전체 책 구성

1. 테이블(직원, 휴가, 매출)RDB
2. FastAPI로 간단한 회사 CRUD 사이트 (JinJa 템플릿) - 여기까지만 만들어서 gitclone하기 ---- git clone
3. 사내문서들 존재(사내규정, 신입사원알아야될내용, 프로세스)
4. 사내문서들 (hwp, pdf, excel 널려있음)
5. RAG+Vector구축
6. 회사 사내 사이트에 ChatGPT같은 화면있음 질문 : A직원 휴가 이번년도 몇개 남았어? - 2개남았어!! RDB + MCP 질문 : 새로운 직원들어왔는데, 사내규정 알려줘 (VectorDB + RAG)
7. Rag 튜닝
    - ReRanker, Hybrid Search, PDF 이미지 처리 등 (PART 7 참고)

사용기술 : RAG+VectorDB Postgre Python FastAPI+Jinja템플릿 MCP 울라마 + 딥씨크 R1 (컴퓨터 메모리 16GB 이상이어야 함)

예제: “사내 문서 기반 AI 업무 비서 (RAG + MCP)”

PART 0. 이 책의 목표와 최종 완성본 미리보기

> 목적: 독자가 "무엇을 만들 수 있는지" 먼저 보여주고 동기 부여

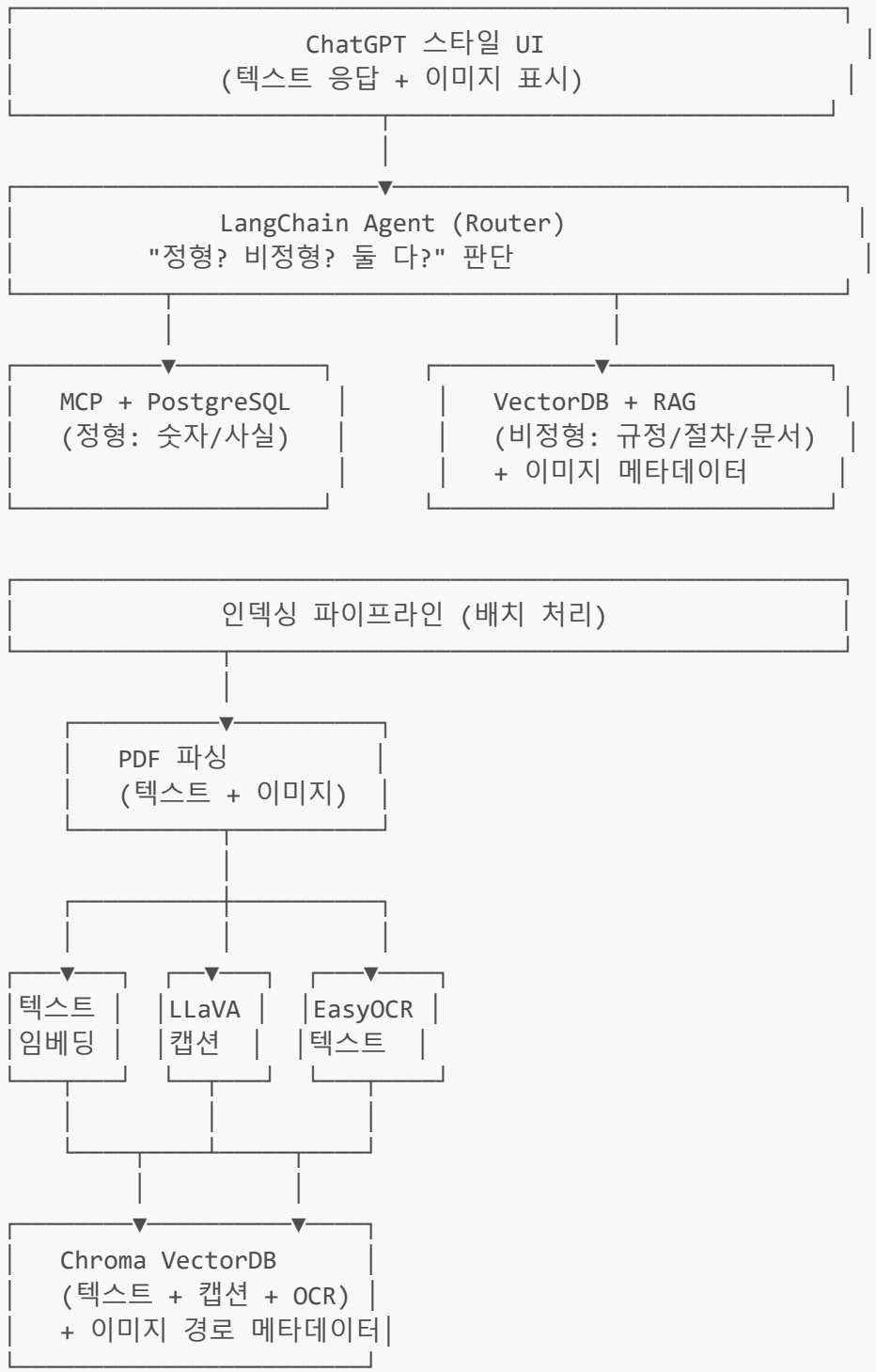
0.1 이 책이 다루는 범위

- **Fine-tuning이 아니라 RAG + MCP**
- 모델을 재학습하지 않고, 기존 LLM에 "회사 맥락"을 주입하는 방식
- 비용 효율적 + 데이터 보안 유지 가능

0.2 최종 결과물 데모 시나리오

질문 유형	예시 질문	처리 방식
정형+비정형 복합	"김대리 남은 연차 + 휴가 규정까지 같이 알려줘"	MCP(DB) + RAG(문서)
정형 데이터 조회	"이번 달 영업팀 매출 합계 알려줘"	MCP(DB)
비정형 문서 검색	"신입사원 온보딩 절차 알려줘"	RAG(VectorDB)
근거 포함 요약	"이번 달 영업팀 매출 요약해줘(근거 포함)"	MCP + RAG

0.3 아키텍처 한 장 요약



0.4 사용 기술 스택

영역	기술	용도	메모리 요구사항
텍스트 LLM	Ollama + DeepSeek R1	질의응답 생성 (런타임)	8-16GB
Vision LLM	Ollama + LLaVA / Qwen2-VL	이미지 캡션 생성 (인덱싱 시)	4-8GB
OCR	EasyOCR	이미지 텍스트 추출 (인덱싱 시)	1-2GB
Backend	FastAPI + Jinja2	웹 서버 + 템플릿	1GB

