CYSHIELD

ID Card Line Extraction System

Technical Report

Computer Vision Assessment

Task: CVCY000

Submitted to

Cyshield – Computer Vision Team (Hiring Process)

Submitted by

Ahmed Gamal Noureddine

Email: eng.ahmed.gamal.cu@gmail.com

Date: 3 September 2025

Contents

Objective

End-to-end pipeline for extracting individual text regions from scanned Egyptian ID cards for downstream OCR.

Artifacts covered: rectify.py, derotate.py, roi_select.py, ROIs.json, roi_extract.py, infer_det.py, infer_seg.py, main.py, config.py, plus the three training notebooks (YOLOv8 detection, RT-DETR detection, YOLOv8 segmentation).

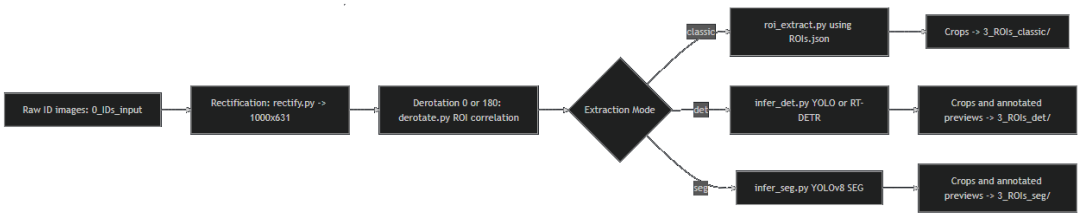


Figure 1. Full pipeline of the system

Full Project Workflow / Architecture

1) Input

- **Source:** Raw scanned or photographed Egyptian ID images.
- **Orientation:** Any orientation is accepted (from 0° to 360°).
- **Location:** Place images in 0_IDs_input/ (.png, .jpg, .jpeg, …).

Figure 2. Sample input for the system

2) Rectification — rectify.py

Purpose

Detect the card boundary, correct perspective, and standardize every image to a single coordinate frame.

Method

- Preprocess the image (grayscale, normalization, gamma adjustment for bright scans, CLAHE for local contrast).
- Denoise and find edges (median/bilateral filtering, Canny).
- Recover the card quadrilateral (contours + Hough Lines; intersect lines to get corners; check geometry).
- Apply a four-point perspective transform and resize to 1000×631 which is the standard dimension of Egyptian ID cards.
- Handles rotations internally, so downstream images share the same axes.
- Returns ID card with 0°/180°, so we will need another method to fix the 180° rotated cards

I/O

- **Input Images Directory:** 0_IDs_input/
- **Output Images Directory:** 1_IDs_rectified/ (dimensions: 1000×631)

Figure 3. The input image after card rectification

3) Derotation (0°/180°) — derotate.py

Purpose

Ensure the final correct orientation after rectification (fix 180° rotated images).

Method

- Load several pre-rectified correct reference images from 0_IDs_derotation_ref/.
- Compute the average value of a specific area of the correct images. This ROI area is a fixed rectangle in the 1000×631 frame. It is exactly the part of the image that contains green written words (جمهورية مصر العربية – بطاقة تحقيق شخصية)
- Compare each rectified image's green ROI area to the reference using correlation returning value within [-1, 1].
- Rotate 180° If the score is low (at certain threshold = 0.2) as they are not correlated, else if they are positively correlated (the score > 0.2) keep it as it is.

I/O

- **Correlation Reference Images:** 0_IDs_derotation_ref/ (upright, rectified samples)
- **Input Images Directory:** 1_IDs_rectified/
- **Output Images Directory:** 2_IDs_derotated/

Figure 4. The rectified card after derotation using correlation

4) ROI / Line Extraction Modes

We can extract crops in three ways. Pick one mode depending on your setup (**config.py**).

(a) Classical method - fixed ROIs: (set extract_option = "classic" in config.py)

- **Set ROIs once:** Use roi_select.py on any rectified sample to draw and name ROIs rectangles for cropping. This creates ROIs.json in the 1000×631 coordinate system.
- **Crop:** roi_extract.py applies those boxes to all derotated images and saves one file per ROI (firstname.png, second_name.png, address.png, id_number.png).

