친환경 소재 휴대성 강조
```

4개 언어로 병렬 번역

```
translate_prompt = ChatPromptTemplate.from_template(
    "다음 문장을 {target_lang} 로 자연스럽게 번역. 번역만 출력:\n\n{text}"
)
translate_chain = translate_prompt | llm | StrOutputParser()

# 4개 언어 병렬 번역
multi_translate = RunnableParallel({
    "english":  translate_chain.bind(target_lang="English"),
    "japanese": translate_chain.bind(target_lang="Japanese"),
    "chinese":  translate_chain.bind(target_lang="Chinese (Simplified)"),
    "french":   translate_chain.bind(target_lang="French"),
})

result = multi_translate.invoke({"text": "오늘 회의는 생산적이었습니다."})
for lang, trans in result.items():
    print(f"[{lang:>10}] {trans}")
```

응용 — 입력 언어 자동 감지로 정확도 향상

```
from langchain_core.runnables import RunnablePassthrough

# 1. 입력 언어 자동 감지
detect_prompt  = ChatPromptTemplate.from_template(
    "다음 문장의 언어를 ISO 639-1 코드 1개로만 답:\n{text}"
)

# 2. 감지 → 번역 체인 결합
chain = (
    RunnablePassthrough.assign(
        source_lang=detect_prompt | llm | StrOutputParser()
    )
    | multi_translate
)
```

- 감지 결과를 `source_lang` 으로 전달해 번역 품질을 높인다.

출처: Day 2 Prompt Engineering

한 문장 / 한 단락 / 핵심 5포인트

```
summary_prompt = ChatPromptTemplate.from_messages([
    ("system", "당신은 한국어 요약 전문가. 군더더기 없이 핵심만."),
    ("user", """다음 글을 {mode} 로 요약하세요:

{text}

요약:"""),
])

summary_chain = summary_prompt | llm | StrOutputParser()

article = """(긴 기사 본문 — 1000자)"""

# 3가지 길이 병렬
modes = RunnableParallel({
    "one_line":   summary_chain.bind(mode="한 문장 (최대 30자)"),
    "paragraph":  summary_chain.bind(mode="한 단락 (3~5문장)"),
    "bullets":    summary_chain.bind(mode="핵심 5개 bullet point"),
})

result = modes.invoke({"text": article})
print(result["one_line"])
```

💡 RAG (Day 2) 에서 chunk 마다 이 요약 체인을 적용해 "chunk 카드"를 생성하면 검색 정확도가 향상된다.

🛡️ **환각 억제**: RAG 프롬프트에 "문서에 없으면 'unknown'" 한 줄을 추가하면 모델의 날조 응답이 크게 줄어든다 — 프롬프트가 RAG 답변 품질을 좌우한다.

키워드 → 제목 5개 + 개요 + SEO

```
from pydantic import BaseModel, Field
from typing import List

class BlogIdea(BaseModel):
    title: str = Field(description="끌리는 제목 (40자 내외)")
    hook: str  = Field(description="첫 문장 후크")
    outline: List[str] = Field(description="목차 5개")
    seo_keywords: List[str] = Field(description="SEO 키워드 5개")
    target_reader: str = Field(description="타겟 독자 한 줄")

llm_struct = ChatOpenAI(model="gpt-4o-mini", temperature=0.9)\
    .with_structured_output(BlogIdea)

prompt = ChatPromptTemplate.from_messages([
    ("system", "한국어 블로그 기획자. 클릭 유도 + 본문 가치 모두 고려."),
    ("user", "주제: {topic}\n\n블로그 글감 1개 제안."),
])

chain = prompt | llm_struct

idea = chain.invoke({"topic": "비전공자가 시작하는 LLM API 개발"})
print(f"📌 {idea.title}")
print(f"👉 {idea.hook}")
print("📋 목차:")
for i, item in enumerate(idea.outline, 1):
    print(f"  {i}. {item}")
print(f"🔍 SEO: {', '.join(idea.seo_keywords)}")
print(f"🎯 타겟: {idea.target_reader}")
```

진단 + 개선 코드 + 점수