영역	기술	용도	메모리 요구사항
정형 DB	PostgreSQL	직원/휴가/매출 데이터	1GB
벡터 DB	Chroma	문서 임베딩 저장	1GB
오케스트레이션	LangChain	RAG + Agent 구성	1GB
도구 연동	MCP	LLM ↔ DB 연결	-
임베딩	ko-sroberta-multitask	텍스트 벡터화	1GB

최소 요구사항: RAM 16GB (DeepSeek R1 + LLaVA 동시 실행 불가 시 인덱싱/런타임 분리)

0.5 이 책을 마치면 할 수 있는 것

- 사내 문서 기반 AI 챗봇 구축
- 정형(DB) + 비정형(문서) 데이터 통합 질의응답
- RAG 파이프라인 설계 및 튜닝
- 로컬 LLM 활용 (비용 절감 + 보안)

PART 0.5. 개발 환경 설정

> 목적: 독자가 막힘 없이 따라할 수 있도록 환경 구축 가이드 제공

0.5.1 필수 요구사항

- Python 3.10 이상
- RAM 16GB 이상 (로컬 LLM 실행 시)
- 저장공간 20GB 이상

0.5.2 Ollama + DeepSeek R1 설치

- Ollama 설치 (Windows/Mac/Linux)
- DeepSeek R1 모델 다운로드: `ollama pull deepseek-r1`
- 모델 테스트: `ollama run deepseek-r1`
- (옵션) GPU 설정 및 메모리 최적화

0.5.3 Python 가상환경 및 의존성

```
python -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate
pip install -r requirements.txt
```

0.5.4 PostgreSQL 설치 및 설정

- Docker로 PostgreSQL 실행 (권장)
- 데이터베이스/유저 생성
- 접속 테스트

0.5.5 프로젝트 클론 및 초기 설정

```
git clone <repository-url>
cp .env.example .env
# .env 파일 수정 (DB 접속 정보, API 키 등)
```

0.5.6 주요 의존성 목록

패키지	용도
fastapi	웹 프레임워크
uvicorn	ASGI 서버
sqlalchemy	ORM
psycopg2-binary	PostgreSQL 드라이버
langchain	RAG 오케스트레이션
chromadb	벡터 DB
sentence-transformers	임베딩 모델
pypdf, python-docx, openpyxl	문서 파싱

