ChatGPT

LLM을 활용한 질의 의도 분류 시스템 개발 방안

문제 개요: 정형 vs 비정형 데이터 검색

사내 데이터에는 **정형 데이터**(데이터베이스의 실적, 고객사 정보 등)와 **비정형 데이터**(수천~수만 건의 회의록 텍스트 등)가 공존합니다. 사용자의 자연어 질의가 들어왔을 때, 어떤 데이터 소스를 이용해 답해야 할지 판단하는 것이 핵심입니다. 예를 들어:

- "김영철이 고객사 담당자로 언급된 문서를 찾아 요약해줘." – 사람 이름과 '문서' 언급으로 미루어 **비정형 문서 검색**이 필요합니다 (회의록에서 해당 이름 검색 후 요약).
- "공정별 실적을 알려줘." – '실적'과 '공정별'이라는 키워드로 보아 **정형 데이터베이스 질의**로 판단됩니다 (DB에서 공정 단위 실적 데이터 조회).

이처럼 **질의 의도 파악**을 통해 **어떤 툴/데이터소스**를 사용할지 결정해야 합니다 [1] [2]. 의도 분류가 잘못되면 엉뚱한 방법으로 답변하게 되므로, LLM을 활용해 정확히 분류하는 시스템을 개발하고자 합니다.

의도 분류를 위한 레이블 정의와 데이터셋 활용

먼저 **의도 분류 체계**를 정의해야 합니다. 귀사에는 이미 질의 -> 해당 데이터베이스 테이블 레이블이 달린 데이터셋이 있다고 했습니다. 이를 활용하여 **카테고리**를 명확히 정합니다. 예시 카테고리:

- **회의록_검색** (비정형 문서 검색이 필요한 질의)
- **실적_DB** (정형 데이터베이스 중 "실적" 테이블 질의)
- **고객_DB** (정형 데이터베이스 중 "고객사" 테이블 질의)
- ... 필요에 따라 다른 테이블이나 데이터소스를 세분화.

카테고리는 가능한 **정해진 명칭**으로 한정해야 합니다. LLM에 별도 지시가 없으면 임의의 표현으로 답할 수 있기 때문에, **허용된 출력 레이블을 명시**하는 것이 중요합니다 [3]. 예를 들어 "회의록_검색" 또는 "실적_DB"처럼 미리 정한 용어로만 답하게 유도해야 합니다.

또한 **질의로부터 추가 파라미터 추출** 규칙도 고려합니다. 예컨대: - 회의록_검색 의도라면 **검색 키워드**(예: "김영철")를 추출하고, - 실적_DB 의도라면 **필요한 필드나 범주**(예: "공정별")를 식별하는 식입니다.

이러한 정보 추출도 출력 포맷에 포함될 수 있도록 설계합니다.

Few-Shot 프롬프트 엔지니어링을 통한 LLM 분류

모델을 추가 학습(fine-tuning)하지 않을 계획이므로, **프롬프트 엔지니어링**으로 LLM의 분류 능력을 이끌어냅니다. Few-shot learning 기법을 활용하면, 별도 학습 없이도 프롬프트에 **예시를 몇 개 제공**하여 성능을 크게 향상시킬 수 있습니다 [4].

프롬프트 설계 방법:

1. **시스템 역할 지시**: 모델에게 자신이 "질의 의도를 분류하는 전문가"임을 알려줍니다. 예를 들면: "너는 입력 질문을 보고, 그것이 정형 DB 질의인지 비정형 문서 검색인지 판단하여 미리 정해둔 카테고리로 분류하고, 추가로 검색 키워드나 필요한 필터 파라미터를 식별해야 한다." 등의 지침을 줍니다.

2. **카테고리 목록 제시**: 프롬프트에 가능한 출력 카테고리를 명시합니다. 예: “의도 분류 결과는 다음 중 하나여야 합니다: [회의록_검색, 실적_DB, 고객_DB].” LLM이 정해진 범위만 선택하도록 유도하는 것입니다 [3].