```
class CodeReview(BaseModel):
    bugs: List[str] = Field(description="발견된 버그/이슈")
    style_issues: List[str] = Field(description="스타일/네이밍 이슈")
    suggestions: List[str] = Field(description="개선 제안")
    refactored_code: str = Field(description="개선된 전체 코드")
    quality_score: int = Field(ge=0, le=100, description="0~100 점수")

llm_review = ChatOpenAI(model="gpt-4o-mini", temperature=0.0)\
    .with_structured_output(CodeReview)

prompt = ChatPromptTemplate.from_messages([
    ("system", """당신은 Python 시니어 리뷰어.
PEP8 + 가독성 + 버그 + 보안 관점에서 검토하세요."""),
    ("user", "다음 코드를 리뷰:\n\n```python\n{code}\n```"),
])
```

⬇️ 코드가 길어 다음 장에 이어집니다.

진단 + 개선 코드 + 점수 (이어서)

```
chain = prompt | llm_review

code_to_review = '''
def calc(l):
  s=0
  for i in range(len(l)):
    s=s+l[i]
  return s/len(l)
'''

review = chain.invoke({"code": code_to_review})
print(f"점수: {review.quality_score}/100")
print(f"버그: {review.bugs}")
print(f"스타일: {review.style_issues}")
print(f"\n개선 코드:\n{review.refactored_code}")
```

> 💡 GitHub Actions 에 연동하면 PR 자동 리뷰 봇으로 활용 가능.

PART G

PART G
프롬프트 엔지니어링 6원칙

> 모델은 동일, 결과는 2~3배 — 약 15분

좋은 프롬프트 6원칙

#	원칙	예시
1	**명확한 역할**	"Python 시니어 리뷰어" 〉"AI 도우미"
2	**구체적 작업**	"PEP8 위반 + 보안 이슈 + 개선 코드" 〉"리뷰해줘"
3	**출력 형식 명시**	"JSON · 한국어 · 5 bullet · 100자 이내"
4	**예시 (Few-shot)**	"입력→출력 예 2개" 〉그냥 설명
5	**금지 사항 명시**	"사담 금지, 코드 외 텍스트 없음"
6	**검증 단서**	"확신도 0~1 함께, 모르면 'unknown'"

한 줄 슬로건

> **모호함은 비용**. 모호하면 모델이 추측 → 결과 분산.

예시 제공 여부의 판단 기준

> 프롬프트에 예시를 포함해 그 자리에서 패턴을 학습시키는 방식을 **인컨텍스트 학습**이라 한다. 가중치는 변경하지 않고 맥락만으로 동작을 바꾼다.

Zero-shot — 예시 없음

```
다음 영화 리뷰의 감성을 positive/negative/neutral
중 하나로 분류하세요.

리뷰: "스토리는 진부했지만 연기가 살렸다"
감성:
```

- 적합: **분류·번역·요약** 같은 일반 작업
- 부적합: 특정 도메인 규칙이 얽힌 작업

Few-shot — 예시 2~5개

```
다음 영화 리뷰의 감성을 분류하세요.

예시 1:
리뷰: "최고의 영화, 또 보고 싶다"
감성: positive

예시 2:
리뷰: "돈 날렸다. 2시간 아까움"
감성: negative

예시 3:
리뷰: "그냥 그렇다"
감성: neutral

리뷰: "스토리는 진부했지만 연기가 살렸다"
감성:
```

- 적합: **도메인 규칙·미묘한 라벨링**

> 💡 **3~5개가 적정 지점**. 그 이상은 토큰 낭비 대비 효과 증가가 미미하다.

`FewShotPromptTemplate` vs `FewShotChatMessagePromptTemplate`

① FewShotPromptTemplate — 한 user 메시지에 예시

```
# 3.fewshot/3.1_fewshot_chat.py
from langchain_core.prompts import (
    PromptTemplate, FewShotPromptTemplate)

example_prompt = PromptTemplate(
    input_variables=["sentence", "result"],
    template="문장: {sentence}\n분석: {result}",
)

fewshot_prompt = FewShotPromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    prefix="다음은 감정 분석 예시입니다. "
           "같은 형식으로 분석하세요.\n\n=== 예시 시작 ===",
    suffix="=== 예시 끝 ===\n\n문장: {sentence}\n분석:",
    input_variables=["sentence"],
)
```

- 예제 **전부를 하나의 user 메시지** 문자열로 합침
- `prefix` / `suffix` 로 "예시 구간 ↔ 진짜 질문" 경계 표시

② FewShotChatMessagePromptTemplate — human/ai 쌍

