

# Pre-MVP Demo Spec

Contact: [limsihyun@kaist.ac.kr](mailto:limsihyun@kaist.ac.kr) or [kjh0209@kaist.ac.kr](mailto:kjh0209@kaist.ac.kr)  
작성자: KAIST 기술경영학부 김지혁, KAIST 전기및전자공학부 임시현

## 개발 단계 요약

1. 주행 화면 데이터 수집을 진행한다.
2. 데이터 라벨링을 수행한다.
3. YOLO를 학습한다: 이미지 → 라벨로 매핑되도록 학습한다.
4. platform\_sign, traffic\_sign에 대해 OCR을 수행해 표지판 세부 내용을 인식한다.
5. 라벨 결과와 OCR 결과를 결합해 instruction 생성 플랫폼을 구현한다.

## 주행 화면 데이터 수집

활용한 데이터셋은 다음과 같다.

- Kaggle: [bdd100k](#)
- 유튜브 주행 영상:
  - [\[4K\] Driving Downtown Seoul Gangnam, Teheran, Cheongdam POV](#)
  - [4K 주행영상 대전 불명동에서 둔산동까지 드라이브 DAEJEON CITY DRIVING DOWNTOWN KOREA ROAD 4K 60P](#)
  - [대전 서남부터미널에서 대전역까지 주행영상 8K 시내 드라이브 신호편집 8K DRIVE KOREA ROAD DRIVE ASMR](#)
  - <https://www.youtube.com/watch?v=l34xYxTqKfU>
  - [4K 주행영상 강원도 강릉시 GANGNEUNG CITY DRIVING DOWN TOWN KOREA ROAD ASMR 4K 60P](#)
  - [인천시청 - 인천공항터미널 주행영상 / ASMR Driving Incheon, Korea 4k60p](#)
  - 등을 사용한다.
- 유튜브 영상 다운로드:
  - [\[4K\] Driving Downtown Seoul Gangnam, Teheran, Cheongdam POV](#)
  - [4K 주행영상 대전 불명동에서 둔산동까지 드라이브 DAEJEON CITY DRIVING DOWNTOWN KOREA ROAD 4K 60P](#)
  - [대전 서남부터미널에서 대전역까지 주행영상 8K 시내 드라이브 신호편집 8K DRIVE KOREA ROAD DRIVE ASMR](#)
  - [강원도 강릉시 GANGNEUNG CITY DRIVING DOWN TOWN KOREA ROAD ASMR 4K 60P](#)
  - [인천시청 - 인천공항터미널 주행영상 / ASMR Driving Incheon, Korea 4k60p](#)
  - 등을 활용한다.

이번 데모에서는 약 3,500장의 이미지로 YOLO를 학습하며, 실서비스 수준의 성능을 위해서는 이보다 훨씬 더 많은 데이터 수집이 필요하다.

## 프레임 추출 및 전처리

- 영상은 원본 FPS 기준으로 1초당 1프레임씩 샘플링한다.
- 각 프레임마다 Laplacian 분산(선명도 지표)을 계산해 **90 이상**인 선명한 프레임만 선택한다.
- 이미지 데이터셋은 폴더를 재귀적으로 탐색하며 동일한 선명도 기준을 적용한다.
- 모든 프레임은 가로 1280px로 리사이즈하고 JPEG 품질 92%로 저장한다.

선명도 필터링 방식은 다음과 같다.

- 이미지를 그레이스케일로 변환한 뒤 Laplacian 연산자를 적용하여 엣지(경계)의 변화량 분산을 측정한다.
- 흐릿한 이미지는 분산이 낮고, 선명한 이미지는 분산이 높게 나타나므로 threshold 90을 기준으로 품질을 보장한다.

이 과정을 통해 다음을 달성한다.

- 유튜브 영상에서는 움직임이 적고 선명한 정적 장면만 자동으로 추출된다.
- Kaggle 데이터에서는 저품질 이미지를 자동 제거한다.
- 최종적으로 YOLO 학습에 적합한 고품질 데이터셋을 구성한다.

데이터 예시는 다음과 같다.





## 데이터 라벨링

클래스 정의는 다음과 같다.