3. **예시(Q&A) 제공 (Few-shot)**: 레이블링된 데이터셋에서 **대표 질의-의도 쌍** 몇 가지를 고릅니다. 각 예시에 질문과 기대 출력(의도 분류 결과 및 파라미터)을 보여주세요. 한국어 데이터셋이므로 예시도 한국어 질의로 제시합니다. 예를 들면:

4. **예시 1**:
   질문: 김영철이 고객사 담당자로 언급된 문서를 검색하고 요약해줘.
   의도: 회의록_검색 (검색어: "김영철")

5. **예시 2**:
   질문: 우리 회사 분기별 실적을 찾아줘.
   의도: 실적_DB (필드: "분기별 실적")

6. **예시 3**:
   질문: ABC회사와 진행한 프로젝트 내역 알려줘.
   의도: 고객_DB (키워드: "ABC회사")

위와 같은 형식으로 2~5개 예시를 프롬프트에 나열합니다. 모델이 예시를 학습하여 새로운 질문도 유사한 형식으로 분류하게 됩니다 [4]. 예시에선 의도 카테고리와 함께 괄호 안에 주요 파라미터를 기술했습니다. 출력 형식은 일관되게 유지하는 것이 좋습니다.

1. **출력 형식 지시**: 마지막으로 실제 **사용자 질의**를 던지고 모델의 답변이 일정한 포맷이 되도록 요구합니다. JSON 등 **구조화된 출력**을 사용하면 파싱이 용이합니다 [5]. 예컨대 JSON 형식 예시:

```
질문: "김영철이 고객사 담당자로 언급된 문서를 요약해줘."
의도 분류 결과: {
 "type": "회의록_검색",
 "keyword": "김영철"
}
```

모델에게 “위 예시처럼 JSON 형식으로 답하라”고 지시하면, 결과를 코드로 바로 처리하기 수월합니다. 다만 JSON 구조를 완전히 따르게 하는 것이 어려울 수 있으므로, 적절히 `"type": 카테고리명, "keyword": 추출어` 형태의 텍스트라도 일관되게 출력하도록 유도합니다.

팁: 생성 결과의 신뢰성을 높이려면 **온도(temperature)** 파라미터를 낮춰서(예: 0.2 이하) 모델의 출력을 deterministic하게 만드는 것이 좋습니다 [6] [7]. 이렇게 하면 매번 같은 질문에 같은 분류 결과를 낼 확률이 높아집니다.

LlamaIndex Router를 활용한 툴 선택 통합

위와 같이 LLM이 의도를 분류하면, 실제 **검색/조회 액션**을 수행할 차례입니다. 이를 수동으로 분기 코드를 짜서 처리할 수도 있지만, **LlamaIndex** 라이브러리의 RouterQueryEngine 기능을 활용하면 보다 세련되게 구현 가능합니다. LlamaIndex의 **Router**는 LLM을 활용해 다중 데이터소스 중 **적절한 후보를 자동 선택**해주는 모듈입니다 [8] [9].

RouterQueryEngine을 사용하는 방식은 다음과 같습니다:

- 각 데이터 소스별로 **QueryEngineTool**을 만듭니다. 예를 들어, 회의록 벡터검색용 `vector_tool`과 DB 질의용 `sql_tool`을 정의하고, 각각에 "이 툴은 어떤 질문에 유용한지" 설명을 붙입니다 [10] [11]. 예: `vector_tool`에는 "회의록 등 비정형 문서에서 특정 키워드를 검색하고 요약할 때 사용", `sql_tool`에는 "사내 DB에서 실적, 고객 정보 등 정형 데이터를 조회할 때 사용" 같은 설명을 첨부합니다.

