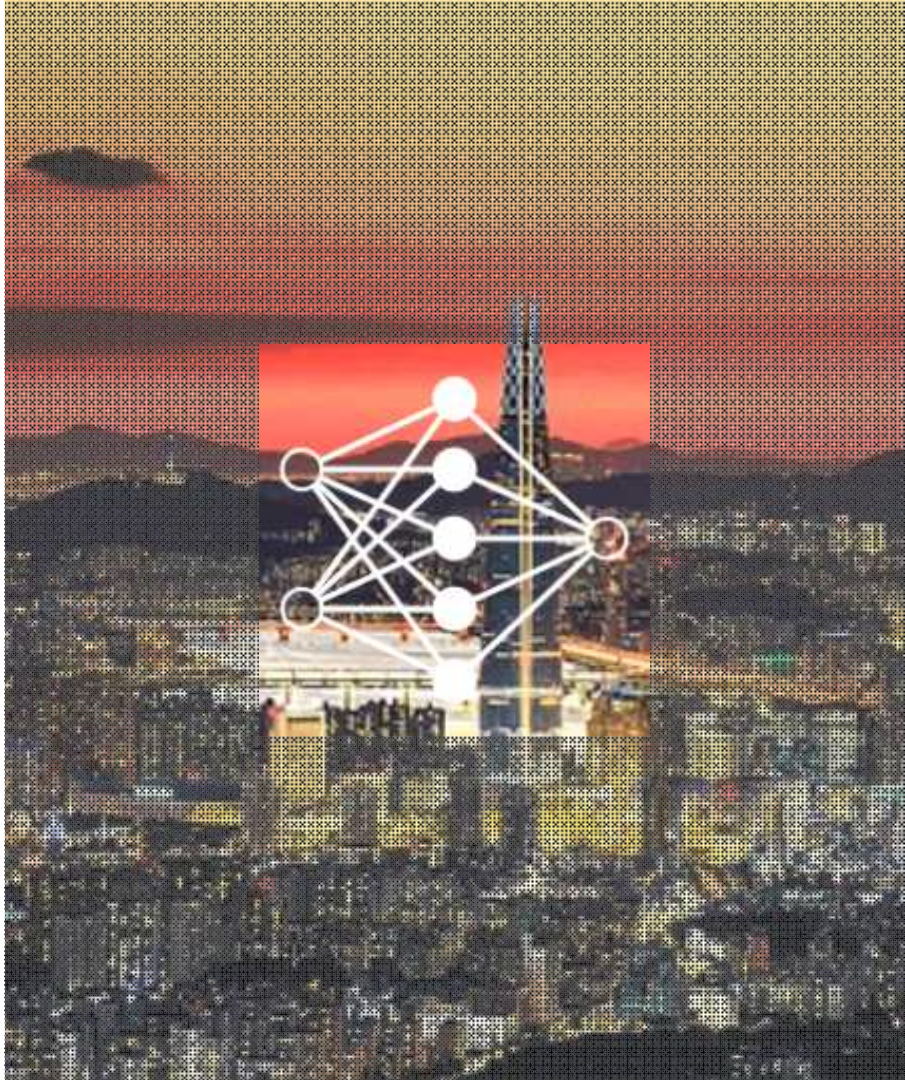
서울시 상권 AI 컨설팅 시스템

Multi-Agent 기반 매출예측·경쟁분석·입지평가 통합 플랫폼



최두석 | 2025.01.12

목차: 시스템 설계와 문제 해결의 흐름

연구 배경: 소상공인 창업 성공률 제고의 필요성

The Problem

창업 실패율의 심각성

매년 증가하는 소상공인 폐업률은 막대한 사회적 비용과 개인적 손실을 야기합니다.

기존 서비스의 한계

단편적인 정보 제공에 그치며, 실시간 변화를 반영하지 못해 의사결정 도구로서 부족합니다.

연도별 폐업 사업자 및 폐업률 추이 (출처: 국세청)

The Solution

"강남구에서 카페 창업하면 성공할까?"
-> AI 기반 종합 컨설팅
→ 데이터 근거 의사결정 지원

프로젝트 특징: 4대 핵심 에이전트의 유기적 협업

Time-Series Prediction

매출예측 (LSTM)

딥러닝 기반의 정교한 시계열 예측 모델을 적용하여 미래 수익성을 분석합니다.

Competitive Analysis

과거 사례 분석 (RAG)

FAISS Vector Store를 활용하여 주변 경쟁 업체를 실시간으로 검색하고 분석합니다.

Location Evaluation

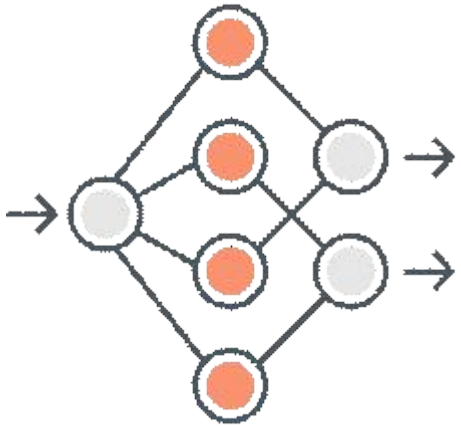
입지, 경쟁 평가 (Rule-based)

교통, 유동인구 등 다각도 지표를 통해 객관적인 입지 등급을 산출합니다.

Natural Language Report

종합리포트 (LLM)

Google Gemini를 통해 분석 결과를 전문가 수준의 자연어 리포트로 생성합니다.



LangGraph 기반
Multi-Agent 오케스트레이션

시스템 아키텍처: LangGraph 기반 지능형 워크플로우

01 사용자 인터페이스 (Gradio)

사용자가 분석 조건을 입력하면 시스템이 가동됩니다.

