Local LLM Evaluation with LangSmith: Dependency Conflicts & Solutions

- LangSmith를 활용한 로컬 LLM 평가 - 의존성 충돌과 해결책

> 날짜: 2025.10.18.
>
> 환경: M1 MacBook Air, Python 3.12, pyenv virtualenv
>
> 프로젝트: LangChain RAG Evaluation with Local LLM (Qwen2.5-Coder-7B)

---

1. 🎯 목표

1) Local-Only RAG 평가 파이프라인 구현:

- 로컬 LLM: Qwen2.5-Coder-7B (Ollama 기반)
- LangSmith: 평가 프레임워크
- LangChain 0.3.x: 안정 버전
- Pydantic v2: 최신 검증 라이브러리

2) 목표:

- Gemini, GPT-4 등 상용 LLM 수준의 평가 품질을 유지하면서 API 비용 줄이기

---

2. 🐛 주요 문제점

이슈 1: Pydantic v1 ↔ v2 충돌 (튜토리얼 요구사항 불일치)

- 문제

```
ModuleNotFoundError: No module named 'langchain.evaluation'
```

- 원인 분석:
  - *기존 튜토리얼* = Pydantic v1.10.18 요구 (*구버전*)
  - 최신 LangChain 1.0.0+ = Pydantic v2만 지원
  - langchain.evaluation 모듈은 LangChain 1.0.0에서 제거됨
  - langsmith 평가자는 LangChain 평가 스키마에 의존함
- 환경 혼란:
  - lc_eval_env (Pydantic v1) → 튜토리얼용 환경

    - `lc_local_env` (Pydantic v2) → 로컬 LLM용 환경

- 해결책: `LangChain`을 `0.3.x` 시리즈로 **다운그레이드**

```
pip install langchain==0.3.7
pip install angchain-core==0.3.15
pip install langchain-community==0.3.5
```

- **왜 `0.3.7` 버전인가?**
    - `langchain.evaluation` 모듈이 아직 포함됨
    - `Pydantic v2`와 호환
    - `LangSmith` 평가자와 호환
    - 프로덕션 환경에서도 안정적

이슈 2: `langchain-huggingface` 버전 **충돌**

- 문제

```
ImportError: cannot import name 'convert_to_json_schema' from
'langchain_core.utils.function_calling'
```

- 원인:

```
langchain-huggingface 1.0.0          → langchain-core 1.0.0+ 필요
langchain-core 0.3.15                → convert_to_json_schema 없음
```

- 해결책

```
pip install langchain-huggingface==0.1.2
```

- 결과:
    - `langchain-core==0.3.15`와 완벽히 호환
    - 임베딩 모델 정상 작동
    - 함수 호출 충돌 없음

이슈 3: `langchain-ollama` 버전 충돌

- 문제

```
    ImportError: cannot import name 'is_data_content_block' from
'langchain_core.messages'
```

- 원인:

```
    langchain-ollama 1.0.0              → langchain-core 1.0.0+ 필요
    langchain-core 0.3.15               → is_data_content_block 없음
```

- 해결책

```
        pip install langchain-ollama==0.2.0
```

- 결과:
  - `LangChain 0.3.x`와 완전 호환
  - `Qwen2.5-Coder-7B` 완벽 구동
  - `Streaming`, `Async` 기능 모두 유지

이슈 4: `langchain-classic` 간섭 문제

- 문제: 자동 설치된 **`langchain-classic==1.0.0`** 이 **네임스페이스 충돌을 유발**
- 해결책

```
        pip uninstall -y langchain-classic
```

- 이유:
  - `langchain-classic` = *구버전 호환용 패키지*
  - LangChain 0.3.x에는 **불필요**
  - `import 경로 혼란`을 일으킴

