2차 프로젝트 가이드 v4

2차 팀 프로젝트 가이드

LLM 기반 서비스 구축과 운영

문제 정의 · LLM 제어 · 서비스 준비도 · 관측과 평가

FastAPI · Docker · Skill + MCP Server 필수

Langfuse — 관측 · 프롬프트 운영 · 평가(LLM-as-judge)

파인튜닝(선택) · 멀티모달 · 모델 라우팅 가산

v4 · 2026년 9월 17일

이 프로젝트의 정체성

이 프로젝트의 본질은 “LLM을 제품으로 만들어 운영하는 것”입니다. 1차가 딥러닝 모델을 직접 손으로 다루는 경험이었다면, 2차는 그 모델을 **다른 사람이 쓸 수 있는 서비스로 세우고, 그것이 잘 돌아가는지 숫자로 증명하는** 경험이에요.

세 프로젝트의 자리는 이렇게 나뉩니다.

차수	핵심 질문	다루는 것
1차	모델을 이해했는가	딥러닝 모델 직접 사용·튜닝 (TTS·STT)
2차	**서비스로 세울 수 있는가**	**LLM API 제어 · 서빙 · 관측 · 평가**
3차	자율 시스템을 설계할 수 있는가	**2차 서비스 위에** RAG 그래프 · LangGraph · 멀티에이전트

1차 가이드에서 LLM API 호출을 금지했던 이유가 여기 있습니다. 그때 아껴둔 것을 지금 씁니다.

📌 이 프로젝트는 3차에서 계속 씁니다

3차(RAG & LangGraph) 캡스톤은 **"2차에서 만든 서비스에 RAG 그래프를 얹는다"**로 설계돼 있습니다. 즉 지금 만드는 FastAPI 백엔드는 2주 쓰고 버리는 과제가 아니라, **3차 프로젝트가 올라앉을 토대**예요.

실질적인 의미가 두 가지 있습니다.

1. **대충 짜면 3차에서 본인이 고생합니다.** Docker가 안 뜨는 프로젝트를 3차에 그대로 들고 가게 돼요.
2. **LLM 호출부를 갈아끼울 수 있게 두세요.** 3차에서 그 자리에 RAG 그래프가 들어옵니다. 호출을 서비스 계층 한 곳에 모아두면 교체가 쉽고, 컨트롤러·UI에 흩뿌리면 그때 전부 뜯어야 합니다.

이건 좋은 설계 습관이기도 합니다. 외부 의존을 한 겹으로 감싸 두는 것은 실무에서도 기본이에요.

이 자료를 관통하는 한 줄 원칙

"돌아간다"는 주장이 아니라 증거다. 증거가 없으면 돌아가지 않는 것이다.

LLM은 같은 입력에 다른 출력을 냅니다. "잘 되던데요"는 이 세계에서 근거가 아니에요. 몇 건 중 몇 건이 계약을 지켰는지, 실패하면 무엇으로 떨어지는지, 한 번 처리에 얼마가 드는지를 남이 확인할 수 있는 형태로 내놓는 것이 이 프로젝트의 목표입니다.

6장 "서비스 엔지니어링 5원칙"을 먼저 읽고 시작하세요.

평가의 4가지 축

축	평가 내용	배점
① 문제 해결력	디자인 씽킹 기반의 사용자 중심 페인 포인트 정의와 솔루션 설계의 완성도	25
② LLM 제어력	출력 계약·검증·재시도 설계, 모델 선택 근거, 실패 경로 처리의 깊이	25
③ 서비스 완성도	FastAPI·Docker 기반 배포, 사용성, 장애 대응, 재현 가능성	25
④ 관측과 평가	Langfuse 세 축(관측·프롬프트·평가) 운영, LLM-as-judge 설계, 비용과 품질을 함께 놓고 내린 판단	25

1차의 4축(문제 해결력 · 모델 성능 · 사용성 · 서비스 준비도)에서 ①은 그대로 계승하고, ②③④를 2차 성격으로 갈아끼웠습니다. 사용성은 ③ 안에 포함됩니다.

---

1. 프로젝트 일정

구분	일시
착수	9월 18일 (금) 오후 세션
구현 마감 (코드·배포·리포트)	10월 8일 (목) 자정
발표 자료 마감	10월 11일 (일) 자정
발표	10월 12일 (월)

> 마감이 둘로 나뉘어 있습니다. **코드는 목요일에 멈추고, 그 뒤 사흘은 발표 준비에만 씁니다.** 10/8 자정 이후의 커밋은 채점에 반영하지 않아요. 만든 것을 어떻게 보여줄지에 집중하는 기간입니다.

1-1. 달력을 믿지 마세요

착수부터 마감까지 달력으로는 21일이지만, **실제 작업 가능한 평일은 12일**입니다. 이 구간에 한국 공휴일이 세 번 들어갑니다.

날짜	요일	구분
9/24~9/25	목·금	추석 연휴
10/5	월	개천절(10/3 토) 대체공휴일
10/9	금	한글날

주말까지 합치면 **9일이 통째로 빠집니다.** "3주니까 여유 있다"고 읽는 순간 일정이 무너져요. 실질은 2주 조금 넘는 분량입니다.

1-2. 권장 진행 흐름

구간	작업일	이 구간에 끝내야 할 것
9/14~9/17 (월~목)	사전 과제	**(사전기획)** 개인별 문제 후보 3개 · 사용자/대안 메모 · 문제 정의 초안 · 평가셋 후보 10건
9/18 (금) 오후	0.5일	팀 편성, 사전기획 비교, 주제·역할·저장소 작업 규칙 확정
9/21~9/23 (월~수)	3일	**문제 정의·범위 확정 · API/출력/관측 계약 · 평가셋 30건과 score rubric 확정**
9/24~9/27	—	추석 연휴
9/28~10/2 (월~금)	5일	핵심 구현 (FastAPI · LLM 파이프라인 · 출력 계약 · Docker · Langfuse trace/observation)
10/5 (월)	—	개천절 대체공휴일
10/6~10/8 (화~목)	3일	실패 observation 기반 prompt v1 등록·재조정 → Evals score 비교 → 동일 평가셋 재측정·리포트 → **구현 마감**
10/9~10/11 (금~일)	—	한글날 + 주말. **코드 동결, 발표 자료 작성** → 일요일 자정 마감
10/12 (월)	—	**발표**

⚠️ 추석 전에 반드시 끝내야 할 것

착수하고 사흘 만에 나흘을 쉽니다. 프로젝트가 가장 잘 무너지는 지점이 여기예요.

9/23(수)까지 문제 정의와 평가셋 초안을 확정하세요. 이 둘이 없으면 연휴 동안 아무것도 진행되지 않고, 연휴가 끝난 뒤 처음부터 다시 이야기하게 됩니다. 반대로 이 둘만 정해두면 연휴 중 각자 자료 조사·환경 셋업·프롬프트 실험을 개인 속도로 진행할 수 있습니다.

평가셋을 초반에 만들라는 원칙 3이 이 일정에서 특히 중요한 이유입니다.

> ⚠️ **10/9~10/11은 코드를 고치는 기간이 아닙니다**
>
> 구현 마감(10/8 목) 뒤로 한글날과 주말이 이어지고 월요일이 발표입니다. 이 사흘은 **발표 자료를 만드는 시간**이지 기능을 더 붙이는 시간이 아니에요.
>
> **코드 동결**: 10/8 자정 이후의 커밋은 채점에 반영하지 않습니다. 마감 시점의 레포와 배포 URL로 평가해요. 배포 URL은 발표일까지 살려 두세요.
>
> 이렇게 나눈 이유가 있습니다. 마감 직전까지 기능을 붙이다 발표를 급조하면, 잘 만들고도 설명을 못 합니다. 이 프로젝트는 **측정하고 설명하는 것**이 절반이라(축 ②·④), 그 절반을 준비할 시간을 따로 뒀어요.

1-3. 중간 점검

- **일시: 9월 30일 (수)** (연휴 복귀 후 3일차, 전체 일정의 중간)
- 확인 내용: 문제 정의 · 평가셋 · 아키텍처 · **무엇을 빼기로 했는가**

> 중간 점검에서는 코드를 보지 않습니다. "누가 무엇 때문에 막혀 있고, 우리는 무엇을 만들며, 무엇을 빼기로 했는가"만 확인합니다. 이 시점에 기능 목록이 늘어나 있는 팀은 마감을 못 지킵니다.

---

2. 주제 및 요구 사항

(1) 핵심 기능 조건 (필수)

아래 6가지는 충족하지 못하면 해당 축에서 점수를 받지 못합니다.

