팀 프로젝트 최종 보고서

헬스케어 혁신을 위한
경구약제 이미지 인식 및 다중 객체 탐지 시스템 개발

AI 모델링부터 모바일 서비스 구현까지, End-to-End 프로토타입 구축기

전재완 · 김효진 · 이태훈 · 황인홍

1. 프로젝트 개요 및 목표

1.1. 배경 및 필요성

현대 헬스케어 산업에서 환자의 안전한 의약품 복용은 오투약 방지 및 환자 안전 관리에 직결되는 핵심 요소입니다. 특히 다제복용 환자나 노령층의 경우 다수의 알약을 동시에 처방받는 경우가 많아, 복용 전 직관적이고 정확하게 알약을 식별할 수 있는 시스템의 필요성이 대두되고 있습니다. 본 프로젝트는 이러한 실제 사용자의 불편함을 해결하는 실천적 솔루션을 도출하는 것을 목표로 합니다.

1.2. 프로젝트 핵심 컨셉

본 프로젝트의 핵심 컨셉은 사진 촬영을 통해 이미지 속에 존재하는 최대 4개의 알약 이름(클래스)과 위치(바운딩 박스)를 동시에 검출(Detection & Classification)할 수 있는 고성능 알고리즘 모델을 구현하는 것입니다. 나아가 이를 실제 활용 가능한 모바일 소프트웨어(App) 인터페이스와 연동함으로써, 제한된 교육 과정 안에서도 AI 모델링부터 서비스 구체화까지 아우르는 헬스케어 서비스의 End-to-End 프로토타입을 구축하고자 했습니다.

1.3. 프로젝트 핵심 목표 및 지향 가치

본 부트캠프 과정의 프로젝트 가이드라인이 제시하듯, 우리 팀은 단순히 기술적으로 화려한 단기적 결과물을 내는 것보다 과정 속에서 팀원 개개인이 폭발적으로 성장하고 진정성 있는 협업 프로세스를 구축하는 것을 최우선 가치로 삼았습니다.

프로젝트 초반에는 집단지성을 발현하고자 데이터 분석(EDA)부터 초기 모델링 단계까지 팀원 전원이 전반적인 과정을 함께 고민하며 지식을 공유하는 방식으로 출발했습니다. 그러나 이 과정에서 마주한 기술적 병목 현상과 일정 관리의 한계를 혼자 끌어안지 않고, 강사님과 멘토님께 적극적으로 질문하며 조언을 구했습니다. 그 결과 '프로젝트의 전체적인 숲을 다 함께 본 뒤에는 역할을 명확히 나누어야 한다'는 멘토링 피드백을 수용하여, 이후 단계부터는 파트별로 업무를 체계적으로 분할해 수행 효율성을 극대화하는 방향으로 협업 구조를 유연하게 안착시켰습니다.

특히 업무가 분할된 이후에도 지식의 고립이나 독점을 막기 위해 '하루 2회 정기 커뮤니케이션' 시스템을 도입했습니다. 매일 두 번씩 진행되는 미팅을 통해 자신이 담당하지 않은 파트(데이터 정제, 모델 고도화, API 및 앱 개발 등)의 진행 상황과 핵심 기술 노하우를 실시간으로 공유받을 수 있었습니다. 이러한 긴밀한 소통 덕분에 각자의 업무 효율성을 챙기는 동시에, 동료의 작업물로부터 함께 배우고 성장하는 경험을 할 수 있었습니다.

2. 팀 구성 및 역할 분담

제한된 일정 속에서 집단지성의 가치를 극대화하고 프로젝트의 성공 가능성을 높이기 위해, 우리 팀은 타임라인의 흐름에 따라 협업 구조를 유연하게 발전시켜 왔습니다.

2.1. 1 주차 — 전원 공동 학습으로 기반 다지기

프로젝트의 첫 단추를 꿰었던 1주차에는 팀원 전원이 프로젝트의 전체적인 숲을 함께 보고 기술적 배경지식을 동기화하는 데 집중했습니다. 5월 19일부터 20일까지는 팀원 전체가 참여하여 원천 데이터셋의 특징을 파악하기 위한 기초 탐색적 데이터 분석(EDA)을 공동으로 수행하였으며, 이를 통해 데이터의 구조와 클래스 불균형 문제를 함께 진단했습니다. 이와 동시에 김효진 팀원이 모바일 소프트웨어 개발을 위한 초기 환경 조사와 구조 기획에 착수하며 서비스의 밑그림을 그리기 시작했습니다.

이어지는 21일에는 전재완 팀원이 중심이 되어 Faster R-CNN 기반의 초반 베이스라인 모델을 구축하고 학습 파이프라인을 수립하였으며, 이태훈 팀원은 데이터 분석 과정을 더욱 세부적으로 고도화했습니다. 황인홍 팀원은 향후 개발 환경 이관 및 배포 효율성을 검토하기 위해 Docker 인프라 도입 타당성을 확인했습니다. 이후 팀원 전체가 구축된 베이스라인 모델을 각자의 환경에서 직접 구동해 보며 전체적인 End-to-End 개발 프로세스를 상호 학습하고, 발생 가능한 기술적 병목 구간을 함께 점검하는 시간을 가졌습니다.

2.2. 2 주차 — 분업 체제로 효율 극대화

이처럼 초반에 모든 과정을 공유하며 기반을 다진 후, 강사님과 멘토님의 조언을 수용하여 2주차부터는 명확한 역할 분담을 통해 업무 수행의 효율성을 극대화하는 분업 체제로 전환했습니

다. 본격적인 고도화가 시작된 5월 26일, 황인홍 팀원은 모델의 강건성을 확보하기 위해 다양한 추가 데이터셋을 조사·확보하는 데 집중하였고, 이태훈 팀원은 어노테이션 오류 등 결측치·이상치 현황을 전수 조사했습니다.

27일에는 파트별 세부 고도화 작업이 한층 더 유기적으로 맞물려 진행되었습니다. 황인홍 팀원은 데이터의 BBox 오류를 검출하여 데이터를 정제하였고, 김효진 팀원은 API 표준 인터페이스를 정의하고 모델을 앱에 붙이는 과정을 실험했습니다. 이태훈 팀원은 결측치·이상치 검출 로직을 코드로 구현하여 직접 확인했으며, 전재완 팀원은 이 모든 협업 과정과 기술적 도전 과제들을 취합하여 중간 보고서 초안을 작성하고 팀의 방향성을 최종적으로 정렬했습니다.

업무는 이처럼 효율적으로 분할되었으나, 매일 오전과 오후 두 차례씩 정기 커뮤니케이션을 진행하여 자신이 담당하지 않은 파트의 기술적 노하우와 진행 상황을 실시간으로 교환했습니다. 그 덕분에 팀원 모두가 독립된 영역에 고립되지 않고 서로의 작업물로부터 배움을 얻으며 동반 성장하는, 진정한 집단지성을 실현해 나갈 수 있었습니다.

3. 데이터 탐색적 분석 (EDA) 및 정제 과정

3.1.원천 데이터셋 구조 분석 (Train Set 232 장 기준)

- **총 파일 수**: 232 개 (70 도: 72 장 / 75 도: 75 장 / 90 도: 85 장으로 균등하게 분포하였습니다)
- **고유 알약 클래스(종류)**: 총 56 종류 (전체 검출 알약 객체 수: 763~771 개)를 확인하였습니다.
- **파일명 역공학을 통한 세팅 규칙 분석:** K-알약조합코드_촬영환경변수_카메라위도_경도_배율.png 형태로 조합되어 있으며, 동일 조합이라도 알약의 배치를 흐트러뜨리거나 촬영 각도를 바꾸어 데이터 증강(Augmentation)이 되어 있음을 규명하였습니다.

3.2. 클래스 불균형 (Data Imbalance) 진단

특정 알약 클래스의 등장 빈도가 극단적으로 치우치는 현상이 관찰되었습니다.

- **최다 빈도** 클래스: 일양하이트린정 2mg (153 개, 약 20.1%)
- **최소 빈도** 클래스: 에스원엠프정 20mg, 레일라정 등 (각 3 개, 약 0.4%)
- 대응 방향: 학습 데이터가 심각하게 부족한 하위 47 개 독점 클래스 및 희귀 의약품에 대해 데이터 증강 기법(Crop, Cutmix 등) 도입이 필수라고 판단하였습니다..

3.3. 데이터 오류 및 결측치(Missing Value) 정제

• **BBox 누락 발견**: 실제 이미지상에는 알약이 존재하나, 9 장의 파일에서 BBox 어노테이션이 누락되거나 오기 표기된 오류를 발견하였습니다 (예: K-003351 계열 누락 등)
• **BBOX 겹침 발생** : 데이터 중 알약에 2 개의 BBOX 가 겹쳐져 있는 오류를 발견하였습니다.
-> 데이터 결측치를 제거하였습니다.

3.4 train 데이터셋과 test 데이터셋의 공통점

train 데이터셋 <총 232 개 파일> test 데이터셋 <파일 넘버는 1500 까지. TEST SET 862 개 파일>
• 공통점 투명한 형태의 OMACOR 파일 발견하였습니다 (EX : 1455 번 파일)
• 공통점 기호 형태가 적힌 알약 발견하였습니다(EX : 819 번 파일)
• 공통점 기호 형태가 적힌 알약 발견하였습니다 (EX : 819 번 파일)

3.5 train 데이터셋과 test 데이터셋의 차이점

• 차이점 test 파일 732 와 같이 알약의 형태가 파인 형태는 없었습니다, test 파일 1136 번 30 이라고 적힌 알약처럼 여러 점 형태가 찍혀있는 특이한 형태의 알약은 없었습니다.
• 차이점 오메가 3 같은 알약이 OMACOR 말고도 아예 아무 글자나 표시도 없는 알약이 있었으며, HN-C 라는 알약도 오메가 3 같은 형태였습니다.
• 차이점 파인 형태의 알약 발견하였습니다. (EX : 732 번 파일), 여러 점형태가 찍혀있는 알약을 발견하였습니다 (EX :1136 번 파일)

3.6 모든 데이터 취합 EDA (결측치 및 이상치 제거)

• 구조적 결측치 명시화 세부 사항, 공백 문자열("") => 정상_인쇄없음 등으로 치환 완료하였습니다.
• 성상 텍스트 후방 공백 예외 제거 : chart 데이터 길이 53 자 => 38 자로 노이즈 제거 완수하였습니다.
• 수치형 스케일 오류 교정 실적 : 5.0 미만 소수점 이상치 데이터 정상 mm 스케일 정밀 변환 자동 완수하였습니다.
• 형태적 날짜 이상치 포맷 통일 : EX) 2012-01-19 규격의 표준 하이픈 구조로 일제 일치하였습니다.

3.7 스무스한 데이터 EDA (2 차 전처리) (결측치 및 이상치 제거)

• 1 차 전처리때 너무 많은 결측치와 이상치가 제거 되었기에 스무스한 2 차 전처리 작업에 들어갔습니다.1 차 때는 전처리 기준이 너무 높아서 세세한 것까지 결측치 이상치 기준에 포함시켰기 때문에 결측치가 과하게 제거 되어 오히려 결과가 안좋을 수 있다는 판단을 하였습니다. 그리고 2 차 때는 스무스한 기준을 적용해서 보다 완화된 관점으로 결측치 및 이상치를 제거하여 EDA 를 진행하고자 했습니다.
• 결측 라벨(Missing Labels) 오류, 빈 이미지(Empty images) 0 장 확인하였습니다.
• 잘못된 클래스 매핑, 규격을 벗어난 극단적 크기 0 장 확인하였습니다.
• 비정상적 종횡비 0 장 확인하였습니다.

그렇게 다양한 결측치 이상치 기준을 적용한 파이프라인 코드를 통해 여러 가지를 확인함으로서 돌다리도 두드리고 간다는 관점으로 여러 문제점들을 체크함으로서 데이터가 최대한 문제가 없다는 것을 확인하시는 것이 중요하다고 생각했습니다.
(데이터셋이 추가되고 전처리 기준이 변화되면 다른 결과가 나올 수 있습니다)

<3.6 모든 데이터 취합 EDA (결측치 및 이상치 제거)의 데이터 EDA 팀원들의 조사 정보 취합 및 결측치 이상치 제거 후 구글 검색(내장 AI)을 통한 코랩 버전 파이프라인 코드 추출 결과>

1.분석 및 정제 대상 알약 메타데이터 세트

-> 초기 원본 데이터 구조 확보 완료. 데이터 크기: (232, 14)

2.구조적 결측치(공백 문자열) 스크리닝 및 'None/정상_인쇄없음' 상수 대체 진행

변수별 공백 문자열("") 결측치 발견 수량 ---

- 컬럼 [di_company_mf]: 결측치 22 개 탐지됨

- 컬럼 [di_company_mf_en]: 결측치 22 개 탐지됨

- 컬럼 [print_back]: 결측치 58 개 탐지됨

- 컬럼 [color_class2]: 결측치 116 개 탐지됨

- 컬럼 [line_front]: 결측치 78 개 탐지됨

- 컬럼 [line_back]: 결측치 78 개 탐지됨

- 컬럼 [mark_code_front]: 결측치 116 개 탐지됨

-구조적 결측치 도메인 상수 대체 프로세스 완료.