---

3. ✅ 최종 안정 구성

1) 최적 패키지 버전

```
# Core LangChain (0.3.x series)
langchain==0.3.7
langchain-core==0.3.15
langchain-community==0.3.5
langchain-text-splitters==0.3.2

# LLM Providers (downgraded)
langchain-ollama==0.2.0        # Local LLM
langchain-huggingface==0.1.2   # Embeddings
langchain-openai==1.0.0        # Optional

# Evaluation & Tracking
langsmith==0.1.147
pydantic==2.12.3
pydantic-core==2.41.4

# Additional Dependencies
python-dotenv==1.0.1
faiss-cpu==1.9.0.post1
pypdf==5.1.0
```

2) 환경 설정 명령어

```
# 새 가상환경 생성
pyenv virtualenv 3.12 lc_eval_env

# 활성화
pyenv activate lc_eval_env

# pip 최신화
pip install --upgrade pip

# 1. Core LangChain
pip install langchain==0.3.7
pip install langchain-core==0.3.15
pip install langchain-community==0.3.5
pip install langchain-text-splitters==0.3.2

# 2. LLM Providers
pip install langchain-ollama==0.2.0
pip install langchain-huggingface==0.1.2
pip install langchain-openai==1.0.0
```

```
# 3. Evaluation
pip install langsmith==0.1.147
pip install pydantic==2.12.3
pip install python-dotenv==1.0.1
pip install faiss-cpu==1.9.0.post1
pip install pypdf==5.1.0

# 4. 검증 (중요!)
pip list | grep -E "langchain|pydantic"
```

---

4. 🧪 평가 결과

1) 성능 비교

항목	API LLM (Gemini)	Local LLM (Qwen2.5-Coder)
정확도	3/5 (60%)	5/5 (100%)
평가 시간	약 5분	약 24분
API 비용	$0.02	$0.00
동시성	4 threads	1 thread

2) 핵심 인사이트

- Local LLM이 상용 LLM과 동등하거나 더 높은 평가 품질 달성
- Qwen2.5-Coder-7B는 평가용으로 매우 우수
- 무료이지만 느림 → 충분히 수용 가능
- 결과 재현성 보장 (랜덤 시드 고정)

---

3) 🔧 추가 수정 사항

- **`TOKENIZERS_PARALLELISM` 경고**

```
# 코드 상단에 추가
os.environ["TOKENIZERS_PARALLELISM"] = "false"
```

- **`Jupyter Kernel` 충돌**
  - 문제: `evaluate()` 호출 시 `Jupyter 커널` 다운됨
  - 해결: `Python 스크립트`로 실행

```
```

```
python myrag.py
#  ...
python myrag5.py

python eval_context.py
python eval_labeled_criteria.py
python eval_labeled_score.py
```

- 파일 경로 문제

```
# 절대경로 사용
script_dir = os.path.dirname(os.path.abspath(__file__))
pdf_path = os.path.join(script_dir, "data", "SPRI_AI_Brief_2023년12월호_F.pdf")
```

---

5. 평가 구조

1) 📊 구조: `myrag5.py`

```
class PDFRAG:
    def create_chain_with_context(self, retriever):
        """
        평가 시 Context와 Answer를 함께 반환.
        - Context 기반 평가 (context_qa, cot_qa)
        - Labeled Criteria (기준 비교형)
        - Labeled Score (정량 점수형)
        """
        prompt = ChatPromptTemplate.from_template(
            "Based ONLY on the following context, answer the question.\n"
            "Context:\n{context}\n\n"
            "Question: {question}\n"
            "Answer:"
        )

        chain = (
            {
                "context": retriever | self._format_docs,
                "question": RunnablePassthrough()
            }
            | RunnablePassthrough.assign(
```

```
                    answer=prompt | self.llm | StrOutputParser()
                )
            )
            return chain
```

- 이 설계의 이유
  - `context` = 평가 참조로 활용 가능
  - `answer` = LLM이 생성
  - `question` = 추적용으로 유지
  - 데이터 구조가 일관되어 평가에 적합