1. **동작 가능한 소프트웨어가 배포된 URL이 존재할 것** 로컬에서만 도는 것은 서비스가 아닙니다.
2. **FastAPI 백엔드 + Docker로 패키징할 것** `docker compose up` 한 줄로 전체 서비스가 떠야 합니다. 채점자가 여러분의 노트북을 빌릴 수는 없어요.
3. **LLM API를 핵심 로직에 사용하고, 출력 계약을 설계할 것** 프롬프트를 던져서 받은 문자열을 그대로 쓰는 것은 인정되지 않습니다. 스키마 정의(Pydantic), 검증, 재시도까지가 한 세트입니다.
4. **Langfuse로 관측·프롬프트·평가 세 축을 운영할 것** ① 트레이싱(입출력·모델·토큰·지연·비용 + `user_id` / `session_id`) ② 프롬프트를 코드에서 떼어내 버전 관리 ③ 평가셋 30건 이상을 Dataset으로 두고, **무언가를 바꾼 뒤 다시 재서 회귀를 숫자로** 보일 것 **한 번만 재고 끝내면 미충족입니다.** 실습 교안 B·C 파트에서 한 것을 여러분 서비스로 하는 것입니다.

5. **Domain LLM으로 구현하여 Skill + MCP Server를 만들 것** 일반적인 지식을 가진 에이전트를 만드는 게 아닌, 도메인 지식과 노하우를 담은 에이전트로 만들어야 합니다. 이를 위해서 Skill을 구성하고 MCP 서버로 연관된 API를 조합하여 만드세요.

6. **소개 페이지를 별도 구성하고, 팀 레포지토리를 깔끔하게 정리할 것** `.env.example` 포함, 민감값 커밋 금지.

> ## 5번이 이 프로젝트의 성격을 정합니다
>
> "LLM을 붙였다"와 "우리 도메인을 아는 에이전트를 만들었다"는 다릅니다. 범용 모델에 질문을 그대로 넘기면 누구나 만들 수 있는 것이 나와요. 이 프로젝트가 요구하는 것은 **여러분 팀이 고른 도메인의 지식과 판단 기준이 시스템 안에 들어가 있는 상태**입니다.
>
> 그 지식을 넣는 통로가 두 개입니다.
>
> | 통로 | 무엇을 담나 | 형태 |
> | --- | --- | --- |
> | **Skill** | 도메인 규칙·판단 기준·용어·예외 처리 노하우 | 문서(예: `SKILL.md`)와 프롬프트 자산 |
> | **MCP Server** | 그 도메인에서 실제로 필요한 데이터·동작 | `FastMCP`로 감싼 도구(API 조합) |
>
> Skill은 "이 도메인에서는 이렇게 판단한다"를 적어 모델에 주는 것이고, MCP는 "그 판단에 필요한 것을 가져오거나 실행하는" 손입니다. 참고 자료는 3주차 특강 `SKILL.md 스킬 만들기`와 2주차 17·20강입니다.

도구와 외부 데이터는 신뢰 경계 밖입니다 (v4 신설)

5번에서 MCP 서버를 만들라고 했는데, 도구를 만든다는 것은 **여러분 서비스에 외부에서 값이 들어오는 통로를 낸다**는 뜻입니다. 그 통로로 들어온 문장이 지시로 읽히면 계약도 검증도 소용이 없어요.

3주차 11강에서 직접 해본 그것입니다. 사용자가 공격한 적이 없는데도, 데이터 안에 숨어 들어온 문장 때문에 뚫렸습니다.

아래 셋을 요구합니다.

시점	할 것	참고
설계	무엇을 믿지 않을지 한 장으로 정한다 — 사용자 입력, 외부에서 가져온 데이터, 도구가 돌려준 값	3주차 11강
구현	외부에서 온 내용과 시스템 지시를 프롬프트에서 분리한다(출처 태깅). 부작용이 있는 도구는 명시적 요청과 권한 없이 호출되지 않게 한다	11강 Step 4 · 12강
검증	평가셋에 인젝션 케이스를 3건 이상 넣고 방어 전후를 같은 기준으로 잰다	11강 Step 5·6

특히 **MCP 도구의 이름·설명·스키마·반환값도 신뢰하지 않습니다.** 도구 설명 안에 "이전 지시를 무시하라"거나 환경변수·비밀값·내부 경로를 요구하는 문장이 들어 있을 수 있어요. 그것을 다른 도구 호출에 옮겨 담지 마세요.

11강이 알려준 것을 그대로 옮깁니다. **"통과"는 안전하다는 뜻이 아니라, 우리가 아는 방식으로는 안 뚫렸다는 뜻입니다.** 그러니 막았다고 쓰지 말고, 무엇을 시도했고 무엇이 막혔고 무엇이 남았는지 쓰세요.

왜 4번이 다른 조건들과 다른가

나머지 조건은 "만들었는가"를 묻습니다. 4번만 "얼마나 잘 만들었는지 스스로 아는가"를 묻습니다.

시장에서 신입과 경력을 가르는 지점이 정확히 여기예요. 만들 줄 아는 사람은 많지만, 자기가 만든 것의 품질을 숫자로 말할 수 있는 사람은 드뭅니다. 발표에서 "정확도가 올랐습니다"라고 하면 반드시 "무엇으로 쟀고, 몇 건이었고, 전에는 얼마였나"를 되묻습니다.

(2) 공통 배포 구조와 endpoint 제출 포맷 (필수)

이 프로젝트의 결과물은 "Vercel에 챗봇 하나를 올린 것"이 아닙니다. **내가 3개월 이상 경험한 버티컬 도메인의 반복 업무를, UI·Agent API·MCP 도구·평가 루프로 운영 가능한 Agent로 만든 것**입니다.

```
사용자 → Vercel / Next.js (서비스 UI)
              → Cloud Run / FastAPI (POST /api/agent, GET /health)
                    ├─ /mcp : Streamable HTTP MCP endpoint
                    └─ Langfuse · LLM · 도메인 API
```

층	공통 역할	제출할 증거
Vercel / Next.js	사용자가 실제로 쓰는 서비스 화면, 입력·결과·실패 안내	Vercel URL, 정상·실패 사용자 흐름 영상
Cloud Run / FastAPI	Agent 실행, Pydantic 출력 계약, 검증·재시도	Cloud Run URL, `GET /health`, `POST /api/agent` 호출 예시
MCP Server	도메인 데이터·행동을 도구로 제공	`/mcp` 연결 방법, tool 목록과 실제 호출 결과
Docker	로컬 재현과 배포 가능한 패키징	`Dockerfile`, `docker compose up` 실행 방법
Langfuse + Evals	관측 → 프롬프트 버전 변경 → 동일 평가셋 재측정	trace, prompt v1/v2, `EVAL_REPORT.md` 전후 score

endpoint를 섞지 마세요. 사용자 화면이 호출하는 Agent API는 `/api/agent`, MCP 클라이언트가 연결하는 프로토콜 endpoint는 `/mcp`로 구분합니다. MCP endpoint를 공개 데모 API처럼 무제한 노출하지 말고, 토큰·허용 클라이언트 등 접근 제어 방식을 README에 적으세요. 브라우저가 Cloud Run을 직접 호출할 경우에는 CORS·인증·비밀값 노출을 설명해야 합니다.

(3) 유형 선택

아래 두 유형 중 하나를 골라 설계합니다. 두 유형 모두 LLM을 서비스로 세우는 것이 핵심입니다.

[유형 1] 도메인 특화 LLM 서비스

특정 도메인의 문서·데이터·업무 흐름에 LLM을 붙여 실무자의 반복 작업을 줄이는 서비스입니다.

예시: - 계약서·약관에서 위험 조항을 뽑아 근거와 함께 제시하는 검토 보조 - 고객 문의를 분류하고 초안 답변을 만들어 상담사에게 넘기는 CS 보조 - 제조 설비 점검 기록을 구조화해 이상 징후를 요약하는 리포트 생성기 - 진료 기록·상담 로그에서 필요한 항목만 표준 양식으로 추출하는 도구

[유형 2] 멀티모달·복합 입력 LLM 서비스

텍스트만이 아니라 이미지·음성·표 등 다른 형태의 입력을 함께 다루는 서비스입니다. 1차에서 만든 TTS·STT 자산을 재활용해도 좋습니다.

예시: - 사진을 올리면 상태를 판정하고 조치를 안내하는 진단 보조 - 회의 녹음을 받아 발언자별 액션 아이템을 구조화하는 도구 - 영수증·명세서 이미지에서 항목을 추출해 장부로 정리하는 서비스 - 화면 캡처를 읽고 오류 원인과 해결 절차를 제시하는 기술지원 보조

---

3. 프로젝트 진행 절차

본 프로젝트도 1차와 같이 **디자인 씽킹 기반 문제 해결 절차**를 따릅니다.

단계	수행 내용
① 주제 선정	팀별 목표 설정, 해결하고 싶은 문제 선정
② 사용자 중심적 사고	사용자 입장의 불편함에 공감하고 가설 수립
③ 문제 정의	가설 검증, 누구에게 무엇이 왜 문제인지 정의
④ 고객 여정 맵	경험 흐름과 단계별 pain point 시각화
⑤ 솔루션 제안	LLM 기반 서비스로 문제를 어떻게 푸는지 제안

절차의 세부 수행 방법(사용자 분석, 문제 정의 문장, 고객 여정 맵 양식)은 1차 프로젝트 가이드 4장과 동일합니다. 1차 가이드를 다시 펴서 참고하세요.