PART 1. FastAPI로 "초간단 사내 시스템" 만들기

> 목적: git clone으로 바로 시작할 수 있는 기본 시스템 구축

1.1 프로젝트 구성

- 폴더 구조 설명
- 환경변수 설정 (.env)
- 실행 방법: `uvicorn main:app --reload`

1.2 데이터 모델 설계 (3테이블)

테이블	컬럼	설명
employee	id, name, dept, email, hire_date	직원 정보
leave_balance	id, employee_id, year, total, used, remaining	휴가 잔여
sales	id, dept, amount, date, description	매출 데이터

1.3 CRUD API 구현

- 직원: 등록/조회/수정/삭제
- 휴가: 잔여 조회/사용 등록/수정
- 매출: 입력/기간별 조회/부서별 집계

1.4 관리자 Admin UI

- Jinja2 템플릿 기반 간단한 입력 화면
- 데이터 조회/수정 폼

PART 2. 사내 문서 수집 전략과 문서 표준 만들기

> 목적: "RAG는 문서 품질과 구조가 반"을 체감하게 만들기

2.1 어떤 문서를 넣을 것인가(교재용 추천 세트)

- **사내 규정**: 인사/휴가/근태/보안 규정
- **업무 매뉴얼**: 신규 입사자 온보딩, 결재/보고 프로세스
- **기술 문서**: 운영 가이드, 장애 대응(runbook), 배포 절차
- **교육 자료**: 사내 교육 슬라이드/핸드북(요약하기 좋음)
- (옵션) **FAQ**: 자주 묻는 질문/답변(정답형 데이터)

2.2 문서 형식 지원 범위

형식	지원	비고
Markdown (.md)	☑	가장 깔끔
PDF (.pdf)	☑	텍스트 추출 필요
Word (.docx)	☑	python-docx
Excel (.xlsx)	☑	표→텍스트 변환 필요
HWP	⚠	PDF로 변환 후 처리

2.3 문서 표준 규칙 (RAG 품질의 핵심)

- 파일명 규칙: `{부서}_{문서종류}_{버전}.pdf`
- 메타데이터: 문서명, 버전, 발행일, 담당 부서
- 섹션 헤더 규칙: `##`, `1.`, `1.1` 형식 통일
- 개정 이력: 문서 상단에 최신 순 기록

2.4 문서 수집 파이프라인

```
docs/
├── hr/          # 인사/휴가 규정
├── ops/         # 운영/업무 매뉴얼
├── security/    # 보안 정책
├── onboarding/  # 신입사원 자료
└── faq/         # 자주 묻는 질문
```

PART 3. VectorDB 구축: 문서를 "검색 가능한 지식"으로 바꾸기

> 목적: 문서 → 청크 → 임베딩 → VectorDB 구축의 정석

3.1 텍스트 추출 전략(형식별)

- PDF: `pypdf` 또는 `pdfplumber`
- DOCX: `python-docx`
- XLSX: `openpyxl` → 표를 텍스트로 정규화

3.2 Chunk 설계 (초기 버전)

- fixed-size chunk (500~1000자) + overlap (10~20%)
- `RecursiveCharacterTextSplitter` 사용

3.3 메타데이터 설계 (검색 품질 핵심!)

메타데이터	용도
doc_id	문서 고유 식별자
title	문서 제목
section	섹션/챕터
version	버전 (최신 필터링용)
date	작성/수정일
department	부서
source_path	원본 파일 경로

3.4 임베딩 모델 선택

- 로컬: `sentence-transformers` (all-MiniLM, multilingual)
- 클라우드: OpenAI `text-embedding-3-small`
- 한국어 특화: `ko-sroberta-multitask`

3.5 VectorDB 선택 (교재 기준)

- **Chroma**: 로컬, 빠른 시작, 교재 기준
- (참고) Pinecone, Weaviate: 클라우드/프로덕션용

3.6 인덱싱 실행 & 검증

- 인덱싱 스크립트 실행
- 테스트 쿼리로 검색 동작 확인
- 검색 결과 품질 검토

PART 4. RAG로 Q&A 엔진 만들기

> 목적: "일단 되는 RAG"를 빨리 만든 뒤, 이후에 튜닝 파트로 확장

4.1 RAG 최소 동작 구현 (LangChain)

```
질문 → Retriever(검색) → Prompt(컨텍스트 삽입) → LLM → 답변
```

4.2 RAG 프롬프트 기본 템플릿

```
"""다음 문서를 참고하여 질문에 답하세요.
문서에 없는 내용은 '확인되지 않음'이라고 답하세요.
반드시 출처(문서명, 섹션)를 함께 제시하세요.

[문서]
{context}

[질문]
{question}
"""
```

