LoRA + 앙상블 RAG기반
할루시네이션을 줄인 업무시스템 구축기

By JAEWOO, Kim.

1. 할루시네이션 억제만으로 충분할까?

이 프로젝트에서는 필자는 원래 LoRA(Low-Rank Adaptation)를 활용해 LLM 할루시네이션(Hallucination)을 줄이는 파인튜닝에 집중하고 담당하고 있었습니다.

즉, "모르는 것은 모른다고 대답하는 정적한 AI를 만들자"라는 목표였습니다.

- 1B(10억 파라미터급)의 소형모델에 수백개의 학습 데이터를 넣어 LoRA로 반복 튜닝 진행
- 그 결과, 모호하거나 잘못된 정보를 요구하는 질문에는 "정보가 없습니다"라고 잘 대응하게 되었고 꽤 성공적인 케이스

그런데 문득 이런 생각을 하였습니다.

"기업 내에서 사용하는 RAG구조라면, 애초에 할루시네이션이 잘 일어나지 않나?"

LoRA란?
파인튜닝 기법중 하나인 LoRA는 원래 모델을 직접 업데이트하지 않고 학습 파라미터를 어뎁터로서 보존하고 어뎁터는 업데이트 대상이 되는 레이어의 파라미터(행렬)를 계산하기 쉽게 2개로 분해한 것을 새롭게 작성해 학습시킨 것을 말합니다.
추론시에는 학습시킨 어뎁터를 설치하여 학습내용을 반영합니다.
QLoRA(Quantized Low-Rank Adaptation)는 양자화(Quantization)한 모델을 LoRA로 파인튜닝시키는 기법입니다.

2. 기업용 RAG는 기본적으로 할루시네이션에 강하다!

회사 내부에서 RAG를 구성하면 일반적으로 DB에서 SQL로 필요한 정보를 추출하고 그 결과를 LLM에게 전달하여 요약하게 만듭니다.

- 데이터소스가 명확하므로 모델이 상상할 여지를 줄이고,
- 질문에 대해서는 DB에 기반한 정답만 주게 된다.

즉, LoRA를 통한 파인튜닝으로 할루시네이션을 막는게 핵심이 아니었던 것입니다. 정답은 이미 DB에 있었고, 모델은 요약머신 역할에 가까웠습니다.

3. 새로운 발상: 모델을 분업화하면 더 낫지 않을까?

여기서 관점을 바꿔보았습니다.

할루시네이션을 막는 것보다, 전문분야별로 모델을 나누는게 더 효과적인 구조가 아닐까?

사람도 마찬가지입니다.

- 제조 이슈는 제조부서가
- 제품사양은 설계팀이
- 클레임 대응은 품질보증팀이

→ 각 분야 전문가에게 물어보는 것이 정확하다!

AI도 마찬가지로, 모든 걸 한 모델에서 시키지 말고, 전문영역별로 LoRA모델을 분리하면 좋겠다는 아이디어로 발전시켰습니다.

4. 앙상블 RAG 구조는?

다음과 같은 3단계 구성으로 시스템을 설계했습니다.

① SQL로 DB에서 정보 추출

- 여러 DB에서 동시에 가능
- 예: 제조데이터, 제품사양, 클레임 이력 등

② 카테고리별 LoRA모델에게 각각 전달

- 예:
  - 제조일지 → 제조전용 LoRA
  - 제조사양 → 설계전용 LoRA
  - 고객불만 → 품질전용 LoRA

③ 마지막으로 하나의 LLM이 통합 및 요약

- 예: DeepSeek같은 LLM이 전체 응답을 정리

이 구조는 마치 "사내에 AI인턴들이 여러 명이 있고, 그 위에 정리해주는 리더 AI가 한명 있는 팀"처럼 작동합니다.

4. 핵심은 소형모델로도 충분하다!

가장 중요한 메시지:

- LoRA를 쓰면 소형모델(예: 1B~3B정도)도 특정 업무영역에 대해 고도화된 튜닝이 가능합니다.
- 각 모델은 한 분야만 책임지므로 전체를 한 모델이 커버할 필요가 없습니다.
- 결과적으로 이런 분업형 구조는 리소스가 적은 팀(소규모 사내 AI팀)에서도 충분히 구축이 가능합니다.