> **2차에서 특히 조심할 것**
>
> 1차보다 기술 스택이 화려해서, 문제 정의를 건너뛰고 아키텍처부터 그리는 팀이 나옵니다.
>
> ✗ 안 되는 사고: “FastAPI랑 Langfuse 써야 하니까 뭘 만들지?” ✓ 맞는 사고: “이 사용자가 이 작업에 하루 2시간을 쓴다. LLM이 그걸 20분으로 줄일 수 있나?”
>
> 배점 25점이 여전히 문제 해결력에 있습니다. 기술 조건은 통과 요건이지 점수가 아니에요.

---

4. 기술 활용 가이드

4-1. 필수 스택

영역	도구	왜 필수인가
백엔드	FastAPI	채용 공고에서 가장 자주 요구되는 파이썬 서빙 프레임워크. 비동기·자동 문서화·Pydantic 통합이 LLM 서비스와 잘 맞습니다
컨테이너	Docker / docker compose	“제 컴퓨터에서는 되는데요”를 없애는 최소 장치. 재현 가능성이 곧 협업 가능성입니다
관측·평가	Langfuse	트레이스·비용·지연을 한곳에서 보고, 평가셋으로 회귀를 잡습니다. 셀프호스팅 가능(강의 자료의 docker-compose 재사용)
LLM	**OpenRouter 권장**	강의 특강이 이미 OpenRouter 기반입니다. 키 하나로 여러 회사 모델을 같은 코드로 부를 수 있어, 축②가 요구하는 “2개 이상 모델 비교”가 훨씬 쉬워집니다. OpenAI·Anthropic·Gemini 직접 호출도 자유

4-1-1. 강의에서 그대로 가져다 쓸 것

새로 배울 것은 거의 없습니다. 프로젝트 요구사항 대부분이 이미 실습한 내용이에요.

프로젝트 요구사항	그대로 쓸 강의 자료
출력 계약 (필수 3)	1주차 12·13강, **특강 LLM API 기초1** (구조화 출력·ToolCall)
재시도 전략 (필수 3)	1주차 14·15강 (3단계 재시도, 100건 실측)
모델 비교 (축 ②)	1주차 8·9강 (closed vs open 실측, 모델 선택 3축)
모델 라우팅 (가산)	1주차 10강
비용·SLA (원칙 4)	1주차 17강, **특강 Prompt Caching과 비용 최적화**
체감 지연 개선 (축 ③)	**특강 스트리밍 응답 처리**
멀티모달 (유형 2)	**특강 LLM API 기초2** (이미지 입력)
검색·임베딩 (유형 1)	**특강 LLM API 기초3** (임베딩·웹검색)
도구 설계 (tool use)	2주차 6·7·8강 (도구 설계, description 작성법, 에러·타임아웃)
MCP 서버 (**필수 5**)	2주차 17강, 20강 (resource·prompt)
평가셋 설계·실패 진단 (필수 4)	3주차 15강 (평가셋 만들기, 배치 실행, KEEP/DISCARD 판단)
평가 지표·출시 게이트 (축 ④)	3주차 14강, **특강 Agent Evaluation Gate** (8지표, 점수와 게이트의 구분)
프롬프트 주입 방어	3주차 11·12강
근거 제시·할루시네이션	3주차 13강, **특강 근거카드**

> 노트북의 실습 코드를 프로젝트로 복사해도 됩니다. 그건 표절이 아니라 재사용이에요. 다만 그 코드가 무엇을 하는지는 설명할 수 있어야 합니다(5장 AI 사용 정책과 같은 기준).

4-1-2. 필수 스택 정리

- **FastAPI · Docker · Langfuse**
- **LLM 사용** (OpenRouter 또는 로컬 SLM) + **자체 Skills 탑재** + **MCP Server 구현**
- **KDT 7기 GitHub org**(강사 전달) + **GCP 배포**

새로 배울 것은 없습니다. 모두 실습을 이미 했어요.

4-2. 선택 스택 (자유)

영역	도구
프론트엔드	Next.js, Streamlit, Gradio, Flutter
데이터베이스	**Postgres 권장** (Docker 컨테이너 / Supabase / RDS)
벡터 검색	pgvector, FAISS, Chroma — **분류·라우팅·추천·중복 탐지까지 허용**. 경계는 4-3-1 표 2 참조
배포	Railway, Fly.io, Render, Vercel, Cloud Run, AWS
API 테스트	Bruno (FastAPI 워크숍 참조)
캐시·큐	Redis, Celery, FastAPI BackgroundTasks

4-3. 사용 금지·제한 사항

> **✗ 사용 금지 — 두 가지입니다**
>
> **× LangGraph × RAG**
>
> 둘 다 3차 프로젝트 준비의 몸통입니다. `1. RAG 파이프라인`과 `2. LangGraph 런타임`이 통째로 여기에 걸려 있어요. Part IV의 설계는 **"Part III가 손으로 고생시킨 것에 `checkpointer`·`interrupt()`로 답한다"**이고, 청킹·하이브리드 검색·리랭킹·RAG 평가도 그 주차의 약속입니다. 2차에서 미리 덮으면 3차에서 "왜 이게 필요한가"를 느낄 기회가 사라집니다.
>
> **LangChain은 허용합니다.** MCP 클라이언트를 붙이거나 모델 래퍼로 쓰는 정도는 막지 않아요. 막는 것은 상태 그래프(LangGraph)와 검색증강 생성(RAG), 두 가지입니다.

패키지별로 정확히 적습니다.

구분	패키지	2차 프로젝트
모델 호출 SDK	`openai`, `anthropic`, `google-genai`	✅
벤더 에이전트 SDK	`openai-agents`, `claude-agent-sdk`, `google-adk`	✅ **조건부**
MCP 서버	`mcp` (FastMCP)	✅ **필수**
LangChain·MCP 어댑터	`langchain`, `langchain-mcp-adapters`	✅
상태 그래프	`langgraph`	❌

> **어디까지 되고 어디부터 안 되나**
>
> "벡터 검색을 쓰면 RAG인가요?"를 많이 묻습니다. 아닙니다. 둘은 층위가 달라요. **벡터 검색은 기법이고, RAG는 그 기법을 쓰는 구조 하나**입니다. 아래 4-3-1의 세 표로 판정하세요.

4-3-1. RAG 경계선 — 세 개의 표로 판정합니다

표 1. 이게 RAG인가? — 세 질문에 모두 "예"면 RAG입니다

#	질문	RAG라면
1	지식이 **잘게 쪼갠 문서 조각(청크) 더미**로 저장돼 있나?	예
2	**질문이 바뀌면 골라오는 조각도 바뀌나?**	예
3	그 조각을 **프롬프트 컨텍스트로 붙여** 답을 생성하나?	예

하나라도 "아니오"면 RAG가 아닙니다. 예를 들어,

- **Skill 문서**를 프롬프트에 넣는 것은 1·2번이 아니오입니다. 통짜 문서이고, 질문이 바뀌어도 같은 내용이 들어가니까요.
- **MCP 도구**가 API로 가져온 값을 LLM이 받는 것은 1번이 아니오입니다. 문서 조각이 아니라 구조화된 데이터를 조회한 것이에요.

이 둘은 오히려 **필수 조건 5번**입니다. 마음껏 쓰세요.

표 2. 사례별 판정

하려는 것	판정	이유
Skill 문서(도메인 규칙)를 시스템 프롬프트에 항상 넣기	✅	고정 지식, 청크 검색 아님 (필수 5)
MCP 도구로 DB·API를 조회해 그 결과를 LLM이 요약	✅	구조화 조회, 문서 검색 아님 (필수 5)
임베딩으로 문의를 분류해 담당 부서로 라우팅	✅	프롬프트에 안 들어감
임베딩으로 유사 문서를 찾아 **사용자에게 목록으로** 표시	✅	생성에 개입 안 함
임베딩으로 중복 접수·유사 사례 탐지	✅	판정 용도
벡터 유사도로 상품·콘텐츠 추천 목록 만들기	✅	생성에 개입 안 함
카테고리가 5개라 **카테고리별 Skill 문서**를 갈아 끼우기	✅	조각이 아니라 통짜 문서, 소수 분기
사용자가 올린 파일 1건을 통째로 프롬프트에 넣어 요약	✅	코퍼스 검색이 아님
FAQ를 임베딩해 가장 가까운 항목의 **정해진 답변을 그대로** 반환	✅	LLM 생성 없음
FAQ를 임베딩해 찾은 항목을 **프롬프트에 넣어** 답을 다듬기	❌	세 질문 모두 예 → RAG
PDF·규정집을 청킹해 벡터DB에 넣고 질문마다 top-k 검색 후 생성	❌	전형적인 RAG
웹 검색 결과 본문을 긁어 프롬프트에 붙여 답변 생성	❌	검색 대상만 다른 RAG
대화 이력을 청킹·검색해 관련 부분만 컨텍스트에 복원	❌	메모리 검색도 RAG 계열 (3차 몫)
하이브리드 검색(BM25+벡터)·리랭킹·청킹 전략 튜닝	❌	Part IV 1주차 주제 그대로

표 3. RAG로 하려던 것을 2차에서 하는 방법

금지만 보면 막막하니, 바꿔 만드는 길을 함께 적습니다.