02 중앙 제어 (LangGraph Orchestrator)

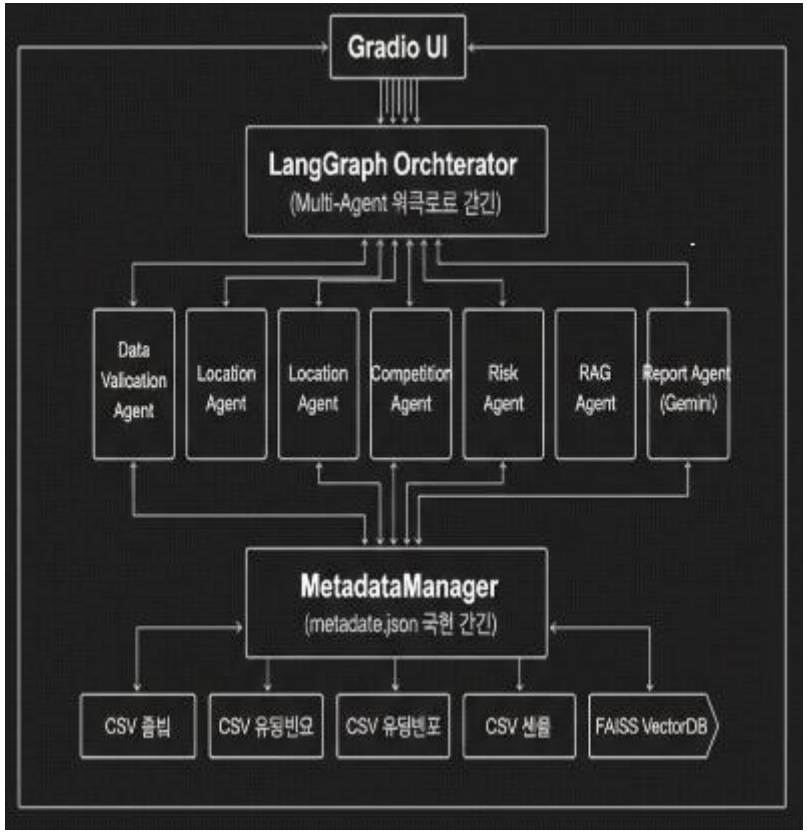
LangGraph가 각 에이전트의 실행 순서와 데이터 흐름을 관리합니다.

03 에이전트 레이어

데이터 검증, 매출, 경쟁, 입지, 위험, Rag, 리포트 에이전트가 유기적으로 작동합니다.

04 MetadataManager

모든 Agent가 metadata.json을 참조 ->
어떤 CSV 파일을 써야 하는지, 어떤 컬럼을 봐야 하는지를 중앙에서 관리



기술 스택: 데이터와 AI 기술의 결합

Data Sources

서울시 공공데이터

상권별 매출, 유동인구, 점포 현황 등 핵심 행정 데이터 수집

Core Frameworks

LangChain / Graph

Multi-Agent 오케스트레이션 및 복잡한 워크플로우 제어

TensorFlow / Keras

LSTM 기반 고성능 시계열 매출 예측 모델 구현

FAISS Vector Store

고속 유사도 검색을 통한 RAG 기반 경쟁 분석 시스템

Model & UI

Gemini 2.0 Flash

최신 LLM을 활용한 자연어 리포트 생성 및 데이터 요약

Gradio

사용자 친화적인 웹 기반 인터페이스 및 결과 시각화

매출예측 Agent: LSTM 기반 비선형 시계열 분석

Model Selection

왜 LSTM인가?

- 상권 데이터의 복잡한 **비선형 관계** 학습 최적화
- 과거 8분기 데이터의 **장기 의존성** 효과적 처리
- 시계열적 특성을 반영한 정교한 미래 매출 예측

Data Input

예측 목표 및 데이터

- **목표**: 다음 분기 예상 매출액 산출
- **입력**: 서울시 공공데이터 기반 2년치 시계열
- **변수**: 유동인구, 업종 밀집도, 과거 매출 등

Regional Insights

강남구의 다양한 업종 고매출, 동작구의 수산물 특화 상권, 용산구의 컴퓨터 업종 밀집 등 지역 및 업종별 비선형 특이점을 모델 학습에 반영하여 예측 정확도를 극대화했습니다.

- 매출예측 Agent 에 사용된 LSTM모델 소개 페이지

https://dooseok913.github.io/seoul-commercial-district/index.html

데이터 전처리 및 모델 성능: 높은 예측 정확도 확보

MAE

13.26억

평균 절대 오차

R^2 Score

0.985

결정 계수 (설명력)

MAPE

24.57%

평균 절대 백분율 오차

Winsorization

이상치를 처리하면서도 시퀀스 데이터를 유지하여 모델 안정성 확보

Square Root Transform

데이터의 압축 강도를 조절하여 정규성을 확보하고 학습 효율 증대

StandardScaler

평균 0, 분산 1로 정규화하여 LSTM 모델의 수렴 속도 최적화

LangGraph Multi-Agent

왜 Multi-Agent 인가?

단일 LLM의 한계:

- 모든 것을 한 번에 처리-> 정확도 저하
- 역할 분리 불가 -> 전문성 부족
- 디버깅 어려움 -> 어디서 틀렸는지 모름

Multi-Agent의 장점:

- 전문성 분리: 각 Agent가 하나의 역할에 집중
- 병렬 처리 가능: 독립적인 Agent는 동시 실행
- 디버깅 용이: Agent별로 결과 추적 가능
- 확장성: 새로운 Agent 추가 용이

LangGraph 선택 이유:

- LangChain 공식 Multi-Agent 프레임워크
- 상태 기반 워크플로우 (StateGraph)
- 조건부 분기 지원
- 에러 처리 및 재시도 내장

```python
# langgraph 워크플로우 정의 (조건부 분기 포함)
from langgraph.graph import StateGraph, END

workflow = StateGraph(AgentState)

# Agent 노드 추가
workflow.add_node("data_validation", data_validation_agent)
workflow.add_node("early_exit", early_exit_node)  # 조건부 분기: 조기 종료
workflow.add_node("location", location_agent)
workflow.add_node("competition", competition_agent)
workflow.add_node("risk", risk_agent)
workflow.add_node("rag", rag_agent)
workflow.add_node("report", report_agent)

# 시작점 설정
workflow.set_entry_point("data_validation")

# ★ 조건부 분기: 데이터 신뢰도에 따라 경로 결정
def route_after_validation(state):
    if state["validation_passed"]:  # 신뢰도 >= 30%
        return "continue"
    else:
        return "early_exit"  # 신뢰도 < 30% → 조기 종료

workflow.add_conditional_edges(
    "data_validation",
    route_after_validation,
    {
        "continue": "rag",      # 정상 경로
        "early_exit": "early_exit"  # 조기 종료 경로
    }
)

# 정상 경로: 순차 실행
workflow.add_edge("rag", "location")
workflow.add_edge("location", "competition")
workflow.add_edge("competition", "risk")
workflow.add_edge("risk", "report")
workflow.add_edge("report", END)

# 조기 종료 경로
workflow.add_edge("early_exit", END)
```

Agent 워크플로우

DataValidationAgent
- CSV 파일 존재 여부 확인
- 데이터 무결성 검증
- 신뢰도 점수 산출 (100점 만점)

RAGAgent
- 유사 창업 사례 검색
- 정책 문서 검색
- Hybrid Search (Vector + BM25 + Reranking)

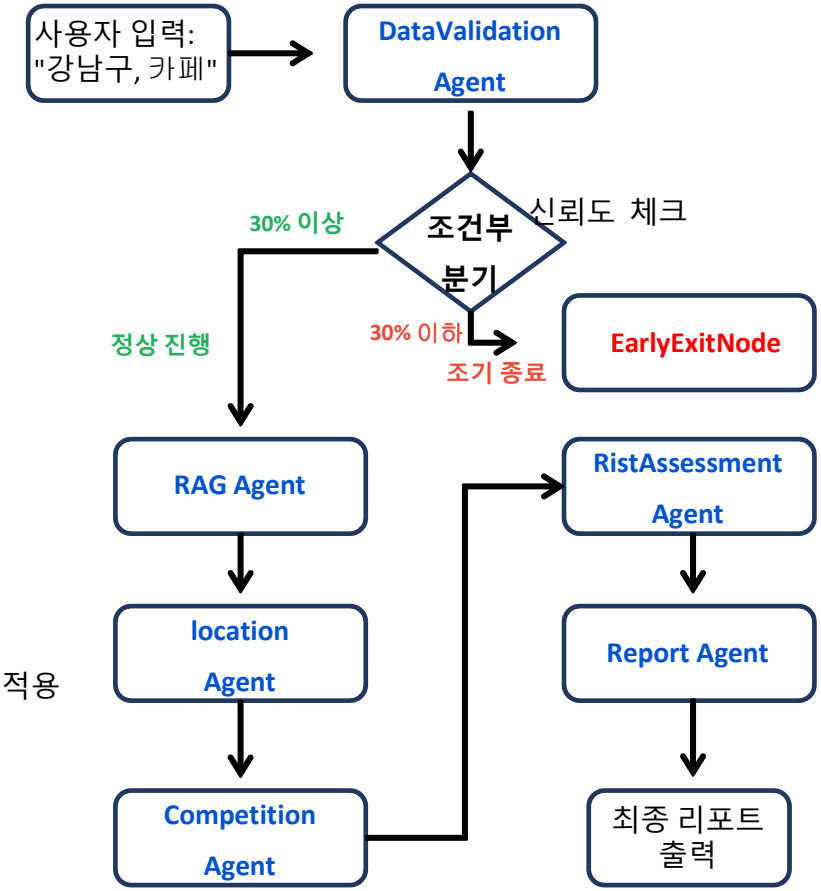
LocationAgent
- 상권 활성도 (유동인구 기반)
- 평균 소득 분석
- 소비 패턴 분석 (서울 평균 대비)

EarlyExitNode
- 데이터 부족 경고
- 분석 불가 사유 안내
- 해결 방법 제시

CompetitionAgent
- 동종 업종 점포 수
- 프랜차이즈 비율
- 개업/폐업률 분석

RiskAssessmentAgent
- 경쟁 리스크
- 입지 리스크
- 계절성 리스크

ReportAgent
- 모든 Agent 결과 통합
- 자연어 리포트 생성
- Hallucination 방지 프롬프트 적용



메타데이터 중심 설계

왜 메타데이터 인가?

문제점 (하드코딩):

- 파일 경로가 코드 곳곳에 흩어져 있음
- 컬럼명 변경 시 모든 코드 수정 필요
- 어떤 Agent가 어떤 파일을 쓰는지 파악 어려움

해결책 (metadata.json):

- 모든 데이터 소스 정보를 한 곳에 관리
- 파일 경로, 인코딩, 컬럼, 사용 Agent 명시
- 데이터 업데이트 정책까지 포함

구조:

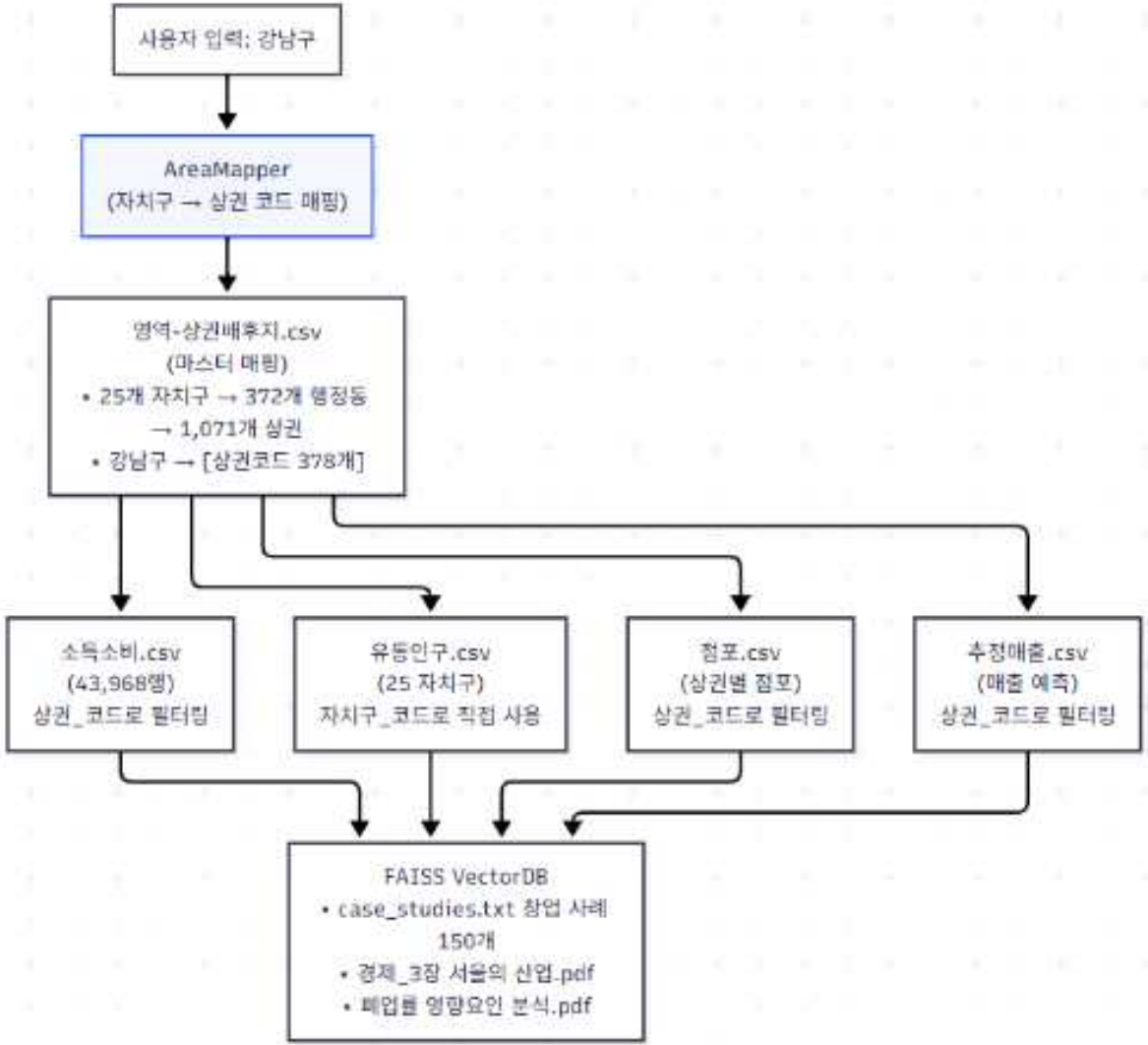
data_sources: CSV/PDF/TXT 파일 정보
rag_collections: 벡터 스토어 정보
mapping_relationships: 자치구-행정동-상권 매핑
agents: 각 Agent별 설정
system_config: 버전, 업데이트 정책

```
"area_mapping": {
    "file_path": "data/서울시 상권분석서비스(영역-상권배후지).csv",
    "file_type": "csv",
    "encoding": "cp949",
    "description": "마스터 매핑 테이블: 자치구-행정동-상권 연결",
"store_data": {
    "mapping_key": "상권_코드",
    "requires_mapping": true,
    "mapping_source": "area_mapping",
"used_by": [
    "ALL_AGENTS",
    "AreaMapper",
    "LocationAgent",
    "CompetitionAgent",
    "RAGAgent"
]
```

```
def list_sources_by_agent(self, agent_name: str) -> List[str]:
    """특정 에이전트가 사용하는 데이터 소스 목록"""
    sources = []
    for name, info in self.metadata.get('data_sources', {}).items():
        used_by = info.get('used_by', [])
        if agent_name in used_by:
            sources.append(name)
    return sources
```

```
"rag_collections": {
    "case_studies_vectorstore": {
        "type": "faiss",
        "path": "data/vectorstore_hybrid",
        "embedding_model": "sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2",
        "search_methods": ["vector", "bm25", "reranking"],
        "documents_indexed": 150,
        "last_update": "2026-01-24",
        "includes": [
            "case_studies",
            "policy_docs_economy",
            "policy_docs_closure"
```

데이터 연결 구조


사용자 입력: 강남구
AreaMapper
(자치구 → 상권 코드 매핑)
영역-상권매후지.csv
(마스터 매핑)
• 25개 자치구 → 372개 행정동
→ 1,071개 상권
• 강남구 → [상권코드 378개]
소득소비.csv
(43,968행)
상권_코드로 필터링
유동인구.csv
(25 자치구)
자치구_코드로 직접 사용
점포.csv
(상권별 점포)
상권_코드로 필터링
추정매출.csv
(매출 예측)
상권_코드로 필터링
FAISS VectorDB
• case_studies.txt 창업 사례
150개
• 경제_3장 서울의 산업.pdf
• 폐업률 영향요인 분석.pdf