```
# 3.fewshot/3.2_fewshot_messages_chat.py
from langchain_core.prompts import (
    ChatPromptTemplate, FewShotChatMessagePromptTemplate)

# 예제 한 건을 'human → ai' 메시지 쌍으로
example_prompt = ChatPromptTemplate.from_messages([
    ("human", "{sentence}"),
    ("ai",    "{result}"),
])

fewshot_messages = FewShotChatMessagePromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
)

final_prompt = ChatPromptTemplate.from_messages([
    ("system", "당신은 한국어 감정 분석기입니다. ..."),
    fewshot_messages,        # 예제들이 메시지 쌍으로 펼쳐짐
    ("human", "{sentence}"), # 진짜 입력
])
```

- 예제마다 **human/ai 메시지 쌍**으로 분리 → 진짜 대화 이력처럼 삽입

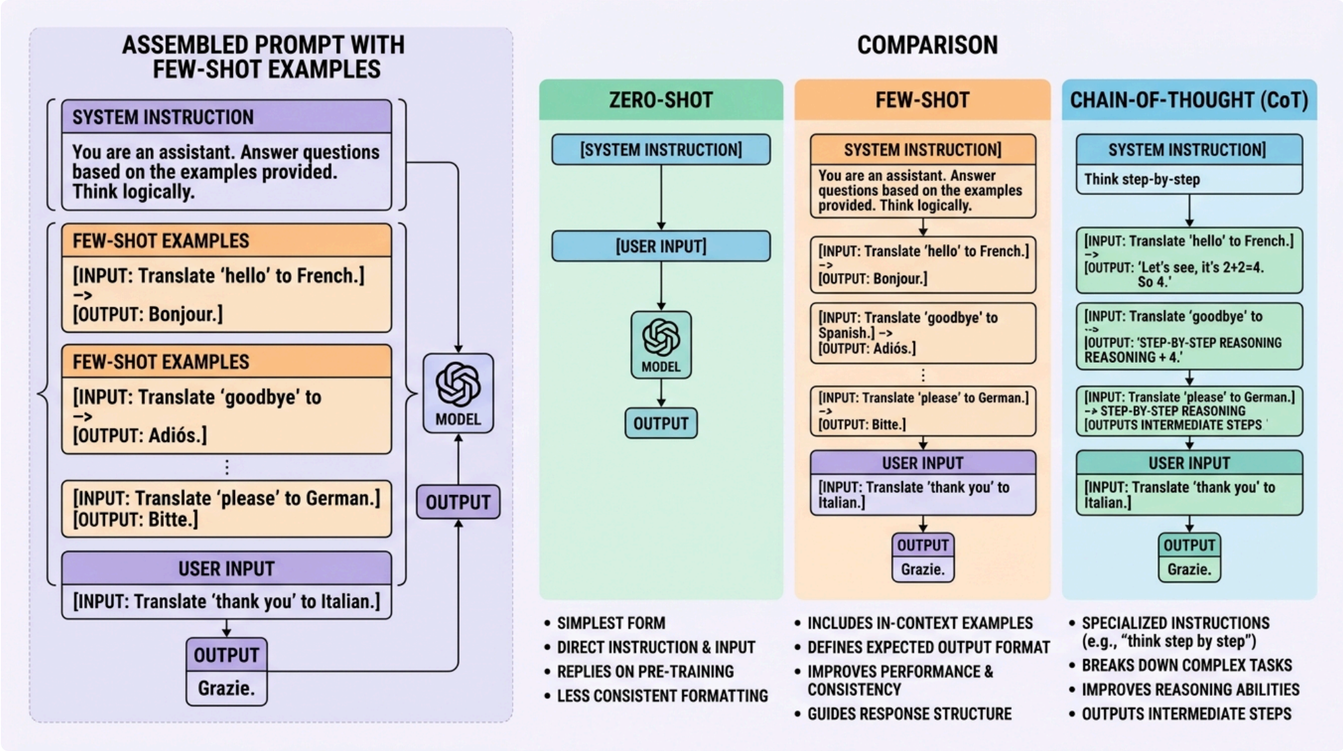
메시지 구조에 따른 선택

구분	FewShotPromptTemplate	FewShotChatMessagePromptTemplate
메시지 구조	예제 전부가 **하나의 user 메시지**	예제마다 **human/ai 메시지 쌍**
모델 관점	길게 예시 늘어놓고 질문	여러 번 대화한 뒤 다음 차례
권장 모델	Legacy completion	**Chat 모델 (gpt-4o 등)**

> 🎯 **Chat 모델에는 후자가 유리**하다. Chat 모델은 human/ai 턴이 교대하는 형식으로 학습되어, 예제를 대화 쌍으로 제공하면 형식 추종이 더 정확하다.

출처: Day 2 | Prompt Engineering

Zero-shot ↔ Few-shot ↔ CoT



프롬프트 · 시스템 지시 + Few-shot 예시 + 사용자 입력 · zero-shot↔few-shot(인컨텍스트 학습)↔CoT(단계적 추론)

출처: Day 2 Prompt Engineering

복잡한 추론은 "단계별로 생각해" 한 줄로 정확도 향상

Before — 단순 질문