4.3 "출처 표시" 응답 포맷

```
{
  "answer": "답변 내용",
  "sources": [{ "title": "휴가규정", "section": "2.3", "snippet": "..." }]
}
```

4.4 ChatGPT 스타일 UI 연결

- 백엔드: `/api/chat` (POST) - 스트리밍 응답
- 프론트: Jinja2 + JavaScript 대화형 UI
- 히스토리 관리 (세션/DB)

PART 5. 정형 MCP, 비정형 VectorDB+RAG: 통합 에이전트 설계

> 목적: LLM이 "검색/DB조회"를 판단해서 조합하는 구조 만들기

5.1 정형/비정형 분리 원칙

유형	예시	처리
정형 (사실/숫자)	휴가 잔여일, 매출 합계	MCP + SQL
비정형 (설명/규정)	휴가 규정, 온보딩 절차	VectorDB + RAG
복합	휴가 잔여 + 규정 설명	MCP + RAG 조합

5.2 질문 라우팅 전략

- **규칙 기반**: 키워드 매칭 ("몇 개", "합계" → DB)
- **스키마 기반**: 테이블 컬럼명과 매칭
- **LLM 판단**: 위 규칙으로 판단 어려울 때

5.3 통합 응답 전략 (핵심!)

```
1. 질문 분석 → 정형? 비정형? 둘 다?
2. 필요한 소스에서 데이터 수집
3. 통합 컨텍스트 구성
4. LLM에게 최종 답변 생성 요청
```

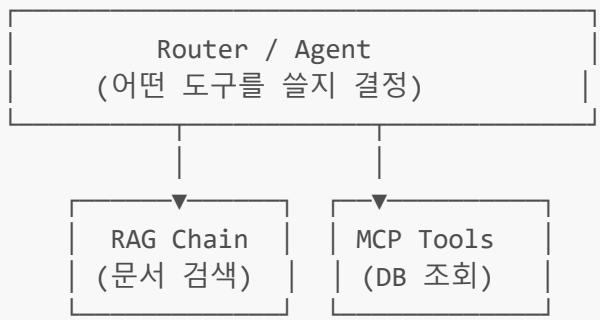
5.4 대표 질문 시나리오 10개

#	질문	유형
1	김대리 연차 며칠 남았어?	정형
2	연차 사용 규정 알려줘	비정형
3	김대리 연차 몇 개고, 사용 규정은?	복합
4	이번 달 영업팀 매출 합계	정형
5	매출 보고서 작성 방법	비정형
6	신입사원 첫 주에 뭐 해야 해?	비정형
7	재택근무 신청 방법	비정형
8	부서별 매출 비교해줘	정형
9	보안 정책 위반 시 처리 절차	비정형
10	영업팀 직원 누구야?	정형

PART 6. LangChain으로 연결 전략 세팅

> 목적: "어떤 체인/구성으로 묶을지"를 책의 표준으로 제시

6.1 기본 구성 3종 세트



6.2 Router 전략 (추천)

1. 질문 분석: "문서 검색 필요?" 판단
2. 질문 분석: "DB 조회 필요?" 판단
3. 실행 순서: **DB 조회 → 문서 검색 → 최종 조합**
4. 응답 생성

6.3 MCP Tool 설계

```python
@tool
def get_leave_balance(employee_name: str) -> dict:
    """직원의 휴가 잔여일 조회"""

@tool
def get_sales_sum(dept: str, start_date: str, end_date: str) -> dict:
    """부서별 기간 매출 합계"""

@tool
def list_employees(dept: str = None) -> list:
    """직원 목록 조회 (부서 필터 옵션)"""

@tool
def search_documents(query: str) -> list:
    """사내 문서 검색"""
```

6.4 운영 설정

- **Timeout/Retry**: LLM 호출 타임아웃, 재시도 정책
- **로깅**: 프롬프트, 컨텍스트, 응답 시간 기록
- **캐싱**: 동일 질문 캐시, 임베딩 캐시
- **비용 관리**: 토큰 사용량 모니터링

PART 7. RAG 튜닝: "되는 수준"에서 "쓸만한 수준"으로

> 목적: 이 파트가 책의 하이라이트. 실전 문제를 해결하는 순서대로 구성

7.1 증상으로 시작하는 튜닝(문제 → 처방)