(문제는 1 차 전처리는 결측치가 너무 많이 탐지된다는 부분입니다 그래서 필요한 데이터도 삭제 될 위험이 높았습니다)

3 대 데이터 이상치(자연어 공백, 스케일 불일치, 날짜 포맷) 정밀 교정 가동

-[이상치 1 교정] 성상 정보(chart) 데이터 우측 대량 공백 제거 정제 완료.

-[이상치 2 교정] 단위 유실 수치형 이상치 4 건 감지 및 mm 단위로 스케일 통합 완료.

-[이상치 3 교정] 날짜 형식 불일치 데이터 116 건 통합 가동 및 포맷 정규화 완료.

1. 구조적 결측치 명시화 세부 사항 : 공백 문자열("") ➔ 정상_인쇄없음 등으로 치환 완료

2. 성상 텍스트 후방 공백 예외 제거 : chart 데이터 길이 53 자 ➔ 38 자로 노이즈 제거 완수

3. 수치형 스케일 오류 교정 실적 : 5.0 미만 소수점 이상치 데이터 정상 mm 스케일 정밀 변환 자동 완수

4. 형태적 날짜 이상치 포맷 통일 : 2012-01-19 규격의 표준 하이픈 구조로 일제 일치

3.7 의 스무스한 데이터 EDA 기준 (2 차 전처리) (결측치 및 이상치 제거)

1.EDA 조사를 통한 누락 파일 포함

2.저화질 및 블러(Blur)

3.극단적 조명 및 조도 불균형

4.알약 잘림(Clipping)

5.이물질 흔입

6.결측 라벨(Missing Labels)

7.빈 이미지(Empty images)

8.잘못된 클래스 매핑

9.규격을 벗어난 극단적 크기.

10.비정상적 종횡비

11.과도한 알약 겹침(이후에 1 장이 에러로 추가로 발견)

11 번의 경우 알약 겹침 사례가 추가로 1 개 이상 나왔는데 코드로만 돌렸을 때 모든 것을 확인하기가 쉽지 않은 상황에서 추가적인 에러가 나올 가능성을 생각해볼 수 있었습니다.

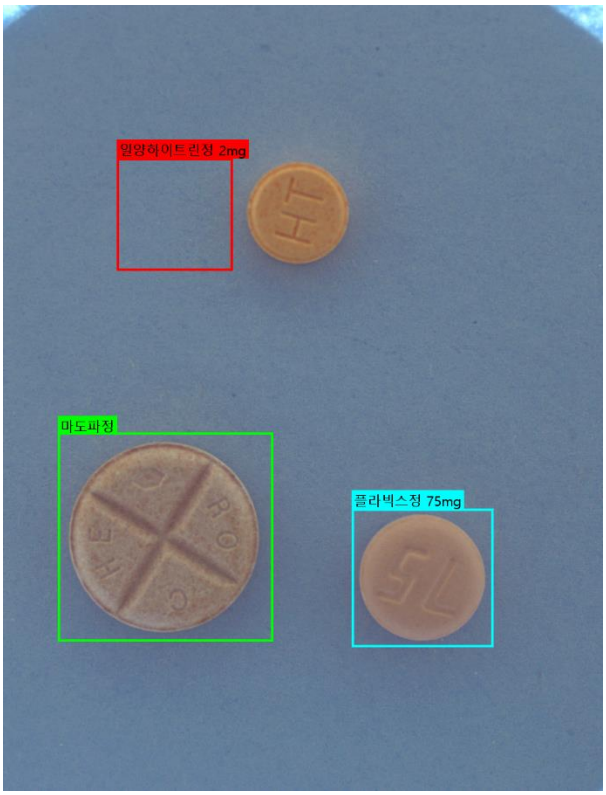
3.8. 데이터 전처리 자동화 파이프라인

기본적으로 train_images의 파일명과 train_annotations의 BBox 좌표, category_id를 활용하여, 시각화 없이 결측치를 찾아내는 코드를 구현해 잘못된 데이터를 1차적으로 전처리했습니다. 크게 네 가지 방향에서 자동화를 진행했으며, 이를 통해 train_images에서 12건의 결측치를 찾아냈습니다.

- Annotation 의 개수가 파일명에 명시된 알약 개수와 일치하지 않는 경우 (8 건)
- BBox 좌표가 이미지 크기를 넘어서는 경우 (1 건)
- BBox 좌표가 다른 BBox 좌표와 겹치는 경우 (3 건)

추가적으로 train_annotations와 train_images를 합성해 이미지화하여 전수 조사를 진행했습니다. train_images의 개수가 232개에 불과해 결측치 하나가 학습에 적지 않은 영향을 끼칠 수 있고, BBox가 이미지 안에 존재하지만 알약과 동떨어진 곳에 위치한 경우는 위의 코드만으로 잡아내기가 불가능하기 때문입니다. 이 과정을 통해 결측치를 1개 더 찾아내, 총 13개의 결측치를 확보할 수 있었습니다.

이후 약 4,000장의 추가 확보 데이터에 대해서도 같은 프로세스를 거쳐 200여 장의 결측치를 찾아냈습니다. 데이터가 충분히 확보됐다고 판단해, 결측치를 수정하여 재사용하지 않고 제거하는 방향으로 결정했습니다.



(예시: 육안 확인을 통해 찾아낸 추가 결측치 — K-003351-020014-020238_0_2_0_2_90_000_200)

4. 모델링 및 체계적 실험 (과정 및 결과)

모델링 단계는 단순한 성능 경쟁이 아니라, 매 단계의 가설과 검증을 누적해 나가는 체계적인 실험의 연속이었습니다. 베이스라인 구축 과정의 시행착오부터 아키텍처 전환, 후처리 고도화, 외부 데이터 수혈, 그리고 최종 3중 앙상블에 이르기까지의 여정을 순서대로 정리합니다.

4.1. 베이스라인 스코어 0 점 기록 단계

현상 및 추이: Faster R-CNN 모델로 설계한 초기 베이스라인의 첫 채점 스코어가 0점으로 기록되었습니다.

원인 파악: 로컬 개발 환경에서 연산 편의성을 확보하기 위해 클래스 인덱스를 0번, 1번, 2번 형태로 라벨 인코딩하여 재매핑한 것이 화근이었습니다. 실제 평가 서버는 임의의 인덱스가 아닌 데이터 고유의 고유식별 ID 숫자를 기준으로 채점을 수행하기 때문에, 매핑된 임의 값을 인식하지 못하고 전량 오답으로 필터링했던 것입니다.

핵심 인사이트: 아무리 딥러닝 모델의 구조적 성능이 뛰어나더라도, 입력단부터 최종 추론 출력단까지 데이터 고유의 식별 체계가 일치하지 않으면 결과물은 무용지물이 됩니다. 따라서 학습 파이프라인을 구축할 때는 초기 단계부터 최종 단계까지 데이터의 정렬 상태를 End-to-End로 완벽하게 검증·추적하는 방어적 코드 설계가 필수적입니다.

4.2. Faster R-CNN 기반 하이퍼파라미터 최적화 단계

현상 및 추이: 초기 파이프라인 수립 후 하이퍼파라미터 제어를 통해 Public Score를 0.63361점부터 최고 0.83284점까지 견인하며 모델의 한계 성능을 시험했습니다.

세부 실험 내역 및 결과 추이

실험	조정 내용	Public Score 추이
2-1 (Base)	Epoch 60, Batch 8, LR 0.005 (기본 구동)	0.65597
2-2 (배치 감소)	Epoch 60→30, Batch 8→4	0.65597 → 0.74304 (상승)
2-3 (학습률 하향)	2-2 상태에서 LR 0.005→0.001	0.74304 → 0.63361 (정체)
2-4 (규제+스케줄러)	Batch 8→16, LR 0.005→0.02, Epoch 30 고정, 20에폭 스케줄러 + 그래디언트 클리핑	0.65597 → 0.83284 (최고점)

실험 방향에 따른 원인 분석

첫째, 배치 사이즈와 학습률의 적절한 균형점을 찾는 방향으로 실험을 전개했습니다. 실험 2-2와 2-3을 통해 무작위로 파라미터를 낮추는 것은 성능 정체를 유발한다는 점을 확인했습니다. 이에 실험 2-4에서는 배치 사이즈를 16으로 키우는 동시에 학습률도 0.02로 함께 높여, 대량의 데이터를 모델이 안정적으로 소화하며 학습할 수 있도록 유도했습니다.

둘째, 제한된 학습 시간 안에서 점수를 최대한 쥐어짜내는 방향으로 스케줄러를 도입했습니다. 총 학습을 30에폭으로 제한했기 때문에, 초기 20에폭까지는 큰 학습률(0.02)로 정답을 향해 빠르게 학습하게 하고, 이후 남은 10에폭 동안은 학습률을 1/10 수준(0.002)으로 떨어뜨려 미세한 특징까지 꼼꼼하게 다듬도록 조율한 것이 성능 향상에 주효했습니다.

셋째, 초기 학습 과정에서 발생할 수 있는 모델 붕괴를 막는 안전장치를 탐색했습니다. 알약의 위치 좌표와 클래스를 동시에 맞춰야 하는 태스크 특성상 초반에 오차가 크게 튀는 리스크가 있었기에, 가중치 업데이트가 일정 수준(1.0)을 넘지 못하도록 강제하는 그래디언트 클리핑을 적용하여 실험 전반의 안정성을 크게 높였습니다.

핵심 인사이트: 최적화 단계에서 무조건적인 에폭 증가나 단일 파라미터 튜닝은 오히려 과소적합이나 학습 정체를 유발합니다. 배치 사이즈와 학습률의 비례 관계를 고려하는 선형 스케일링 법칙을 준수하고, 안정적 수렴을 유도하는 스케줄러와 그래디언트 클리핑 등의 안전장치를 조화롭게 설계해야만 모델 본래의 성능 한계치까지 점수를 효율적으로 끌어올릴 수 있습니다.

4.3. 아키텍처 전환 단계 (Faster R-CNN 83% → YOLO11 95%)

현상 및 추이: 기존 2-스테이지 기반 모델(Faster R-CNN) 체제에서는 하이퍼파라미터를 아무리 정밀하게 튜닝해도 Public Score가 83%에서 성능 한계에 봉착했습니다. 이를 극복하기 위해 1-스테이지 기반 최신 아키텍처인 YOLO11 모델로 전환한 결과, 스코어가 95%로 유의미하게 향상되었습니다.

1 단계: Faster R-CNN 체제의 명확한 성능 한계와 원인

고정된 생각의 틀(Anchor Box)이 가진 구조적 한계입니다. Faster R-CNN은 미리 규격화해 둔 표준 네모 틀을 이미지에 대보며 물체를 찾는 방식을 씁니다. 하지만 우리가 다루는 알약은 종류에 따라 세로로 서 있거나 눕기도 하고 크기가 제각각이어서, 이 뻣뻣한 표준 틀로는 알약 테두리를 정밀하게 밀착해 잡지 못하고 오차가 계속 누적되는 한계가 있었습니다.

2 단계: YOLO 패러다임으로의 전격 전환

앞선 한계를 뼈저리게 느끼고, 이미지 전체를 한 번에 훑으며 훨씬 유연하게 물체를 찾아내는 YOLO 계열로의 전환을 결정했습니다. YOLO 아키텍처는 인위적인 틀을 만들지 않고 알약의 중심점을 곧바로 예측하는 '틀 없는 방식(Anchor-Free)'을 취하므로 불규칙한 알약 배치도 정확히 잡아낼 수 있었습니다. 또한 단순 테두리 끝점이 아니라 알약이 차지하는 '전체 면적과 겹침 비율'을 통틀어 계산하는 현대적 채점 방식(CIoU, DFL 등)을 지원하기 때문에, 밀집 구역에서의 탐지 성능을 근본적으로 개선할 수 있는 열쇠라고 판단했습니다.

3 단계: 구형 버전을 배제하고 'YOLO11'을 선택한 기술적 근거

YOLO를 도입하더라도 v7, v8, v9처럼 이미 지나간 과도기적 구형 버전을 쓰는 것은 의미가 없다고 보았습니다. 구형 버전들의 한계까지 완전히 극복하고 나온 최신 'YOLO11'만이 본 프로젝트를 구원할 최종 무기라고 확신한 이유는 다음과 같습니다.