하고 싶은 것	RAG 방식 (3차)	2차에서 하는 방법
사내 규정·매뉴얼 기반 답변	문서 청킹 → 검색 → 주입	핵심 규칙을 **Skill 문서로 정리**해 항상 주입. 분량이 크면 카테고리별로 나눠 분기
최신·실시간 데이터 반영	문서 인덱싱 후 검색	**MCP 도구로 API 조회.** 애초에 검색보다 정확합니다
많은 FAQ 대응	FAQ 임베딩 → top-k → 생성	임베딩으로 **분류**한 뒤 카테고리별 템플릿·Skill 적용
사용자가 올린 문서 처리	업로드 문서 청킹 → 검색	**파일 1건을 통째로** 넣어 처리. 길면 요약을 단계로 나눔
근거 문장 제시	검색된 청크를 근거로 인용	MCP 도구가 **원본 위치를 함께 반환**하게 설계

> **왜 이렇게까지 가르나요?** 지식을 넣는 방법이 곧 시스템 설계이기 때문입니다. 2차는 "지식을 **적어 넣고**(Skill) **가져오는**(MCP)" 방식을, 3차는 "**찾아서 넣는**"(RAG) 방식을 다룹니다. 두 방식은 비용·지연·정확도·유지보수가 전부 다르고, 실무에서는 둘을 섞어 씁니다. 그 차이를 체감하려면 한 번은 따로 만들어 봐야 해요.

> **판단이 서지 않으면** 물어보세요. 애매한 채로 2주를 가는 것이 가장 나쁩니다. 특강 `LLM API 기초3`에서 배운 임베딩 활용은 표 2의 ✅ 범위에서 그대로 쓰시면 됩니다.

> ### 벤더 SDK를 쓸 때의 조건
>
> 3주차 16~20강에서 배운 다섯 SDK(ADK·OpenAI Agents·Claude Agent·Vertex·Azure)는 써도 됩니다. **Part IV에서는 이걸 다루지 않으므로, 2차가 여러분이 그걸 실제로 써 볼 유일한 기회예요.**
>
> 다만 축 ②(LLM 제어력)는 "직접 짰는가"가 아니라 **"알고 골랐는가"**로 봅니다. SDK를 썼다면 리포트와 발표에서 아래에 답할 수 있어야 합니다.
>
> - 이 SDK가 **재시도·구조화 출력·에러 처리를 어떻게 하는가** (기본값이 무엇인가)
> - 그 기본값이 **우리 서비스 요구에 맞는가**, 안 맞으면 어디를 바꿨는가
> - SDK가 삼킨 실패를 **트레이스에서 볼 수 있는가** (Langfuse에 남는가)
>
> 답하지 못하면 그 부분은 제어력으로 인정되지 않습니다. 반대로 직접 구현했다면 "왜 SDK를 쓰지 않았는지"를 같은 깊이로 설명하면 됩니다. **둘 다 정답이고, 모르고 쓰는 것만 오답입니다.**
>
> ### ○ MCP 서버는 필수입니다 (필수 조건 5번)
>
> 17강에서 배운 그대로 `mcp.server.fastmcp.FastMCP` 로 만들면 됩니다. 붙여서 확인하는 방법은 자유예요 — 18강처럼 `langchain-mcp-adapters` 로 연결해도 되고, Claude Desktop이나 Cursor에 물려 자연어로 불러도 됩니다. **강의 실습을 그대로 가져다 쓰세요.**
>
> 중요한 것은 도구를 무엇으로 노출했는가입니다. 도메인에 필요한 동작이 tool로 잘 갈라져 있고 `tool_description` 만 읽고도 언제 쓰는 도구인지 알 수 있으면 좋은 설계예요(2주차 6·7강).

왜 LangGraph와 RAG만 막을까요?

이 둘은 3차에서 **주제 자체로** 다룹니다. 미리 써버리면 3차가 복습이 되고, 무엇보다 "왜 필요한가"를 느낄 기회를 잃어요. Part IV는 여러분이 2차에서 손으로 겪은 불편에 답하도록 설계돼 있습니다.

반대로 LangChain·벤더 SDK를 막지 않는 이유도 같습니다. 3차에서 다시 다루지 않으니, 지금 쓰지 않으면 배운 것을 써 볼 자리가 없습니다.

개발 도구로서의 LLM은 자유: Cursor, Claude Code, Copilot으로 코드를 쓰고 디버깅하는 것은 1차와 동일하게 전면 허용입니다. 산출물 코드 안에 금지 프레임워크가 들어가지 않으면 됩니다.

---

5. AI 사용 정책

이 과정에서 AI 사용은 금지가 아니라 권장입니다. 다만 **어떻게 썼는지가 평가됩니다.**

구분	내용
✅ 허용·권장	AI 에디터 전면 허용, 전체 파일 스캐폴딩 요청, 에러 해결 질의, 리팩터링·네이밍·주석 위임, 코드 리뷰 받기
⚠️ 조건부	아키텍처 결정(예: 어떤 모델을 쓸지)은 AI 제안을 받되, **이유를 본인 언어로** 리포트에 정리
❌ 금지	이해 없이 복붙, 타 팀 코드를 AI에 넣어 자기 것으로 포장

> **발표·평가 시 확인 방법**: 교수자가 여러분 코드의 임의 지점을 짚고 "이거 왜 이렇게 했어요?"라고 묻습니다. 답하지 못하면 그 부분은 점수에 반영되지 않습니다. AI가 쓴 코드여도 상관없습니다. 설명할 수 있으면 됩니다.

---

6. 서비스 엔지니어링 5원칙

1차의 "제품 엔지니어링 5원칙"이 물리적 제약(FPS·지연·발열)을 다뤘다면, 2차의 5원칙은 **확률적 시스템을 신뢰할 수 있게 만드는 법**을 다룹니다. 7·8·9장은 이 원칙들의 실천편입니다.

> **"LLM은 실패한다. 실패를 설계에 넣은 사람만 서비스를 만든다."**

원칙 1 — 출력을 계약으로 고정하라

LLM의 출력은 자연어가 아니라 데이터여야 합니다. 스키마를 먼저 정하고, 그 스키마를 만족하지 못한 응답은 실패로 처리하세요.

- Pydantic 모델로 응답 형태를 정의한다
- 검증 실패 시 무엇을 할지 미리 정한다 (재시도 / 폴백 / 사용자 안내)
- "대부분 잘 나와요"는 계약이 아니다

Part III 1번 커리큘럼 12·13·14강(출력 계약 설계 · 검증 파이프라인 · 3단계 재시도)이 그대로 여기 쓰입니다.

원칙 2 — 실패 경로를 먼저 그려라

정상 경로는 저절로 만들어집니다. 서비스의 품질은 실패했을 때 결정돼요.

실패 유형	준비해야 할 것
API 타임아웃·5xx	재시도 횟수와 백오프, 최종 실패 시 사용자 메시지
스키마 위반	재요청 프롬프트, 그래도 실패하면 폴백 값
레이트 리밋	큐잉 또는 다른 모델로 라우팅
비용 초과	요청당 상한, 초과 시 차단 정책
유해·부적절 입력	입력 검증, 거부 응답 설계

> 발표에서 "장애가 나면 어떻게 되나요?"를 반드시 묻습니다. "안 나게 만들었습니다"는 답이 아닙니다.

원칙 3 — 측정하지 않은 것은 개선할 수 없다

"프롬프트를 고쳤더니 좋아졌다"는 느낌입니다. 같은 평가셋으로 전후를 재야 사실이 됩니다.

- 평가셋은 프로젝트 초반에 만든다 (나중에 만들면 이미 튜닝된 것에 맞춰진다)
- 최소 30건, 정답 또는 판정 기준을 함께 적는다
- 실패 사례를 반드시 포함한다 (잘 되는 것만 모으면 평가가 아니라 자랑이다)

원칙 4 — 비용과 지연은 기능이다

응답이 30초 걸리거나 한 번에 500원이 나가면, 그 기능은 없는 것과 같습니다.

측정해야 할 것: - 요청당 토큰 수와 비용 - p50 / p95 지연 시간 (평균만 보면 느린 소수가 숨습니다) - 동시 요청 시 큐잉 지연

줄이는 수단: 프롬프트 압축, 캐싱, 모델 라우팅(쉬운 요청은 작은 모델로), 스트리밍으로 체감 지연 낮추기. Part III 1번 9·10·17강과 특강(Prompt Caching)이 이 내용입니다.

원칙 5 — 기능을 줄이는 것이 제품을 완성하는 것이다

1차 5원칙에서 그대로 가져옵니다. 작업일 12일 안에 여섯 기능을 어설프게 만드는 것보다 한 기능을 끝까지 미는 편이 언제나 낫습니다.

> ✗ 주니어의 사고: "이 기능도 넣으면 좋을 것 같은데?" ✓ 시니어의 사고: "이 기능을 빼면 더 단단해질까?"

