Agentic AI를 활용한

지역아동센터 수요 예측 서비스

26.01.05

김세준

GitHub

목차

01 기획서 – 기획의도(1/2)

여성농업인신문

종합 농촌여성 농민단체 소식

Since2005

홈 > 농촌여성

아이돌봄서비스 수요 느는데 인력은 태부족

김수현 기자 | 입력 2025.11.07 14:01 | 댓글 0

전국 아이돌봄센터 232개소,
돌보미수 2만9천여명
지난해 13만9천명 신청
평균 대기 기간 32.8일

사회 교육

초등 학부모 50% "수업 전후 돌봄 필요"...4년새 20%p 급증

2023년 범정부 온종일 돌봄 수요조사

김민제 기자

수정 2023-03-06 13:35 등록 2023-03-06 13:35

가족 형태 변화 + 맞벌이 증가로
방과후 돌봄 수요 확대

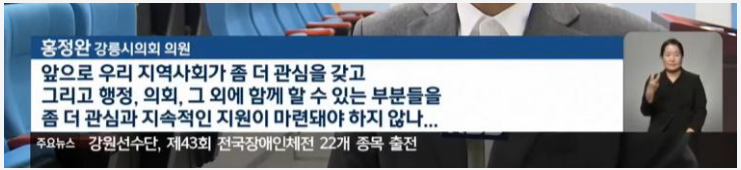
↓

가정에서 아이 돌봄 감당하기 어려워
야간, 방과후 돌봄 공백 발생

↓

"지역아동센터 이용자 수요를 예측할 필요"

01 기획서 – 기획의도(2/2)



돌봄 수요는 전국적으로 증가하지만
지원 여건은 지역마다 제각각

어느 지역은 수요 대비
예산·시설·인력이 부족

"**자치구별 이용자 수를 예측**해
정책 의사결정을 지원 "

01 기획서 - 개발 목표(1/3)

이용자 수 예측

- Target: 자치구별 지역아동센터 이용자 수(연도별)
- Feature(예시):
  - 한부모가구 수
  - 아동인구
  - 학원 수
  - GRDP(지역내총생산) 등
- 학습 데이터: 2015~2020년 / 검증(Test): 2021~2022년
- 데이터 출처 : 서울열린데이터광장, KOSIS국가통계포털

수요 분석

- 예측 결과를 자치구·연도별로 비교하여 변화 추세 파악
- 증가/감소 폭이 큰 구를 식별해 원인 지표(가구·인구·경제)와 함께 해석
- 예측값과 현재값의 차이를 근거로 정책 대응 우선순위 참고자료 제공
- 도출 결과를 자치구별 그래프로 시각화해 한눈에 비교

01 기획서 - 개발 목표(2/3)

목 표

- 자치구별 이용자 수 예측을 위한 회귀 모델 구축
- 다양한 지역 지표(인구·가구·경제)를 활용해 예측 성능 개선
- 교차검증 기반 하이퍼파라미터 튜닝으로 최적 모델 선정
- 예: n_estimators, max_depth, learning_rate, subsample, colsample_bytree

예측 결과 조회·비교 UI

- 지역 / 기간 선택만으로 지도 / 그래프 기반 통계, 예측 결과를 한 번에 조회
- 시각화 된 결과를 통해 수요가 높은 지역을 직관적으로 파악
- 지원 우선순위를 세우는 참고자료로 활용할 수 있는 웹 화면 제공

01 기획서 - 개발 목표(3/3)

LLM 기반 정책 지원

- 목표: 정책/운영 문서 기반으로 근거 있는 답변·요약제공
- HuggingFace: Transformers(모델/토크나이저), PEFT(LoRA), pipeline(요약)로 LLM 기능 구현
- LLM: Llama3-8B(파인튜닝/추론 서버 연동)
- RAG: 정책 PDF/문서를 임베딩 후 Chroma Vector DB에 저장 → 관련 근거 검색
- 요약: KoBART 기반 요약 모델로 긴 문서 핵심 요약

생성형 AI 기능 UI

- 보고서 생성: 자치구/기간 입력을 자동 파싱해 요약·근거·시사점 형식 보고서 출력
- 정책 Q&A: 정책/지침 문서 기반 근거(RAG) 포함 답변 제공
- 정책 제안: 예측 결과 + 정책 문맥을 반영해 대안/우선순위 제시
- 텍스트 요약: 긴 민원/기사/문서 입력을 핵심 요약 + 키포인트로 압축

02 사용 기술 목록/경험

사용 기술 목록

Language

[Python]
[3.11]

[HTML5]

[JavaScrpit]

[CSS3]

Library

[Numpy]
[1.26.4]

[Pandas]
[2.3.3]

[Scikit-learn]

AI Framework

[Hugging Face]

[LangChain]

Infra

[runpod]

[docker]

Database

[SQL Developer]

[Orcale]
[11g]

[ChromaDB]

FrameWork

[Flask]
[3.0.3]

02 사용 기술 목록/경험 (1/3)

Fine Tuning

Neural Network
- Meta Llama-3-8B-Base
- Transformer 기반 Causal Language Model

Adaptation Techniques
- LoRA Dropout(0.0) 적용
- PEFT/LoRA: Rank 16, Alpha 16 가중치 튜닝

Training & Checkpoint Strategy
- Batch: per_device_train_batch_size=2, gradient_accumulation_steps=8
- Steps: max_steps=300, warmup_steps=50
- 검증/저장: eval_steps=50, save_steps=100 (strategy='steps')
- Seed=42, report_to='none'

Optimization
- Optimizer: adamw_8bit, learning_rate=1e-4, weight_decay=0.01
- Scheduler: constant_with_warmup
- gradient_checkpointing=True (메모리 절감)

RAG

Knowledge Ingestion
- 복지/정책 PDF 문서 수집 및 폴더 단위 로딩(DirectoryLoader, PyPDFLoader)
- RecursiveCharacterTextSplitter 기반 청크 분할(문맥 보존)

Embedding & Vector
- jhgan/ko-sroberta-multitask 임베딩 적용
- Chroma VectorStore 구축 및 persist_directory로 영속화
- 기존 DB 존재 시 재인덱싱 없이 로드(초기 로딩 시간 절감)

Retrieval & Prompting
- 사용자 질문 → 관련 문서 Top-k 검색 후 컨텍스트 주입
- 규칙/수치 기반 답변 형식(3줄 불릿 등) 프롬프트 템플릿 적용
- 할루시네이션 완화(근거 없는 단정/절대값 금지 규칙 적용)

Intergration
- Flask서비스(GenAIService)에서 RAG 모듈 지연 로딩/캐싱
- LLM 응답 + 근거 컨텍스트를 UI에 함께 출력하도록 연동

Machine Learning

Supervised Learning
- Multiple Linear Regression / Random Forest / XGBoost 모델 비교·적용

Feature Engineering & Preprocessing
- 자치구 One-Hot Encoding + 결측/이상치 처리, 학습 컬럼 스키마 고정

Model Interpretation
- RandomForest Feature Importance로 주요 영향 변수 분석

Model Evaluation
- MAE, RMSE 기반 모델 성능 비교 및 검증
- 모델별 결과 비교 후 성능/안정성 기준으로 모델 선정

Hyperparameter Tuning
- RandomizedSearchCV(n_iter=30, cv=3, scoring=R^2)로 XGBoost 튜닝
- 2차 Local Search + log1p(target) 적용 후 expm1로 원복 평가

02 사용 기술 목록/경험 (2/3)

Flask / Backend

Architecture
- Blueprint 기반 모듈 분리 및 라우팅 설계
- ServiceLayer(GenAIservice, RagService, predict등)로 로직 분리

API & Integration
- Flask JSON API 설계(요청/응답 스키마) 및 예외 처리
- RunPod(FastAPI) 추론 서버 연동: /generate, /switch_model 호출

GenAI Features
- 보고서/요약/규칙 기반 응답 등 프롬프트 템플릿 적용 및 결과 후처리
- 입력 파싱 → 필요한 데이터 조회/가공 → LLM 컨텍스트 구성 흐름 구현

Deployment
- 환경변수(.env) 기반 설정 분리 (RUNPOD_API_URL 등) 및 배포 환경 구성

Database

관계형 데이터베이스 개념
- Oracle과 DBMS 기본 이해
- 테이블, 기본키/외래키, 무결성 제약조건
- 정규화(Nomalization)를 통한 중복 최소화

Oracle 활용
- SELECT,JOIN,GROUP BY 등 작성
- 조건문/필터링을 통한 데이터 추출 및 집계

테이블 관리
- 머신러닝 학습용 테이블 생성 및 분할

Python 연동
- cx_Oracle을 활용한 DB ↔ Python 연계
- 학습데이터 추출 및 모델 입력용 데이터 프레임 변환

HTML / CSS / JS

UI composition
- Jinja2 탬플릿 기반 화면 구성 및 공통 레이아웃(상단 메뉴/네비) 적용
- 생성형 AI 탭 포함 다중 기능 페이지 UI 설계

Async Communication
- Fetch API로 Flask 엔드포인트 호출 (JSON 송수신)
- 로딩 상태/에러 처리 등 사용자 인터랙션 흐름 구현

Visualization
- Chart.js로 예측/통계 데이터 시각화 (추세 그래프, 비교차트)
- 자치구/연도 조건 변경 시 그래프 동적 업데이트

02 사용 기술 목록/경험 (3/3)