```
질문: A 매장이 사과 10개를 갖고 있다.
3개를 팔고 5개를 더 받았다.
2개가 상해서 버렸다.
지금 몇 개 있는가?

답:
```

- 직접 답 → 모델이 즉답 시도 → 종종 틀림

After — CoT 트리거

```
질문: ... (위와 동일)

단계별로 생각해서 답하세요:
1.
2.
3.
4. 결론:
```

- 모델이 차근차근 추론 → 정확도 향상

"단계별로 생각해" 한 줄의 효과

Zero-shot CoT 트리거 문장

- "Let's think step by step."
- "단계별로 생각해보자." (한국어)

> 📊 Wei et al. 2022 — 수학 문제 정확도 **17% → 58%** 향상

출처: Day 2 Prompt Engineering

❌ 나쁜 system vs ✅ 좋은 system

❌ 나쁜 예

```
You are a helpful assistant.
```

문제: - 페르소나·도메인·톤·언어 무엇도 없음 - 모델이 일반적 답변 → 사용자 만족도 ↓

✅ 좋은 예

```
당신은 한국어 Python 시니어 리뷰어입니다.

[역할]
- 코드 리뷰 + 개선 제안

[작업 절차]
1. 버그를 찾아라
2. 스타일을 점검하라 (PEP8)
3. 더 나은 코드를 제시하라

[톤]
- 정중하고 구체적
- 항상 "왜" 를 함께

[금지]
- 사담, 인사말
- 코드 외 마크다운 헤더

[출력]
- JSON 형식 (BugList, StyleIssue, Refactored)
- 한국어, 핵심만
```

> 🎯 **잘 설계된 system 한 문단이 응답 품질의 70% 를 좌우**한다.

PART H

PART H
미니 실습

> 본인 도메인 미니앱 1개 — 약 30분

실습 과제 (30분)

1단계 — 미니앱 아이디어 (5분)

예시 후보 (자유): - 채용공고 → 면접 질문 5개 · 영화 리뷰 → 별점·감성·키워드 - 회의록 → 액션아이템 · 메뉴판 텍스트 → 4언어 메뉴

2단계 — Pydantic 모델 + ChatPromptTemplate 작성 (10분)

- 입출력 구조 먼저, system + user 메시지 분리
- few-shot 예시 1~2개 권장

3단계 — LCEL 체인 (10분)

```
chain = prompt | llm.with_structured_output(MyOutput)
chain.invoke({...})
```

4단계 — 입력 3개로 테스트 + 결과 비교 (5분)

- 출력이 일관적인가?
- 일관성이 부족하면 → system 프롬프트를 더 구체화

> 📌 **결과 보고**: 어떤 system 한 줄을 추가했을 때 결과가 개선되었는가 — Day 2 시작 전 공유.

핵심 6가지

1. ✅ LCEL 미니앱 = `prompt | model | parser` 한 패턴의 반복
2. ✅ **RunnableParallel** 로 여러 출력을 한 번에 (번역·요약·작명)
3. ✅ **구조화 출력**(`with_structured_output`)으로 JSON·타입 보장
4. ✅ 프롬프트 6원칙: **역할·작업·형식·예시·제약·검증**
5. ✅ **Few-shot · CoT** — 예시와 "단계별로 생각" 한 줄로 정확도 ↑ (Chat 모델엔 `FewShotChatMessagePromptTemplate`)
6. ✅ 좋은 **System Prompt** 가 절반 — 역할·톤·금지선을 명시