5원칙 → 평가 4축 매핑

원칙	강하게 연결되는 축
1 — 출력 계약	② LLM 제어력
2 — 실패 경로	② LLM 제어력 / ③ 서비스 준비도
3 — 측정	④ 관측과 평가
4 — 비용·지연	④ 관측과 평가 / ③ 서비스 준비도
5 — 기능 줄이기	① 문제 해결력

---

7. LLM 제어력 평가 (축 ②)

7-1. 출력 계약

제출물에 다음이 드러나야 합니다.

- 응답 스키마 정의 (Pydantic 모델 코드)
- 검증 로직과 실패 시 동작
- 재시도 전략 (몇 번, 어떤 프롬프트로, 언제 포기하는지)
- 100건 이상 실행했을 때의 계약 준수율

벤더 SDK를 쓴 경우에도 위 네 가지는 그대로 답해야 합니다. 다만 답이 "우리가 짰다"가 아니라 "SDK가 이렇게 하고, 우리는 이 부분을 이렇게 바꿨다"가 됩니다.

	직접 구현한 팀	벤더 SDK를 쓴 팀
스키마	Pydantic 모델 코드	SDK의 구조화 출력 설정 + 그 한계
재시도	재시도 함수 코드	SDK 기본 재시도 정책 + 조정한 값과 이유
실패 처리	폴백 분기	SDK가 던지는 예외 종류와 우리 처리
준수율	100건 실측	동일

> 이 표의 어느 열이든 25점 만점이 가능합니다. 채점자가 보는 것은 **"이 시스템이 실패할 때 무슨 일이 일어나는지 이 팀이 아는가"** 하나예요. 직접 짜면 코드로 증명되고, SDK를 쓰면 설명으로 증명합니다.

7-2. 모델 선택 근거

최소 2개 이상의 모델을 같은 입력으로 비교하고, 그 결과로 최종 모델을 골라야 합니다.

비교 항목	예시
품질	평가셋 정확도 또는 판정 통과율
비용	요청당 토큰 비용
지연	p50 / p95
안정성	계약 준수율, 실패율

비교 대상은 상용 모델끼리여도 되고, 상용 vs 로컬 SLM이어도 됩니다. 중요한 것은 **같은 조건에서 쟀는가** 입니다.

7-3. 프롬프트 관리

- 프롬프트를 코드에 흩뿌리지 말고 한곳에서 관리한다
- 버전을 남긴다 (무엇을 왜 바꿨는지)
- **바꾸기 전에 되돌릴 길을 먼저 확인한다** — v2를 올렸는데 나빠졌을 때 배포 없이 v1으로 돌아갈 수 있어야 합니다. 되돌아갈 수 없으면 실험을 못 합니다 (v4 추가)
- 3주차 10강(Langfuse Prompt Release Lab — 평가·승격·Rollback) 참조. release manifest를 고정하고 같은 평가셋으로 승격을 판단하는 절차가 그대로 나옵니다

> v3까지 이 항목의 참조를 "Part III 1번 커리큘럼 10강"으로 적어 두었는데 잘못된 안내였습니다. 1주차 10강은 모델 라우팅이고, 프롬프트 버전관리는 **3주차 10강**입니다.

---

8. 서비스 준비도 평가 (축 ③)

8-1. 재현 가능성 (필수)

채점자가 여러분 레포를 클론해서 아래를 할 수 있어야 합니다.

```
git clone <repo>
cp .env.example .env    # 키만 채우고
docker compose up       # 한 줄
```

10분 안에 로컬에서 뜨지 않으면 감점합니다. README에 실행 절차를 적으세요.

8-2. API 설계

- FastAPI 자동 문서(`/docs`)가 살아 있을 것
- 요청·응답 모델이 Pydantic으로 정의돼 있을 것
- 헬스체크 엔드포인트 존재
- 구조화 로그: 요청 ID, 사용 모델, 토큰 수, 지연 시간을 로그에 남길 것

8-3. 사용성

항목	확인 내용
즉각 피드백	처리 중임을 사용자가 알 수 있는가 (로딩·진행률·스트리밍)
오류 안내	실패했을 때 사용자가 무엇을 해야 하는지 알 수 있는가
입력 가이드	무엇을 넣어야 하는지 화면에서 알 수 있는가

인터페이스는 자유입니다. Streamlit·Gradio도 인정하되, 실제 사용자가 써볼 만한 흐름이어야 합니다.

8-4. LLM 호출부는 한곳에 모으세요

3차에서 이 자리에 RAG 그래프가 들어옵니다. **지금 5분 들여 두면 그때 며칠을 법니다.**

```
✗ 흩어진 구조                          ✓ 모인 구조
routers/summary.py                    services/llm.py    ← LLM 호출은 여기만
  → openai.chat.completions             routers/*.py     ← 이 함수만 부른다
routers/extract.py
  → openai.chat.completions
```

라우터·UI 코드가 `openai`를 직접 import하지 않게만 해도 충분합니다. 3차에서 `services/llm.py` 내부만 갈아끼우면 나머지는 그대로 돌아가요.

> 이건 2차 점수를 위한 요구가 아니라 **여러분을 위한 조언**입니다. 축 ③ 채점 항목에 넣지 않았어요. 지금은 2차에 집중하고, 이 한 가지만 기억해 두세요.

---

9. 관측과 평가 (축 ④)

이 축이 2차 프로젝트를 1차와 가르는 지점이고, 취업 시장에서 가장 강한 신호이기도 합니다.

Langfuse 실습 교안이 정의한 **LLM Ops 세 축**을 그대로 가져옵니다. 프로젝트에서 이 셋을 다 써야 합니다.

축	실무에서 답해야 하는 질문	실습	프로젝트 요구
① **Observability**	이 응답이 왜 이렇게 나왔나? 어디서 느려졌나? 누가 얼마나 썼나?	B 전체 · C-1 · C-2	9-1
② **Prompt Management**	프롬프트 한 줄 바꾸려고 매번 배포해야 하나? 어느 버전이 서비스 중인가?	C-3	9-2
③ **Evaluation · LLM-as-judge**	바꿨는데 품질이 나빠지지 않았다고 어떻게 증명하나?	B-3 · C-4	9-3

> 교안의 경고를 그대로 옮깁니다. **①만 하고 끝내면 "trace는 보이는데 그래서 뭘 고칠지는 모르는" 상태가 됩니다.** 트레이스를 붙이는 데는 30분이면 충분해요. 나머지 둘이 이 축의 실체입니다.

9-1. Observability — 무슨 일이 있었는지 남긴다 (필수)

- 모든 LLM 호출이 트레이스로 남을 것
- 트레이스에 **입력·출력·모델·토큰·지연·비용**이 포함될 것
- 재시도가 일어난 경우 그 사실이 트레이스에서 보일 것
- **`user_id` · `session_id` 를 붙일 것** (B-5) — 요청 축이 아니라 사람 축으로 집계해야 "누가 얼마나 썼나"에 답할 수 있습니다
- 대시보드 스크린샷을 제출물에 포함

> **B-6에서 본 것을 기억하세요.** 응답 시간 숫자만으로는 "느리다"까지만 알고, 타임라인을 봐야 "왜 느린가"를 알 수 있습니다. 느림의 원인이 LLM인지 우리 코드인지 트레이스로 가릴 수 있어야 축 ③(서비스 완성도)의 장애 대응도 설명됩니다.

9-2. Prompt Management — 프롬프트를 코드에서 떼어낸다 (필수)

프롬프트를 소스에 문자열로 박아두면 한 줄 고치는 데 배포가 필요하고, 어느 버전이 서비스 중인지 아무도 모릅니다.

- **프롬프트를 Langfuse Prompt로 관리할 것** (C-3)
- 버전이 남고, **배포 없이 교체·롤백**이 되는 것을 시연으로 보일 것
- 트레이스에 **어느 프롬프트 버전이 쓰였는지** 연결될 것

> 이게 9-3의 전제이기도 합니다. 프롬프트 버전이 기록되지 않으면 "이 점수가 어느 프롬프트에서 나온 건지"를 나중에 되짚을 수 없어요.

9-3. Evaluation — 바꿔도 나빠지지 않았음을 증명한다 (필수)

이 프로젝트에서 가장 무거운 요구입니다. **"한 번 재봤다"는 평가가 아닙니다.**

평가셋

항목	최소 기준
건수	**30건 이상**
구성	정상 케이스 + 경계 케이스 + 실패 유도 케이스
기준	각 건마다 정답 또는 판정 기준, 그리고 **그 판정이 맞다고 본 이유**
관리	**Langfuse Dataset으로** (C-4). 로컬 파일에만 두지 않습니다

> 잘 되는 것만 모으면 평가가 아니라 자랑입니다. 실패 유도 케이스를 반드시 넣으세요.

만드는 시점이 중요합니다. 착수 후 **9/23(수)까지** 초안을 확정하세요. 나중에 만들면 이미 튜닝된 결과에 맞춰지고, 추석 연휴(9/24~27) 전에 이게 없으면 연휴가 통째로 빕니다.

점수는 축을 나눠 냅니다

하나의 종합 점수로 뭉개면 무엇이 무너졌는지 안 보입니다. 교안 C-4 실측이 이걸 정확히 보여줍니다.