Git

- GitHub 기반 브랜치 전략으로 기능 단위 개발/통합 (feature → main)
- PR/merge 중심으로 변경 이력 관리 및 팀 협업 흐름 운영
- 기능 단위 커밋 메시지 규칙 적용 (원인/변경/영향이 보이게)
- README/문서 정리로 실행 방법 및 결과 공유

Docker

- docker-compose로 Flask 앱 + Oracle DB 개발 환경 구성/실행
- 서비스 간 네트워크/포트 구성으로 로컬 통합 테스트 가능
- 실행 환경을 컨테이너로 표준화하여 "동일 환경 재현" 확보
- 환경 변수(.env) 기반 설정 분리로 로컬/배포 환경 대응

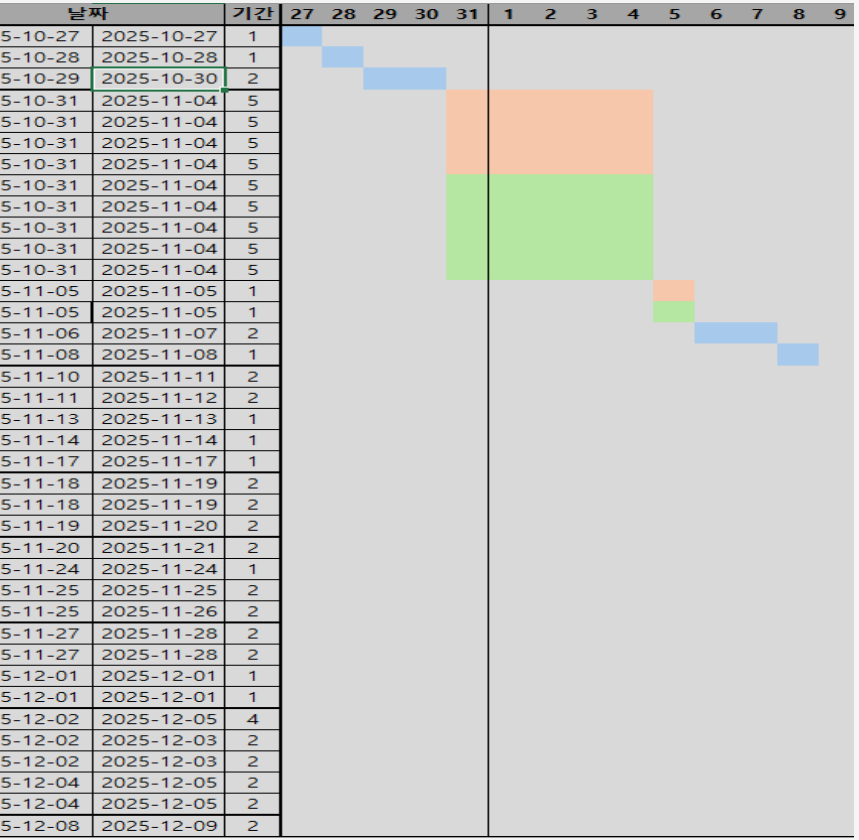
Cloud / RunPod

- RunPod GPU 인스턴스에서 Llama 3 실행 환경 구성 (CUDA/라이브러리 세팅)
- 추론 서버(FastAPI)로 모델 서빙 엔드포인트 제공
- 체크포인트/LoRA 어댑터 등 모델 산출물 버전 관리 및 교체 테스트
- 학습(파인튜닝)과 추론 서빙을 분리 운영하여 비용/안정성 고려

03 개발 스케줄 (1/2)

담당자	색상
김진혁	
김세준	
공동	

구분	담당자	파트	날짜		기간
프로젝트 기획	공동	아이디어 구상	2025-10-27	2025-10-27	1
		화면 목록 구성	2025-10-28	2025-10-28	1
		Information Architecture	2025-10-29	2025-10-30	2
데이터 단계	김진혁	[서울 열린데이터광장]GRDP(서울시 구별, 년)	2025-10-31	2025-11-04	5
		[서울 열린데이터광장]기초생활 수급자(자치구별, 년)	2025-10-31	2025-11-04	5
		[서울 열린데이터광장]다문화가구(자치구별, 년)	2025-10-31	2025-11-04	5
		[서울 열린데이터광장]연령별 등록 인구(자친구별, 년)	2025-10-31	2025-11-04	5
		[KOSIS]시군구별 연도별 이혼율	2025-10-31	2025-11-04	5
	김세준	[서울특별시]서울시 아동복지시설수, 이용자수(자치구별, 년)	2025-10-31	2025-11-04	5
		[서울 열린데이터 광장]서울시 저소득 한부모 가족(자치구별, 년)	2025-10-31	2025-11-04	5
		[KOSIS]서울시 자치구별 출생아수(년)	2025-10-31	2025-11-04	5
		[서울특별시교육청]학생 1만명당 사설학원수(자치구별, 년)	2025-10-31	2025-11-04	5
	김진혁	지원센터 이용자수 데이터 정제	2025-11-05	2025-11-05	1
	김세준	변수 데이터 정제	2025-11-05	2025-11-05	1
	공동	EDA(통계 시각화, 관계분석)	2025-11-06	2025-11-07	2
		모델 피처 확정	2025-11-08	2025-11-08	1
모델링	김진혁	베이스라인 모델 구축	2025-11-10	2025-11-11	2
	김세준	트리 기반 모델 개발	2025-11-11	2025-11-12	2
	공동	모델 성능 검증 및 비교	2025-11-13	2025-11-13	1
		파이프라인 구성	2025-11-14	2025-11-14	1
		모델 최종 선정	2025-11-17	2025-11-17	1
시스템 설계	공동	Flask 구조 설계	2025-11-18	2025-11-19	2
		DB 테이블 설계	2025-11-18	2025-11-19	2
		API 설계	2025-11-19	2025-11-20	2
UI/UX	공동	화면설계서 초안 작성	2025-11-20	2025-11-21	2
		UI 시안	2025-11-24	2025-11-24	1
	김세준	전체 화면 설계서	2025-11-25	2025-11-25	2
	김진혁	시퀀스 다이어그램	2025-11-25	2025-11-26	2
백엔드	김세준	통계 대시보드 API 개발	2025-11-27	2025-11-28	2
		Q&A API	2025-11-27	2025-11-28	2
	김진혁	계정 로그인 / 회원가입 API	2025-12-01	2025-12-01	1
		지역아동센터 수요 예측 API 개발	2025-12-01	2025-12-01	1
프론트엔드	김세준	메인 페이지 UI	2025-12-02	2025-12-05	4
		Q&A 페이지	2025-12-02	2025-12-03	2
	김진혁	대시보드 UI	2025-12-02	2025-12-03	2
		예측 결과 페이지 UI	2025-12-04	2025-12-05	2
		UI소개 페이지 UI	2025-12-04	2025-12-05	2
테스트	공동	프론트-백엔드 연동 및 최종 테스트	2025-12-08	2025-12-09	2



03 개발 스케줄 (2/2)

프로젝트 기획	개인 프로젝트	AI 서비스 기획	2025-12-10	2025-12-10	1		
		사전학습 모델 벤치마킹	2025-12-11	2025-12-12	2		
데이터 단계		도메인 지식 문서 수집	2025-12-15	2025-12-15	1		
		학습 데이터셋 가공	2025-12-15	2025-12-15	1		
모델링		Llama-3 파인튜닝	2025-12-16	2025-12-18	3		
		KoBART 요약 모델 구축	2025-12-19	2025-12-20	2		
시스템 설계		분산 추론 아키텍처 설계	2025-12-22	2025-12-23	2		
		RAG 및 벡터 DB 설계	2025-12-24	2025-12-25	2		
UI/UX		생성형 AI 화면 설계	2025-12-26	2025-12-26	1		
백엔드		LLM 연동 API 개발	2025-12-29	2025-12-30	2		
		텍스트 요약 API 개발	2025-12-31	2025-12-31	1		
프론트엔드		생성형 AI 통합 UI 개발	2026-01-02	2026-01-02	1		
테스트		AI 추론 및 성능 테스트	2026-01-05	2026-01-05	1		
배포 및 운영		Docker 이미지 빌드	2026-01-06	2026-01-06	1		
		OCI 클라우드 배포	2026-01-07	2026-01-07	1		

04 요구사항 정의서/ 분석서

요구사항 정의서(1/2)