구성항목	설명
LoRA 모델군	1B급 x 3종(제조일지, 제품사양, 클레임)
추론환경	VMware 게스트OS, Mac Mini(M2)
통합모델	DeepSeek v2, Phi-4등 경량모델
실행방식	FastAPI + 비동기 API를 이용한 병렬질의 처리 방식

5. 예상동작: 이렇게 동작합니다. (1개 모델로는 이미 검증 완료)

예시출력

ABC제품은 5월 1일에 A라인에 생산되었습니다. 사양변경으로 인해 용량이 기존 250mL에서 200mL로 줄었으며, 일부 고객으로부터 '이전 사양과 차이를 알기 쉽다'라는 클레임이 접수되었습니다. 해당 클레임은 이미 대응 완료되었지만, 향후 자주하는 질문(FAQ)정리가 필요합니다.

이 문장은 사실상 3개의 LoRA모델과 하나의 통합 LLM이 협력해서 만든 결과입니다.

- 제조일지 모델: 언제, 어디서 생산되었는가?
- 사양모델: 어떤 사양으로 어떻게 바뀌었는가?
- 클레임모델: 어떤 고객이 어떤 문제를 제기했고 어떻게 대응했는가?
- 통합 LLM: 위 정보들을 종합해 전체 맥락이 연결된 문장으로 요약 정리

이 흐름은 사람이 보고서를 작성하는 절차와 거의 동일합니다.

6. 작게 시작해서 확장이 가능하다! (이 구조의 장점)

이 시스템 구조의 장점은 다음과 같습니다.

- 소형모델로 충분: 모델당 1B(10억 파라미터)급이면 Mac mini에서도 실행됩니다.
- 병렬처리로 속보 확보: GPU서버 환경에서는 각 모델을 병렬로 실행해 빠른 응답 가능
- 모델 확장이 쉬움: 기존 모델에 법무전용모델, 영업전용 모델등을 유연하게 추가 가능
- 로컬 구성 가능: 전체를 사내 인프라에서 구축할 수 있어 보안도 문제가 없음

7. 결론 :"할루시네이션 억제"보다 "LLM 팀워크"

처음에는 "할루시네이션을 완전히 억제하는 AI"를 만들기 위해 LoRA를 열심히 튜닝했지만, 그 과정을 거치면서 깨달은 것은 다음과 같습니다.

"모든 걸 다 아는 AI 한명"보다 "여러 AI가 팀처럼 협력해서 답변하는 구조"가 훨씬 현실적이다.

<LLM개발은 코딩이 아니라, 행동설계이다. 코드를 작성하는 것이 아니라, AI의 협력구조를 설계하는 것이다>

```
# ✅ LangChain 기반 멀티 에이전트 구조 + 부서별 LoRA 모델 파이프라인 예시

# 전제: 부서별 LoRA 모델 (제조, 사양, 클레임)을 LangChain tool로 래핑하여 멀티 에이전트 처리
# 통합 LLM이 각 전문 에이전트에게 질의 후 종합 응답을 생성함

from langchain.agents import AgentExecutor, create_tool_calling_agent
from langchain_core.tools import tool
from langchain_core.runnables import RunnableLambda
from langchain_openai import ChatOpenAI
```

```
# 🛠️ 1. 각 LoRA 모델을 호출하는 Tool 정의 (여기선 mock 함수로 대체)
@tool
def manufacturing_tool(query: str) -> str:
    """제조 관련 정보를 응답하는 LoRA 모델"""
    return f"[제조정보] 제품 ABC는 4월 1일에 라인 A에서 생산되었습니다."

@tool
def spec_tool(query: str) -> str:
    """제품 사양 변경에 대한 정보를 응답하는 LoRA 모델"""
    return f"[사양정보] 용량이 250mL에서 200mL로 변경되었습니다."

@tool
def claim_tool(query: str) -> str:
    """클레임 처리 내역에 대한 정보를 응답하는 LoRA 모델"""
    return f"[클레임정보] 고객이 불만을 제기했고, 대응이 완료되었습니다."

# 📦 2. 통합 LLM (최상위 요약 역할)
llm = ChatOpenAI(model="gpt-3.5-turbo", temperature=0)

# 🧠 3. 에이전트 정의 (tool 호출을 통한 ReAct 기반 추론)
tools = [manufacturing_tool, spec_tool, claim_tool]
agent = create_tool_calling_agent(llm, tools)
executor = AgentExecutor(agent=agent, tools=tools, verbose=True)

# 🚀 4. 실행 예시
query = "제품 ABC의 생산일과 사양 변경, 클레임 대응 내역을 알려줘"
result = executor.invoke({"input": query})

print("₩n📌 최종 응답:")
print(result["output"])
```