모델	`tool_selection`	`grounded`	무엇이 무너졌나
A 모델	1.00	1.00	없음
B 모델	**1.00**	**0.43**	도구는 다 골랐는데 **답변이 계산 결과를 근거로 제시하지 않음**
C 모델	**0.67**	1.00	도구를 하나 빠뜨렸는데 **답변은 근거 있어 보임**

B 모델을 종합 점수 하나로만 봤다면 "그럭저럭"으로 보였을 겁니다. **도메인에 맞는 축을 2개 이상** 정해서 따로 재세요.

축을 뭐로 나눌지 모르겠다면 (v4 추가)

3주차 참고문서 `reference/langfuse-vertical-agent-prompt-config.md`에 버티컬 도메인 에이전트용 grader가 네 개로 정리돼 있습니다. 그대로 가져다 도메인만 바꿔 쓰세요.

Grader	무엇을 보나
Scope	우리 서비스가 답할 질문과 아닌 질문을 구분했는가. 기준이 없을 때 추측하지 않았는가
Tool trajectory	필요한 도구만 골랐는가. 같은 도구를 불필요하게 반복 호출하지 않았는가. 사실 조회와 판정을 섞지 않았는가
Grounding	도구가 준 결론·근거·출처·한계를 보존했는가. 사실과 추정을 구분했는가
Safety and authority	보장할 수 없는 것을 보장하지 않았는가. 권한 없이 쓰기 도구를 호출하지 않았는가. 도구 안에 숨은 지시를 따르지 않았는가

같은 문서에 프롬프트 본문(Scope · Decision policy · Evidence policy · Safety and authority · Failure handling · Response format)과 Config JSON, 권장 평가셋, Prompt Release 절차까지 들어 있습니다. 394줄짜리 템플릿이니 처음부터 쓰지 말고 이것을 출발점으로 삼으세요.

LLM-as-judge

- 판정 방식은 자유입니다 — 정확 일치 · 스키마 통과 · 규칙 기반 · LLM-as-judge
- LLM-as-judge를 쓴다면 **생성 모델과 판정 모델을 다르게** 두세요. 같으면 후한 점수가 나옵니다
- **판정 근거를 따라갈 수 있어야 합니다** — 점수에서 judge가 실제로 호출된 trace로 들어갈 수 있는 상태 (C-4 체크포인트)

⭐ **두 번 이상 재고 비교합니다 (이 항목이 축 ④의 핵심)**

> **한 번 재고 끝내면 필수 조건 4는 미충족입니다.**
>
> 무언가를 바꾼 뒤 **같은 평가셋으로 다시 재서 어느 문항이 회귀했는지 숫자로** 보여야 합니다. 교안 C-4에서 환경변수만 바꿔 세 모델을 나란히 돌려본 그 작업을, 여러분 서비스로 하는 것입니다.
>
> 바꿀 대상은 자유입니다 — 모델 · 프롬프트 버전 · 재시도 정책 · 컨텍스트 구성. 다만 **한 번에 하나만** 바꾸세요. 여러 개를 동시에 바꾸면 나아져도 무엇 덕분인지 모릅니다.

`EVAL_REPORT.md` 에 아래가 있어야 합니다.

#	바꾼 것	축별 점수 (전 → 후)	회귀한 문항	판단
1				KEEP / DISCARD

버린 시도도 적으세요. "이걸 해봤는데 효과가 없었다"가 실제로 가장 값진 내용입니다.

9-4. 비용과 품질을 같은 화면에서 판단한다 (필수)

교안이 **C-2와 C-4를 한 쌍**으로 둔 이유입니다.

> "무료 모델로 바꿔서 비용을 90% 줄이자"와 "그런데 품질이 얼마나 떨어지나"는 같은 결정의 양면입니다. 비용만 보면 무조건 싼 모델이 이기고, 품질만 보면 무조건 비싼 모델이 이깁니다.

발표 자료에 이 표가 있어야 합니다.

모델	평가셋 점수 (축별)	요청당 비용	p95 지연	우리 선택

그리고 **왜 그 모델을 골랐는지** 한 문장. "제일 좋아서"가 아니라 "이 정도 품질 손실을 이 정도 비용 절감과 바꿀 만하다고 봤다"가 되어야 합니다.

9-5. 운영 지표 (필수)

지표	개선 전	개선 후
평가셋 점수 (축별로)		
계약 준수율 (스키마 검증)		
p50 / p95 지연		
요청당 평균 비용		

숫자가 나빠도 괜찮습니다. **재고, 원인을 알고, 무엇을 시도했는지**가 평가 대상이에요.

9-6. 점수와 게이트를 구분하세요 (권장)

점수는 순위를 매기고, 게이트는 문을 닫습니다.

평균 82점짜리 서비스가 개인정보를 한 번 유출하면 그 82점은 의미가 없어요. 가중 평균으로 뭉뚱그리면 이런 치명적 실패가 좋은 점수에 묻힙니다.

게이트 조건	왜
스키마 검증 통과율 < 95%	계약을 못 지키면 뒷단이 깨진다
p95 지연 > 목표치	느려서 아무도 안 쓴다
유해·거부 케이스 오작동 1건이라도	한 번으로 서비스가 끝난다
요청당 비용 > 상한	쓸수록 손해다

9-7. 측정 도구를 의심하세요 (가산 요소)

> 지난 기수에서 가장 인상적이었던 리포트는 **자기 점수가 나쁜데 그 나쁜 숫자마저 못 믿겠다고 말한 팀**이었습니다.
>
> 립싱크 품질이 목표 미달로 나왔는데, 결론을 내기 전에 측정 코드를 프레임 단위로 대조했어요. 그랬더니 판정 모델이 **캐릭터의 입술 색에 반응해 입을 다문 프레임을 "최대로 벌어짐"으로 뒤집어 읽고** 있었습니다. 그 팀은 세 가지 측정 방법을 차례로 시도한 뒤, 결과를 "참고용 보조 지표"로 격하하고 최종 판단은 사람 평가에 맡겼습니다.

측정 결과가 직관과 크게 어긋나면 **결론보다 측정 도구를 먼저 의심하세요.** 특히,

- LLM-as-judge의 판정이 사람 눈으로 본 것과 다를 때
- 점수가 이상하게 높거나 낮을 때 (0점·만점이 몰려 나올 때)
- 평가 환경이 실제 서비스와 다른 분기를 타고 있을 때

`EVAL_REPORT.md`의 **"판정 방식 · 한계 · 검증 과정"** 칸에 이 확인을 적으면 축 ④에서 높게 평가합니다. 도구가 틀렸음을 발견하고 다른 방법으로 바꾼 과정은, 좋은 점수를 낸 것보다 값집니다. —

10. 가산점

필수 조건을 모두 충족한 뒤에만 가산됩니다. 필수를 부실하게 하고 가산을 채우는 것은 감점 요인입니다.

> **모델링(파인튜닝)은 선택입니다.** 1차에서 이미 필수로 다뤘고, 2차의 중심은 **만든 것을 운영하고 측정하는 일**입니다. 파인튜닝을 한다면 그것도 축 ④의 문법으로 증명하세요 — 베이스 모델과 같은 평가셋으로 재서 어느 축이 얼마나 올랐는지. "파인튜닝했습니다"만으로는 가산되지 않습니다.

항목	배점	인정 기준
파인튜닝 (선택)	+5	LoRA·QLoRA·PEFT로 도메인 특화 모델을 만들고, **베이스 모델과 같은 평가셋으로 비교**. 학습 데이터 구성 과정 설명 필수
멀티모달 확장	+5	이미지·음성 등 텍스트 외 입력을 실제 기능으로 통합
모델 라우팅	+3	요청 난이도에 따라 모델을 자동 선택하고, 비용 절감 효과를 수치로 제시

> MCP 서버는 **필수 조건 5번**입니다. 가산으로 중복 계산하지 않습니다.

> **파인튜닝을 가산점에 둔 이유**
>
> 1차에서 이미 파인튜닝을 필수로 요구했습니다. 2차에서 다시 필수로 두면 같은 것을 두 번 평가하게 돼요.
>
> 다만 채용 시장에서 LoRA·QLoRA 경험은 여전히 강한 신호(공고 출현율 55% 수준)입니다. 여력이 되는 팀은 도전하세요. 특히 **파인튜닝한 소형 모델이 큰 상용 모델보다 특정 작업에서 싸고 빠르다**는 것을 평가셋으로 증명하면, 그 자체가 포트폴리오의 한 챕터가 됩니다.

11. 도메인 선택 가이드 (취업 관점)

주제는 자유지만, 어차피 만들 것이라면 시장이 사람을 뽑고 있는 곳을 고르는 편이 낫습니다. 아래는 2026년 기준 참고 자료이며, 강제가 아닙니다.