RQ-ID	화면 명	요구사항 명	요구사항 상세
RQ-ID-0001	메인 페이지	로고	웹 페이지 왼쪽 상단 클릭 시, 첫 화면으로 이동할 수 있는 로고를 삽입
RQ-ID-0002	메인 페이지	상단 네비게이션 바	네비게이션 바 각 영역에 머신러닝/통계 대시보드/로그인/회원가입으로 구분되어 있고 클릭 시 해당 페이지로 이동
RQ-ID-0003	메인 페이지	웹페이지 하단	웹 페이지 하단에 위치하며 서비스 이용 약관/개인정보 처리방침/정보 등을 포함
RQ-ID-0004	메인 페이지	이미지 슬라이드(캐러셀)	자동으로 전환하는 이미지를 페이지 중앙에 설정
RQ-ID-0005	메인 페이지	스크롤 스냅	메인 페이지 내 스크롤 시 섹션 단위로 자동 정렬되며 스크롤 이동이 부드럽게 처리
RQ-ID-0101	서비스 소개	메인 서비스 이동 버튼	주요 서비스 페이지로 빠르게 이동할 수 있는 버튼 제공, 클릭 시 해당 서비스 페이지로 이동
RQ-ID-0102	서비스 소개	데이터 기반 안내	서비스가 실제 통계 기반 머신러닝 모델로 동작함을 사용자에게 안내
RQ-ID-0201	통계 대시보드	연도·지역 필터	사용자는 연도 범위 및 자치구 선택을 통해 데이터를 조회
RQ-ID-0202	통계 대시보드	자치구 검색 자동완성 기능	자치구 이름 일부 입력 시 일치/유사 자치구를 자동완성 목록으로 제공
RQ-ID-0203	통계 대시보드	동적 데이터 갱신	필터 변경 시 새로고침 없이 값이 즉시 갱신
RQ-ID-0204	통계 대시보드	카드형 지표	이용자 수·시설 수 지표가 카드 UI로 구성
RQ-ID-0205	통계 대시보드	연도 범위 선택 제약 기능	종료 연도는 시작 연도와 같거나 이후 연도만 선택 가능
RQ-ID-0206	통계 대시보드	데이터 예외 처리	선택된 기간에 데이터가 없을 경우 '데이터가 없습니다.' 메시지가 표시
RQ-ID-0301	예측 화면	연도 및 자치구 선택	사용자가 예측 조회를 위해 자치구 및 연도를 선택
RQ-ID-0302	예측 화면	예측 그래프	2015~2022 실제값과 2023~2030 예측값이 동일 그래프에서 비교 표시
RQ-ID-0303	예측 화면	성능 정보 제공	예측 결과 하단에 모델 성능 정보 표시
RQ-ID-0304	예측 화면	비교 분석 기능	서울 평균 대비 선택 지역의 예측 추세를 비교할 수 있는 기능이 포함
RQ-ID-0401	지도 UI	SVG 지도 표시	서울시 기반 SVG 지도 UI가 화면에 표시
RQ-ID-0402	지도 UI	선택 연동 기능	특정 자치구 클릭 시 예측 데이터 및 그래프가 동기화
RQ-ID-0501	Q&A	목록 조회	게시글 목록 조회 기능
RQ-ID-0502	Q&A	글 작성	Q&A 게시판 목록 화면에서는 각 게시글의 번호, 제목, 작성일시가 표 형태로 함께 표시
RQ-ID-0504	Q&A	댓글 기능	게시글에는 댓글 작성 및 표시 기능이 제공
RQ-ID-0601	로그인	로그인 기능	사용자 아이디 + 비밀번호로 로그인
RQ-ID-0602	로그인	오류 처리	로그인 실패 시 오류 메시지가 표시

04 요구사항 정의서/ 분석서

요구사항 정의서(2/2)

RQ-ID	화면 명	요구사항 명	요구사항 상세
RQ-ID-0603	회원가입	회원 등록	사용자 정보 입력 후 회원가입
RQ-ID-0604	회원가입	회원가입 입력값 유효성 검사	이메일 형식이 올바르지 않거나 비밀번호.비밀번호 확인이 일치하지 않을 경우 각각 오류 메시지를 표시
RQ-ID-0605	회원가입	암호화 저장	비밀번호는 암호화(HASH) 처리되어 저장
RQ-ID-0606	로그아웃	로그아웃 알림 메시지	사용자가 로그아웃할 경우, "로그아웃 되었습니다."와 같은 안내 내용을 팝업으로 표시
RQ-ID-0607	인증	아이디/비빈번호 찾기	이메일 인증 기반으로 계정 정보 찾기
RQ-ID-0701	생성형AI	AI 페이지 UI 구성	생성형 AI 전용 대시보드 및 채팅/리포트 출력 영역 구성
RQ-ID-0702	생성형AI	자치구별 분석 리포트 생성	DB 예측 데이터를 컨텍스트로 활용하여 3줄 요약(요약, 요인, 데이터) 형태의 분석 결과 제공계정 정보 찾기
RQ-ID-0703	생성형AI	정책 아이디어 제안	파인튜닝된 모델을 통해 자치구 상황에 맞는 3가지 정책 아이디어 리스트 도출
RQ-ID-0704	생성형AI	전문 Q&A	RAG를 참조하여 아동 복지 정책 관련 신뢰도 높은 질의응답 기능 제공
RQ-ID-0705	생성형AI	텍스트 요약	사용자가 입력한 복지 정책 원문이나 긴 텍스트를 경량 모델(KoBART)을 활용하여 핵심 내용 요약 제공
RQ-ID-0706	생성형AI	인프라 연동(FastAPI)	Flask 백엔드와 외부 FastAPI(RunPod) 추론 서버 간 비동기 API 통신 구현
RQ-ID-0707	생성형AI	모델 버전 비교 모드	Base, Middle, Final 등 파인튜닝 단계별 모델 답변 비교 기능
RQ-ID-0708	생성형AI	추론 성능 로깅	AI 답변 생성 시간 및 추론 성공 여부를 DB(Chat Log)에 기록

04 요구사항 정의서/ 분석서

요구사항 분석서(1/2)

1.1 메인 페이지 기능↵

1.1.1 메뉴 이동 기능↵

- 상단 메뉴 클릭 시 각 페이지로 이동할 수 있어야 한다.↵
- 상단 메뉴는 모든 페이지에서 동일한 레이아웃으로 고정되어야 한다.↵
- 로그인 상태일 경우 "로그아웃" 버튼이 표시되고 클릭 시 로그아웃 처리 후 메인 화면으로 이동해야 한다.↵

1.1.2 캐러셀 기능↵

- 메인 페이지에는 자동 재생되는 캐러셀이 있어야 한다.↵
- 캐러셀은 약 5 초 간격으로 자동 전환되어야 한다.↵
- 좌우 네비게이션 버튼 또는 인디케이터 클릭을 통해 수동 제어가 가능해야 한다.↵

1.1.3 스크롤 스냅 기능↵

- 메인 화면 스크롤 시 주요 섹션 단위로 화면이 자연스럽게 정렬 되도록 스크롤 스냅이 적용되어야 한다.↵
- 사용자 스크롤을 내리거나 올릴 때, 각 섹션이 부드럽게 전환되며 화면이 고정되어야 한다.↵

1.1.4 주요 서비스 이동 버튼 기능↵

- 메인 페이지에 서비스 이동 버튼(통계 대시보드, 머신러닝, QnA)이 제공되어야 한다.↵
- 버튼 클릭 시 해당 기능 페이지로 이동해야 한다.↵

1.2 서비스 소개 기능↵

1.2.1 서비스 목적 소개 기능↵

- 서비스 목적이 첫 화면에서 명확히 전달되어야 한다.↵
- 예측 기반 서비스의 필요성과 사회적 활용성을 문구 및 시각 요소로 설명해야 한다.↵

1.2.2 데이터 기반 설명 기능↵

- 서비스가 실제 데이터 기반으로 동작한다는 안내가 포함되어야 한다.↵

↵

1.3 통계 대시보드 기능↵

1.3.1 연도·지역 필터 기능↵

- 사용자는 연도 범위 및 자치구를 선택하여 데이터를 조회할 수 있어야 한다.↵
- 필터 변경 시 새로고침 없이 동적으로 값이 갱신되어야 한다.↵

1.3.2 표 및 지표 표시 기능↵

- 이용자 수, 시설 수, 증감률 등 핵심 수치가 카드형 UI 로 표시되어야 한다.↵
- 조회된 데이터는 표 및 그래프 형태로 시각화되어야 한다.↵

1.3.3 데이터 예외 처리 기능↵

- 조회 범위 내 데이터가 없을 경우 "데이터가 없습니다." 메시지를 표시해야 한다.↵

1.4 머신러닝 화면 기능↵

1.4.1 연도 및 자치구 선택 기능↵

- 사용자가 예측 값 조회를 위해 자치구 및 연도를 선택할 수 있어야 한다.↵
- 선택된 값은 즉시 화면에 반영되어야 한다.↵

1.4.2 예측 그래프 표시 기능↵

- 2015~2022 년 실제 데이터와 2023~2030 년 예측 데이터를 동일 그래프에서 비교하여 표시해야 한다.↵
- 그래프는 마우스 오버 시 수치가 강조 표시되어야 한다.↵
- 예측 그래프 밑에는 모델 성능 표시가 있어야 한다.↵

1.4.3 비교 분석 기능↵

- 서울 평균과 선택 구의 예측 값 비교 기능이 제공되어야 한다.↵
- 전년 대비 증감 수치 및 변화율(%) 정보가 포함되어야 한다.↵

↵

1.5 지도 연동 기능↵

1.5.1 SVG 지도 표시 기능↵

- 서울시 자치구 기반 SVG 지도 UI 가 제공되어야 한다.↵
- 자치구 영역은 Hover 및 선택 시 색상 변화가 적용되어야 한다.↵

1.5.2 선택 연동 기능↵

- 지도 선택, 드롭다운 선택 등 UI 요소 간 동기화가 이루어져야 한다.↵
- 선택된 구의 예측 분석 데이터가 우측 UI 에 업데이트되어야 한다.↵

04 요구사항 정의서/ 분석서

요구사항 분석서(2/2)