증상	원인	처방
엉뚱한 문서를 가져온다	검색 품질 낮음	ReRanker, Hybrid Search
정확한데 근거가 없다	출처 추적 미흡	메타데이터 강화, 프롬프트 수정
문서에 없는 걸 지어낸다	Hallucination	프롬프트 튜닝, 근거 강제
최신 규정이 아닌 옛 문서를 인용	버전 관리 미흡	metadata filtering
긴 문서에서 핵심을 못 찾음	청크 설계 문제	Parent Document Retriever

7.2 Chunk 튜닝

- fixed-size chunk → **semantic chunk**(제목/문단 기반)로 전환
- overlap 조정 실험 (10~20% 권장)
- **RecursiveCharacterTextSplitter** 활용

7.3 Retriever 튜닝

- k 값 실험 (3~10 범위)
- similarity threshold 설정
- metadata filtering (최신 버전만, 부서별 필터)

7.4 ReRanker (검색 품질 핵심!)

> 1차 검색(Vector Search) 후 결과를 **재순위화**하여 정확도 향상

- **Cross-Encoder ReRanker**: 질문+문서를 함께 입력하여 관련도 점수 산출
- **Cohere Rerank API**: 간편한 클라우드 ReRanker
- **BGE Reranker**: 로컬에서 돌릴 수 있는 오픈소스 옵션
- 적용 패턴: `Vector Search(top_k=20) → ReRanker → top_k=5 사용`

7.5 Hybrid Search (키워드 + 벡터 결합)

> 벡터 검색만으로는 정확한 용어/고유명사 검색이 약함

- **BM25**(키워드 검색) + **Vector Search** 결합
- **Ensemble Retriever**: 여러 검색기 결과를 가중 평균
- 적용 예시:
  - "휴가" 검색 → 벡터는 "연차, 휴식" 등도 찾음
  - "HR-2024-001" 검색 → 키워드가 정확히 매칭

7.6 고급 Retriever 전략

- **Parent Document Retriever**: 작은 청크로 검색 → 큰 원본 문서 반환
- **Self-Query Retriever**: LLM이 메타데이터 필터를 자동 생성
- **Contextual Compression**: 검색된 문서에서 관련 부분만 추출

7.7 Query Rewrite / Multi-Query

- **Query Expansion**: 질문을 여러 버전으로 확장
- **HyDE**: 가상 답변을 먼저 생성 → 그걸로 검색
- 약어/동의어 처리 (예: "연차" ↔ "휴가" ↔ "annual leave")

7.8 프롬프트 튜닝

- 근거 우선 프롬프트: "반드시 문서 내용을 인용하라"
- "모르면 모른다" 강제: "문서에 없으면 '확인되지 않음'이라고 답하라"
- 답변 포맷 고정: JSON, 표 형식 등

7.9 PDF 이미지 처리 전략 (하이브리드 방식)

> 텍스트 추출이 어려운 이미지(차트, 다이어그램, 표 이미지)를 어떻게 할 것인가?

핵심 문제: DeepSeek R1은 텍스트 LLM

- **DeepSeek R1은 텍스트만 처리 가능**: PDF 내 이미지를 벡터DB에 그대로 넣어도 **이미지 의미를 이해하지 못함**
- **해결책**: 이미지를 "텍스트로 변환"하여 인덱싱 + 원본 이미지는 메타데이터로 저장
- **중요**: DeepSeek R1 못해도 **LLaVA는 가능** (역할이 다름)
  - DeepSeek R1: 런타임 질의응답 (항상 실행)
  - LLaVA: 인덱싱 시 이미지 캡션 생성 (배치 처리 가능)

메모리 상황별 권장 구성

메모리	텍스트 LLM	Vision 모델	OCR	전략
16GB+	DeepSeek R1	LLaVA-7B	EasyOCR	하이브리드 (LLaVA + OCR)
8-12GB	Llama 3.2 3B / Phi-3-mini	BakLLaVA / LLaVA-7B (양자화)	EasyOCR	하이브리드 (경량)
4-8GB	Llama 3.2 1B / Phi-3-mini	없음 (인덱싱 시만 LLaVA 실행 후 종료)	EasyOCR	OCR 중심 + 이미지 저장

