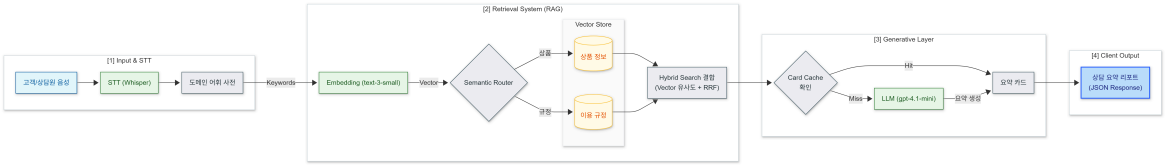
LLM 활용 소프트웨어



1. 음성을 텍스트로 변환 (STT)

고객 또는 상담원의 발화 음성을 텍스트로 변환

`app/audio/whisper.py`

```
# OpenAI API 호출
transcript = self.client.audio.transcriptions.create(
        model="whisper-1",
        file=audio_file,
        language="ko",
)

text = transcript.text.strip()
```

- WebSocket을 통해 실시간 오디오 스트림을 수신
- Whisper API 호출
  - `model="whisper-1"` : OpenAI의 Whisper 모델 사용
  - `language="ko"` : 한국어로 지정하여 정확도 향상

2. 도메인 어휘사전 기반 키워드 추출

STT 결과에서 카드명, 금융 활동, 결제수단 등의 핵심 키워드를 추출하여 문의 유형 파악

`app/api/v1/endpoints/routers.py`

```
def_build_processor(synonyms: Dict[str, List[str]]) → KeywordProcessor:
    kp = KeywordProcessor(case_sensitive=False)
```

```
for canonical, termsin synonyms.items():
        kp.add_keyword(canonical, canonical)
for termin terms:
            kp.add_keyword(term, canonical)
return kp
```

- **FlashText 매칭**
  - O(n) 시간복잡도로 모든 동의어를 동시에 검색
  - 텍스트 정규화
    (예: "나라사랑" → "나라사랑카드")
    (예: "잃어버렸어요" → "분실")

3. 임베딩 처리

사용자 쿼리를 벡터로 변환하여 의미 기반 검색

OpenAI text-embedding-3-small 임베딩 모델 (1536차원)

`app/rag/retriever.py`

```
def embed_query(text:str, model:str ="text-embedding-3-small") → List[flo
at]:
    client = get_openai_client()
    resp = client.embeddings.create(model=model,input=text)
return resp.data[0].embedding
```

```
def vector_search(
        query:str,
        table:str,
        limit:int,
        filters: Optional[Dict[str,object]]=None,
        ) → List[Tuple[object,str, Dict[str,object],float]]:
    table = _safe_table(table)
    emb = Vector(embed_query(query))
```

- 쿼리 템플릿 생성
  - 원본: `"나라사랑카드 혜택"`

    - 확장: `"나라사랑카드 혜택 나라사랑카드 혜택이 뭐예요?"`
- OpenAI Embedding API 호출
    - 입력 텍스트를 1536차원 벡터로 변환
    - PostgreSQL의 pgvector 확장을 사용하여 저장