11-1. 시장이 지금 보내는 신호

- 채용 무게중심이 "모델 연구자"에서 "AI 응용 엔지니어"로 이동했습니다. 대기업들이 자기 도메인에 LLM을 즉시 적용할 사람을 찾고 있습니다.
- 평가 역량(eval literacy)이 "LLM으로 실제로 만들어본 사람"과 "강의만 들은 사람"을 가르는 가장 큰 단일 신호로 꼽힙니다. 이 프로젝트의 축 ④가 그것입니다.
- 구조화 로그·요청 ID·요청당 토큰 비용을 갖춘 Docker화된 FastAPI가 응용 직군의 사실상 기본값으로 언급됩니다.
- 정부 정책상 제조 AI 전환에 큰 예산이 걸려 있습니다. AI 팩토리를 2030년까지 500개로 확대하는 계획이 진행 중입니다.
- 고용 전망상 보건·사회복지, 정보통신, 전문·과학·기술 서비스업에서 취업자 증가가 예상됩니다.

11-2. 도메인별 특성

도메인	강점	주의할 점
제조·설비	정책 예산이 크고 경쟁자가 적음. 이력서에서 눈에 띔	실제 데이터 확보가 어려움. 공개 데이터셋이나 시뮬레이션으로 대체
금융·보험	채용 규모가 크고 문서 처리 수요가 명확	규제·개인정보 이슈. 공개 약관·공시 자료로 우회
의료·헬스케어	수요 증가 전망. 사회적 의미가 큼	의료 판단은 절대 자동화하지 말 것. 기록 정리·요약까지만
법률·행정	문서 구조가 명확해 LLM과 잘 맞음	오답의 대가가 큼. 근거 제시와 HITL 필수
커머스·CS	데이터 구하기 쉽고 결과가 눈에 보임	경쟁 프로젝트가 많음. 차별점을 분명히
교육	팀원이 사용자 맥락을 잘 앎	흔한 주제. 평가 설계로 차별화해야 함

11-3. 도메인을 고를 때의 질문

1. 팀원 중 이 도메인의 불편함을 직접 겪은 사람이 있는가?
2. 학습·평가에 쓸 데이터를 합법적으로 구할 수 있는가?
3. 이 서비스가 틀렸을 때 사용자가 얼마나 다치는가? (많이 다치면 HITL을 넣어야 합니다)
4. 면접에서 3분 안에 설명했을 때 상대가 고개를 끄덕일 주제인가?

> **유사 제품을 먼저 찾아라**
>
> 1차 가이드의 원칙 4를 그대로 가져옵니다. “독창성”보다 “실효성”이 우선입니다. 이미 있는 제품을 찾아 구조를 분석하고, 한 가지를 더 잘 하는 방향으로 차별화하세요.

---

12. 제출물

구현 마감: 10월 8일 (목) 자정 · 발표 자료 마감: 10월 11일 (일) 자정

항목	형식	마감	비고
GitHub 레포	KDT 7기 org	10/8	`.env.example` 포함, 민감값 커밋 금지
Vercel 서비스 URL	Vercel	10/8	사용자가 실제로 쓰는 화면. 정상·실패 흐름이 보일 것
Cloud Run Agent API	GCP Cloud Run	10/8	`GET /health`, `POST /api/agent`가 문서대로 동작할 것
README	레포 내	10/8	10분 내 재현 가능한 실행 절차
Skill 자산	레포 내 `skills/`	10/8	도메인 규칙·판단 기준 (필수 조건 5)
MCP 서버	레포 내 + 연결 스크린샷	10/8	`/mcp` 연결 방법, tool 목록과 호출 결과, 접근 제어 방식을 보이게
평가 리포트	레포 내 `EVAL_REPORT.md`	10/8	평가셋 구성, 개선 전후 지표표, 실패 사례 분석
신뢰 경계 1장	`EVAL_REPORT.md` 안의 절로 두어도 됨	10/8	무엇을 믿지 않기로 했는지, 인젝션 케이스 3건의 방어 전후, 아직 막지 못한 것
Langfuse 대시보드	스크린샷 3장 이상	10/8	트레이스 상세 1장 포함
데모 영상	링크	10/8	정상 경로 1개 + **실패 경로 1개**
발표 자료	PDF	**10/11**	문제 정의 → 아키텍처 → 지표 → 한계 순

> 발표 자료만 사흘 늦게 받습니다. 나머지는 10/8 자정에 멈춰요. 그 이후 커밋은 채점에 반영하지 않으니, 10/9~11은 만든 것을 설명하는 데 쓰세요.

> 실패 경로 데모를 요구하는 이유: 정상 경로만 찍은 영상은 어느 팀이나 만듭니다. 장애 상황에서 서비스가 어떻게 반응하는지 보여주는 팀이 준비된 팀입니다.

---

13. 발표에서 반드시 나올 질문

미리 답을 준비해 두세요. 이 질문들이 곧 채점 기준입니다.

1. 누가, 무엇 때문에, 얼마나 절실하게 막혀 있습니까?
2. 무엇을 만들지 않기로 결정했고, 왜입니까?
3. 이 모델을 고른 근거가 무엇입니까? 다른 모델과 비교했습니까?
4. 출력이 스키마를 어겼을 때 무슨 일이 일어납니까?
5. 평가셋은 어떻게 만들었고, 몇 건이며, 무엇을 측정합니까?
6. 개선 전후 숫자가 얼마나 달라졌습니까?
7. 요청 한 번에 얼마가 들고, p95 지연은 얼마입니까?
8. 지금 이 서비스의 가장 큰 약점은 무엇입니까?

> 8번에 "없습니다"라고 답하는 팀은 자기 서비스를 모르는 팀입니다. 약점을 정확히 아는 것이 그 자체로 실력이에요.

---

14. 이 프로젝트를 취업에 쓰는 법

14-1. 이 프로젝트가 커버하는 채용 요구

이력서에 한 줄로 적으려고 만드는 게 아닙니다. 아래 항목들이 실제 채용 공고에 반복해서 나오는 것들이고, 이 프로젝트를 끝내면 각각에 대해 **자기 경험으로** 답할 수 있게 됩니다.

채용 공고에 나오는 표현	이 프로젝트에서 해당하는 것
"LLM 기반 서비스 개발 경험"	필수 조건 1~3 전체
"FastAPI 등 백엔드 서빙 경험"	축 ③, 필수 조건 2
"Docker 컨테이너 환경 경험"	필수 조건 2
"LLMOps / 모델 모니터링"	축 ④ Langfuse 트레이싱
"프롬프트 엔지니어링"	축 ② 출력 계약·검증
"LLM 평가 체계 수립"	축 ④ 평가셋, `EVAL_REPORT.md`
"비용 최적화 경험"	원칙 4, 모델 라우팅 가산
"MCP / 도구 연동"	MCP 서버 (필수 조건 5)
"LoRA·PEFT 파인튜닝"	파인튜닝 가산

14-2. GitHub README에 반드시 넣을 것

채용 담당자가 레포를 여는 시간은 길지 않습니다. 아래 세 가지가 상단에 없으면 읽히지 않아요.

1. **한 줄 정의** — "누구의 어떤 문제를 어떻게 푸는가". 기술 나열이 아니라 문제로 시작합니다.
2. **지표 표** — 평가셋 통과율, p95 지연, 요청당 비용. 숫자가 있는 README는 드뭅니다.
3. **실행 방법** — `docker compose up` 한 줄. 이게 되면 "돌려볼 수 있는 프로젝트"가 됩니다.

아키텍처 다이어그램과 데모 GIF는 그다음입니다.

> 흔한 실수: README에 사용 기술 로고만 잔뜩 붙이는 것. "FastAPI, Docker, OpenAI, Langfuse 사용"은 누구나 씁니다. 숫자와 문제 정의가 변별력이에요.

14-3. 면접에서 이 프로젝트로 답할 수 있는 질문

발표 질문(13장)과 겹치지만, 면접에서는 표현이 이렇게 바뀝니다.

면접 질문	이 프로젝트의 어느 부분으로 답하나
"LLM이 이상한 답을 내면 어떻게 처리하셨나요?"	출력 계약 + 3단계 재시도 + 폴백
"품질을 어떻게 검증하셨나요?"	평가셋 30건, 개선 전후 비교
"비용은 어떻게 관리하셨나요?"	요청당 토큰 비용 측정, 캐싱·라우팅
"모델은 왜 그걸 고르셨나요?"	2개 이상 비교 실험 결과
"장애가 나면 어떻게 되나요?"	실패 경로 설계 표
"협업은 어떻게 하셨나요?"	팀 레포, 역할 분담, 중간 점검

이 질문들에 "해봤습니다" 대신 **숫자와 함께** 답할 수 있는 것이 이 프로젝트의 목적입니다.

14-4. 부풀리지 마세요

면접관은 금방 압니다. 실제로 한 것만 적되, 정확하게 적으세요.

- ✗ "대규모 트래픽 처리" — 동시 요청 5건 테스트했으면 그렇게 씁니다
- ✗ "정확도 95% 달성" — 무슨 평가셋 몇 건인지 없으면 의미가 없습니다
- ✓ "자체 평가셋 30건 기준 통과율 62% → 84%, p95 지연 4.1초"

한계를 아는 사람으로 보이는 편이 만능으로 보이는 것보다 낫습니다.