첫째, 새로운 신경망 블록(C3k2) 도입을 통한 '뭉뚱그림 현상' 해결입니다. 구형 YOLO 모델들은 알약이 너무 붙어 있으면 하나의 거대한 덩어리로 뭉뚱그려 인식하는 고질적 문제가 있었습니다. 반면 YOLO11은 신경망 뼈대를 최신형 'C3k2 블록'으로 설계하여, 연산량 부담은 대폭 낮추면서도 알약과 알약 사이의 미세한 경계선 및 곡선 특징을 칼로 자른 듯 정밀하게 분리해낼 수 있었습니다.

둘째, 향후 확장성까지 고려한 완성형 다중 작업(Multi-task) 헤드 구조입니다. 이번 프로젝트는 단순히 알약 위치만 찾고 끝나는 것이 아니라, 장기적으로 알약 표면의 미세한 각인(글자·로고)을 식별하고 영역을 정밀하게 분할·분류하는 단계까지 확장되어야 합니다. v9 등은 단일 탐지 기법에 치중되어 있었으나, YOLO11은 위치 탐지와 영역 분할을 담당하는 두 신경망이 하나의 핵심 정보를 실시간으로 공유합니다. 따라서 여러 모델을 무겁게 따로 돌릴 필요 없이 하나의 가벼운 모델로 향후 확장 작업까지 매끄럽게 선행 처리할 수 있는 구조적 장점이 있었습니다.

핵심 인사이트: 하이퍼파라미터 숫자 몇 개를 고치는 것에 앞서, 우리가 다루는 도메인 데이터셋이 가진 객체의 크기·분포·형태적 제약 조건을 먼저 면밀히 분석해야 한다는 중요한 사실을 깨달았습니다. 알약처럼 형태가 일정 범위 안에서 정형화되어 있고 배경 대비가 뚜렷한 태스크에는, 연산이 무거운 기존 방식보다 이미지 전체를 빠르게 훑으며 전역 특징을 파악하는 1-스테이지 계열의 현대적 아키텍처를 영리하게 선택하는 것이 실험 효율을 극대화하는 길임을 배웠습니다.

4.4. 제한된 자원 환경을 고려한 모델 체급(YOLO11m) 선정

이론적으로는 가장 체급이 큰 YOLO11x 모델이 최고의 성능을 낼 것으로 기대했으나, 실제 우리의 제한된 하드웨어 및 시간 환경을 고려했을 때는 다른 접근이 필요했습니다.

초대형 체급인 X 모델은 신경망 용량이 거대하여 완전히 수렴하고 제 성능을 내기까지 물리적으

로 훨씬 더 많은 학습량(Epoch)과 연산 시간을 요구합니다. 정해진 기한 내에 X 모델을 불완전하게 학습시키는 것보다, 우리 환경에서 빠르게 정답 근처까지 수렴할 수 있는 중간 체급 YOLO11m 모델을 선택하는 것이 자원 대비 실험 효율성을 극대화하는 영리한 전략이라고 판단했습니다.

4.5. 임계값 조절 및 테스트 단계 데이터 증강(TTA)

현상 및 추이: 아키텍처 전환으로 확보한 기본 성능 95% 상태에서, 모델을 재학습시키지 않고 후처리 설정 변경과 추론 기법 고도화만으로 Public Score를 각각 95.2%와 96.5%로 단계별 돌파했습니다.

1 단계: Confidence 임계값 미세 조정 (95% → 95.2%)

모델이 알약을 찾아낸 뒤 '이것이 알약이 맞다'고 확신하는 기준선인 Confidence 점수를 0.2% 수준으로 미세하게 낮추어 적용했습니다. 알약 데이터의 특성상 배경이 정형화되어 있어 엉뚱한 배경을 알약으로 잘못 짚는 오탐지 리스크가 매우 낮았습니다. 이 도메인 특성을 역이용하여 임계값을 정밀하게 조율하자, 모델이 아슬아슬하게 확신하지 못해 놓칠 뻔했던 흐릿한 알약이나 경계면의 소형 객체들까지 끝까지 찾아내며 점수를 95%에서 95.2%로 견인할 수 있었습니다.

2 단계: TTA(Test-Time Augmentation) 도입 (95.2% → 96.5%)

최종 스코어를 극대화하기 위해 테스트 단계 데이터 증강(TTA) 기법을 도입했습니다. 이는 학습이 끝난 모델에게 시험 이미지를 그냥 보여주는 것이 아니라, 이미지의 좌우를 뒤집거나 살짝 확대하는 등 다양한 변형을 주어 여러 번 추론하게 한 뒤 그 결과들을 종합하는 기법입니다. 데이터 증강을 추론 단계에 적용하자 최종 Public Score가 96.5%로, 1.3%의 성능 향상을 보였습니다.

핵심 인사이트: 학습이 끝난 이후에도 도메인 특성을 반영한 하이퍼파라미터(Confidence 0.2%) 제어와 고도화된 추론 기법(TTA) 같은 정교한 엔지니어링 후처리가 중요하다는 것을 깨달았습니다.

4.6. 데이터 증강 기법 적용 및 실패 단계

현상 및 추이: 모델의 강건성을 높이기 위해 Copy-Paste, Degrees(회전), Flipud(상하 반전) 등 다양한 데이터 증강 기법을 시도했으나, 오히려 기존 최고 성능(96.5%)보다 스코어가 전량 하락하

는 역효과를 기록했습니다.

기법별 실패 원인 분석

첫째, Copy-Paste 기법의 공간·물리적 맥락 파괴 (94.7%): 알약 데이터셋은 정형 바닥 구도, 빛의 방향에 따른 자연스러운 그림자, 알약 간 물리적 거리감이 만드는 '공간 통계적 맥락'이 매우 중요합니다. 그러나 임의의 알약을 잘라 다른 위치에 무작위로 붙여넣는 Copy-Paste는 이러한 물리 규칙을 완전히 파괴합니다. 그림자 방향과 맞지 않는 인위적 객체가 유입되고 알약이 비현실적으로 겹치면서, 모델이 정상적 형태학적 특징을 학습하는 파이프라인을 심각하게 방해해 성능 하락을 초래했습니다.

둘째, Degrees(회전) 기법의 픽셀 뭉개짐 현상 (93%): 알약 식별의 핵심 단서는 표면에 새겨진 미세한 각인 글자와 로고입니다. 임의의 각도로 이미지를 회전시키는 과정에서 글자와 로고의 경계선이 흐릿하게 뭉개지며 식별력을 떨어뜨리는 치명적 부작용을 낳았습니다.

셋째, Flipud(상하 반전) 기법의 비현실성 유입 (90%): 상하 반전 증강은 중력을 무시하는 비현실적 배치를 모델에게 강제 학습시켰습니다. 더불어 알약 각인의 고유한 방향성까지 상하로 뒤틀어놓아, 모델이 물체를 판별할 때의 확신도를 급격히 떨어뜨리는 원인이 되었습니다.

핵심 인사이트: 모델이 이미 96% 이상의 고성능을 기록하며 원본 데이터의 고유 분포를 마스터한 고도화 상태에서는, 구조를 깨뜨리는 기하학적 변형이 오히려 역효과를 낸다는 것을 실험적으로 체득했습니다.

4.7. 데이터 추가 확보 단계 (분포 정합성 기반 외부 데이터 수혈)

현상 및 추이: 데이터 증강이 일관되게 성능 하락으로 귀결되던 상황에서, 분포가 일치하는 외부 원천 데이터를 정밀 선별해 추가 수혈하는 전략으로 선회했습니다. 이 과정에서 train·test 데이터 간 클래스 구성 차이를 발견해 라벨링 정책을 두 차례 정교화하였고, 그 결과 기존 최고점을 0.96540점까지 끌어올린 뒤 최종적으로 0.98435점까지 견인했습니다.

첫째, 증강이 실패한 근본 원인은 데이터 출처에 있었습니다. Notion 안내 문서를 역추적한 결과, 대회에서 제공된 train_set과 test_set은 모두 AI Hub 경구약제 데이터의 동일한 TS_2·TL_2 묶

음에서 분기된 데이터였습니다. 즉 두 집합의 분포가 본래부터 거의 일치했기 때문에, 증강은 이미 정답 분포에 근접해 있던 양질의 학습 데이터를 인위적으로 변형해 오히려 test 분포에서 멀어지게 만드는 역효과를 낳았던 것입니다. 증강의 본질은 실제 사용 환경 분포에 모델을 근접시키는 것인데, 이 경우 정반대로 작용한 셈입니다.

둘째, 이 통찰은 곧 해법으로 이어졌습니다. AI Hub 경구약제 데이터에는 TL_1~TL_8, TS_1~TS_8까지 총 여덟 개의 묶음이 존재하므로, 대회에 사용된 TL_2·TS_2를 제외한 나머지 묶음은 test_set과 동일한 원천 분포를 공유하면서도 모델이 학습 시 본 적 없는 미사용 데이터, 사실상 '분포가 보장된 추가 학습 자원'이었습니다. 다만 train_set은 56종 클래스만 사용하는 반면 외부 원천 데이터에는 133종 클래스가 혼재해 있어, 어떤 알약을 학습 대상으로 인정하고 어떤 알약을 무시할지에 대한 라벨링 기준 설계가 핵심 변수가 되었습니다.

셋째, 1차 수혈에서는 이미지에 등장하는 알약이 전부 train_set의 56종 안에 포함되는 조합만 선별해 610장을 확보했습니다. 기존과 동일하게 '이미지 내 모든 알약을 BBox 처리'하는 단순 정책을 그대로 유지할 수 있어 안전했고, 즉시 0.96540점이라는 최고점을 기록하며 효과를 입증했습니다. 그러나 Kaggle 페이지에 명시된 안내 문구를 강사님께 재확인하는 과정에서, 실제로는 test 데이터에 train에 없는 클래스가 일부 섞여 있으며 이러한 알약은 탐지가 아니라 배경으로 처리해야 채점된다는 사실을 파악했습니다. 이는 '알약이 있으면 무조건 BBox'라는 1차 정책의 전제를 깨뜨리는 동시에, 더 넓은 데이터 활용의 문을 열어주었습니다.

넷째, 2차 수혈에서는 '56종에 완전히 일치하는 조합'이라는 제약을 풀고, 일부 알약만 56종에 속하면 그 알약만 라벨링하고 나머지는 배경 처리하는 방식으로 전환했습니다. 예컨대 한 이미지에 네 종이 있어도 그중 두세 종만 56종에 해당하면 해당 알약만 학습 타깃으로 삼는 식입니다. 이로써 활용 가능한 데이터의 폭이 크게 넓어져 총 3,206장을 추가 확보했고, 이를 학습에 투입해 최종 0.98435점을 달성했습니다.

핵심 인사이트: 데이터 증강이 통하지 않을 때, 정답은 더 정교한 변형이 아니라 분포가 일치하는 원천 데이터의 확보일 수 있습니다. 모델 성능의 상한은 결국 학습 데이터의 분포가 평가 환

경과 얼마나 정합하는가에 의해 결정되며, 출처를 끝까지 추적해 분포가 보장된 데이터를 발굴하는 작업은 어떤 아키텍처 변경보다 강력한 지렛대가 됩니다. 나아가 test셋의 미학습 클래스를 배경으로 처리해야 한다는 평가 규칙을 정확히 해석하자, 그 규칙이 곧 제약에서 자원으로 전환되어 활용 가능한 데이터를 다섯 배 이상 확장시켰습니다. 안내 문구의 미세한 어색함을 그냥 넘기지 않고 끝까지 검증한 태도가 정책 전환의 결정적 계기였습니다.

4.8. 국소 대비 강화(CLAHE) 실험

현상 및 추이: 알약 표면 각인 정보가 식별 성능에 얼마나 영향을 미치는지 검증하고자, 국소 부위 대비를 높여주는 CLAHE 필터 기법을 도입했습니다. 그러나 학습·추론 전반에 적용한 결과, 최종 Public Score가 96.5%에서 94.9%로 오히려 하락하는 현상이 발생했습니다.

첫째, 이질적 픽셀 유입에 따른 혼란 (추론 단계만 적용 시): 학습은 원본 이미지로 유지한 채 추론 단계에만 CLAHE를 적용했을 때는, 모델이 학습 과정에서 본 적 없는 이질적·왜곡된 픽셀 데이터가 급격히 유입되었습니다. 이로 인해 모델이 심각한 혼란을 느껴 스코어가 91%로 급락하는 원인을 확인했습니다.