- **RouterQueryEngine**에 `selector=LLMSingleSelector`(또는 OpenAI 함수호출 지원 시 `PydanticSingleSelector`)를 설정하고, 위에서 정의한 두 가지 툴을 등록합니다 [7] [12]. 그러면 RouterQueryEngine은 내부적으로 LLM에게 질의와 각 툴의 설명을 넣은 프롬프트를 보내 **JSON 형태로 어느 툴을 쓸지 선택**하게 합니다 [7] [13].

- 사용자가 질문을 하면 `query_engine.query(user_query)`를 호출하고, Router가 자동으로 적절한 툴 (벡터검색 또는 DB질의)을 실행합니다 [8] [9]. LLM이 선택한 결과에 따라 해당 경로의 처리(엘라스틱서치/벡터DB 검색 또는 SQL 조회)가 진행되고 최종 답변을 생성합니다.

  참고: Router를 쓰지 않고 직접 분류 결과를 사용할 수도 있습니다. 예를 들어 앞서 LLM 출력이 `"type": "회의록_검색"`이라면 Python 코드로 if/else 분기하여 벡터DB에서 검색을 수행하고 요약하도록 할 수 있습니다. 반면 `"type": "실적_DB"`이면 미리 정의된 SQL 쿼리를 실행하거나, LLM에게 해당 테이블에 맞는 SQL을 작성하게 할 수도 있습니다. 어느 방법이든 **LLM의 의도 분류 출력**을 기반으로 처리 로직을 분기하면 됩니다.

메타데이터 필터링을 통한 권한 제어

비정형 문서 검색을 할 때는 **사용자 권한에 따른 결과 필터링**도 필요합니다. 벡터DB나 LlamaIndex는 문서에 메타데이터(예: 접근 가능 부서, 보안 등급)를 저장해 두고 **검색 시 필터**를 적용할 수 있습니다 [14] [15]. 구현 예시:

- 각 문서 노드에 `allowed_departments`나 `allowed_roles` 같은 메타데이터 필드를 추가합니다. 예컨대 A문서는 `{"allowed_departments": ["영업팀","경영지원"]}` 식으로 태그.
- 사용자의 부서나 권한 정보를 인증 후 애플리케이션 측에서 알고 있다고 가정합니다.
- **검색 질의 실행 시 필터 적용**: LlamaIndex의 `.query()`나 `.as_retriever().retrieve()` 메소드에 필터 조건을 넣어 해당 사용자에게 허용된 문서만 검색되도록 제한합니다 [14]. 만약 벡터 스토어 자체가 필터를 지원하지 않으면, 우선 상위 N개의 후보를 가져온 뒤 `if current_user.department in node.metadata["allowed_departments"]` 같은 코드로 걸러낼 수도 있습니다 [14].
- 이러한 필터링을 거쳐 최종적으로 LLM이 요약에 사용할 문맥이나 DB 응답을 구성하면, **민감한 정보가 섞이지 않도록 안전장치**가 됩니다.

정형 데이터의 경우도 유사하게, 사용자가 접근 권한 없는 테이블이면 질의 자체를 수행하지 않도록 하거나 결과를 숨기는 로직을 추가해야 합니다. 이는 주로 어플리케이션 계층에서 제어하되, 필요하면 LLM에게 "이 사용자는 X 부서이니 Y 데이터는 제외하고 답변하라"는 식으로 프롬프트에 제약을 줄 수도 있습니다.

오픈소스 모델 활용 및 추가 팁

프롬프트 엔지니어링만으로도 의도 분류가 대부분 가능하지만, **분류 정확도**를 높이기 위한 몇 가지 고려사항을 덧붙입니다:

- **모델 선택**: 회사 서버의 LLaMA 기반 모델을 사용한다고 했는데, 해당 모델이 한국어 질의에 충분히 능숙한지 확인이 필요합니다. 한국어 데이터셋으로 few-shot 프롬ππ를 주면 큰 문제는 없겠지만, 혹시 성능이 부족하면 KoGPT, XLM-RoBERTa 등 **한국어/멀티언어**에 강한 오픈소스 LLM을 시도해볼 수 있습니다. 또는 해당 데이터

셋으로 **경량 모델을 파인튜닝**한 후, 이 모델을 분류 전용으로 쓰고 이후 큰 LLM을 생성용으로 쓰는 방법도 있습니다 [16]. 다만 추가 학습을 하지 않는다는 조건에서, 우선은 현재 사용 중인 LLM의 힘을 믿고 프롬프트 방식으로 접근합니다.