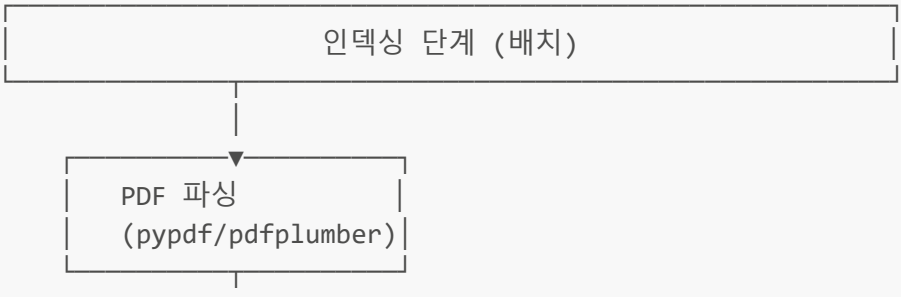
LLaVA vs OCR 선택 가이드

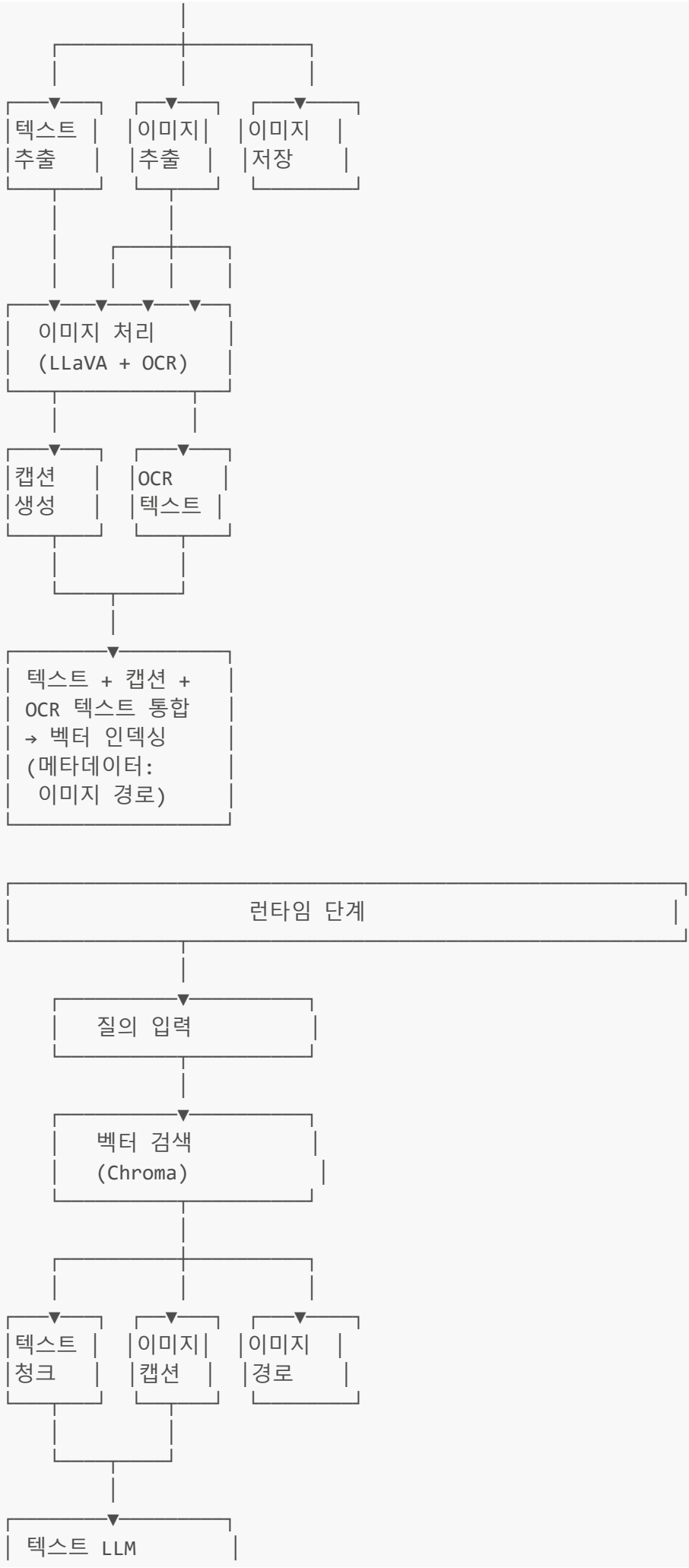
항목	LLaVA (Vision LLM)	OCR (EasyOCR)
용도	이미지 전체 의미 이해	이미지 내 텍스트 추출
적합한 경우	차트, 다이어그램, 그래프	스캔 문서, 표 이미지
메모리	4-8GB	1-2GB
속도	느림 (초당 1-2개)	빠름 (초당 10개+)

권장: 하이브리드 방식 (이미지 타입 자동 판단)