---

2) 🎯 평가 스크립트 예시

- 1. Context-Based Evaluators

```
    # eval_context.py

    context_qa_evaluator = LangChainStringEvaluator(
        "context_qa",
        config={"llm": eval_llm},
        prepare_data=lambda run, example: {
            "prediction": run.outputs["answer"],
            "reference": run.outputs["context"],
            "input": example.inputs["question"],
        },
    )
```

- 2. Labeled Criteria

```
    # eval_labeled_criteria.py

    helpfulness_evaluator = LangChainStringEvaluator(
        "labeled_criteria",
        config={
            "criteria": {
                "helpfulness": "Is this helpful compared to the
reference?"
            },
            "llm": eval_llm,
        },
        prepare_data=lambda run, example: {
            "prediction": run.outputs["answer"],
            "reference": example.outputs["answer"],
            "input": example.inputs["question"],
        },
    )
```

- 3. Labeled Score

```
# eval_labeled_score.py

accuracy_evaluator = LangChainStringEvaluator(
    "labeled_score_string",
    config={
        "criteria": {
            "accuracy": "Score 1-10: How accurate is the answer?"
        },
        "llm": eval_llm,
        "normalize_by": 10.0,
    },
    prepare_data=lambda run, example: {
        "prediction": run.outputs["answer"],
        "reference": example.outputs["answer"],
        "input": example.inputs["question"],
    },
)
```

6. 💡 핵심 교훈

- 1. 버전 호환성 매트릭스
  - LangChain의 핵심 모듈, 통합 패키지(ollama, huggingface), Pydantic, LangSmith 간 버전 호환성 반드시 확인할 것.
- 2. 다운그레이드 전략
  - 충돌 발생 시:
    - langchain==0.3.7로 시작
    - 관련 통합 패키지들을 이에 맞게 하향 조정
    - 0.3.x ↔ 1.0.x 혼용 금지
- 3. 로컬 LLM의 장점
  - 💰 비용 0원
  - 🔒 데이터 프라이버시 보장
  - 🧩 결과 재현성 확보
  - ⚙️ 완전한 제어권 (Prompt, Parameter)
- 4. 평가 Best Practice
  - 로컬 LLM = max_concurrency=1
  - prepare_data 매핑을 신중히 설계
  - 소규모 테스트 후 전체 데이터 평가
  - LangSmith에 기록해 비교 분석

7. 🚀 빠른 시작 (복붙용)

```
# 1. 환경 생성
pyenv virtualenv 3.12 lc_eval_env
pyenv activate lc_eval_env

# 2. 버전별 설치
pip install langchain==0.3.7
pip install langchain-core==0.3.15
pip install langchain-community==0.3.5
pip install langchain-ollama==0.2.0
pip install langchain-huggingface==0.1.2
pip install langsmith==0.1.147
pip install pydantic==2.12.3
pip install faiss-cpu pypdf python-dotenv

# 3. Ollama 설치 및 모델 다운로드
brew install ollama
ollama pull qwen2.5-coder:7b-instruct

# 4. 평가 실행
python eval_labeled_criteria.py
```

---

8. 📚 참고 문서

1) 공식 가이드

- LangChain Docs
- LangSmith Evaluators
- Ollama Models
- Qwen2.5-Coder

---

2) 📝 Note

- 많은 튜토리얼이 `Pydantic v1`을 사용하므로 주의
- `LangChain Migration Guide` 참고 필수
- LangChain Discord 커뮤니티는 매우 빠른 대응
- 초기엔 소규모 데이터로 테스트 후 확장

---

3) 🎉 성과 지표

- ✅ 5문항 기준 100% 정확도
- ✅ API 비용 $0
- ✅ 완전 재현 가능
- ✅ Pydantic v1, v2 충돌 해결
- ✅ M1 MacBook Air (16GB RAM) 완벽 동작

> *최종 업데이트: 2025.10.18.*
>
> *작성자: Jay*
>
> *상태: ✅ Production-Ready*