- **platform\_sign:** 공항 플랫폼 등을 나타내는 표지판
- **traffic\_light:** 신호등
- **crosswalk:** 횡단보도
- **traffic\_sign:** 교통 표지판(속도 제한 등)
- **vehicle:** 자동차, 버스 등 차량
- **pedestrian:** 보행자

## 자동 라벨링 파이프라인

### 1단계: Florence-2 객체 감지

- Microsoft Vision-Language 모델 Florence-2의 `\<OD\>` 태스크로 이미지 내 일반 객체(차량, 보행자, 신호등, 표지판 등)를 감지한다.
- 반환된 바운딩 박스와 라벨을 매핑 함수로 6개 타겟 클래스(platform\_sign, traffic\_light, crosswalk, traffic\_sign, vehicle, pedestrian)로 정규화한다.
- 너무 작은 객체(10px 미만)는 노이즈로 간주해 제거한다.

### 2단계: EasyOCR 텍스트 인식

- Florence-2가 놓칠 수 있는 작은 텍스트(플랫폼 번호 등)를 EasyOCR로 보완한다.
- 이미지에서 텍스트를 감지한 뒤 정규식 `^\[A-Z\]\?-\?\d{1,2}\$`로 플랫폼 번호 패턴만 필터링한다. (예: "A-3", "12")
- O→O 치환 규칙으로 전형적인 OCR 오인식을 보정한다.
- 신뢰도 80% 이상인 결과만 platform\_sign 클래스로 추가한다.

### 3단계: YOLO 포맷 변환 및 저장

- Florence-2와 EasyOCR 결과를 병합해 YOLO 형식(`class_id center_x center_y width height`)으로 변환한다.
- xyxy 핀셀 좌표를 이미지 크기로 나누어 0~1 범위로 정규화하고 중심 좌표를 계산한다.
- 각 프레임마다 `.txt` 라벨 파일을 생성하여 `labels/train`, `labels/val` 디렉토리에 저장한다.
- Train/Val 분할은 10% 비율로 무작위 분할한다.

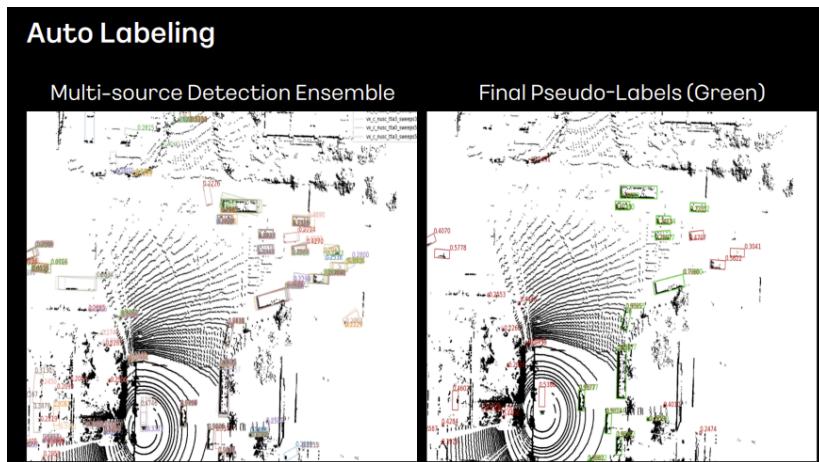
### 안전장치

- 10개 단위로 체크포인트를 저장하고 `os.fsync()`로 즉시 동기화하여, 세션 중단 시 최대 10개까지만 손실되도록 한다.
- 이미 처리한 파일 목록을 JSON으로 기록하여 재실행 시 자동으로 스킵한다.

### 정확도 및 활용 방식

- 자동 라벨링 예상 정확도는 **60~70% 수준**으로, 초안 생성을 위한 단계이다.
- 실 사용을 위해서는 샘플 검수와 Active Learning 기반 재라벨링으로 품질을 끌어올려야 한다.
- Florence-2는 공항 환경에 특화되지 않아 작은 플랫폼 표지판을 자주 놓치며, EasyOCR도 원거리·비스듬한 각도에서는 오인식 비율이 높다.
- 이를 보완하기 위해 사람이 데이터와 bounding box를 직접 검수하는 과정이 필수적이다.