- 텍스트 많음 → OCR 우선
- 차트/다이어그램 → LLaVA 캡션 생성

최종 아키텍처 (무료/오픈소스 기반)




텍스트 추출
이미지 추출
이미지 저장
이미지 처리 (LLaVA + OCR)
캡션 생성
OCR 텍스트
텍스트 + 캡션 + OCR 텍스트 통합 → 벡터 인덱싱 (메타데이터: 이미지 경로)
런타임 단계
질의 입력
벡터 검색 (Chroma)
텍스트 청크
이미지 캡션
이미지 경로
텍스트 LLM

```
| (DeepSeek R1/     |
|  Llama 3.2)       |
| 응답 생성          |
└──────────┬─────────┘
           |
┌──────────▼─────────┐
| UI 응답            |
| (텍스트 + 이미지)  |
└────────────────────┘
```

구현 핵심 코드

```python
# 1. 이미지 처리 (하이브리드)
def process_image_hybrid(image_path):
    """이미지 타입에 따라 자동 선택"""
    # OCR 시도 (빠름)
    ocr_text = easyocr.Reader(['ko', 'en']).readtext(image_path)
    ocr_text = ' '.join([r[1] for r in ocr_text])

    if len(ocr_text.strip()) > 50:  # 텍스트가 많으면
        return {'type': 'document', 'text': ocr_text, 'method': 'ocr'}
    else:  # 차트/다이어그램
        caption = ollama.chat(
            model='llava',
            messages=[{
                'role': 'user',
                'content': '이 이미지를 자세히 설명해주세요.',
                'images': [image_path]
            }]
        )['message']['content']
        return {
            'type': 'diagram',
            'text': f"{caption}\n\n[OCR 텍스트]\n{ocr_text}",
            'method': 'llava'
        }

# 2. 인덱싱
def index_with_images(pdf_path, doc_id):
    # 텍스트 + 이미지 추출
    text_chunks, images = extract_from_pdf(pdf_path)

    # 이미지 처리
    for img in images:
        processed = process_image_hybrid(img['path'])
        # 벡터DB에 인덱싱 (텍스트 + 이미지 경로 메타데이터)
        vectorstore.add_texts(
            texts=[processed['text']],
            metadatas=[{
                'doc_id': doc_id,
                'type': 'image',
                'image_path': img['path'],
```

```
                'page': img['page']
            }]
        )

# 3. 검색 및 응답
def search_and_respond(query, vectorstore, llm):
    results = vectorstore.similarity_search_with_score(query, k=5)

    # 텍스트와 이미지 분리
    text_context = [r[0].page_content for r in results if r[0].metadata['type'] ==
'text']
    images = [r[0].metadata['image_path'] for r in results if
r[0].metadata['type'] == 'image']

    # LLM 응답 생성
    response = llm.invoke(f"문서: {text_context}\n질문: {query}")

    return {
        'answer': response,
        'images': images  # UI에서 표시
    }
```

무료/오픈소스 스택

구성 요소	무료 도구	메모리	용도
텍스트 LLM	DeepSeek R1 / Llama 3.2	8-16GB	질의응답 생성
Vision LLM	LLaVA / Qwen2-VL (Ollama)	4-8GB	이미지 캡션 생성
OCR	EasyOCR	1-2GB	이미지 텍스트 추출
벡터DB	Chroma	1GB	임베딩 저장
임베딩	ko-sroberta-multitask	1GB	텍스트 벡터화

실전 팁

상황	권장 방식	도구
차트/그래프	LLaVA 캡션 + 이미지 저장	LLaVA
스캔 문서	OCR → 텍스트 변환	EasyOCR
표 이미지	OCR + LLaVA 구조 설명	EasyOCR + LLaVA
메모리 부족	OCR만 사용 + 이미지 저장	EasyOCR

7.10 평가 체계 (필수!)

- **테스트 질문 세트**: 최소 30개 이상의 평가용 질문
- **정답 데이터셋**: 예상 답변 + 출처 문서 매핑
- **평가 지표**:

    - Retrieval 정확도: 올바른 문서를 가져왔는가?
    - Answer 정확도: 답변이 맞는가?
    - Hallucination Rate: 지어낸 내용 비율
- 실패 케이스 기록 및 분류
- 개선 전/후 비교 리포트