1.6.1 목록 조회 기능↵

- 사용자는 게시글 목록을 확인할 수 있어야 한다.↵
- 목록에는 제목, 작성자, 작성일, 댓글 수가 포함되어야 한다.↵

1.6.2 글 작성·수정·삭제 기능↵

- 로그인 사용자는 질문을 작성할 수 있어야 한다.↵
- 게시글 수정 및 삭제는 작성자 본인만 가능해야 한다.↵

1.6.3 댓글 기능↵

- 게시글에는 댓글 작성 기능이 제공되어야 한다.↵

↵

1.7 사용자 인증 기능↵

1.7.1 로그인 기능↵

- 사용자는 ID(또는 이메일)과 비밀번호로 로그인할 수 있어야 한다.↵
- 로그인 실패 시 오류 메시지가 표시되어야 한다.↵

1.7.2 회원가입 기능↵

- 사용자 정보 입력 후 회원가입이 가능해야 한다.↵
- 비밀번호는 암호화(HASH) 처리되어 저장되어야 한다.↵

1.7.3 비밀번호·아이디 찾기 기능↵

- 이메일 또는 인증 방식으로 정보 찾기 기능이 제공되어야 한다.↵

1.8 생성형 AI 기능↵

1.8.1 로그인 기능↵

1.8 생성형 AI 기능↵

1.8.1 로그인 기능↵

- 사용자가 특정 자치구를 선택할 경우, Oracle DB 내의 수요 예측 데이터를 자동으로 추출하여 프롬프트의 컨텍스트로 구성해야 한다.↵
- 파인튜닝된 Llama-3 모델은 제공된 데이터를 분석하여 리포트를 생성해야 한다.↵

1.8.2 맞춤형 정책 제안 기능↵

- 자치구별 복지 특성을 고려하여 실현 가능한 정책 아이디어를 제시해야 한다.↵
- 답변 생성 시 불필요한 서술은 생략하고 출력해야 한다.↵

1.8.3 RAG 기반 Q&A 기능↵

- 사용자 질문 의도를 분석하여 Vector DB 에서 관련 컨텍스트를 검색해야 한다.↵
- 검색된 자료를 바탕으로 답변을 생성함으로써 할루시네이션을 최소화하고 정보의 정확성을 확보해야 한다.↵

1.8.4 텍스트 요약 기능↵

- 사용자가 긴 정책/지침/사용자 입력 텍스트를 입력하여 요약 결과를 확인할 수 있어야 한다..↵
- 요약 기능은 Hugging Face Transformers 기반 요약 파이프라인을 사용하며, 한국어 요약 모델을 적용해야 한다.↵

1.8.5 모델 성능 비교 및 추론 관리 기능↵

- 파인튜닝 과정에 따른 모델 버전(Base Model, 최종학습모델 등)을 선택하여 동일 질문에 대한 답변 차이를 확인할 수 있는 인터페이스를 제공해야 한다.↵

05 화면 설계서

메인화면(1/5)

화면 코드	MA-01	화면 경로	Main	페이지 명	메인 페이지 (캐러셀)	페이지	2

Description	
1	• 캐러셀 • 좌우 화살표 클릭 시 이전·다음 슬라이드로 이동
2	• 하단 인디케이터 • 하단 점(●) 클릭 시 해당 슬라이드로 이동
3	• Snap Scroll • 마우스 휠 또는 스크롤 시 다음 섹션으로 자동 이동
4	• 같은 페이지 내 '서비스 소개(스냅 스크롤-1)' 섹션으로 자동 스크롤

05 화면 설계서

머신러닝 페이지(1/2)

화면 코드	ML-02	화면 경로	Main - ML	페이지 명	머신러닝	페이지	8

Description	
1	• 원하는 연도·자치구 선택 후 조회 시 지도·그래프·표가 동시에 갱신 • 페이지 최초 진입 시 기본값 : 최신 연도, 서울시 전체
2	• 지도에서 자치구 클릭 시 조회 (선택 구 색상 강조)
3	• 조회 조건의 실제·예측 이용자 수 그래프 • 전년·서울 평균 대비 증감 표시 • 마우스 오버 시 연도별 실제/예측 값과 증감률 툴팁 표시
4	• 선택 연도·자치구의 예측 이용자 수와 주요 지표 표

05 화면 설계서

생성형 AI 페이지(1/5)

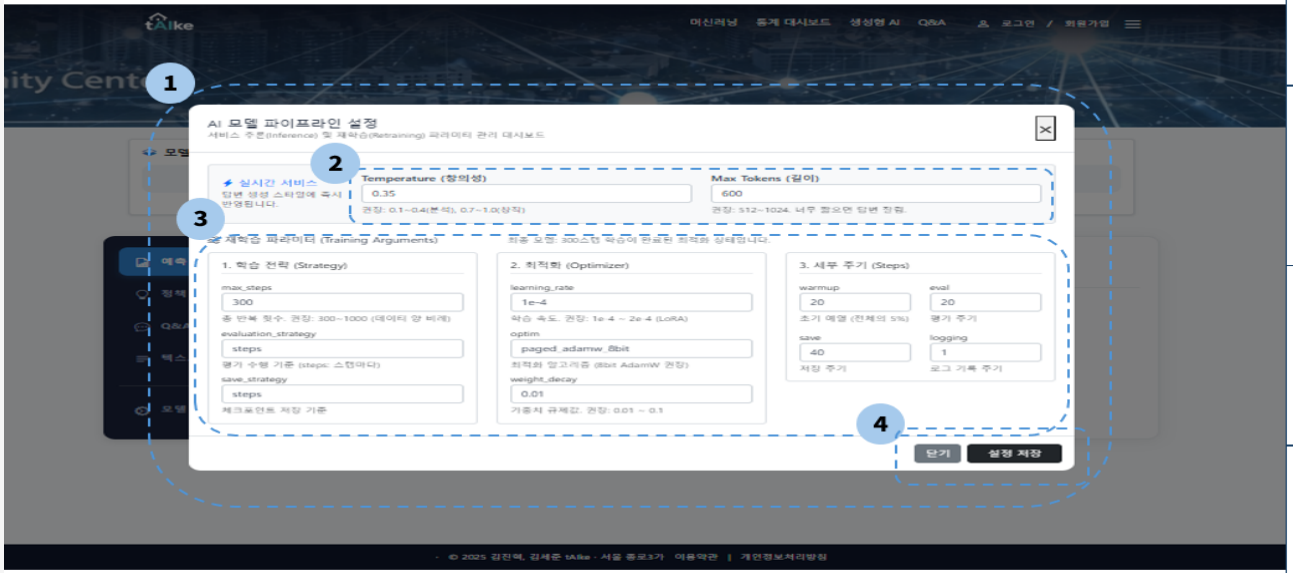
화면 코드	GEN-01	화면 경로	Main – GEN	페이지 명	GEN	페이지	10

	Description
1	• LoRA 체크포인트 선택 영역을 표시한다 • Base/cp100/cp200/최종 모델 버전을 선택할 수 있다. • 모델 선택 시 서버에 모델 교체 요청을 전송하고, 이후 생성 결과에 반영한다
2	• 생성형 AI 기능 메뉴를 표시한다. • 메뉴 클릭 시 해당 기능 화면으로 이동한다.
3	• 보고서 생성에 필요한 자치구/기간 입력 폼을 표시한다. • 사용자가 값을 입력하면 입력값을 기준으로 보고서 생성 요청을 준비한다.
3	• 버튼 클릭 시 입력값을 전송하여 요약 보고서 생성 요청을 수행하고 결과를 화면에 출력한다

05 화면 설계서

생성형 AI 페이지(5/5)

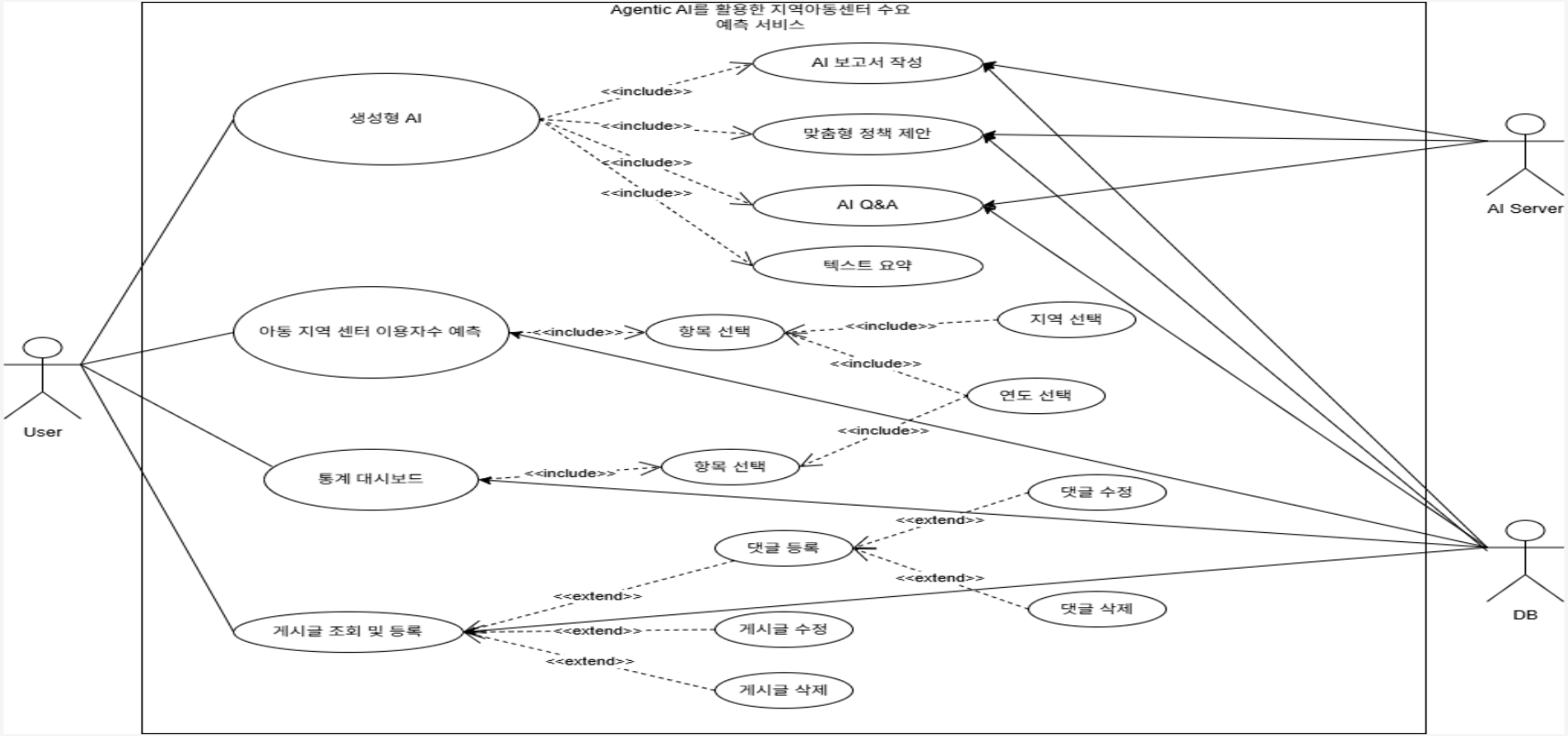
화면 코드	GEN-05	화면 경로	Main – GEN	페이지 명	GEN	페이지	14