- 검수 과정에서 카카오 자율주행 연구팀의 Object Detection용 데이터와 상호 보완적으로 협업할 수 있다. 이 경우, 우리는 교통 상황 판단 모델의 정확도를 높이고, 검수된 데이터셋은 카카오 자율주행 연구팀의 연구에도 기여한다.
  - c.f. 협력 가능한 카카오 자율주행 연구 분야: Auto Labeling



(출처: <https://if.kakao.com/2025/session?sessionId=43>)

예시 라벨은 다음과 같다.

```
5 0.966000 0.916000 0.033000 0.085000
10.861000 0.983000 0.019000 0.033000
```

- 1행: 보행자(클래스 5)가 이미지 우측 하단(중심 96.6%, 91.6%)에 있으며, 크기는 이미지 대비  $3.3\% \times 8.5\%$ 이다. 화면 기준으로 오른쪽 아래 모서리 근처의 사람이다.
- 2행: 신호등(클래스 1)이 이미지 하단(중심 86.1%, 98.3%)에 있으며, 크기는  $1.9\% \times 3.3\%$ 로 매우 작은 객체이다. 화면 기준 하단 중앙~우측의 멀리 있는 신호등을 의미한다.

## YOLO 학습

YOLO 학습은 전이학습(Transfer Learning) 방식으로 진행한다. COCO 데이터셋으로 사전 학습된 YOLOv8n(nano, 약 6MB 경량 모델)을 공항·도로 환경에 특화시키는 것을 목표로 한다.

YOLO를 선택한 이유는 다음과 같다.

- 객체 탐지 성능이 이미 검증된 모델이며 연구와 자료가 풍부하다.
- 추론 속도가 빨라 실시간(Real-time) 객체 탐지가 가능하다.

### 학습 설정

- 입력 이미지는 960px로 리사이즈한다.
- batch size는 -1로 두어 GPU 메모리에 맞게 자동으로 조정되도록 한다.
- 총 40 epoch 동안 학습한다.
- Cosine learning rate 스케줄러를 사용해 초반에는 빠르게, 후반에는 미세 조정을 수행한다.
- Early stopping patience는 12 epoch로 설정하여 검증 성능이 개선되지 않으면 조기 종료한다.
- `cache="ram"` 설정으로 데이터를 메모리에 캐싱해 I/O 오버헤드를 줄인다.

### 학습 메커니즘

- YOLOv8은 anchor-free 방식으로 각 그리드 셀에서 객체 중심, 너비, 높이를 직접 예측한다.
- 손실 함수 구성은 다음과 같다.
  - **Clou Loss:** 바운딩 박스 위치 손실
  - **BCE Loss:** 클래스 분류 손실
  - **DFL Loss:** 분포 초기 손실
- 학습 중 자동 Data Augmentation을 적용한다.

- Mosaic(4장 합성), MixUp(2장 블렌딩)
- Random Flip / Scale / HSV 변환
- 학습 중 자동 Data Augmentation:
  - Mosaic(4장 합성), MixUp(2장 블렌딩)
  - Random Flip / Scale / HSV 변환을 사용한다.

#### 평가 지표 및 모니터링

- mAP@0.5, mAP@0.5:0.95를 주요 지표로 사용한다.
- 클래스별 Precision, Recall, F1-score를 함께 확인한다.
- 학습 과정은 TensorBoard로 모니터링한다.
- 최고 성능 모델은 'best.pt', 마지막 체크포인트는 'last.pt'로 저장한다.

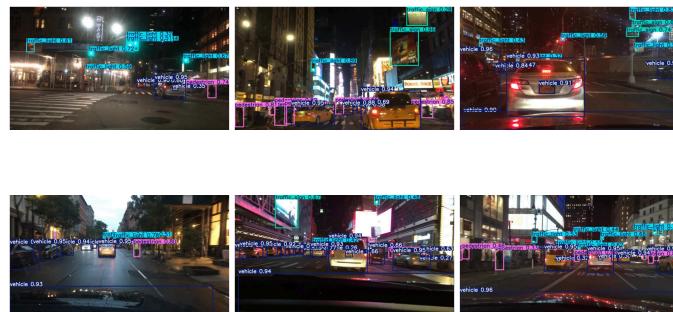
#### Active Learning (선택)

- 1차 학습 후 검증셋에서 Hard Case(False Negative, 낮은 Confidence, False Positive)를 자동으로 수집한다.
- 이 샘플을 수동으로 재라벨링한 뒤 2차 학습을 수행하면 일반적으로 mAP가 5~10%p 개선된다.
- 이 과정을 2~3회 반복해 mAP 0.85~0.90 수준의 실용 모델을 목표로 한다.