둘째, 원천 데이터의 라벨링 노이즈 발견 (학습·추론 모두 적용 시): 필터 왜곡을 줄이기 위해 학습·추론 파이프라인 전체에 CLAHE를 적용했음에도 성능이 94.9%로 정체된 원인을 파악하고자 Confusion Matrix를 심층 분석했습니다. 그 결과 필터링으로 알약의 형체와 경계가 더 뚜렷해졌음에도, 모델이 특정 알약을 엉뚱하게 오탐하거나 실제 존재하는 알약을 배경(Background)으로 무시하는 현상이 포착되었습니다. 이는 모델의 식별력 문제가 아니라, 학습 데이터 자체에 정답 BBox가 누락되어 있거나 라벨링이 잘못 섞여 들어간 전형적인 '원천 데이터 품질 오류'였습니다. 필터가 알약 윤곽을 너무 선명하게 강조하다 보니, 오히려 데이터셋 내부의 숨은 라벨링 오류들이 모델 학습에 독으로 작용한 것입니다.

핵심 인사이트: 국소 대비를 강제로 높이는 필터는 각인을 선명하게 만들어줄 순 있지만, 동시에 알약 표면의 미세한 질감이나 배경과의 경계선 부분에 인위적인 격자 노이즈(어둡고 거친 얼룩)를 발생시키는 부작용이 있음을 확인했습니다. 또한 필터가 강조한 선명함이 데이터셋 내부에

잠재된 라벨링 오류를 도리어 부각시킬 수 있다는 점도 함께 깨달았습니다.

4.9. 트랜스포머 기반 아키텍처(RT-DETR) 도입 및 최종 앙상블

현상 및 추이: 최신 트랜스포머 기반 객체 탐지 모델 RT-DETR을 도입하여 단독 성능 Public Score 87.5%를 기록했습니다. 이후 앞서 구축한 YOLO11 모델과 RT-DETR을 유기적으로 결합하는 최종 앙상블을 수행하여 98.6%라는 압도적이고 안정적인 성능을 확보했습니다.

RT-DETR 단독 성능(87.5%)의 한계 원인

이미지 속 물체들의 문맥과 관계를 스스로 파악하게 만드는 트랜스포머 구조는, 본래 주변 물체 간의 연관성이 깊을 때 강력한 힘을 발휘합니다. 그러나 사전 데이터 분석 결과, 우리 데이터셋은 정형화된 바닥 위에 알약들이 독립적으로 흩어져 있을 뿐 알약 상호 간의 유기적 연관성이나 문맥적 흐름이 전혀 없는 특성을 가지고 있었습니다. 이 때문에 RT-DETR의 장점이 발휘되기 어려울 것을 기술적으로 미리 예측했으며, 실제 단독 점수 87.5%로 이 예측이 적중함을 확인했습니다.

그럼에도 RT-DETR 을 도입해 앙상블(98.6%)한 전략적 이유

단독 성능이 낮을 것을 알면서도 RT-DETR을 끝까지 포기하지 않고 투입한 이유는, 향후 웹 서비스 상용화 단계에서 어떤 돌발 상황(특이한 조명, 그림자, 카메라 흔들림 등)에서도 강건한 모델을 완성하기 위함이었습니다. 구조가 완전히 다른 두 모델 — 이미지 전체를 빠르게 훑는 YOLO 계열과 물체의 특징을 다각도로 뜯어보는 트랜스포머 계열 — 을 합치면 서로의 약점을 보완하는 시너지가 발생합니다. 실제 최종 웹 서비스 환경을 가정하고 두 모델의 예측을 앙상블하자, 단독 구동 시 발생할 수 있는 미세한 오탐지들이 깨끗하게 교정되면서 안정적으로 98.6%를 달성할 수 있었습니다.

핵심 인사이트: 단일 모델의 점수가 최고점보다 낮다고 해서 그 모델이 가치 없는 것이 아님을 배웠습니다. AI 엔지니어링에서 가장 중요한 것은 단순 스코어 경쟁이 아니라, 실제 상용화 환경에서 시스템이 오류 없이 버텨줄 수 있는 구조적 안정성을 설계하는 것임을 깨달았습니다.

4.10. 웹 서비스 최적화 아키텍처(YOLO26) 전환 및 최종 3 중 앙상블 SOTA

현상 및 추이: 최고 성능 라인(98.6%)을 확보한 후, '실제 웹 서비스 환경에서의 실시간 상용화'를 실현하기 위해 웹 최적화 아키텍처 YOLO26 모델로 전환을 시도했습니다. YOLO26 단독으로도 뛰어난 효율(98.7%)을 확인한 뒤, 최종 단계에서 [YOLO11 + RT-DETR + YOLO26] 세 핵심 모델을 모두 결합한 3중 앙상블을 수행하여 마침내 Public Score 99.0%라는 SOTA 성능을 달성했습니다.

'YOLO26'을 선택한 상용화 관점의 이유

우리가 개발한 알약 인식 AI는 향후 모바일이나 PC 등 '웹(Web) 브라우저 환경'에서 실시간으로 가볍고 빠르게 구동되어야 하는 명확한 서비스 목표가 있었습니다. 아무리 점수가 높아도 모델이 너무 무거우면 웹에서 로딩이 느려지거나 서버 운영 비용이 폭발하기 때문입니다. YOLO26은 성능을 유지하면서도 가벼운 용량과 빠른 연산 속도를 보장하도록 뼈대가 설계되어 있어, 웹 브라우저 자원을 최소한으로 사용하면서도 정확도를 극대화할 수 있는 구조적 장점이 있었습니다. 이에 최종 상용화 방향성에 가장 부합하는 최적의 무기라고 판단해 선택했습니다.

최종 SOTA 99% 달성 원인 분석

첫째, 웹 최적화 구조가 가져온 단독 모델의 성능 이점: YOLO26은 웹 구동 효율을 높이기 위해 불필요하고 중복된 신경망 계산 과정을 과감하게 걷어냈습니다. 이 과정에서 이미지 내부의 핵심 특징들만 더 깔끔하게 추려지고, 경량화 모델 특유의 과적합 방지 효과(배경 노이즈 무시 및 알약 형태 집중)가 더해지며 단독 모델만으로도 98.7%를 기록하는 놀라운 효율성을 보여주었습니다.

둘째, 세 가지 핵심 엔진이 결합된 3중 앙상블의 시너지: 프로젝트를 거치며 구축한 '고성능 탐지 엔진(YOLO11)', '문맥 강건형 엔진(RT-DETR)', '웹 최적화 엔진(YOLO26)'을 최종적으로 하나로 묶었습니다. 속도가 빠르고 명확한 YOLO 계열의 장점과 사각지대를 다각도로 보완하는 트랜스포머의 장점이 삼위일체로 융합되며, 개별 모델이 유발하던 마지막 1%의 미세한 위치 오차와 노이즈까지 완벽하게 상쇄시켰습니다. 그 결과 프로젝트 최종 SOTA 점수인 99.0%를 완성할 수 있었습니다.

핵심 인사이트: AI 프로젝트의 진정한 종착지는 연구실 안에서 높은 점수를 받는 것을 넘어, 실제 사용자가 쓰는 웹 환경에서 빠르고 안정적으로 서비스되는 동시에 압도적인 정확도까지 유지하는 것임을 깊이 깨달았습니다. '웹 최적화'라는 명확한 목적을 가지고 스마트한 아키텍처(YOLO26)를 선제적으로 도입하고, 이를 기존의 강력한 자산들과 영리하게 결합(3중 앙상블)했을 때, 연산 속도와 상용화 가능성은 물론 99%라는 SOTA 성능까지 모두 거머쥐는 최고의 엔지니어링 성과를 낼 수 있음을 실험적으로 증명한 값진 경험이었습니다.

4.11. 모델 실험 종합 요약

아래 표는 본 장에서 수행한 핵심 실험들의 흐름과 성능 추이를 한눈에 정리한 것입니다. 각 단계의 상세 분석은 위 절을 참고하십시오.

단계	핵심 시도	Public Score	결과
1	Faster R-CNN 베이스라인 (인덱스 재매핑 오류)	0점	실패 → 정렬 검증 교훈
2	Faster R-CNN 하이퍼파라미터 최적화	0.656 → 0.833	한계 도달
3	YOLO11 아키텍처 전환	0.83 → 0.95	대폭 향상
4	체급 선정 (YOLO11m)	—	자원 효율 최적화
5	Confidence 조정 + TTA	0.95 → 0.965	후처리 향상
6	데이터 증강 (Copy-Paste/회전/반전)	0.965 → 0.90~0.947	역효과(실패)
7	외부 데이터 수혈 (분포 정합)	0.965 → 0.984	최대 향상
8	CLAHE 국소 대비 강화	0.965 → 0.949	역효과(실패)
9	RT-DETR 도입 + 2중 앙상블	0.875 → 0.986	안정성 확보
10	YOLO26 전환 + 3중 앙상블 (SOTA)	→ 0.990	최종 SOTA

5. 모바일 활용 소프트웨어(SW App) 및 API 개발

앞서 구축한 고성능 알약 인식 모델은 그 자체로 끝이 아니라, 실제 사용자의 손끝에 닿는 서비스로 완성될 때 비로소 의미를 가집니다. 본 장에서는 학습된 모델을 모바일에서 동작하는 End-to-End 서비스로 구현한 과정을 다룹니다. 시스템 아키텍처와 API 설계에서 출발해, 핵심 기능(Feature)과 UI/UX 설계, 프레임워크 전환(Streamlit → FastAPI)의 배경, 그리고 버전별 진화와 실측 성능 프로파일링까지 순서대로 정리했습니다. 이 파트는 모델링 파트와 동일한 한 줄기의 흐

름 위에 있으며, 모델의 성능이 실제 서비스 품질로 이어지는 '마지막 1마일'을 어떻게 설계했는지를 보여줍니다.

5.1. 시스템 아키텍처

전체 시스템은 모바일 단말과 추론 서버를 분리한 클라이언트–서버 구조로 설계했습니다. 폰에서 촬영한 이미지를 JPEG로 압축하여 전송하면, Ngrok 터널을 통해 우분투 서버로 전달되어 YOLO 모델로 추론이 수행됩니다. 그 결과는 JSON 형태로 다시 앱으로 전송되어 화면에 표시됩니다.

이러한 구조는 무거운 추론 연산을 서버가 전담하고 단말은 촬영·표시에만 집중하게 함으로써, 저사양 기기에서도 고성능 모델을 활용할 수 있게 해줍니다.

5.2. SW 구조 및 API 설계

생산성과 유지보수성을 높이기 위해 프론트엔드 UI와 백엔드 알약 인식 엔진을 분리하여 개발했습니다. 모듈 간 인터페이스를 API로 명확히 정의함으로써 책임 소재를 분명히 하고, 각 모듈을 독립적으로 개발·검증할 수 있도록 했습니다.

- **프론트엔드–백엔드 분리:** UI 와 추론 엔진을 독립 개발하여 모듈별 병렬 작업과 검증이 가능.
- **API 인터페이스 정의:** 입력은 이미지(image), 출력은 시각화 이미지(image)·바운딩 박스(bbox)·클래스(class).

5.3. 핵심 기능(Feature)

1) MVP (Minimum Viable Product)

- 정적 / 실시간 카메라 뷰파인더 (크롭 가이드라인: Reticle)
- YOLO 기반 객체 검출: 바운딩 박스 및 클래스 표시
- 글래스모피즘(Glassmorphism) 기반 심미성 극대화
- HITL(전문가 판별)을 위한 알약 및 판별 정보 저장 기능 (불확실 / 아님 / 맞음)
- 복약 가이드: 다빈도 알약 DB lookup matching (현재 54 종)
- 위치기반 서비스: 공공데이터 API 기반 현재 위치 주변 영업 중 약국 가시화 (dummy)

2) 차기 고도화 기능

- 신약 추가 및 사용자 피드백 활용을 통한 자동 모델 재학습 파이프라인 구축 (MLOps / Airflow)
- 미세한 손떨림을 역이용한 입체 Depth 정보 추가로 인식률 향상 (RGB + Depth)
- 시력 저하 고령층을 위한 고대비 / 거대 폰트 접근성 모드 추가
- 상용앱 수준의 반응성과 디자인을 위한 FastAPI → Android Studio 전환
- 반응 시간 4 초 → 0.5 초 이내로 단축 (YOLOv8n 수준의 경량 버전 확보)

5.4. UI / UX 설계 원칙

기술 검증 단계를 지나며 우리가 도달한 결론은, 사용자 입장에서는 YOLO의 인식률 자체보다 '체감되는 사용 경험'이 더 중요하다는 점이었습니다. T4 GPU 환경의 YOLO11m 기준 촬영–추론–폰 표시까지 약 3초가 소요되지만, 사용자의 기대 수준은 0.5초 이하였습니다. 또한 이상적인 학습(train) 환경과 달리 실제 사용 환경은 명암·조도·크기·초점·흔들림에서 현격한 차이가 있어, 이를 흡수하는 UI 설계가 필요했습니다.