Description	
1	• 생성형 AI 기능의 설정 모달을 표시한다. • 모달 내에서 추론/학습 관련 옵션을 확인, 조정할 수 있다.
2	• Temperature, Max Token 입력값을 표시한다. • 사용자가 값을 변경하면 생성 결과의 창의성과, 출력길이에 반영된다.
3	• LoRA 학습에 사용된 Training Arguments(학습 파라미터)를 표시한다. • 사용자가 값을 변경하면 생성 결과의 창의성과, 출력길이에 반영된다.
4	• 닫기 / 설정 적용 버튼을 표시한다. • 설정 적용 클릭 시 변경된 값을 서버로 전송하여 설정을 저장하고, 이후 생성 요청에 적용한다.

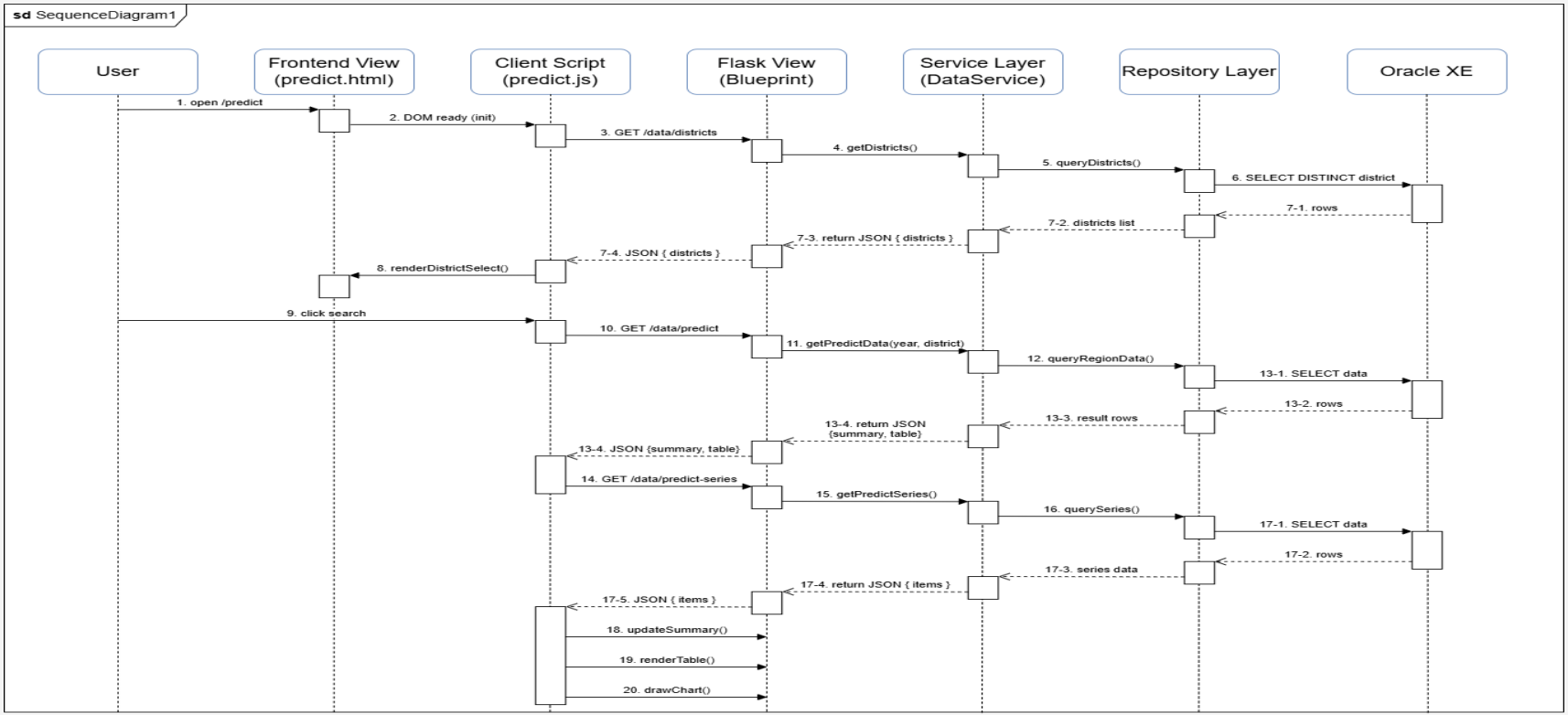
06 UML

Usecase Diagram



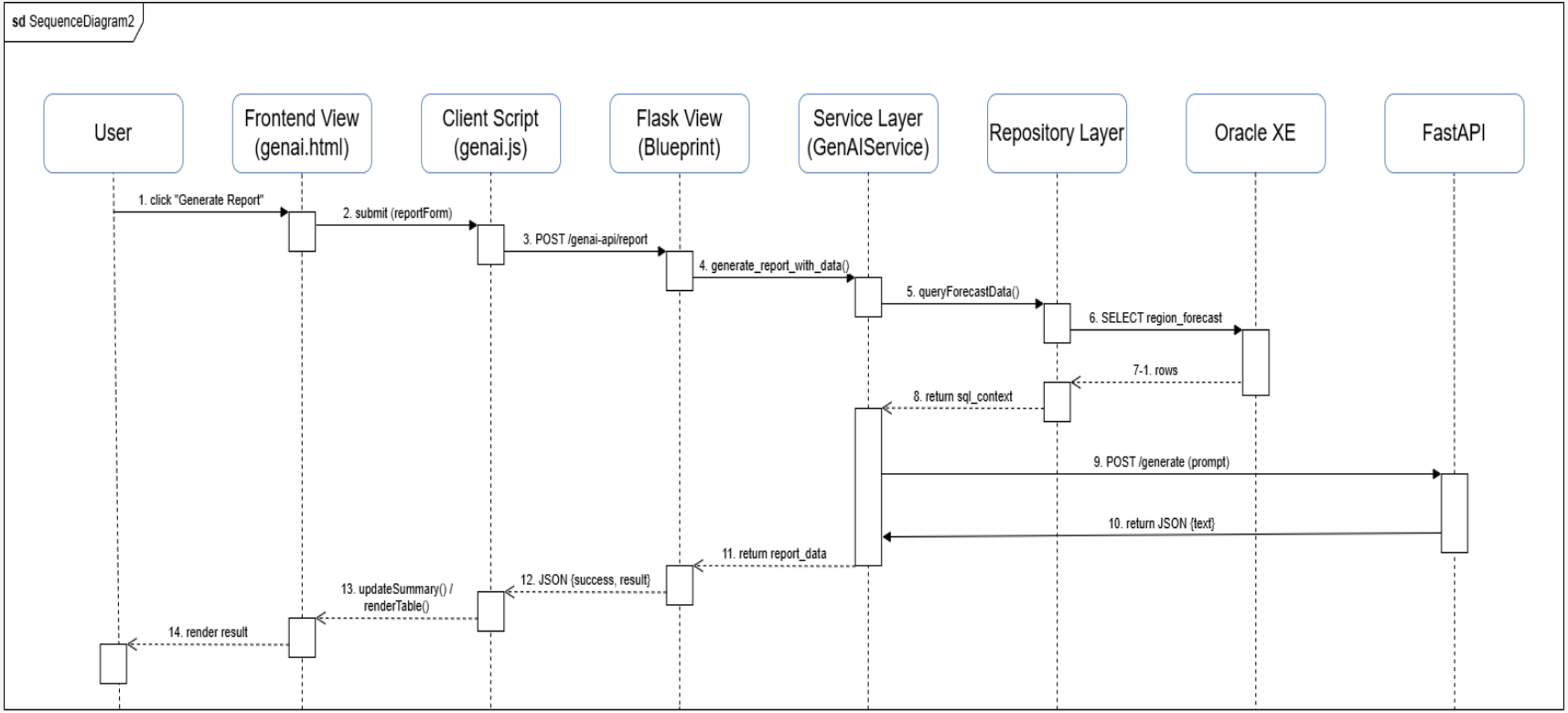
06 UML

Sequence Diagram (1/2)