#### 전이학습 효과

- 사전학습 덕분에 From-scratch 대비 학습 시간은 약 1/5 수준으로 줄어들며, 요구 데이터는 약 1/10 수준으로 감소한다.
- 최종 모델은 약 6MB로, 라즈베리파이 등 엣지 디바이스에서도 30 FPS 이상 실시간 추론이 가능하다.

학습된 YOLO 모델 예시는 다음과 같다.



## OCR로 표지판 세부 내용 인식

- 감지된 platform\_sign, traffic\_sign 영역을 crop하여 OCR을 수행한다.
- 목적은 승객에게 정확히 어떤 플랫폼/어떤 표지판으로 이동해야 하는지 안내하기 위함이다.

#### 동작 방식 요약

- Tesseract **digits-only** 모드를 사용해 platform\_sign, traffic\_sign에서 숫자만 추출한다.

#### 전처리 및 다중 시도 전략

- YOLO가 감지한 bbox를 원본과 2.6배 확장 버전으로 나누어 crop한다.
- traffic\_sign의 경우 HSV 색 분할로 빨간 원형 테두리를 찾은 뒤, 내부 70% 영역만 추출한다. 실패 시 HoughCircles로 보완 시도를 진행한다.
- 각 crop에 대해 다음을 수행한다.
  - CLAHE 히스토그램 평활화를 적용한다.
  - adaptive threshold (블록 크기 31/51)를 적용한다.
  - inverted binary 등 5가지 전처리 variant를 생성한다.
- Tesseract PSM(Page Segmentation Mode) 7, 8, 6, 13을 모두 시도하여 총 20~30회의 OCR을 수행한다.
- 모든 결과에 `tessedit_char_whitelist=0123456789`를 적용해 숫자만 남기고, heuristic scoring으로 가장 그럴듯한 숫자를 선택한다.

- 길이 1~3자리면 +0.3점, 2자리면 추가 +0.2점 등 가중치를 부여한다.

#### 클래스별 처리 차이

- traffic\_sign의 경우 원형 내부 ROI → raw → expanded 순으로 시도한다.
- platform\_sign의 경우 raw / expanded 두 버전을 중심으로 처리한다.
- confidence와 bbox 크기를 기준으로 각 클래스당 상위 10개 detection만 처리하며, 숫자가 하나라도 안정적으로 읽히면 즉시 반환한다.

#### 한계 및 개선 방향

- traffic\_sign 중 숫자·문자가 아닌 그림 기반 표지판(예: 어린이 보호, 버스 정류장 등)은 현재 시스템으로는 구체 인지가 어렵다.
- 이를 개선하려면 YOLO 클래스 설계를 더 세분화해야 한다.
  - 예: traffic\_school\_zone\_sign, traffic\_bus\_station\_sign 등 세밀한 클래스를 정의한다.
- 그에 맞추어 데이터 전처리와 라벨링을 재설계하면, 의미 수준까지 포함한 안내 문장을 생성할 수 있다.

## Instruction 생성

YOLO 감지 결과와 OCR 숫자를 결합해 **우선순위 기반 안내 메시지를** 생성한다.

#### 핵심 로직

- platform\_sign에서 숫자가 읽히는 경우
  - "플랫폼 X 표지 방향으로 이동"을 최우선으로 사용한다.
- platform\_sign이 없고 traffic\_sign 숫자만 있는 경우
  - "근처 표지 숫자 'X' 방향으로 이동" 문구를 사용한다.
- 둘 다 실패하는 경우
  - "가까운 플랫폼 표지판이 보이는 방향으로 이동"과 같은 일반(generic) 메시지를 출력한다.