RAG 시스템 개요

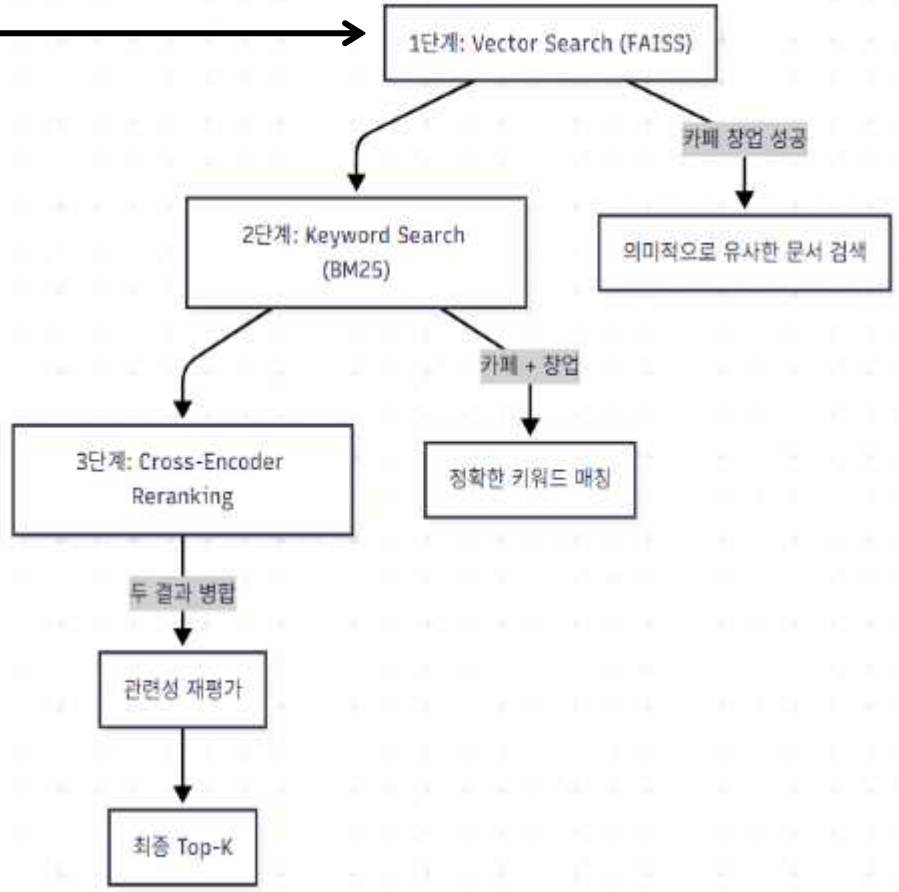
RAG (Retrieval Augmented Generation) 란?

- LLM의 지식 한계를 외부 문서 검색으로 보완
- "LLM이 모르는 것은 검색해서 알려준다"

우리 시스템의 RAG 용도:

- 과거 창업 성공/실패 사례 검색
- 서울시 산업 정책 문서 검색
- 폐업률 영향 요인 연구 검색

Hybrid Search (3단계):



FAISS vs Chroma

FAISS 선택 이유:

1. 속도 우선

- 150개 문서 검색에 수 밀리초
- 실시간 응답 필요

2. 검증된 기술

- Meta의 프로덕션 검색 시스템에서 사용
- Facebook, Instagram 검색에 적용됨

3. 메모리 효율

- 임베딩 압축 지원 (PQ, IVF)
- 대용량에도 메모리 부담 적음

4. LangChain 통합

- langchain_community.vectorstores.FAISS
- 표준 인터페이스 지원

```python
from langchain_community.vectorstores import FAISS
from langchain_community.embeddings import HuggingFaceEmbeddings

# 임베딩 모델 로드
embeddings = HuggingFaceEmbeddings(
    model_name="sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2"
)

# FAISS 벡터 스토어 생성
vectorstore = FAISS.from_documents(
    documents=split_docs,
    embedding=embeddings
)

# 저장
vectorstore.save_local("data/vectorstore_hybrid")

# 로드
vectorstore = FAISS.load_local(
    "data/vectorstore_hybrid",
    embeddings,
    allow_dangerous_deserialization=True
)

# 검색
results = vectorstore.similarity_search(query, k=5)
```

BM25 + Reranking

왜 Vector Search만으로 부족한가?

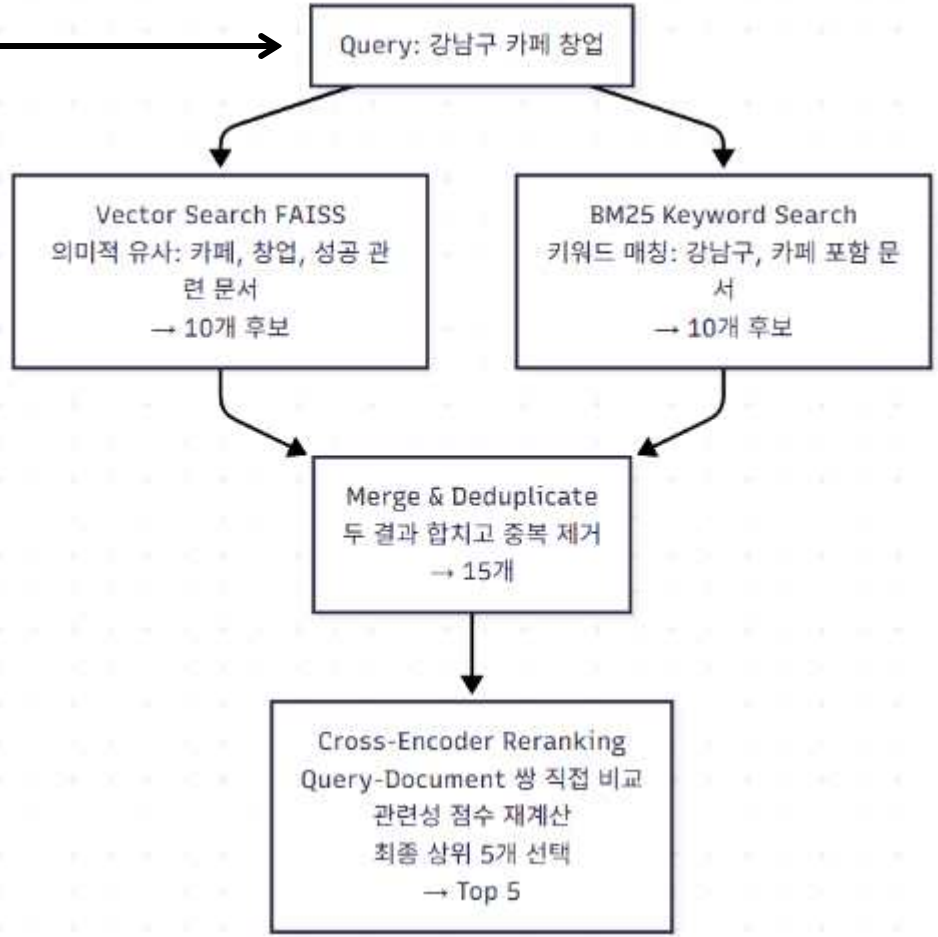
Vector Search 한계:

- "강남구 카페" → "서초구 음식점"도 유사하다고 판단
- 의미적 유사성만 보고, 정확한 키워드는 놓침

BM25 (Best Matching 25):

- 전통적인 키워드 검색 알고리즘
- TF-IDF 개선 버전
- "강남구"가 정확히 포함된 문서 우선

Hybrid Search 프로세스:



BM25 + Reranking

관련 코드

```python
from rank_bm25 import BM25Okapi
from sentence_transformers import CrossEncoder

class HybridSearcher:
    def __init__(self, documents, vectorstore):
        self.documents = documents
        self.vectorstore = vectorstore

        # BM25 인덱스 생성
        tokenized = [doc.split() for doc in documents]
        self.bm25 = BM25Okapi(tokenized)

        # Reranker 모델
        self.reranker = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-6-v2')

    def search(self, query, k=5):
        # 1. Vector Search
        vector_results = self.vectorstore.similarity_search(query, k=10)

        # 2. BM25 Search
        bm25_scores = self.bm25.get_scores(query.split())
        bm25_top_idx = bm25_scores.argsort()[-10:][::-1]
        bm25_results = [self.documents[i] for i in bm25_top_idx]

        # 3. Merge & Deduplicate
        all_results = list(set(vector_results + bm25_results))

        # 4. Reranking
        pairs = [[query, doc] for doc in all_results]
        scores = self.reranker.predict(pairs)

        # 상위 k개 반환
        ranked = sorted(zip(all_results, scores), key=lambda x: x[1], reverse=True)
        return [doc for doc, score in ranked[:k]]
```

임베딩 모델 선택

선택한 모델:

paraphrase-multilingual-MiniLM-L12-v2

왜 이 모델인가?

1. 다국어 지원 (Multilingual)
   - 한국어 지원
   - 영어, 중국어, 일본어 등 50+ 언어
   - 한국어 창업 사례 문서 임베딩에 필수

2. 경량 모델 (MiniLM)
   - 파라미터 수: 117M (BERT-base의 1/3)
   - 추론 속도: 5-10ms per document
   - 메모리: ~500MB

3. 의미 보존 (Paraphrase)
   - "카페 창업 성공"과 "커피숍 오픈 성공" 유사하게 인식
   - 다양한 표현을 같은 의미로 매핑

4. 벡터 차원
   - 384차원 출력
   - 768차원(BERT) 대비 절반 → 저장공간/속도 최적화

RAG 검색은 '정밀 분류'가 아니라
'후보 검색'이 목적
최종 판단은 reranking이 담당

이 모델은 한국어 문서에 대한 의미 표현 성능을 유지하면서도,
BERT 대비 1/3 수준의 파라미터와 384차원 벡터를 사용해,
대규모 문서 임베딩과 실시간 벡터 검색에 적합한 경량 구조를 제공합니다.
이를 통해 저장 공간과 검색 속도를 최적화하면서도,
RAG 기반 의미 검색 품질을 안정적으로 확보할 수 있었습니다.

Agent 상세 (1) - DataValidation + Location

1. DataValidationAgent

역할:

데이터 품질 검증 (파이프라인의 첫 번째 관문)

검증 항목:

- CSV 파일 존재 여부
- 필수 컬럼 존재 여부
- 데이터 무결성 (NaN, 이상치)
- 최신 분기 데이터 확인

2. LocationAgent

역할:

입지 평가

분석 항목:

- 상권 활성도: 유동인구 기반 (1-10점)
- 평균 소득: 월 평균 소득 기반 (1-10점)
- 소비 패턴: 서울 평균 대비 업종 추천

핵심 기술:

- AreaMapper: 자치구 → 상권 코드 변환
- 데이터 분포 기반 점수화: 백분위수 활용
- 서울 평균 대비 분석: 상대적 강점 파악

데이터 소스:

- 소득소비.csv (43,968행)
- 유동인구.csv (25개 자치구)

Agent 상세 (1) - DataValidation + Location

2. LocationAgent

```python
# 서울 전체 평균 계산 (v2.4)
def _calculate_seoul_average_consumption(self):
    total_food = self.income_df['식료품_지출_총금액'].sum()
    total_leisure = self.income_df['여가_지출_총금액'].sum()
    total_education = self.income_df['교육_지출_총금액'].sum()
    total_all = total_food + total_leisure + total_education

    return {
        '식료품': (total_food / total_all * 100),  # 58.9%
        '여가': (total_leisure / total_all * 100),  # 8.9%
        '교육': (total_education / total_all * 100)  # 32.2%
    }

# 소비 패턴 분석 - 서울 평균 대비 비교
def _analyze_consumption_pattern(self, food, leisure, education):
    seoul_avg = self._calculate_seoul_average_consumption()

    # 해당 구의 비율 계산
    food_pct = food / (food + leisure + education) * 100

    # 서울 평균 대비 차이
    food_diff = food_pct - seoul_avg['식료품']  # 예: -6.4%p

    # 평균보다 높은 업종만 추천
    if food_diff > 0:
        return f"식료품 업종 유망 (서울 평균 대비 +{food_diff:.1f}%p)"
```

Agent 상세 (2) - Competition + Risk

3. CompetitionAgent

역할:

경쟁 현황 분석

분석 항목:

- 동종 업종 총 점포 수
- 프랜차이즈 비율
- 개업률 / 폐업률
- 상권당 평균 점포 밀집도

4. RiskAssessmentAgent

역할:

종합 리스크 평가

리스크 요인:

- 경쟁 리스크: 점포 밀집도 기반
- 입지 리스크: 유동인구, 소득 기반
- 계절성 리스크: 업종별 계절 변동

핵심 기술:

- AreaMapper: 자치구 → 상권 코드 변환
- 데이터 분포 기반 점수화: 백분위수 활용
- 서울 평균 대비 분석: 상대적 강점 파악

데이터 소스:

- 소득소비.csv (43,968행)
- 유동인구.csv (25개 자치구)

Agent 상세 (2) - Competition + Risk

3. CompetitionAgent

```python
def analyze_competition(self, district, industry):
    # 해당 구 + 업종 데이터 필터링
    industry_data = self.df[
        (self.df['자치구'] == district) &
        (self.df['업종'] == industry)
    ]

    # 전체 점포 수 계산 (v2.3.2 수정)
    independent_count = industry_data['점포_수'].sum()
    franchise_count = industry_data['프랜차이즈_점포_수'].sum()
    total_stores = independent_count + franchise_count  # 전체!

    # 프랜차이즈 비율
    franchise_ratio = franchise_count / total_stores * 100

    # 폐업률 계산 (100% 상한 적용)
    closure_count = industry_data['폐업_수'].sum()
    closure_rate = min((closure_count / total_stores * 100), 100.0)

    return {
        'total_stores': total_stores,
        'franchise_ratio': franchise_ratio,
        'closure_rate': closure_rate
    }
```

Agent 상세 (3) - RAG + Report

5. RAGAgent

역할:

유사 사례 & 정책 검색

검색 대상:

- case_studies.txt: 창업 성공/실패 사례 150개
- 경제_3장 서울의 산업.pdf: 산업 구조 정책
- 폐업률 영향요인 분석.pdf: 폐업 리스크 연구

검색 기술:

- FAISS Vector Search
- BM25 Keyword Search
- Cross-Encoder Reranking

6. ReportAgent (Gemini 2.0 Flash)

역할:

최종 리포트 생성 (유일한 LLM Agent)

LLM: Google Gemini 2.0 Flash

- 빠른 응답 속도
- 한국어 품질 우수
- 비용 효율적

Hallucination 방지:

- 명시적 데이터 범위 제한
- "주어진 데이터만 사용" 지시
- 근거 제시 강제

Agent 상세 (3) - RAG + Report

6. ReportAgent (Gemini 2.0 Flash)

```python
from langchain_google_genai import ChatGoogleGenerativeAI

class ReportAgent:
    def __init__(self):
        self.llm = ChatGoogleGenerativeAI(
            model="gemini-2.0-flash",
            temperature=0.3,  # 낮은 창의성 → 사실 기반
            top_p=0.8
        )

    def generate_report(self, agent_results):
        prompt = f"""
당신은 창업 컨설턴트입니다.
아래 분석 결과를 바탕으로 종합 리포트를 작성하세요.

⚠️ 중요 규칙:
1. 제공된 데이터만 사용하세요. 추측하지 마세요.
2. 모든 수치에는 출처를 명시하세요.
3. 데이터에 없는 내용은 "정보 없음"으로 표시하세요.

=== 분석 결과 ===
[입지 평가]
{agent_results['location']}

[경쟁 분석]
{agent_results['competition']}

[리스크 평가]
{agent_results['risk']}

[유사 사례]
{agent_results['rag']}

위 내용을 바탕으로 창업 추천 여부와 핵심 조언을 작성하세요.
"""
        response = self.llm.invoke(prompt)
        return response.content
```

Hallucination 방지

Hallucination이란?

LLM이 사실처럼 보이지만 실제로는 틀린 정보를 생성하는 현상

1. LLM 사용 최소화

- 6개 Agent 중 5개는 LLM 없이 동작
- 데이터 분석은 Pandas로 직접 계산
- LLM은 최종 리포트 생성에만 사용

2. Citation Tracking (출처 추적)

- 모든 수치에 출처 명시
- 예: "폐업률 12.3% (출처: 서울시 점포 CSV)"
- 사용자가 검증 가능

3. 계산 근거 제공

- 점수 산정 기준 공개
- "유동인구 9천만명 → 7점 (기준: 1.18억=10점)"
- 블랙박스 제거

4. 프롬프트 제약

- "제공된 데이터만 사용"
- "추측 금지"
- "정보 없으면 '정보 없음' 표시"

Hallucination 방지

Hallucination 관련 코드

```python
# 1. Citation Tracking (출처 추적)
def format_output(self, district, profile):
    if 'avg_income_raw' in profile:
        income_raw = profile['avg_income_raw']
        income_basis = f"월 평균 소득 {income_raw:,.0f}원"
        income_basis += " (출처: 서울시 상권분석서비스 소득소비 CSV)"

    return f"""
    평균 소득: {profile['avg_income']}/10점
      └ 산정 근거: {income_basis}
      └ 점수 기준: 440만 이상=10점, 280만(평균)=6점
    """

# 2. 계산 근거 제공
def _calculate_income_score(self, avg_income):
    """
    실제 데이터 분포 기반 점수화:
    - 95% 지점 (443만원) → 10점
    - 평균 (280만원) → 6점
    - 중앙값 (258만원) → 5점
    """
    if avg_income >= 4400000:  # 95%
        return 10
    elif avg_income >= 2800000:  # 평균
        return 6
```