06 UML

Sequence Diagram (2/2)



07 주요 서비스 기능 및 코드 (1/6)

[train.py]

```
# Quantization Setup
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.bfloat16,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type='nf4'
)
model_id = "meta-llama/Meta-Llama-3-8B"
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    quantization_config=quantization_config,
    device_map={"": 0}
)
```

- Llama-3 8B 모델을 4-bit(NF4) 양자화 + double quant 설정으로 로드해 GPU 메모리 사용량을 크게 줄인 구성이다.
- 연산은 bfloat16으로 수행해 안정성을 유지하고, device_map={"":0}로 GPU 0번에 모델을 배치한다.

```
# LoRA Setup
peft_config = LoraConfig(
    lora_alpha=16,
    lora_dropout=0,
    r=16,
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=["q_proj", "v_proj", "k_proj", "o_proj", "gate_proj", "up_proj", "down_proj"]
)
```

- LoRA(PEFT)로 Llama-3를 전체 재학습 없이 미세조정하기 위해 r=16, alpha=16, dropout=0 설정을 적용하고 bias는 학습하지 않는다.
- q/k/v/o_proj와 gate/up/down/proj 등 Attention-FFN 핵심 projection 레이어만 타깃으로 어댑터를 붙여 파라미터 효율을 높인다

```
# Training Arguments (300 steps)
output_dir = "/workspace/finetune/outputs"
training_arguments = TrainingArguments(
    output_dir=output_dir,
    report_to="none",
    per_device_train_batch_size=2,
    gradient_accumulation_steps=8,
    warmup_steps=50,
    max_steps=300,
    eval_steps=50,
    save_steps=100,
    evaluation_strategy='steps',
    save_strategy='steps',
    learning_rate=1e-4,
    logging_steps=1,
    optim="adamw_8bit",
    weight_decay=0.01,
    lr_scheduler_type="constant_with_warmup",
    seed=42,
    gradient_checkpointing=True,
    gradient_checkpointing_kwargs={'use_reentrant': True}
)
```

- per_device_train_batch_size=2 : GPU 메모리 한계로 마이크로배치 최소화
- gradient_accumulation_steps=8 : 유효 배치=2×8=16으로 안정성 확보
- max_steps=300 : 짧은 실험 주기로 빠른 검증/비교
- warmup_steps=50 : 초반 발산 방지(학습률 점진 상승)
- learning_rate=1e-4 : LoRA 튜닝에서 안정적인 기본 학습률
- optim=adamw_8bit : 옵티마이저 메모리 절감(저사양 GPU 대응)
- weight_decay=0.01 : 과적합 완화(일반화 보조)
- lr_scheduler_type=constant_with_warmup : warmup 이후 일정 LR로 간단/안정
- gradient_checkpointing=True : activation 메모리 절감(속도↓, 메모리↑)
- eval_steps=50 / save_steps=100 : 50 step마다 성능 점검, 100 step마다 저장
- seed=42 / report_to='none' : 재현성 확보 / 불필요 로깅 제거

07 주요 서비스 기능 및 코드 (2/6)

▪1.2 구성 요소 역할

genai_views.py (Flask Blueprint): /genai-api/* 엔드포인트 (report/policy/qa/summarize/switch-model/config)를 제공하며, 요청값 검증 후 서비스 레이어 호출 결과를 JSON으로 반환한다.

genai_service.py (서비스 레이어): 보고서/정책/QA/요약 기능의 핵심 로직을 담당한다. DB 컨텍스트 구성, 프롬프트 생성, RunPod 추론 서버 호출, 응답 가공을 수행한다.

rag_service.py (RAG): jsonl 규칙/수치 데이터를 Document로 변환하여 Chroma에 적재/로드하고, 질문 유형(doc_type)에 따라 필터링된 유사도 검색으로 근거 컨텍스트를 생성한다.

server.py (RunPod FastAPI): Llama-3 8B 4bit 로딩 및 LoRA 체크포인트 로딩을 수행한다. /switch_model로 활성 모델을 교체하고 /generate로 추론 결과를 제공한다.

genai.js (프론트 연동): 모델 비교 라디오 선택 시 Flask의 /genai-api/switch-model을 호출한다. 보고서/정책/QA/요약 요청을 fetch로 전송하고 로딩 스피너/버튼 비활성화 등 UX 처리를 수행한다.

[server.py]

```python
@app.on_event("startup")
async def load_initial_model():
    global tokenizer, base_model, current_ft_model
    print("Loading Base Model and Default Adapter (300 steps)...")

    bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_compute_dtype=torch.bfloat16,
        bnb_4bit_use_double_quant=True,
        bnb_4bit_quant_type="nf4",
    )

    tokenizer = AutoTokenizer.from_pretrained(BASE_MODEL_ID)
    tokenizer.pad_token = tokenizer.eos_token

    base_model = AutoModelForCausalLM.from_pretrained(
        BASE_MODEL_ID,
        quantization_config=bnb_config,
        device_map={"": 0},
    )

    current_ft_model = PeftModel.from_pretrained(base_model, os.path.join(OUTPUT_DIR, "checkpoint-300"))
    print("Default Server Ready (Final Model).")
```

- FastAPI 서버가 시작될 때(startup) 토크나이저를 초기화하고, Llama-3 Base 모델을 4-bit(NF4) 양자화 설정으로 로드해 GPU0에 올린다.
- pad_token을 eos_token으로 맞춰 패딩 토큰 문제를 방지하고, 입력 처리 일관성을 확보한다.
- 마지막으로 checkpoint-300의 LoRA 어댑터(PEFT) 를 base 모델에 결합해 기본 추론 모델(current_ft_model) 로 설정한 뒤 준비 완료 로그를 출력한다.

07 주요 서비스 기능 및 코드 (3/6)

[server.py]

```python
@app.post("/generate")
def generate(req: GenerateRequest):
    global current_ft_model, tokenizer

    device = next(current_ft_model.parameters()).device
    print("model device:", device, "cuda avail:", torch.cuda.is_available())
    print("req.max_new_tokens:", req.max_new_tokens, "req.temperature:", req.temperature)

    current_ft_model.eval()

    prompt = (
        f"Below is an instruction that describes a task. "
        f"Write a response that appropriately completes the request.\n\n"
        f"### Instruction:\n{req.instruction}\n\n"
        f"### Input:\n{req.input}\n\n"
        f"### Response:\n"
    )

    inputs = tokenizer(prompt, return_tensors="pt")
    print("input(before):", inputs["input_ids"].device, "len:", inputs["input_ids"].shape[-1])


    inputs = {k: v.to(device) for k, v in inputs.items()}
    print("input(after):", inputs["input_ids"].device)

    if device.type == "cuda":
        torch.cuda.synchronize()
        t0 = time.time()

    with torch.inference_mode():
        output_tokens = current_ft_model.generate(
            **inputs,
            max_new_tokens=req.max_new_tokens,
            temperature=req.temperature,
            do_sample=(req.temperature > 0),
            repetition_penalty=1.2,
            pad_token_id=tokenizer.eos_token_id,
            eos_token_id=tokenizer.eos_token_id
        )

    if device.type == "cuda":
        torch.cuda.synchronize()
        t1 = time.time()
        new_tokens = output_tokens.shape[-1] - inputs["input_ids"].shape[-1]
        print(f"[perf] gen_time={t1-t0:.2f}s new_tokens={new_tokens} tok/s={new_tokens/(t1-t0+1e-6):.2f}")

    text = tokenizer.decode(
        output_tokens[0][inputs["input_ids"].shape[-1]:],
        skip_special_tokens=True
    ).strip()

    return {"text": text}
```

- 요청으로 받은 instruction/input을 프롬프트로 합친 뒤 토크나이징해서 모델 디바이스(GPU/CPU)로 올리고, generate()로 지정된 max_new_tokens/temperature 설정에 따라 응답 토큰을 생성한다.
- 생성 시간·토큰 처리량을 로그로 찍고, 입력 길이 이후의 생성 부분만 디코딩해 최종 텍스트를 JSON({"text": ...})으로 반환한다.

[rag_service.py]

```python
class RagService:
    @staticmethod
    def _route_doc_type(question: str) -> Optional[str]:
        q = (question or "").lower()
        if any(k in q for k in ["인건비", "급여", "호봉", "수당", "보수", "연봉", "돈", "월급"]):
            return "salary"
        if any(k in q for k in ["지원", "보조금", "운영비", "배치기준", "정원", "시설장", "생활복지사"]):
            return "support"
        if any(k in q for k in ["법", "조문", "시행령", "시행규칙", "아동복지법"]):
            return "law"
        return None

    def get_relevant_context(self, question: str) -> str:
        if not self.vector_db:
            return "참조할 수 있는 운영 지침 데이터가 없습니다."

        doc_type = self._route_doc_type(question)

        # MMR 대신 속도가 빠른 similarity 검색 사용
        search_kwargs: Dict[str, Any] = {"k": 3}  # 검색 결과 개수를 3개로 최적화

        if doc_type:
            search_kwargs["filter"] = {"doc_type": doc_type}

        # search_type을 similarity로 변경하여 연산 속도 향상
        retriever = self.vector_db.as_retriever(
            search_type="similarity",
            search_kwargs=search_kwargs
        )

        # 랭체인 invoke 메서드 사용
        docs = retriever.invoke(question)
```

- 질문 문자열에 포함된 키워드를 기준으로 문서 타입을 급여(salary) / 지원(support) / 법령(law) 으로 분류해, 검색 범위를 좁히는 라우팅 로직이다.
- 벡터DB가 없으면 안내 문구를 반환하고, 있으면 k=3으로 Top-3 유사도(similarity) 검색을 수행한다(MMR 대신 속도 우선).
- 분류된 doc_type이 있으면 해당 타입으로 filter를 걸어 관련 문서만 가져오고, retriever.invoke(question)으로 컨텍스트 문서를 추출한다.