- YOLO 인식률보다 더 중요한 것은 사용자 경험(UX)이다.
- T4 / YOLO11m 기준 촬영 → 추론 → 폰 표시까지 3 초. 사용자 기대는 0.5 초 이하 → 경량 버전 필요.
- 이상적 train 환경 대비 실제 환경은 명암·조도·크기·초점·흔들림에서 현격한 차이가 존재.

아래는 이러한 원칙을 반영한 모바일 앱의 UI 레이어 설계도입니다. 카메라 뷰(Z-Index 0) 위에 오버레이 레이어(Z-Index 1)와 결과 바텀시트(Z-Index 2)를 쌓는 구조로, 촬영 가이드(Reticle), 바운딩 박스, 촬영 버튼(FAB), 확장형 결과 시트를 단계적으로 노출하도록 설계했습니다.

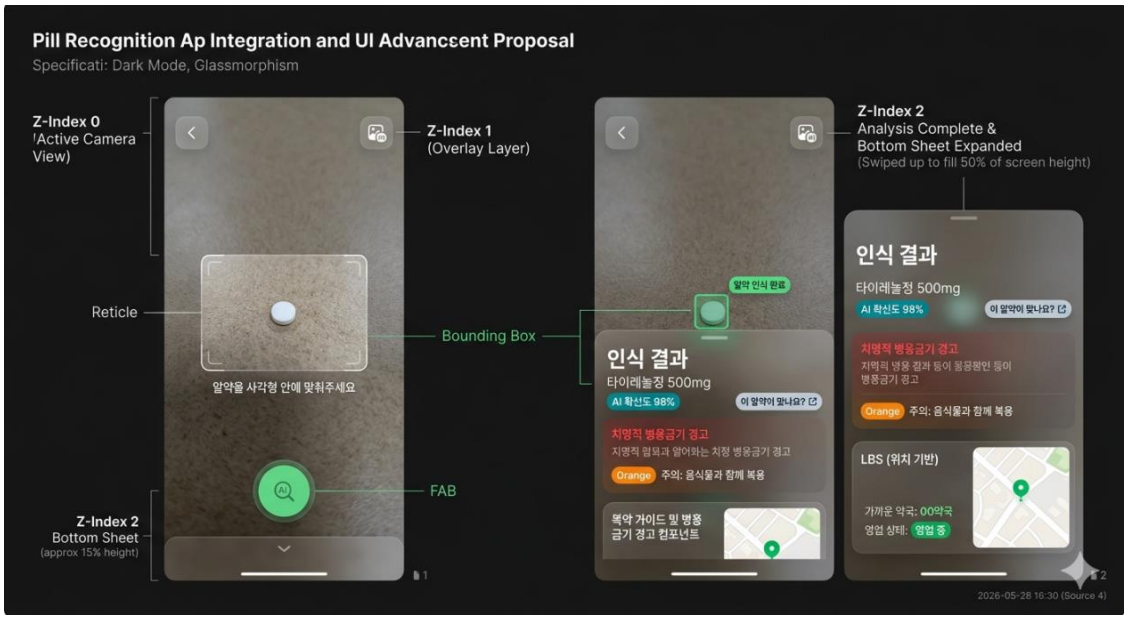


그림 5-1. 모바일 앱 UI/UX 설계안 — Dark Mode·Glassmorphism 기반의 3단계 Z-Index 레이어 구조(카메라 뷰 → 인식 결과 → 확장 바텀시트).

5.5. 앱 동작 메커니즘 (Streamlit / FastAPI, Ngrok, JS)

초기 프로토타입은 Streamlit으로 빠르게 구현했으나, 실제 서비스 요구사항을 검토하며 FastAPI 기반 구조로 전환했습니다. 두 방식의 동작 메커니즘 차이는 다음과 같습니다.

Streamlit (폴링 방식)

- 화면 전체 리셋 동기화: 화면 버튼을 클릭하면 소스 코드 전체가 처음부터 다시 실행됨.
- 동시성 제한: 2 명 이상 동시 접속이 불가능하며, 서버가 마비될 수 있음.

FastAPI (인터럽트 방식)

- 동시 접속 최적화: 화면 전체 리셋이 없고, 고성능(Uvicorn) 서버를 통해 수백~수천 명의 동시 요청 처리 가능.
- 화면 구성 용이성: FastAPI 는 화면이 없는 순수 엔진으로, 프론트엔드와 명확히 분리됨.

FastAPI 구조에서는 화면 구성과 추론·동작의 역할이 다음과 같이 분담됩니다. index.html(JS)이 화면 그리기·버튼 반응·API 호출을 담당하고, main.py(FastAPI)가 YOLO 추론과 결과 JSON 반환(/predict, /hitl, /model-info)을 담당합니다.

사용자 행동	브라우저 감지 이벤트	JS 함수 호출
버튼 클릭	click 이벤트 발생	captureNow()
화면 탭	touchend 이벤트	focusAt()
시트 드래그	touchmove 이벤트	dragMove()
파일 선택	change 이벤트	handleFile()

5.6. 버전별 진화와 비교 (ver1 → ver4)

앱은 UI 프로토타입(ver1)에서 출발해, 실제 추론과 데이터 연동이 결합된 기능 완성형 MVP(ver4)로 발전했습니다. ver1이 실제 추론 없이 화면 레이아웃과 흐름만 완성한 버전이라면, ver4는 카메라 → 추론 → 결과 → 지도 → HITL까지 한 줄기가 실제 데이터로 끝까지 관통되는, 이른바 '수직 슬라이스 MVP(Vertical Slice MVP)'입니다. 전체 기능을 얕게 붙인 것이 아니라 핵심 흐름 하나를 끝까지 실제로 동작시켰다는 점이 핵심입니다.

항목	ver1_0522 (프로토타입)	ver4_0529_FastAPI (기능 완성)
프레임워크	Streamlit	FastAPI + HTML/JS
입력	파일 업로드만	실시간 카메라 + 파일
추론	더미 (원형 crop, detect.py)	실제 YOLO 모델 (app_wrapper.py)
알약명	"타이레놀정 500mg" 하드코딩	모델 출력값
확신도	"96.4%" 하드코딩	pred['confidence']
지도	iframe 플레이스홀더	Leaflet + Overpass API 실데이터
HITL	st.toast만 (알림)	hitl_log.jsonl 실제 저장
카메라	st.camera_input (기본)	흐림 감지 + 자동 포커스 + 2초 버퍼
복약정보	타이레놀 1개 하드코딩	PILL_DB 56종 연동

아래 두 화면은 ver1 프로토타입의 모습입니다. 실제 추론 없이 결과값(타이레놀정 500mg / 96.4% / 0.42 sec)을 하드코딩하여 화면 흐름과 레이아웃을 먼저 검증했습니다.

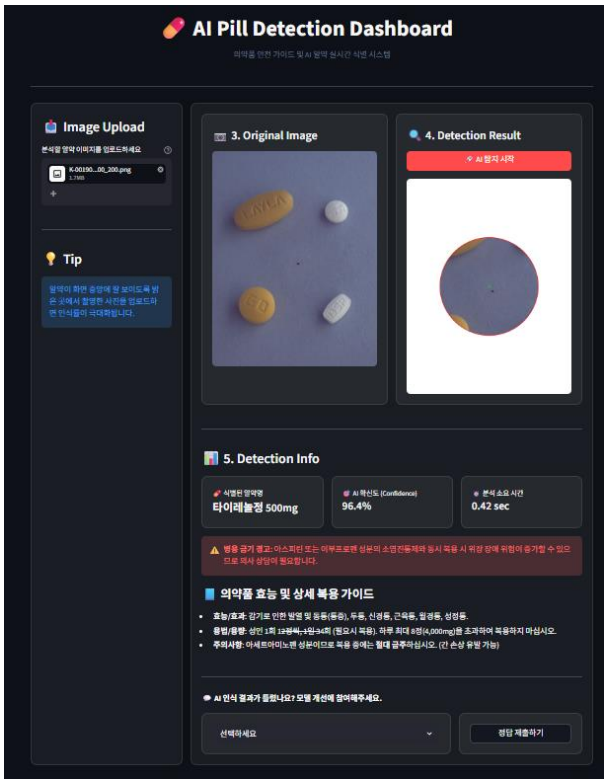


그림 5-2. ver1 프로토타입(Streamlit) — AI Pill Detection Dashboard. 업로드·원본·검출 결과·복약 가이드 화면 흐름을 하드코딩 값으로 선검증.

5.7. 파일 구조 분석

Streamlit 시절과 FastAPI 전환 후의 파일 구조 변화는 다음과 같습니다. 핵심 추론 엔진(app_wrapper.py)과 YOLO 모델은 그대로 재사용하면서, 화면 계층만 streamlit_app.py에서 index.html + main.py 구조로 교체한 것이 특징입니다.

```
[Streamlit 시절]                [FastAPI 전환 후]


 streamlit_app.py                index.html (JS, 신규)
        |                                |
        v                                v
 app_wrapper.py      ===>        main.py (신규)
        |                                |
        v                                v
      YOLO                       app_wrapper.py (재사용)
                                         |
                                         v
                                       YOLO
```

main.py — FastAPI 서버 (라우팅 + 후처리)

FastAPI 서버는 화면 서빙과 추론 요청 라우팅, 후처리를 담당합니다. 서버 시작 시 모델을 1회만

로드하고, 56종 복약 정보(PILL_DB)를 직접 보유합니다.

```
GET  /            -> index.html 서빙
POST /predict     -> app_wrapper 호출 -> JPEG 인코딩 -> base64
                     -> DB 조회 -> JSON 반환
POST /hitl        -> hitl_log.jsonl 에 사용자 수정 데이터 저장
GET  /model-info  -> 클래스 목록 반환


# 직접 들고 있는 것
PILL_DB = { "가바토파정 100mg": {...}, ... }   # 56종 복약 정보
predictor = PillPredictorWrapper()             # 서버 시작 시 1회 로드
```

app_wrapper.py — YOLO 추론 + 시각화 엔진

실제 추론과 시각화를 담당하는 엔진으로, 입력 이미지를 4단계 파이프라인을 거쳐 예측 데이터·시각화 이미지·타이밍 통계로 반환합니다.

```
# 4단계 파이프라인
1_decode_cvtColor  -> PIL -> OpenCV 변환
2_yolo_inference   -> self.model(img_cv)
3_box_parsing      -> box.xyxy, cls, conf 추출
4_pil_drawing      -> 바운딩박스 + 한글 라벨 그리기


# 반환값
return refined_output, np.array(pil_img), timer.result()
#       예측 데이터       시각화 이미지        타이밍 통계
```

index.html — 프론트엔드 (약 1,071 줄)

화면 구조와 스타일, 그리고 모든 사용자 인터랙션 로직을 담은 단일 프론트엔드 파일입니다. 브라우저가 하드웨어(터치스크린·마우스) 이벤트를 감지해 등록된 JS 함수로 전달하면, 각 함수가 촬영·추론 호출·결과 렌더링·지도·HITL 모달 등을 처리합니다.

```
index.html (1071줄)
 ├── <style>  (9~329줄)     <- 색상, 위치, 애니메이션
 ├── <body>   (331~416줄)   <- 버튼, 레이어 배치
 └── <script> (419~1068줄)  <- 로직 전체
      ├── captureNow()        <- 촬영 버튼
      ├── handleFile()        <- 갤러리 버튼
      ├── resetApp()          <- 리셋 버튼
      ├── callPredict()       <- FastAPI /predict 호출
      ├── applyResult()       <- 결과 화면 렌더링
      ├── openHITL/submitHITL <- HITL 모달
```

```
      ├─ fetchGPS/initMap    <- GPS + Leaflet 지도
      └─ dragStart/dragEnd   <- 바텀시트 드래그
```

5.8. 앱 서빙(App Serving) 방식

서비스는 두 가지 방식으로 구동했습니다. 초기에는 Streamlit + Ngrok 조합으로 빠르게 프로토타입을 띄웠고(case 1), 최종적으로는 FastAPI + Uvicorn 서버를 Ngrok으로 외부에 노출하는 방식(case 2)으로 전환했습니다.

case 1 — 프로토타입: Streamlit & Ngrok

```
# (ubuntu / powershell)
cd /mnt/c/Users/kimhy/CodeItPJT/초급PJT
source .venv/bin/activate
streamlit run streamlit_app.py


# 실행 화면
$ streamlit run app.py
Uvicorn server started on 0.0.0.0:8501
  Local URL:   http://localhost:8501
  Network URL: http://172.20.186.173:8501
```

case 2 — 최종: FastAPI & Uvicorn 서버

```
# (ubuntu / powershell)
cd /mnt/c/Users/kimhy/CodeItPJT/초급PJT
source .venv/bin/activate
uvicorn main:app --host 0.0.0.0 --port 8000 --reload
ngrok http 8000


# 실행 화면
[LOG] FastAPI 서버 시작 — YOLO 모델 로딩 중...
✅[INIT] PillPredictorWrapper가 .../outputs/yolo/train_m/weights/best.pt
         모델로 초기화되었습니다.
[LOG] 모델 로드 완료
INFO:  Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:  Application startup complete.


[MIDDLEWARE] GET / -> 34.57 ms        INFO: "GET / HTTP/1.1" 200 OK
[MIDDLEWARE] GET /model-info -> 2.36 ms
[LOG] /predict 수신 — image.jpg
```

5.9. 실측 성능 프로파일링

FastAPI 서버에 7단계 프로파일러(profiler.py)를 도입하여 추론 요청 1건의 처리 시간을 단계별로 측정했습니다. 측정 결과 전체 처리 시간의 대부분(약 94%)이 YOLO 추론 단계에 집중되어 있어, 경량 모델 도입이 체감 반응 속도 개선의 핵심임을 정량적으로 확인할 수 있었습니다.