- **출력 검증**: LLM이 항상 정확한 카테고리를 낼 것으로 가정하지 말고, 분류 결과를 **검증/로깅**하세요. 예를 들어, LLM 출력이 예상과 다르거나 파싱 실패할 경우를 대비해 **fallback 전략**을 두는 것이 좋습니다. 규칙 기반으로 "만약 '문서'나 '요약' 단어가 질문에 포함되면 회의록_검색으로 간주" 같은 백업 룰을 적용할 수 있습니다. Matt Rickard도 언급했듯이, LLM의 분류 출력 80%는 형식이 맞아도 20% 정도는 파싱이 틀릴 수 있어 반복 개선이 필요합니다 [17].

- **지속적인 프롬프트 개선**: 실제 사용자 질의를 모니터링하면서 LLM이 오분류하는 패턴이 보이면, 그에 맞는 **추가 예시를 프롬프트에 보강**하거나, 잘못된 해석을 바로잡는 지침을 넣어 줍니다. 예를 들어 "만약 질문에 시간이나 날짜가 언급되면 실적_DB일 확률 높음" 같은 힌트를 시스템 메시지에 추가해줄 수도 있습니다.

요약하면, **의도 분류 LLM** 개발은 거대한 모델 재학습 없이도 **프롬프트 설계**와 **예시 제공**으로 구현 가능합니다. LLM에게 몇 가지 예제를 보여주고 규칙을 알려주면, 새로운 질의도 적절한 데이터 소스로 매핑할 수 있습니다 [4]. 이후 LlamaIndex의 Router 등을 통해 그 결정을 실제 검색/질의 수행으로 연결하고, 메타데이터 필터링으로 **보안/권한** 문제도 해결하면 완성도 높은 **통합 질의 응답 시스템**을 구축할 수 있을 것입니다 [9] [14].

참고 자료: LLM 기반 분류와 툴 선택에 대한 일반적인 논의 [1] [18], LlamaIndex 공식 문서의 RouterQueryEngine 설명 [2], 그리고 OpenAI API를 활용한 few-shot 분류 기법 소개 [4] 등을 함께 참고하시기 바랍니다. 또한 메타데이터 기반 필터링 구현 사례도 살펴보시면 도움될 것입니다 [14]. 적절한 프롬프트 디자인과 체계적인 설계를 통해 요구하신 LLM 의도 분류 모듈을 성공적으로 개발하시길 바랍니다.

---

[1] [3] [5] [17] [18] Categorization and Classification with LLMs - Matt Rickard
https://blog.matt-rickard.com/p/categorization-and-classification

[2] [8] [9] [13] Routing - LlamaIndex
https://docs.llamaindex.ai/en/stable/module_guides/querying/router/

[4] GPT-based few-shot classification with the OpenAI API - Dataiku Developer Guide
https://developer.dataiku.com/12/tutorials/machine-learning/genai/nlp/gpt-few-shot-clf/index.html

[6] [7] [10] [11] [12] Router Query Engine - LlamaIndex v0.10.17
https://docs.llamaindex.ai/en/v0.10.17/examples/query_engine/RouterQueryEngine.html

[14] [15] How to implement document-level access control in LlamaIndex for a global chat app? : r/Rag
https://www.reddit.com/r/Rag/comments/1k8t6ye/how_to_implement_documentlevel_access_control_in/

[16] LLMs are machine learning classifiers - Wandb
https://wandb.ai/gladiator/LLMs-as-classifiers/reports/LLMs-are-machine-learning-classifiers--VmlldzoxMTEwNzUyNA