07 주요 서비스 기능 및 코드 (4/6)

[genai_views.py]

```python
# 모델 변경
@bp.route("/switch-model", methods=["POST"])
def switch_model():
    data = request.get_json()
    try:
        # 런포드 서버의 /switch_model 엔드포인트 호출
        runpod_url = os.getenv("RUNPOD_API_URL").replace("/generate", "/switch_model")
        res = requests.post(runpod_url, json=data, timeout=60)
        return jsonify({"success": True, "result": res.json()})
    except Exception as e:
        return jsonify({"success": False, "error": str(e)}), 500
```

- flask의 /switch-model POST API로, 클라이언트가 보낸 JSON을 받아 RunPod 추론 서버의 /switch_model 엔드포인트로 그대로 전달한다.
- RUNPOD_API_URL의 /generate를 /switch_model로 바꿔 호출하고, 응답 JSON을 success/result 형태로 반환한다(타임아웃 60초).
- 호출 실패 시 예외를 잡아 success:false와 에러 메시지를 500으로 반환한다.

[genai_service.py]

```python
class GenAIService:
    def generate_report_with_data(self, user_prompt: str, **kwargs) -> str:
        start_all = time.time()
        print(f"\n[로그 1] 함수 진입 완료: {time.time() - start_all:.4f}s")  #

        meta = self._extract_query_meta(user_prompt)
        meta.district = kwargs.get('district', meta.district)
        meta.end_year = kwargs.get('end_year', meta.end_year)
        meta.start_year = kwargs.get('start_year', 2023)
        print(f"[로그 2] 메타데이터 추출 완료: {time.time() - start_all:.4f}s")  #

        print(f"[로그 3] DB 조회 시작...")  #
        sql_context = self._build_forecast_context(meta)
        print(f"[로그 4] DB 조회 및 가공 완료: {time.time() - start_all:.4f}s")  #

        instruction = (
            "너는 서울시 아동복지 정책 전문가다. 반드시 한국어로 답해.\n"
            "아래 형식을 정확히 지켜. (딱 3줄만, 추가 문장/설명 금지)\n"
            "- 요약: 한 문장\n"
            "- 가능 요인: 한 문장\n"
            "- 추가 데이터: 한 문장\n"
        )
        input_text = f"지역:{meta.district}\n기간:{meta.start_year}-{meta.end_year}\n데이터:\n{sql_context}"

        print(f"[로그 5] 런포드 요청 직전: {time.time() - start_all:.4f}s")  #

        raw_response = self._call_llama3(
            instruction,
            input_text,
            max_tokens=128,
            model_version=kwargs.get('model_version', 'final')
        )
```

- 사용자의 프롬프트를 self._extract_query_meta()로 파싱해 자치구/연도 범위 같은 메타정보를 추출하고, 단계별 처리 시간을 로그로 기록한다.
- 추출된 조건을 기반으로 DB 조회를 수행해 sql_context(예측/통계 등 필요한 데이터 컨텍스트)를 만들고, 이를 LLM 입력에 포함시킨다.
- instruction에 출력 형식(예: 3줄 불릿, 단정 금지, 추가 설명 금지 등)을 강하게 지정해 답변 스타일을 통제한다.
- 최종적으로 self._call_llama3()를 호출해 DB 컨텍스트 + 사용자 질의를 합친 요청으로 Llama3 응답을 생성하고 반환한다.

07 주요 서비스 기능 및 코드 (5/6)

[train.py]

```
# district 원핫 인코딩
df = pd.get_dummies(df, columns=["district"], drop_first=False)

district_ohe_cols = [c for c in df.columns if c.startswith("district_")]
```

- district 범주형 변수를 pd.get_dummies()로 원-핫 인코딩해 district_로 시작하는 더미 컬럼들을 생성하고 목록(district_ohe_cols)으로 저장한다.

[train.py]

```
param_grid_local = {
    "max_depth": [4, 5, 6],
    "learning_rate": [0.03, 0.05, 0.07],
    "n_estimators": [600, 700, 800],
    "subsample": [0.5, 0.6, 0.7],
    "colsample_bytree": [0.5, 0.6, 0.7],
    "gamma": [0.1, 0.3, 0.5],
    "reg_lambda": [1.0, 1.5, 2.0],
    "reg_alpha": [0, 0.1, 0.3]
}

xgb_model_local = XGBRegressor(
    random_state=42,
    tree_method="hist"
)
```

- XGBRegressor를 tree_method="hist"로 빠르게 학습시키면서, 깊이/학습률/트리수·샘플링·정규화 등 주요 하이퍼파라미터의 탐색 범위(param_grid_local)를 정의한 코드다.

[train.py]

```
# Feature 설정
base_features = [
    "year",
    "single_parent",
    "basic_beneficiaries",
    "multicultural_hh",
    "academy_cnt",
    "grdp",
    "population"
]

features = base_features + district_ohe_cols
target = "child_user"

# Train/Test Split
train = df[df["year"] <= 2020]
test  = df[df["year"] >= 2021]

X_train = train[features]
y_train = train[target]

X_test = test[features]
y_test = test[target]

y_train_log=np.log1p(y_train)
```

- 학습 피처(features)와 타깃(child_user)을 정의한 뒤, 2020년까지 train / 2021년 이후 test로 연도 기준 분리하고 X/y를 각각 생성한다.
- 또한 y_train에 log1p를 적용해 스케일을 완화하고 학습 안정성을 높인다.

07 주요 서비스 기능 및 코드 (6/6)

[train.py]

```python
search_local = RandomizedSearchCV(
    estimator=xgb_model_local,
    param_distributions=param_grid_local,
    n_iter=30,
    scoring="r2",
    cv=3,
    verbose=2,
    n_jobs=-1,
    random_state=42
)

search_local.fit(X_train, y_train_log)

best_xgb_local = XGBRegressor(
    **search_local.best_params_,
    random_state=42
)
best_xgb_local.fit(X_train, y_train_log)

best_xgb_local.district_ohe_cols = district_ohe_cols
best_xgb_local.base_features = base_features

pred_local_log = best_xgb_local.predict(X_test)
pred_local = np.expm1(pred_local_log)
```

- RandomizedSearchCV로 XGBoost 하이퍼파라미터를 30회 랜덤 탐색(n_iter=30) 하고, 3-fold CV(cv=3) 에서 R^2(scoring="r2") 기준으로 최적 조합을 찾는다
- 찾은 best_params_로 best_xgb_local 모델을 다시 생성해 전체 학습 데이터(X_train, y_train_log) 로 재학습한다.
- 모델 객체에 district_ohe_cols, base_features를 속성으로 저장해 추론 시 컬럼 스키마를 고정하고, 예측은 log 스케일로 나온 값을 expm1로 원래 스케일로 복원해 pred_local을 만든다.

```python
class DataService:
    def get_predict_series(self, district: str) -> dict:
        else:
            actual_rows = self.region_repo.get_total_series_actual()
            for r in actual_rows:
                if r.child_user is None:
                    continue
                items.append({
                    "year": int(r.year),
                    "child_user": int(r.child_user),
                    "is_pred": False,
                })

            pred_rows = self.region_repo.get_total_series_forecast()
            for r in pred_rows:
                if r.child_user is None:
                    continue
                items.append({
                    "year": int(r.year),
                    "child_user": int(r.child_user),
                    "is_pred": True,
                })
            items.sort(key=lambda x: x["year"]) # 마지막으로 순서 점검 year기준으로 정렬
```

- DB에서 실제값 시계열(get_total_series_actual) 과 예측값 시계열(get_total_series_forecast)을 각각 조회해, year / child_user / is_pred 형태의 리스트로 합친다.
- child_user가 없는 행은 건너뛰고(continue), 실제 데이터는 is_pred=False, 예측 데이터는 is_pred=True로 구분해 프론트(차트)에서 구간을 나눠 그릴 수 있게 한다.
- 마지막에 year 기준으로 정렬해 시간 순서가 보장된 시계열 데이터를 반환한다.