#### 보조 안전 정보 추가

- crosswalk, traffic\_light, pedestrian 클래스는 **존재 여부만** 활용한다.
  - 예: "횡단보도에서는 정지 후 좌우를 확인한다.", "신호등을 확인하고 이동한다.", "보행자 통행에 유의한다." 등의 안전 안내를 추가한다.

#### 거리 기반 세부 조정 (가설 단계)

- 승객-기사 간 Haversine 거리에 따라 표현을 다르게 구성한다.
  - 30m 이하: "매우 가깝다."
  - 30~80m: "약간만 더 이동하면 된다."
  - 80m 이상: "직진 이동을 계속한다."
- 현재 데모 단계에서는 실제 GPS를 연동하지 못해, 거리 구간은 임의 설정 상태이다.
- 추후 카카오 지도 API 등을 활용해 실제 GPS 기반 거리를 반영할 수 있다.

#### 최종 문장 구성 예시

- "플랫폼 12 표지 방향으로 이동한다. 횡단보도에서는 정지 후 좌우를 확인한다. 기사님과는 매우 가까운 거리이다."

## 데모 영상

[데모 영상 \(Google Drive\).](#)

Object Detection 결과는 다음과 같다.



예시 Instruction은 다음과 같다.

| 가까운 교통 표지판이 보이는 방향으로 이동한다. 신호등을 확인하고 안전하게 이동한다. 기사님 위치까지 직진 이동을 계속한다.

## 향후 계획

### 1. 객체 탐지 모델 성능 향상

- a. 더 엄격한 데이터 전처리로 데이터셋 품질을 향상한다.
- b. 데이터셋 규모를 확장한다.
- c. 클래스를 세분화해 더 구체적이고 정확한 instruction을 생성한다.

### 2. Instruction 구체화 및 정확도 향상

- a. 현재는 간단한 if문과 템플릿 기반 로직이며 GPS 정보는 미활용 상태이다.
- b. 차량과 승객의 GPS 정보를 활용해 "어느 방향으로 몇 m 이동"까지 포함하는 정량적 안내로 확장한다.
- c. Instruction 템플릿 선택을 위한 트리 기반 머신러닝 모델을 도입해 템플릿 매칭 정확도를 향상한다.
- d. 템플릿 풀을 더 촘촘하게 설계해 승객 상황에 맞는 템플릿 누락 케이스를 최소화한다.

### 3. OCR 정확도 향상

- a. 다양한 실험과 레퍼런스 조사를 통해 OCR 파이프라인과 알고리즘을 개선한다.
- b. platform\_sign, traffic\_sign OCR에 특화된 모델을 별도로 개발·미세조정하여 정확도를 향상한다.
  - 전용 데이터셋을 구축한 뒤 도메인 특화 OCR 모델을 학습하는 방향으로 발전시킨다.

면 됨.

### 1. A/B-로그 내장 개선 루프 개발

- a. 이벤트 로그, 실패 태깅, 리포트를 내장하도록 서버 개발
- b. 주 단위로 가설 → A/B → 개선을 반복하는 유지·보수 시스템 설정

### 2. UI 개발

- a. 승객 UI 개발: 행동 지시 instruction 표시 + 택시의 주행 화면 프레임 스냅샷 제공
- b. 기사 UI 개발: 전송된 내용과 예외 버튼 제공

### 3. (가능하다면) 카카오 자율주행 서비스와 연계

- a. 카카오 자율주행 AI 서비스 데이터 전처리 및 개발팀과 협력
- b. 데이터셋 공유, 모델 공유를 통한 Win-Win 전략 수립 가능

### 4. A/B-로그 내장 개선 루프 개발

- a. 이벤트 로그, 실패 태깅, 리포트를 내장하도록 서버 개발
- b. 주 단위로 가설 → A/B → 개선을 반복하는 유지·보수 시스템 설정

### 5. UI 개발

- a. 승객 UI 개발: 행동 지시 instruction 표시 + 택시의 주행 화면 프레임 스냅샷 제공

b. 기사 UI 개발: 전송된 내용과 예외 버튼 제공

6. (가능하다면) 카카오 자율주행 서비스와 연계

a. 카카오 자율주행 AI 서비스 데이터 전처리 및 개발팀과 협력

b. 데이터셋 공유, 모델 공유를 통한 Win-Win 전략 수립 가능