---

15. 자주 나오는 질문

비용과 환경

Q. LLM API 비용은 누가 부담하나요? 운영 방식은 별도 공지합니다. 다만 **비용을 줄이는 것 자체가 평가 항목(원칙 4)**이라는 점을 기억하세요. 개발 중에는 작은 모델로 파이프라인을 검증하고, 평가셋 실행처럼 꼭 필요한 때만 큰 모델을 쓰는 습관이 그대로 점수가 됩니다.

Q. GPU가 필요한가요? 필수 요건에는 필요 없습니다. LLM은 API로 부르니까요. **파인튜닝 가산점에 도전할 때만** 필요하고, 그 경우 Colab·Kaggle 무료 GPU 또는 1차 프로젝트 때 쓰던 환경을 재사용하세요.

Q. 배포는 어디에 하나요? 무료로 되나요? Railway, Fly.io, Render, Cloud Run 모두 소규모 무료 또는 저비용 티어가 있습니다. **무료 티어 정책은 자주 바뀌니 착수 시점에 직접 확인하세요.** 배포가 막히면 팀 전체가 멈추므로, 구현 초반(9/28 주)에 "빈 FastAPI 하나 띄우기"부터 해두는 것을 권합니다.

Q. Docker가 Apple Silicon(M1~M4)에서 안 떠요. 이미지 아키텍처 불일치가 가장 흔한 원인입니다. `platform: linux/amd64`를 명시하거나 arm64 대응 베이스 이미지를 쓰세요. Docker 실습 STEP 1~9(`5. 특강_실습/docker/`)에 같은 상황의 대처가 정리돼 있습니다.

Q. Langfuse 셀프호스팅이 노트북에서 너무 무거워요. Langfuse 개인 실습의 **STEP 0-A "Docker 없이 Langfuse Cloud로 대신하기"**를 따르세요. 강의가 이미 그 경로를 준비해 뒀습니다. 트레이스가 남으면 필수 조건 4는 충족됩니다.

Q. Windows인데 실습 명령어가 안 먹어요. Langfuse 실습 교안 앞부분에 "Windows 수강생은 반드시 먼저 읽으세요" 섹션이 있습니다. PowerShell Here-String은 작은따옴표 `@' '@`를 써야 하는 것 같은 함정이 정리돼 있어요.

요구사항 해석

Q. 우리가 만든 게 RAG인지 아닌지 헷갈립니다. 4-3-1의 표 1로 판정하세요. ① 지식이 청크 더미로 저장돼 있고 ② 질문마다 골라오는 조각이 바뀌며 ③ 그 조각을 프롬프트에 붙여 생성한다 — **셋 다 예일 때만** RAG입니다. 하나라도 아니면 괜찮아요. 구체 사례는 표 2에 15가지를 정리해 뒀습니다.

Q. Skill 문서를 프롬프트에 넣는 것도 RAG 아닌가요? 아닙니다. 표 1의 ①②가 아니오예요. 통짜 문서이고 질문이 바뀌어도 같은 내용이 들어갑니다. MCP 도구가 API로 가져온 값도 마찬가지로 문서 조각 검색이 아니고요. **둘 다 필수 조건 5번이니 마음껏 쓰세요.**

Q. RAG를 못 쓰면 도메인 지식은 어떻게 넣나요? 그게 필수 조건 5번입니다. 고정된 지식은 **Skill**(규칙·판단 기준을 적은 문서와 프롬프트 자산)로 넣고, 그때그때 필요한 데이터는 **MCP 도구**로 가져옵니다. 검색해서 프롬프트에 붓는 방식은 3차에서 제대로 배웁니다.

Q. OpenAI Agents SDK나 Claude Agent SDK는 써도 되나요? 됩니다(조건부). 3주차 16~20강에서 배운 다섯 SDK는 허용이에요. **Part IV에서는 이걸 다루지 않으므로 2차가 실제로 써 볼 유일한 기회입니다.** 다만 4-3의 조건 세 가지(SDK의 재시도·구조화 출력 처리 방식, 기본값이 우리 요구에 맞는지, 삼킨 실패가 트레이스에 남는지)에 답할 수 있어야 축 ②로 인정됩니다.

Q. 그럼 직접 구현하는 게 손해인가요? 아니요. 두 길 모두 25점 만점이 가능합니다. 직접 짜면 코드로 증명되고, SDK를 쓰면 설명으로 증명해요. 오히려 직접 짜 본 팀이 "SDK가 뭘 대신해주는지"를 더 잘 말합니다. 시간이 빠듯하면 SDK로 뼈대를 세우고 계약·재시도만 직접 손보는 절충도 좋은 선택입니다.

Q. 왜 LangGraph와 RAG만 금지인가요? LangChain은 되는데. 그 둘은 3차에서 **주제 자체로** 다루기 때문입니다. Part IV의 `1. RAG 파이프라인` 20강과 `2. LangGraph 런타임` 20강이 통째로 그 자리예요. 미리 써버리면 3차가 복습이 됩니다. 반면 LangChain·벤더 SDK는 Part IV에서 다시 다루지 않으니, 지금 안 쓰면 배운 것을 써 볼 자리가 없습니다.

Q. MCP 서버를 만들었는데 Claude Desktop이 없어요. Cursor도 MCP를 지원합니다. 둘 다 없으면 18강처럼 `langchain-mcp-adapters`로 붙여도 되고, `mcp` 패키지의 개발용 클라이언트로 검증해도 됩니다. **연결해서 도구가 불린다는 것만 화면으로 보이면 됩니다.**

Q. 평가셋 30건의 정답을 우리가 정해도 되나요? 네, 그게 정상입니다. 중요한 것은 **정답을 정한 기준을 문서에 적는 것**이에요. "이 항목은 이런 이유로 이게 맞다"가 있으면 됩니다. 팀원 간 판단이 갈리는 항목이 나오면 그 자체가 좋은 발견이고, 리포트에 적을 가치가 있습니다.

Q. 모델 비교를 공정하게 하려면? 같은 입력, 같은 프롬프트, 같은 평가셋, 같은 판정 기준. 하나라도 다르면 비교가 아닙니다. temperature 같은 파라미터도 맞추거나, 못 맞추면 그 사실을 적으세요.

Q. 우리 주제와 똑같은 서비스가 이미 있어요. 좋은 신호입니다(원칙 4). 문제가 실재한다는 뜻이니까요. 그 서비스가 못 푸는 한 가지를 찾아 거기에 집중하면 됩니다.

Q. 1차와 같은 팀인가요? 주제를 이어가도 되나요? 팀 구성은 별도 공지합니다. 주제는 이어가도 되고, 그 경우 1차의 TTS·STT 자산을 유형 2에서 재활용할 수 있습니다. 단 **1차에서 이미 평가받은 부분은 2차 점수에 반영되지 않습니다.** 2차의 4축으로 새로 평가해요.

일정과 진행

Q. 추석 연휴에 팀원이 안 모여요. 그래서 9/23까지 문제 정의와 평가셋을 확정하라고 하는 겁니다. 그 둘이 있으면 연휴 중에는 각자 비동기로 진행할 수 있어요. 없으면 연휴가 통째로 사라집니다.

Q. 기능을 다 못 만들 것 같습니다. 기능을 줄이세요(원칙 5). 이 프로젝트는 기능 개수로 평가하지 않습니다. 기능 하나가 계약을 지키고, 실패해도 우아하게 떨어지고, 측정된 상태인 것이 기능 다섯 개가 가끔 되는 것보다 높은 점수를 받습니다.

Q. 평가셋 점수가 개선되지 않았습니다. 그대로 제출하세요. 무엇을 시도했고 왜 안 됐는지 분석하면 점수를 받습니다. 오히려 아무 실패 없이 "다 잘 됐습니다"라는 리포트가 의심받아요. 실제로 LLM 서비스 개선은 대부분 그렇게 매끄럽지 않습니다.

부록. 참고 자료

강의 자료

주제	위치
출력 계약·검증·재시도	Part III / 1. LLM 제어와 출력계약 / 12~16강
모델 선택·라우팅·비용	Part III / 1. LLM 제어와 출력계약 / 9·10·17강
배치·대시보드	Part III / 1. LLM 제어와 출력계약 / 19·20강
도구 설계·tool use	Part III / 2. Agent 런타임과 MCP / 6~9강
MCP 서버 (프로젝트에 쓸 부분)	Part III / 2. Agent 런타임과 MCP / 15·16·17·20강
평가셋·실패 진단	Part III / 3. 컨텍스트·신뢰성·평가 / 15강
평가 8지표·출시 게이트	Part III / 3. 컨텍스트·신뢰성·평가 / 14강 + 특강 Agent Evaluation Gate
prompt injection 방어	Part III / 3. 컨텍스트·신뢰성·평가 / 11·12강
FastAPI 워크숍	Part III / 5. 특강_실습 / fastapi/ (docs 가이드 13종 포함)
Docker 실습	Part III / 5. 특강_실습 / docker/ (STEP 1~9)
Langfuse 실습	Part III / 5. 특강_실습 / langfuse/개인실습/
MCP 서버 실습	Part III / 5. 특강_실습 / mcp_server/