4. 시맨틱 라우터

추출된 키워드를 조합하여 사용자 의도를 분류하고 검색 범위를 1차 결정

`app/rag/router.py`

```
# 1) 카드 + 액션: 둘 다 있으니 가장 강함
if card_names and actions:
    ui_route = ROUTE_CARD_USAGE
    db_route = "both"
    boost = {"card_name": card_names, "intent": actions}
    if payments:
        boost["payment_method"] = payments
    if weak_intents:
        boost["weak_intent"] = weak_intents
    query_template = f"{card_names[0]} {actions[0]} 방법"
    should_trigger = True

# 2) 카드 + 결제수단
elif card_names and payments:
    ui_route = ROUTE_CARD_USAGE
    db_route = "card_tbl"
    boost = {"card_name": card_names, "payment_method": payments}
    query_template = f"{card_names[0]} {payments[0]} 사용 방법"
    should_trigger = True

# 3) 카드 + 약한의도
elif card_names and weak_intents:
    ui_route = WEAK_INTENT_ROUTE_HINTS.get(weak_intents[0], ROUTE_C
ARD_USAGE)
    db_route = "both"
    boost = {"card_name": card_names, "weak_intent": weak_intents}
    if ui_route == ROUTE_CARD_INFO:
```

```
        query_template = f"{card_names[0]} {weak_intents[0]}"
    else:
        query_template = f"{card_names[0]} {weak_intents[0]} 방법"
    should_trigger = True

# 4) 카드만
elif card_names:
    ui_route = ROUTE_CARD_INFO
    db_route = "card_tbl"
    boost = {"card_name": card_names}
    query_template = f"{card_names[0]} 정보"
    should_trigger = True

# 5) 액션만
elif actions:
    ui_route = ROUTE_CARD_USAGE
    db_route = "guide_tbl"
    boost = {"intent": actions}
    if payments:
        boost["payment_method"] = payments
    query_template = f"카드 {actions[0]} 방법"
    should_trigger = any(a in ACTION_ALLOWLIST for a in actions)

# 6) 결제수단만
elif payments:
    ui_route = ROUTE_CARD_USAGE
    db_route = "card_tbl"
    boost = {"payment_method": payments}
    query_template = f"{payments[0]} 사용 방법"
    should_trigger = any(p in PAYMENT_ALLOWLIST for p in payments)

# 7) 아무것도 못 잡으면: fallback 검색
else:
    ui_route = ROUTE_CARD_USAGE
    db_route = "both"
    boost = {}
    query_template = None
    should_trigger = False
```

- 카드명, 액션, 결제수단, 약한의도의 조합 패턴 분석
  - `card_info`: 카드 정보 조회 (카드 테이블만 검색)
  - `card_usage`: 카드 이용 안내 (가이드 테이블 포함)
- 검색 시 사용할 메타데이터 필터 구성

5. Vector DB 검색

임베딩된 쿼리와 유사한 문서를 PostgreSQL + pgvector에서 검색

`app/rag/retriever.py`

```
sql = (
      f"SELECT id, content, metadata, 1 - (embedding ⇔ %s) AS score "
      f"FROM{table}{where_sql} ORDER BY embedding ⇔ %s LIMIT %s"
)
params = [emb, *where_params, emb, limit]
try:
   cur.execute(sql, params)
except Exception:
   conn.rollback()

   sql = (
            f"SELECT id, content, metadata, 1 - (embedding ↔ %s) AS score
"
            f"FROM{table}{where_sql} ORDER BY embedding ↔ %s LIMIT %
s"
   )
   cur.execute(sql, params)
```

- 벡터 검색
  - pgvector의 `<=>` 연산자 사용 (코사인 거리)
  - 의미적으로 유사한 문서 검색
- 키워드 검색
  - PostgreSQL의 `ILIKE` 연산자 사용
  - 정확한 용어 매칭

- 하이브리드 랭킹(RRF)
  - 벡터 검색 결과와 키워드 검색 결과를 결합
  - RRF 알고리즘 적용
  - 최종 점수 = RRF 점수 + 제목 매칭 점수

6. 요약문 생성

검색된 문서를 기반으로 사용자 질문에 대한 간결한 요약 생성

GPT-4.1-mini 텍스트 생성 모델

app/llm/card_generator.py

```
def _build_card_prompt(query:str,docs: List[Dict[str, Any]]) →str:
   parts = []
      for idx, docinenumerate(docs,1):
            content = doc.get("content")or""
            doc_id = doc.get("id")or""
            title = doc.get("title")or""
            parts.append(
                        f"[{idx}] id={doc_id}\n"
                        f"title={title}\n"
                        f"content={_truncate(content, MAX_CARD_DOC_CHAR
S)}"
            )
         joined ="\n\n".join(parts)if partselse"문서 없음"
         doc_count =len(docs)
      return (
      f"""다음은 카드 상담용 문서입니다. 사용자 질문과 문서 내용을 참고해 카드 요약
(content)만 생성하세요.

         ### 지시 사항
         1. 반드시 아래 제공된 JSON 객체 형식만 반환하세요. 추가 텍스트는 금지합니
다.
         2. 카드 수는 총 {doc_count}개이며, 문서의 순서와 동일하게 cards 배열을
구성하세요.
         3. 각 요약은 1~2문장으로 작성하며, 문서에 없는 내용은 절대 포함하지 마세
요.
```

```
        4. content 외의 필드는 출력하지 마세요.

        ### 사용자 질문
        {query}

        ### 상담 문서 내용
        {joined}

        ### 출력 형식 (JSON Schema)
        {{
        "cards": [
            {{ "content": "카드 요약 내용 1-2문장" }}
        ]
        }}"""
    )
```