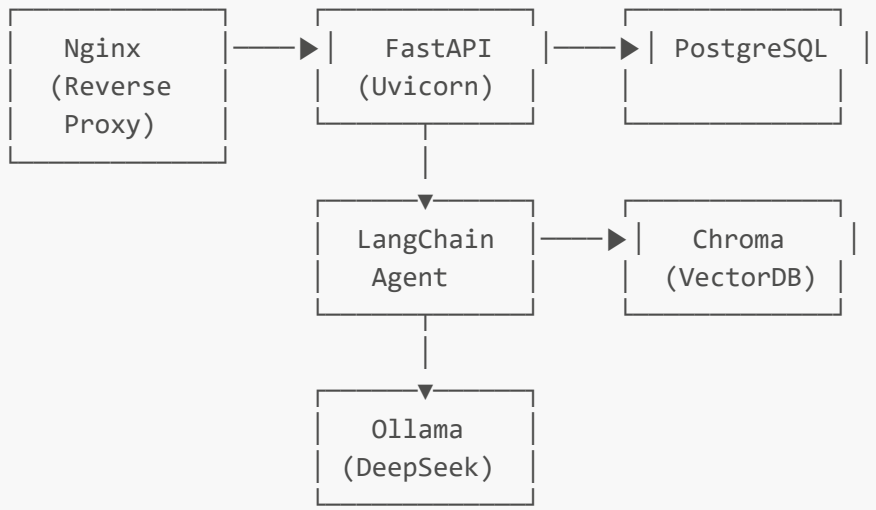
7.11 튜닝 우선순위 가이드

```
1순위: Chunk 설계 + 메타데이터 → 기본기
2순위: ReRanker 적용 → 검색 품질 대폭 향상
3순위: Hybrid Search → 고유명사/코드 검색 보완
4순위: 프롬프트 튜닝 → Hallucination 감소
5순위: 이미지 처리 → PDF 차트/표 대응
6순위: 고급 Retriever → 복잡한 문서 구조 대응
```

PART 8. 배포와 운영

> 목적: 개발 환경에서 실제 운영까지 가는 길 안내

8.1 배포 아키텍처



8.2 Docker Compose 구성

- 서비스별 컨테이너: FastAPI, PostgreSQL, Chroma
- Ollama는 호스트에서 실행 (GPU 접근)
- 볼륨 마운트 설정

8.3 환경 분리

- 개발(dev) / 스테이징(staging) / 운영(prod)
- 환경별 .env 파일 관리

- 시크릿 관리 (API 키, DB 비밀번호)

8.4 로깅 및 모니터링

- 요청/응답 로깅
- LLM 호출 로깅 (프롬프트, 토큰 수, 응답 시간)
- 에러 추적 (Sentry 등)
- 비용 모니터링 (토큰 사용량)

8.5 성능 최적화

- 응답 캐싱 (자주 묻는 질문)
- 임베딩 캐싱
- 동시 요청 처리 (Worker 수 조정)
- 벡터 DB 인덱스 최적화

8.6 보안 체크리스트

- ☐ API 인증/인가 (JWT, API Key)
- ☐ HTTPS 적용
- ☐ SQL Injection 방지
- ☐ Prompt Injection 방지
- ☐ 민감 정보 마스킹
- ☐ Rate Limiting

8.7 문서 업데이트 파이프라인

- 새 문서 추가 시 자동 인덱싱
- 문서 버전 관리
- 구버전 문서 처리 정책

8.8 트러블슈팅 가이드

증상	원인	해결
Ollama 응답 느림	메모리 부족	모델 크기 조정, GPU 확인
검색 결과 없음	인덱싱 안됨	문서 인덱싱 상태 확인
DB 연결 실패	설정 오류	.env, 네트워크 확인
토큰 초과 에러	컨텍스트 너무 김	청크 크기/k값 조정

부록

A. 예제 문서 세트

- 휴가 규정 (샘플)
- 신입사원 온보딩 가이드 (샘플)
- 보안 정책 (샘플)

B. 테스트 질문 30선

- 정형 데이터 질문 10개
- 비정형 문서 질문 10개
- 복합 질문 10개

C. 코드 전체 구조

```
project/
├── app/
│   ├── main.py
│   ├── models/
│   ├── routers/
│   ├── services/
│   │   ├── rag_service.py
│   │   ├── mcp_service.py
│   │   └── agent_service.py
│   └── templates/
├── docs/               # 사내 문서
├── scripts/
│   ├── index_docs.py   # 문서 인덱싱
│   └── seed_data.py    # 초기 데이터
├── tests/
├── docker-compose.yml
├── requirements.txt
└── .env.example
```

D. 참고 자료

- LangChain 공식 문서
- Chroma 공식 문서
- Ollama 공식 문서
- RAG 논문/아티클 추천