08 머신러닝 결과보고서 (1/4)

Modeling

- 사용 모델: Linear Regression, Random Forest, XGBoost
- 최적화 기법: RandomizedSearchCV를 통한 하이퍼파라미터 튜닝
- 검증 전략: 시계열 특성을 고려하여 2020년 이전 데이터를 학습(Train), 2021~2022년 데이터를 검증(Test)용으로 분리

Data Information

- **데이터출처 : KOSIS(국가통계포털) , 서울열린데이터광장**
- **Target(종속변수): child_user(이용자 수)**
- **Features(설명변수):**
  - 한부모 가정 수(single_parent)
  - 기초생활수급 가구 수(basic_beneficiaries)
  - 다문화 가구 수(multicultural_hh)
  - 학원 수(academy_cnt)
  - GRDP(grdp), 인구(population), 연도(year)
  - 자치구(district): 원-핫 인코딩 적용

※ 지역내총생산(GRDP) : 어떤 기간 동안 어떠한 지역에서 생산된 상품과 서비스의 가치를 시장가격으로 평가한 수치

08 머신러닝 결과보고서 (2/4)

```
csv_path = os.path.join(DATA_DIR, "master_2015_2022.csv")
df = pd.read_csv(csv_path)

# district 원핫 인코딩
df = pd.get_dummies(df, columns=["district"], drop_first=False)

district_ohe_cols = [c for c in df.columns if c.startswith("district_")]
```

- 데이터 로드 + district 원핫

```
features = base_features + district_ohe_cols
target = "child_user"

# Train/Test Split
train = df[df["year"] <= 2020]
test  = df[df["year"] >= 2021]
```

- 자치구는 원-핫 인코딩으로 지역 효과를 반영하고, 연도 기준 분리로 시간 누수를 방지(2015~2020 학습 / 2021~2022 평가).

```
y_train_log=np.log1p(y_train)
```

- 타깃 log1p (분포 안정화)

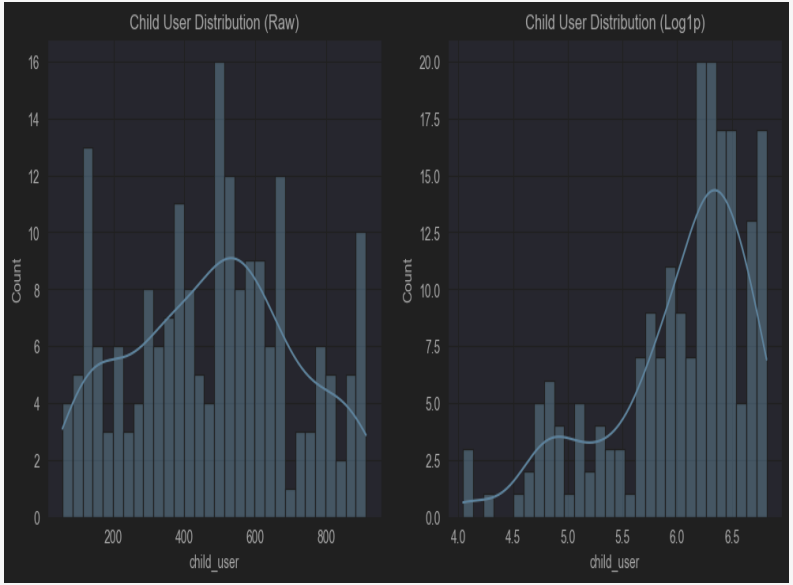
```
search_local = RandomizedSearchCV(
    estimator=xgb_model_local,
    param_distributions=param_grid_local,
    n_iter=30,
    scoring="r2",
    cv=3,
    verbose=2,
    n_jobs=-1,
    random_state=42
)

search_local.fit(X_train, y_train_log)
```

- RandomizedSearchCV 탐색(30회, 3-fold)

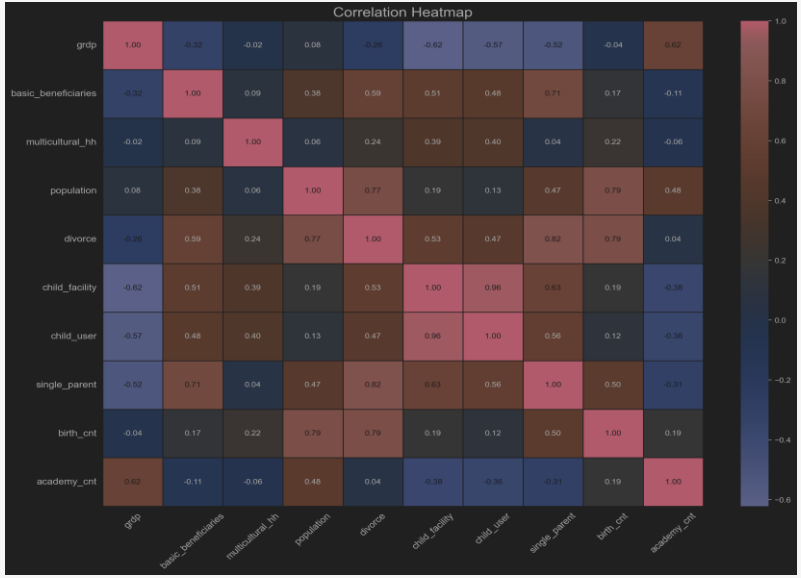
```
best_xgb_local = XGBRegressor(
    **search_local.best_params_,
    random_state=42
)
best_xgb_local.fit(X_train, y_train_log)

best_xgb_local.district_ohe_cols = district_ohe_cols
best_xgb_local.base_features = base_features

pred_local_log = best_xgb_local.predict(X_test)
pred_local = np.expm1(pred_local_log)
```

- XGBoost를 RandomizedSearchCV로 튜닝하고, log 스케일 예측을 expm1로 원복하여 실제 이용자 수 단위로 평가.

08 머신러닝 결과보고서 (3/4)



- 이용자 수 분포가 우측 꼬리 형태 → log1p 변환으로 학습 안정성 확보



- child_user는 취약가구·이혼 지표와 양의 상관, GRDP·학원수와 음의 상관 → 취약도가 높을수록 이용 수요↑, 소득·사교육이 높을수록 이용 수요↓

※ 상관관계는 인과를 의미하지 않음(EDA 기반 경향)

08 머신러닝 결과보고서 (4/4)

모델 선택 요약

- Step1 MLR(기본): R^2 0.56 → 편차 설명 부족
- Step2 XGB(2차): R^2 0.76 → 패턴 개선 확인
- Step3 XGB 고도화: 원-핫+튜닝+log1p → R^2 0.84
- Step4 MLR 재평가: 원-핫만 적용 시 R^2 0.84 →후보 가능

Final

- 최종 채택 기준: 지표보다 예측 현실성
- MLR: 일부 구 비현실 급증
- XGB: 관측 범위 내 완만한 변화
- 보고서/정책 활용 목적상 XGBoost 채택

향후 개선 방향

- **데이터 양(학습 기간) 자체가 부족**

학습 데이터가 2015~2020(6개 연도)로 짧고 검증도 2021~2022(2개 연도)라서, 모델이 패턴을 충분히 학습하기 어려움. 연도 범위 확장(추가 연도 확보) 일반화 성능을 더 정확히 확인할 필요가 있음.

- **피처(설명 변수) 확장 필요**

현재 피처만으로는 '이용자 수'를 설명하는 핵심 요인을 충분히 담지 못할 가능성이 큼. 정책/복지 수요 지표, 지역 인구 구조 변화 등 영향력이 큰 변수를 추가하면 예측 품질이 개선될 여지가 있음.

09 시연

프로젝트 시연

https://drive.google.com/file/d/1DzBx8PQqxOeYAgGnsNaFRTk2noertKaA/view

10 향후 개발 계획

파인튜닝 품질 고도화

현재 LoRA SFT(300 steps)로 체크포인트(cp100/cp200/final) 비교 시연까지 구현했습니다.
다음 단계는 데이터 추가보다 학습 전략을 튜닝해 "말투 변화"가 아니라 지시 이행·답변 품질이 실제로 개선되는 구간을 찾는 방향으로 진행합니다.

학습 설정 튜닝

학습 길이를 300 → 600~1000 steps로 확장하고 저장 전략을 조정해 비교 범위를 넓힙니다.
또한 learning_rate(5e-5~2e-4), scheduler(constant_warmup/cosine), LoRA(r/alpha/dropout) 조합을 실험하여 발산 없이 가장 깔끔한 출력이 나오는 설정을 최적화합니다.

추론 설정 표준화

시연/서비스에서 답변이 흔들리지 않도록 temperature, max_new_tokens, repetition_penalty 기준값을 확정하고,
보고서·정책·QA 기능별로 권장 프리셋을 정리해 일관된 품질을 유지할 계획입니다.

11 프로젝트 수행 소감

소통의 중요성

팀 프로젝트와 개인 프로젝트를 연속으로 경험하면서, 소통의 중요성도 크게 느꼈습니다. 초반에는 필요한 얘기만 하고 역할을 나눠 진행했는데, 마지막에 합치는 단계에서 소통이 부족했던 부분이 한꺼번에 드러나 문제가 많이 생겼습니다. 반대로 주제를 바꾼 이후에는 설계와 기준을 더 명확히 맞추고 진행하면서 효율과 속도가 확실히 좋아졌습니다.

기능 개발 과정과 어려움과 배움

프로젝트를 하기 전에는 머신러닝이나 LLM 흐름을 제대로 이해하지 못한 상태에서 시작했기 때문에 시행착오가 컸고, 그 과정에서 시간 낭비도 많았습니다. 하지만 그만큼 프로젝트를 하면서 계속 찾아보고 직접 부딪히다 보니, 이전에는 모르던 개념들이 연결되면서 이해되는 순간이 많았습니다. 아직 모르는 게 많다고 생각하지만, 이번 경험을 통해 내가 약한 부분이 어디 인지가 분명해졌고, 앞으로 설계를 먼저 확실히 잡고 진행해서 같은 실수를 줄여가고 싶습니다.