처리 단계	구간	소요 시간
1. decode / cvtColor (이미지 디코딩)	추론 전	125.25 ms
2. YOLO inference (모델 추론)	추론	2,417.44 ms
3. box parsing (박스 파싱)	추론	15.09 ms
4. PIL drawing (시각화)	추론	11.25 ms
소계 (app_wrapper)		2,569.03 ms
5. JPEG encode	후처리	4.58 ms
6. base64 encode	후처리	0.15 ms
7. DB lookup	후처리	0.00 ms
소계 (후처리)		4.73 ms

5.10. App / Model 연동 테스트

모델(.pt)과 앱 인터페이스가 실제로 올바르게 연결되는지 검증하기 위해, 별도의 테스트 스크립트 세트를 구성했습니다. 환경 점검(check_env.py), 인터페이스 검증(test_wrapper.py), 추론 엔진(app_wrapper.py)으로 역할을 나누어 단계적으로 확인합니다.

```
.
├── app_wrapper.py    # PillPredictorWrapper 클래스 정의, predict() 추론 수행
├── check_env.py      # 필요한 라이브러리 설치 여부 검사
└── test_wrapper.py   # PillPredictorWrapper 인터페이스 검증 스크립트
```

연동 테스트 절차

1. **환경 확인:** check_env.py 실행으로 필요한 라이브러리 설치 여부를 점검한다 (python check_env.py).
2. **인터페이스 검증:** test_wrapper.py 실행으로 실제 이미지 파일을 바이너리로 읽어 predict()를 호출한다 (python test_wrapper.py).
3. **결과 확인:** 모델 로드 후 predict(image_bytes) 호출 결과(list, 시각화 이미지, box/label/confidence 키)와 반환 타입·구조의 정합성을 검사한다.

이 테스트는 app_wrapper.py와 모델(.pt)을 직접 연결해 실제 추론이 수행되는지 확인하는 역할을 합니다. 실패할 경우 반환 타입·구조가 올바른지, 모델 추론이 정상인지를 즉시 점검할 수 있

습니다.

5.11. FastAPI 개발 디렉토리 비교 분석

5월 29일부터 6월 1일까지 4일간의 개발 과정에서 총 7개의 버전 폴더가 생성되었습니다. 각 버전은 DB 확장, Streamlit 전용화, 한글 폰트 적용, 프로파일링 추가, QA 로그 정리 등 명확한 목적을 가지고 점진적으로 진화했습니다.

진화 흐름: initial / ver1 → ver2(DB 확장) → semifinal(Streamlit) → 0531(한글 폰트) → 0529(프로파일링) → 0601(QA 로그)

기능	initial/ver1	ver2	semifinal	0531	0529/0601
main.py (FastAPI)	O	O	X	O	O
약품 DB 개수	10개	56개	56개	56개	56개
이미지 시각화	YOLO.plot()	YOLO.plot()	YOLO.plot()	PIL 커스텀	PIL 커스텀
한글 폰트	X	X	X	O	O
Profiler	X	X	X	X	O (7단계)
HTML 리포트	X	X	X	X	0601만 O

각 버전 특징 요약

- **ver4_0529_FastAPI_initial — 최소 MVP:** 약품 10 개, YOLO 기본 시각화, 간단한 FastAPI 구조.
- **ver4_0529_FastAPI_ver1 — initial 복제본:** 실험용 백업.
- **ver4_0529_FastAPI_ver2 — DB 대폭 확장:** 약품 56 개로 확대, 상세 정보(성분명) 추가, re 모듈로 라벨 정규화 시작.
- **ver4_0529_semifinal — Streamlit 전용 전환:** main.py 제거, Streamlit 만 사용, 커스텀 모델(yolov8_custom_pill_detection_best.pt) 추가.
- **ver4_0531_FastAPI — 시각화 품질 개선:** YOLO 기본 렌더링 → PIL 커스텀 드로잉 전환, NanumGothic.ttf 한글 폰트 지원, 테마 색상(#00C896) 바운딩 박스.

- **ver4_0529_FastAPI — 성능 모니터링 추가:** profiler.py 신규 추가, 7 단계 측정(디코딩 → YOLO 추론 → 박스 파싱 → PIL 드로잉 → JPEG 인코딩 → base64 인코딩 → DB 조회).
- **ver4_0601_FastAPI — 최종 안정화 버전:** 0529 와 코드 동일, HTML 리포트 파일 추가(QA 로그 4 개, 사용자 가이드).

용도별 권장 버전

- **운영 / 제출용:** ver4_0601_FastAPI (가장 완성된 버전)
- **성능 분석용:** ver4_0529_FastAPI (profiler 포함, 코드 동일)
- **Streamlit 전용:** ver4_0529_semifinal
- **가벼운 테스트:** ver4_0529_FastAPI_initial

5.12. 백오피스 및 데이터 플라이휠 (운영 관점 설계)

단순한 인식 기능을 넘어, 서비스가 운영 단계에서 스스로 개선될 수 있도록 HITL(Human-in-the-Loop) 백오피스와 데이터 플라이휠 모니터링 화면을 함께 설계했습니다. 사용자가 현장에서 찍은 알약이 빛 번짐이나 그림자로 낮은 정확도로 판독되면 Quarantine DB로 비동기 포크 전송되고, 도메인 전문가(약사)가 백오피스에서 원클릭으로 정답 클래스를 매핑해 데이터 무결성을 보존합니다.

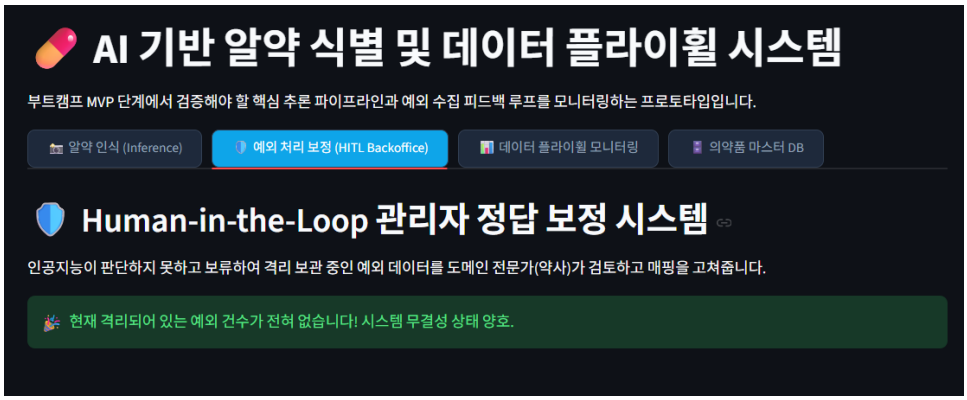


그림 5-3. HITL 백오피스 — 인공지능이 판단을 보류한 예외(격리) 데이터를 도메인 전문가가 검토·보정하는 화면.

보정된 데이터가 일정량(10건) 이상 누적되면 Airflow와 MLflow 파이프라인이 가동되어 기존 가중치 위에 점진적 추가 학습(Fine-Tuning)을 수행하고, 검증 셋 정확도가 개선되었을 때만 추론 서버가 신규 모델 가중치를 무중단(Hot-Swap)으로 교체합니다. 이러한 '데이터 플라이휠' 구조

는 서비스를 사용할수록 모델이 더 똑똑해지는 선순환을 만들어 줍니다.

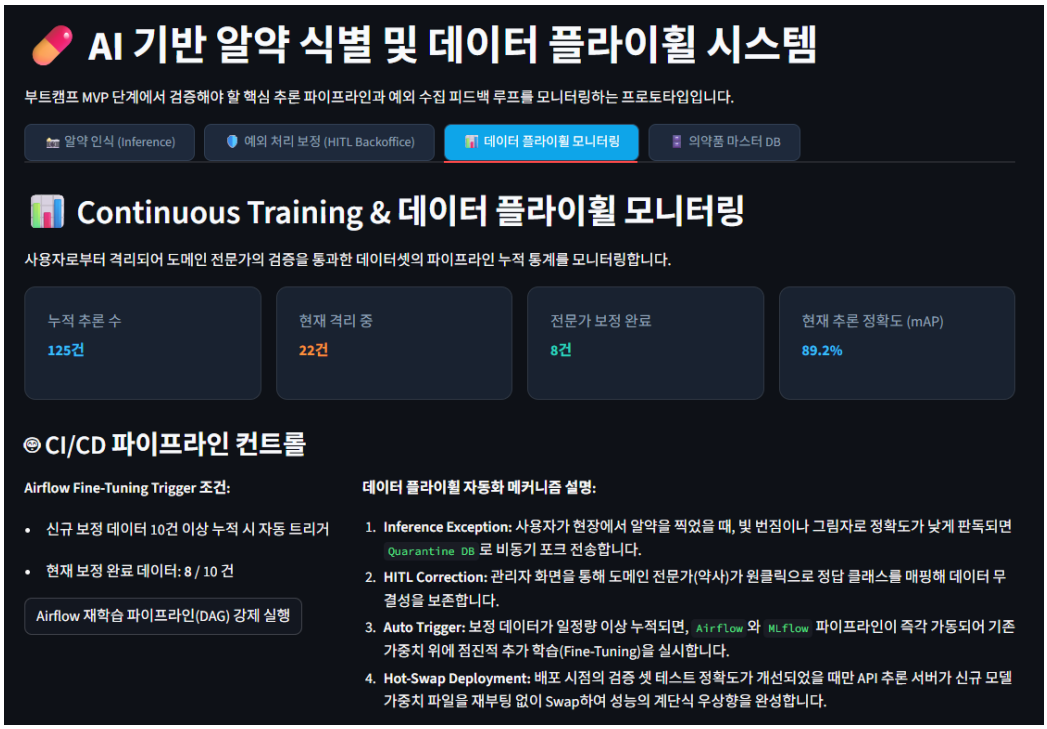


그림 5-4. 데이터 플라이휠 모니터링 & CI/CD 파이프라인 컨트롤 — 누적 추론 수, 격리 건수, 전문가 보정 완료, 현재 추론 정확도(mAP)를 실시간 모니터링.

또한 신규 알약이 시장에 출시되더라도 AI 가중치 자체를 재학습할 필요 없이, 마스터 DB 테이블과 알약 벡터 DB에 특징 임베딩 한 행만 추가하면 즉시 인식 성능을 지원할 수 있도록 설계했습니다. 이는 'AI 모델'과 '의약품 정보 데이터베이스'를 느슨하게 결합(Decoupling)한 하이브리드 파이프라인의 핵심으로, 관리 운영 비용을 획기적으로 낮춰 줍니다.