프롬프트 엔지니어링

ReportAgent 프롬프트 설계

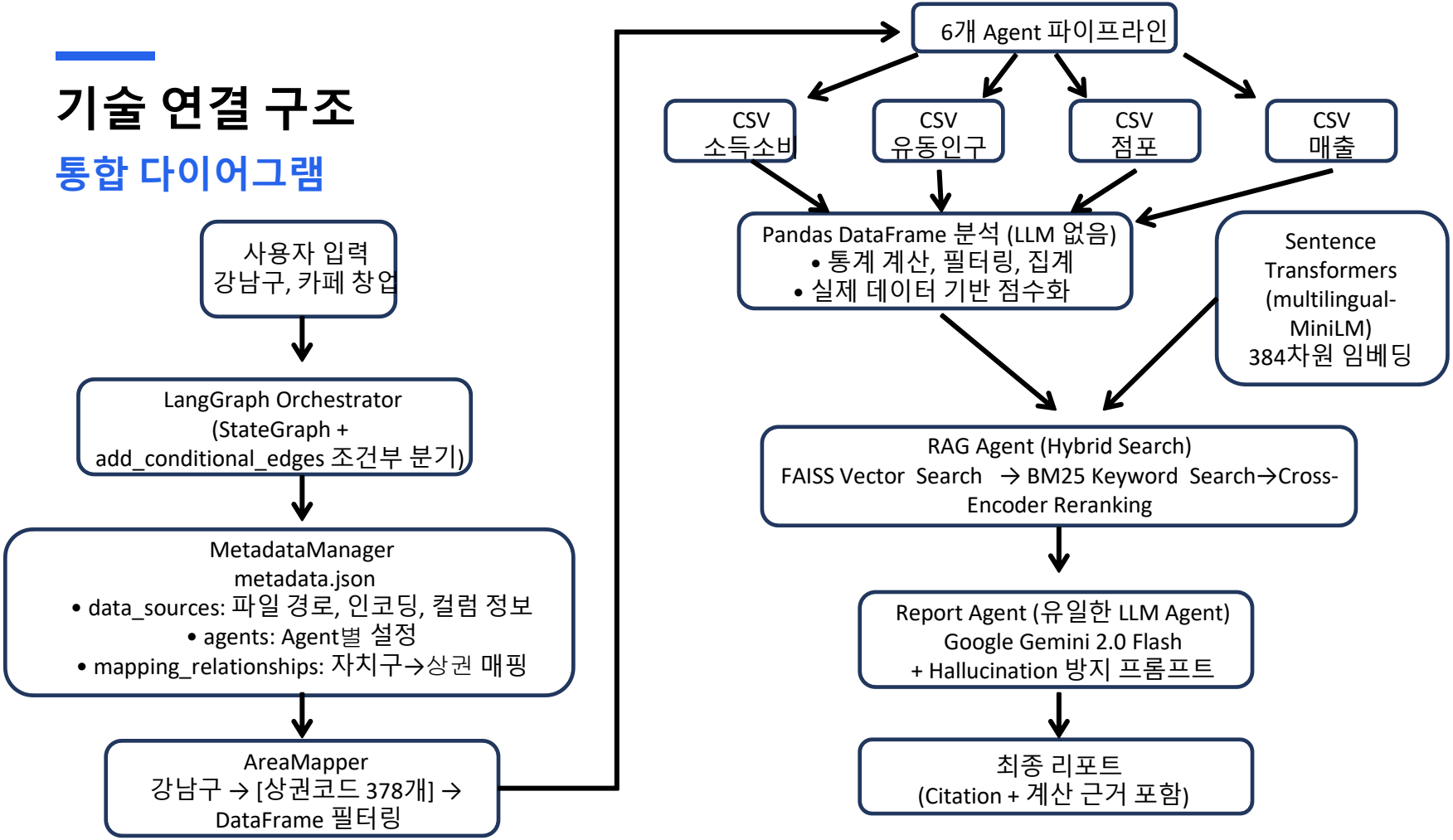
핵심 원칙:

1. 역할 정의: "당신은 창업 컨설턴트입니다"
2. 데이터 범위 제한: "아래 데이터만 사용"
3. 금지 사항 명시: "추측하지 마세요"
4. 출력 형식 지정: 구조화된 리포트 템플릿
5. 폴백 지시: "정보 없으면 '정보 없음' 표시"

```
REPORT_PROMPT = """
당신은 서울시 창업 컨설턴트입니다.
아래 분석 결과를 바탕으로 종합 창업 컨설팅 리포트를 작성하세요.

⚠️ 중요 규칙 (반드시 준수):
1. 제공된 데이터만 사용하세요. 외부 지식이나 추측을 포함하지 마세요.
2. 모든 수치에는 해당 Agent 이름을 출처로 명시하세요.
   예: "폐업률 12.3% (Competition Agent)"
3. 데이터에 없는 내용은 "해당 정보 없음"으로 표시하세요.
4. 확실하지 않은 내용은 "추가 조사 필요"로 표시하세요.

=== 입지 평가 (Location Agent) ===
{location_result}

=== 경쟁 분석 (Competition Agent) ===
{competition_result}

--- 리스크 평가 (Risk Agent) ---
{risk_result}

--- 유사 사례 (RAG Agent) ---
{rag_result}

위 분석을 종합하여 다음 형식으로 리포트를 작성하세요:

## 📊 종합 평가
[창업 추천 여부 및 한줄 요약]

## 💡 핵심 인사이트 (3가지)
1. ...
2. ...
3. ...

## ⚠️ 주의사항 (2가지)
1. ...
2. ...
```

기술 연결 구조

통합 다이어그램



결론 & 성과

핵심 차별점

1. Multi-Agent 아키텍처

- 6개 전문 Agent가 역할 분담
  - LangGraph로 워크플로우 관리
  - 디버깅 및 확장 용이

2. 메타데이터 중심 설계

- metadata.json으로 중앙 집중 관리
  - 하드코딩 제거, 유지보수성 향상
  - 데이터 변경 시 한 곳만 수정

3. Hybrid RAG

- Vector + BM25 + Reranking 3단계
- 검색 품질 대폭 향상
- 한국어 최적화

4. Hallucination 방지

- LLM 사용 최소화 (6개 중 1개)
  - Citation Tracking
  - 계산 근거 공개
  - 프롬프트 제약

배운 점

- LLM은 도구일 뿐, 데이터가 핵심
- Multi-Agent로 복잡한 문제 분할
- 메타데이터로 시스템 통합
- Hallucination은 구조로 방지