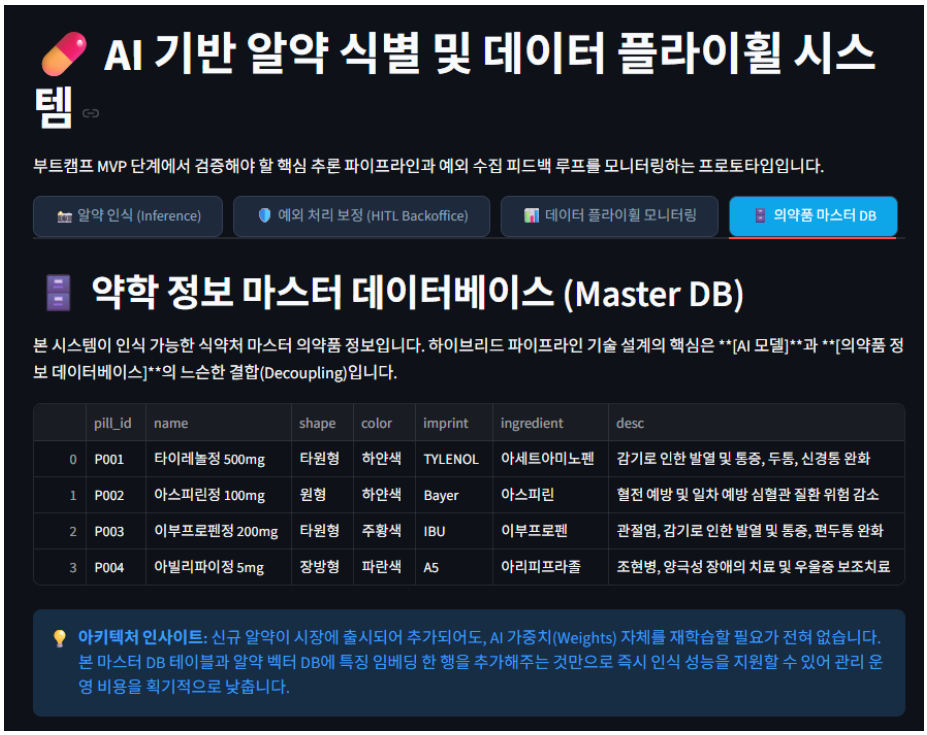


그림 5-5. 약학 정보 마스터 데이터베이스(Master DB) — pill_id·성분명·효능 등을 관리하며, AI 모델과 분리(Decoupling)된 구조로 신약 확장에 유연하게 대응.

5.13. 실제 구동 화면 (End-to-End 시연)

아래는 최종 ver4 앱이 실제 모바일 브라우저에서 구동되는 화면들입니다. 학습 환경(train)과 실제 사용 환경의 차이(조도·초점·배경)를 그대로 보여주는 동시에, 모델 성능이 실제 서비스로 이어지는 전 과정을 확인할 수 있습니다.

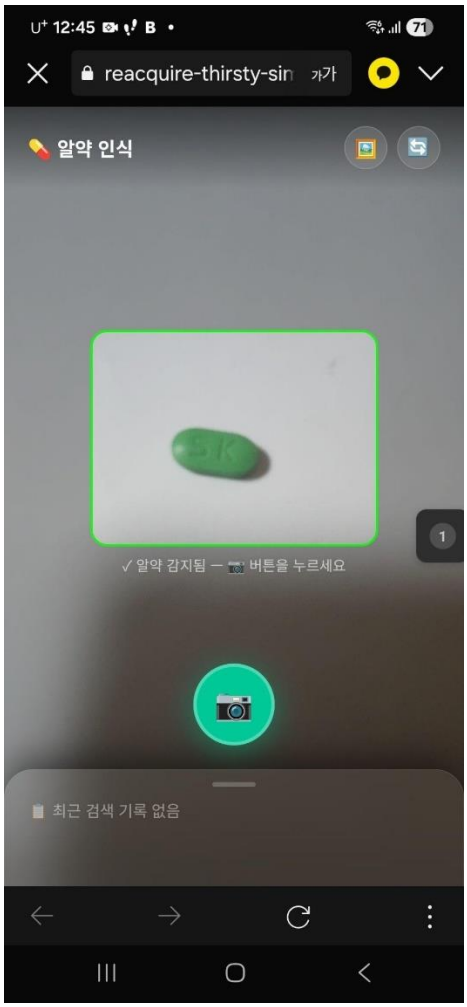


그림 5-6. 실시간 카메라 — 알약을 자동 감지하면 녹색 가이드(Reticle)가 활성화되고 촬영 버튼(FAB)이 강조된다.

그림 5-7. 인식 결과(저신뢰도 케이스) — '기넥신에프정 84%' 인식 후, 저신뢰도 경고·복약 가이드(병용금기/복약주의)·근처 약국(LBS)을 바텀시트로 표시.

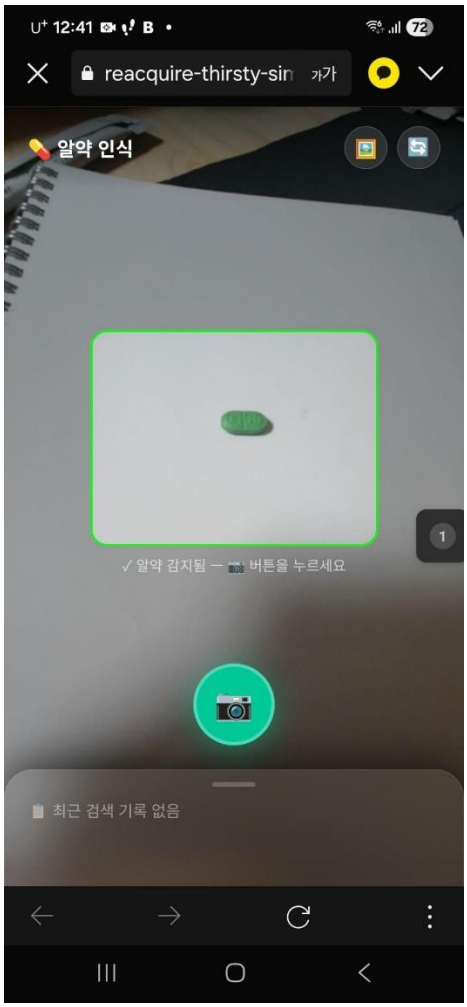


그림 5-8. 다른 알약 촬영 화면 — 형태·각인이 다른 알약도 동일한 가이드 흐름으로 감지한다.

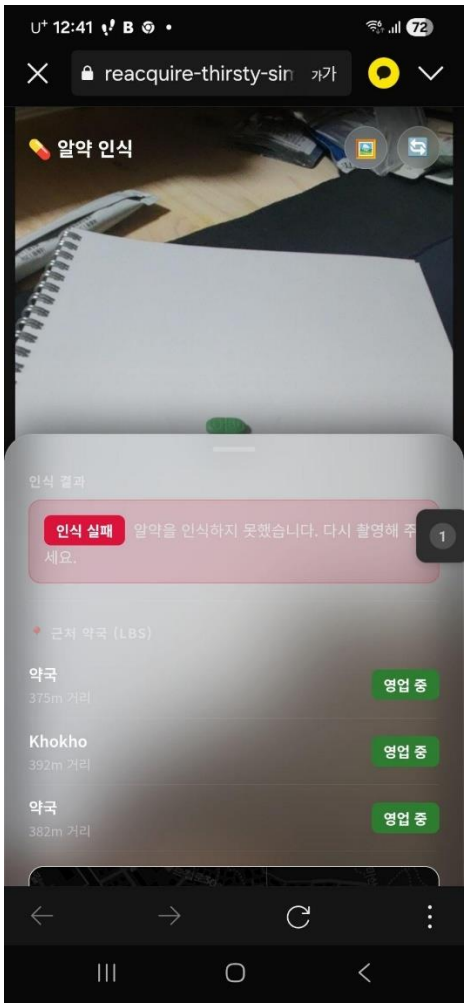


그림 5-9. 인식 실패 케이스 — 알약을 인식하지 못한 경우 재촬영을 안내하며, 근처 약국 LBS 정보는 정상적으로 제공한다.

추가로, 실시간 카메라 입력에 대해 전처리·하이브리드 추론이 적용되는 모니터링 화면(프로토타입)도 함께 구축하여, 추론 신뢰도(Confidence)와 정상 식별 여부를 실시간으로 확인할 수 있도록 했습니다.

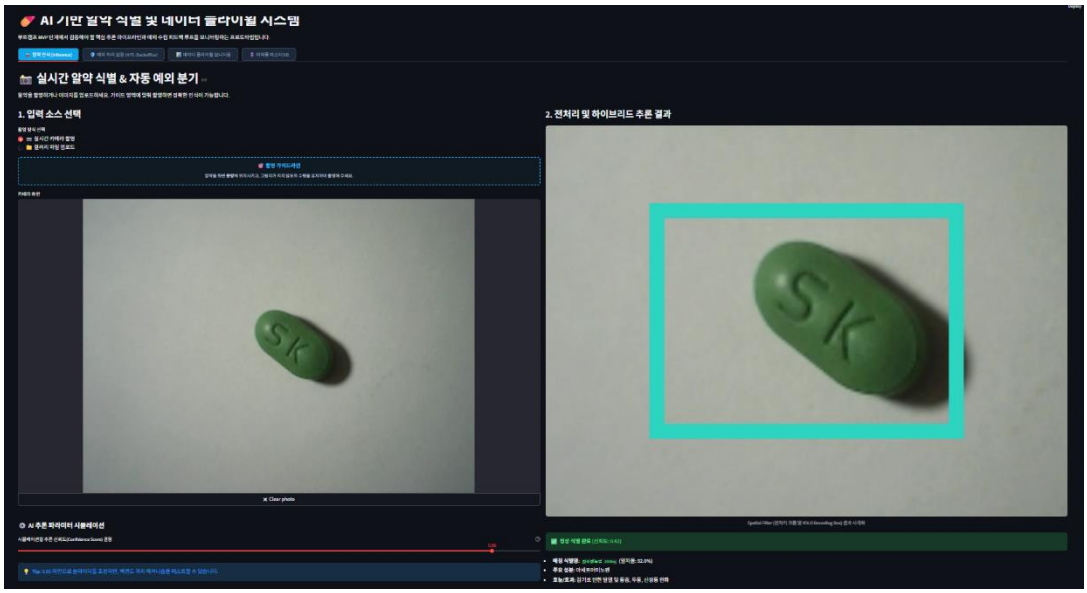


그림 5-10. 실시간 식별 & 자동 예외 분기 모니터링 — 입력 소스 선택, Confidence 슬라이더, 전처리·하이브리드 추론 결과를 한 화면에서 확인.

위치기반 서비스(LBS)는 공공데이터 및 지도 API와 연동하여, 현재 위치 주변의 영업 중인 약국을 지도 위에 시각화하도록 구성했습니다.

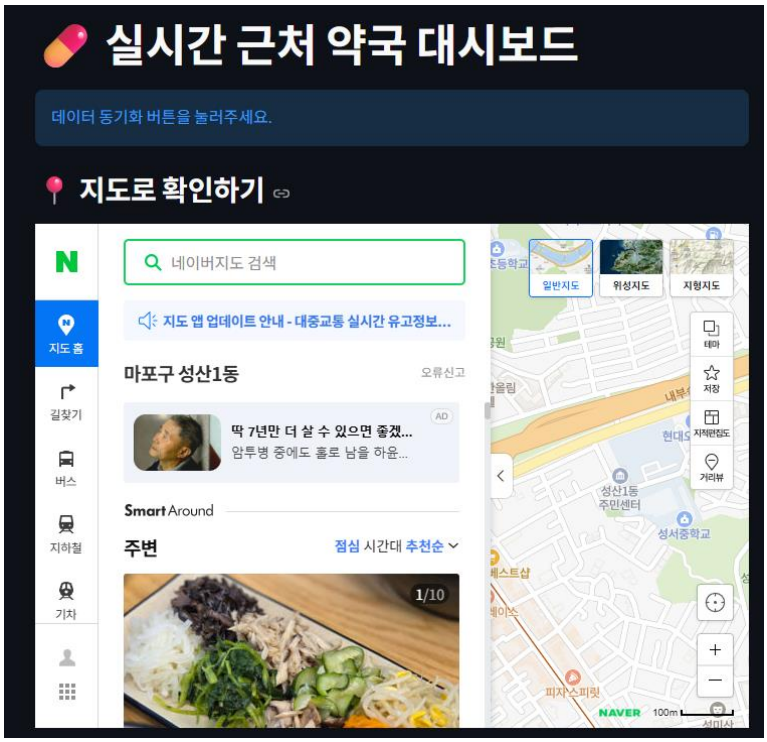


그림 5-11. 근처 약국 지도(LBS) — 사용자 현재 위치를 기준으로 주변 약국을 지도에 표시.

5.14. Lessons & Learned

소프트웨어 개발 파트를 마무리하며, 모델 성능을 실제 서비스로 전환하는 과정에서 얻은 교훈을 정리하면 다음과 같습니다.

- 하드웨어(HW) 개발자도 App 개발 및 서빙(Serving)이 가능함을 직접 확인했다 — '일단 해보자'의 가치.
- API 를 정의함으로써 모듈 간 책임 소재를 명확히 할 수 있었다.
- 모듈별 개발·검증 및 통합(Integration)의 효율성을 경험했다.
- 'Model' 중심 사고에서 '제품·고객' 중심 사고로 전환하며 시야와 관점이 확대되었다.
- 제품 측면에서는 모델 성능은 기본이며, 사용자 환경에 대한 연구·적용이 필수임을 체감했다.
- 알약 App 개발은 부트캠프 과정의 Full Pipeline(서비스 → 전달 방법(App) → AI 모델(YOLO) → 데이터(Kaggle/AI Hub))을 모두 적용해 볼 수 있는 플랫폼이었다.
- 향후 공공서비스 측면에서는 LLM·RAG 를, 모바일(Phone) 측면에서는 경량화·양자화·Docker 를 확장 과제로 도출했다.

6. 결론 및 향후 개선 사항

6.1. 본 프로젝트의 의의 (초급 단계 성과)

본 프로젝트는 한 장의 사진에서 다수의 알약을 동시에 식별하는 AI 모델을 구축하고, 이를 실제 모바일에서 동작하는 서비스로 연결하는 End-to-End 프로토타입을 완성했다는 데 의의가 있습니다. 모델 측면에서는 베이스라인의 시행착오에서 출발해 아키텍처 전환·후처리 고도화·분포 정합 데이터 수혈·3중 앙상블을 거쳐 최종 Public Score 99.0%(SOTA)를 달성했고, 서비스 측면에서는 카메라–추론–결과–지도–HITL로 이어지는 한 줄기를 실제 데이터로 관통시켰습니다.

- 원천 데이터의 불균형성과 BBox 누락 등의 한계를, EDA 와 자동화 정제 파이프라인으로 극복했다.
- 모델 성능과 별개로, 사용자 경험(UX)과 운영(HITL·데이터 플라이휠)까지 고려한 서비스 구조를 설계했다.

- Git 인프라 구축, 소통 그라운드 룰 정립 등을 시도하며 프로젝트 협업 프로세스의 기초를 공고히 다졌다.

6.2. 차기 프로젝트(중급)를 위한 아쉬운 점 및 개선 과제

- **소통 및 동기화:** 프로젝트 진행 중 팀원 간 수행 작업의 실시간 싱크 및 기록 관리가 일부 누락된 점이 아쉬움으로 남음. 차기 단계부터 Notion·Git 기반 프로젝트 관리를 더욱 타이트하게 적용할 예정.
- **기술 인프라 고도화:** 초급 단계 이후에는 Codex·Claude 등 코딩 AI 에이전트를 적극 도입하여 개발 효율성을 극대화하고, CLAUDE.md 규칙 적용을 통한 코드 정확도 향상을 유도할 예정.
- **서비스 경량화:** 현재 약 3~4 초인 반응 시간을 0.5 초 이내로 단축하기 위해, 경량 모델(YOLOv8n 수준) 확보 및 양자화·Docker 기반 배포를 차기 과제로 설정.

7. 법적리스크 및 마케팅 분석

7.1앱서비스의 법적리스트(Law risk) 및 마케팅(Marketing) 분석

- 처방약 사진 촬영과 식별이 현행 의료법(원격 의료금지 등) 저촉 여지에 대한 판례 또는 법령 조사

- •환자 본인의 단순 촬영 및 정보 조회는 합법입니다.
- •개인이 자신이 처방받은 약의 알약 모양이나 처방전을 찍어 AI 나 식별 앱(예: 약학정보원 검색 등)을 통해 이름을 확인하는 행위는 의료법 위반이 아닙니다.
- •의료법상 규제 대상인 의료행위나 원격의료는 타인을 대상으로 하는 진료, 진단, 처방을 의미함. 자신의 건강 정보를 스스로 조회하고 확인하는 것은 환자의 알 권리 및 건강 관리 영역에 해당됩니다.
- •의료업에 종사하고 직접 진찰한 의사&가 아니면 처방전 등을 작성하여 환자에게 교부하지 못한다. 그러나 처방약을 사진 촬영하고 식별만 하는 것은 의료행위가 아니므로 무관합니다.
- •처방약 사진 촬영 및 식별은 이미 나와있는 처방약에 대한 촬영 및 식별일뿐이라 현행 의료법 위반에는 해당이 안됩니다.
- •그러나 주의해야 될 점이 앱이 진단이나 처방을 하게 되면 얘기가 달라지며 의료법상 의사만이 할 수 있는 행위를 앱이 대신하면 안됩니다.

- •의료법 위반이 아닌 합법적 서비스에 해다외는 것은 이미지 인식 기술을 통해 단순히 약의 명칭, 성분, 효능, 부작용, 복약 주의사항 등 공공 데이터에 등록된 객관적 정보만 매칭하여 보여주는 경우입니다.

- •관련 법령:의료법 제 27 조제 1 항 (무면허 의료행위 금지): 의료인이 아니면 누구든지 의료행위를 할 수 없습니다. 의료법 제 34 조 (원격의료): 현행법상 비대면 진료가 제도화(2025 년 12 월 법안 통과)되었으나, 비대면 진료 및 처방의 주체는 엄연히 '사람(의사)'이어야 합니다.

- •보건복지부 행정해석 및 일반 법리에 따르면, 의학적 판단(질병의 진단 및 치료법 결정)이 개입되지 않은 단순 정보 제공, 복약 지도 보조, 의약품 식별 가이드는 의료행위로 보지 않습니다.

- •단, 정부가 정한 비대면 진료의 법적 기준(대상 환자 조건, 마약류 및 오남용 우려 의약품 처방 제한 등)을 철저히 준수해야 하며, 이를 벗어난 임의 원격 처방은 여전히 의료법 제 33 조(개설 기준 위반) 및 제 17 조(직접 진찰 의무) 위반으로 처벌받습니다.

7.2 처방전과 알약 사진이 민감 개인정보에 해당하는지 조사

- •처방전의 경우 (100% 민감정보): 처방전에는 환자의 이름, 주민등록번호뿐만 아니라 질병분류기호(진단명), 투약 법, 구체적인 의약품 명칭이 명시되어 있습니다. 이를 통해 해당 개인이 어떤 질병(예: 우울증, 당뇨, 암 등)을 알고 있는지 완전히 파악할 수 있으므로 명백한 건강정보에 해당됩니다.

- <일반적인 시중의 알약들에 대한 데이터베이스 확보하여 앱에서 활용하는 것의 합법 여부>

- •시중의 일반적인 알약(의약품) 정보 자체를 수집하여 앱의 데이터베이스(DB)로 구축하는 행위는 전혀 불법이 아니며, 100% 합법입니다.

- • '민감정보'는 "특정 개인(A 씨)이 이 약을 먹는다"는 사실과 결합된 사진을 뜻합니다. 이와 달리, 시중의 약품 그 자체에 대한 정보(이름, 모양, 성분, 효능)는 개인정보가 아닌 '공공 데이터 및 객관적 사실'에 해당하기 때문입니다.

7.3 데이터 수집 출처에 따른 법적 판단

- -합법적 데이터 수집

- •약품 데이터를 수집할 때는 '어디서 가져오느냐'가 가장 중요합니다.

- •가장 안전하고 합법적인 방법 (공공 데이터 활용)
- •식품의약품안전처(의약품안전나라)나 공공데이터포털(data.go.kr)에서 제공하는 '의약품 낱알식별 정보' 및 '의약품 개요 정보' Open API 를 활용하는 것은 정부가 공식적으로 허용한 합법입니다.

- •이 공공 API 를 연동하여 앱 내에 DB 를 구축하면 저작권이나 위법성 문제가 전혀 발생하지 않습니다.
- -위법적 데이터 수집
- 위법 소지가 있는 방법 (타사 사이트 크롤링)
- •'약학정보원'이나 '네이버 지식백과(의약품 사전)', 혹은 기존의 다른 알약 식별 앱의 웹사이트를 무단으로 크롤링(상업적 긁어오기)하여 DB 를 구축하는 것은 위험합니다.
- 이유: 의약품의 '사실 정보(성분명 등)' 자체는 저작권이 없지만, 타사가 막대한 비용과 노력을 들여 구축한 '데이터베이스의 체계 및 사진 촬영물'은 저작권법(DB 제작자의 권리) 및 부정경쟁방지법으로 보호받기 때문입니다. 무단 크롤링 시 손해배상 청구 소송을 당할 수 있습니다.

7.4 앱 서비스 운영 시 주의할 점 (약사법 관련)

- •알약 DB 를 안심하고 구축하셨더라도, 앱을 통해 유저에게 정보를 보여줄 때는 주의해야 합니다.
- •의약품 정보의 단순 표기 (합법): "이 알약은 식약처에 등록된 아세트아미노펜 성분의 해열진통제입니다."라고 보여주는 것은 합법입니다.
- •의학적 판단 및 추천 (위법 소지): 유저가 증상을 입력했을 때, DB 에 있는 특정 전문의약품을 지목하며 "당신은 이 약을 먹어야 합니다"라고 추천하거나 진단을 내리는 서비스 구조는 약사법 및 의료법 위반이 될 수 있습니다. (단순 효능 검색 매칭은 괜찮습니다.)

7.5 마케팅 분석

- -이거뭐약 앱의 수익 창출 구조(광고, 제휴 등) 조사
- 구글 플레이스토어 리뷰 기준. 이거뭐약 사용자들이 꼽는 가장 큰 불만 2 가지

<알약/처방전 식별 앱의 수익 창출 구조>

- 앱을 다운 받아 살펴보니 배너 광고도 현재는 전혀 없습니다. 앱을 통해서 제대로 광고를 하고 있는 것 같지는 않습니다. (어플 화면 캡쳐 업로드는 저작권법상 주의해야 하므로 생략)

- 현재 성능 자체의 문제가 있는 것 같기에 사용자 확보도 그다지 쉽지 않아 보이는 만큼 광고를 한다고 해도 제대로 수익을 얻기가 어려운 상황입니다.

7.6 새로운 앱을 만들었을 때 광고 방향 제안

- (이거뭐약 앱의 광고 수익 구조가 제대로 되어 있지 않은 것 같아서 제안형식을 추가)

- 1.우선 앱의 검출 성능이 확실해야 합니다. 그래야 사용자들이 믿고 씁니다.

- 2.앱 하단, 결과 화면, 알림 창 등에 구글 애드몹(AdMob) 등을 통한 디스플레이 광고를 노출합니다.

- 3.건강기능식품, 영양제, 제약회사의 특정 일반의약품(배과사전형 광고) 등 유저의 건강 관심사에 맞춘 타깃 맞춤형 헬스케어 광고를 시도할 수 있습니다.

- 4. 유료 기능을 부분적으로 추가하기: 예시 대량의 알약 한 번에 카운팅하기(이 부분은 머신러닝 성능 필요), 가족 건강 통합 관리(복약 알림) 등.

- 5.이커머스 연계 (영양제/건강 보조식품 큐레이션)

- 의약품에 대해서는 처방 문제가 있어서 까다롭지만 영양제로만 한정할 경우 유저가 처방받은 약의 성분을 AI 가 분석한 후, 함께 먹으면 안 되는 영양제를 걸러주거나 반대로 부족한 영양소를 채워줄 수 있는 맞춤형 영양제 제휴 링크를 노출하여 판매 수수료를 얻습니다.

- 6.B2B 약국 및 제약사 솔루션 연계

- 주변 약국 연계: 현재 조제 가능한 인근 약국을 매칭해주고 약국으로부터 플랫폼 입점비나 홍보 수수료를 받는 구조입니다. 그 부분은 각 약국과 제휴를 맺어야 가능한 부분입니다.

7.7구글 플레이스토어 리뷰 기준 사용자 불만 2가지

- 유사한 AI 비전 기반 알약 식별 및 의약품 정보 앱들에서 유저들이 가장 큰 불만으로 꼽는 것은 다음과 같습니다.

- 성능이 안좋아서 제대로 알약이 검출이 안돈다.

- -원인: 알약은 조명 상태, 촬영 각도, 카메라 화질, 그리고 알약 표면에 새겨진 미세한 식별 문자(각인)의 마모 상태에 따라 AI 인식이 매우 까다롭습니다. 유저들은 완벽한 스캔을 기대하지만 실제 결과가 빗나가는 것이 원인입니다.

- -해결책: 데이터 EDA 의 세부적인 세팅 및 머신 러닝 모델 성능 강화를 통한 검출 역량 향상시키는 것입니다.

7.8경쟁사 대비 마케팅 적으로 강조해야 할 강점 제안

- 강점 제안: 앱을 다운받고 별도의 개인정보 입력이나 회원가입 없이 카메라 촬영한 후 사진 한 장만 찍으면 시니어 층의 중복 투약 문제를 해결하고 가정 내 방치된 폐의약품 및 알약 오남용 방지 할 수 있다는 부분을 어필하면 될 것입니다.

— 이상 —