1_firstname.jpg 2_second_name.jpg 3_location.jpg 4_national_id.jpg

Figure 5. The lines extracted using the fully classical method

(b) Detection - YOLOv8s or RT-DETR-01: (set extract_option = "det" in config.py)

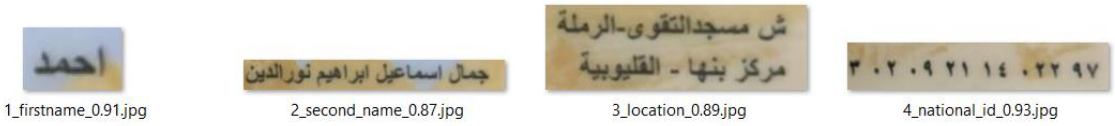
- **Model:** Use a trained detector (yolov8s or RT-DETR-01) to localize lines/regions.
- **Outputs:** A folder of crops for per input ID card first name, second name, address and ID number.

1_firstname_0.80.jpg 2_second_name_0.84.jpg 3_location_0.88.jpg 4_national_id_0.89.jpg

Figure 6. The lines extracted using the DL detection model (YOLOv8)

(c) Segmentation - YOLOv8s-SEG: (set extract_option = "seg" in config.py)

- **Model:** Instance segmentation with YOLOv8s-SEG.
- **Outputs:** A folder of rectangular crops for per input ID card.



1_firstname_0.91.jpg 2_second_name_0.87.jpg 3_location_0.89.jpg 4_national_id_0.93.jpg

Figure 7. The lines extracted using the DL segmentation model (YOLOv8)

Table 1. Overview & Setup comparison

Method	Training needed	Labels needed	Output type	ROIs flexibility	Use
Classical (fixed ROIs)	No	None	Named, fixed crops (from ROIs.json)	Low	Single known template; fastest path
YOLOv8s-det	Yes	Box labels	Rectangular detections + annotated previews	Medium	One/few layouts; good balance of speed/accuracy
RT-DETR-01	Yes	Box labels	Rectangular detections + annotated previews	Medium–High	Lines close together; cleaner deduplication
YOLOv8s-seg	Yes	Mask labels (or polygons)	Rectangular crops (masks in previews)	Medium–High	Need mask supervision and clear previews

5) Automation - main.py + config.py:

Purpose

Run the full pipeline in one go: **Rectify → Derotate → Extract**.

Run the full pipeline

- Set paths and options in config.py (input/output folders, mode, model weights).
- Run main.py. It stage in order.

Final Artifacts for OCR

After a full run, you'll have:

- 1_IDs_rectified/ perspective-corrected images at 1000×631
- 2_IDs_derotated/ upright images (0° or corrected 180°)
- One of:
    - 3_ROIs_classic/ named crops from ROIs.json
    - 3_ROIs_det/ detector crops + annotated previews
    - 3_ROIs_seg/ segmentation crops + annotated previews

Each folder per image contains the line crops ready to feed into OCR pipelines.

Dataset Choice and Reason

Availability & rationale

- High-quality, labelled Egyptian ID datasets are hard to find in public sources. Privacy concerns, legal limits, and the effort required for careful annotation make them rare.
- For this project, I used a Roboflow Universe dataset for Egyptian IDs: Egyptian ID Cards. Egyptian IDs Dataset

Source & format

- I downloaded from Roboflow in YOLOv8 format (images, labels, and a data.yaml file that defines train/val/test splits).
- Annotations are compatible with both detection and segmentation workflows, which matches the two deep learning-based extraction modes used in this project.

Why this dataset?

- **Domain match:** Images and labels reflect the real appearance of Egyptian IDs (colors, typography, and common field layouts).

- **Training-ready:** YOLOv8 export works directly with Ultralytics' training and evaluation code, so there's no extra conversion step.
- **Task alignment:** The labelled regions correspond to the areas we need to crop before running OCR, which keeps the pipeline simple and focused.

Splits & usage

- Training, validation, and test splits are defined in data.yaml.
- During training, we use Ultralytics' model.val() to compute standard metrics (mAP, precision, recall, F1) on the validation set.
- The **best** checkpoint from training is saved and later used by the inference scripts to produce the final crops.

Model Architectures

This project uses three Ultralytics models that cover box detection and instance segmentation. Each serves a slightly different role in the pipeline.

1) YOLOv8s - Object Detection

- **Backbone & neck:** C2f-based backbone with a FPN-style neck for multi-scale features.
- **Head:** Decoupled branches for classification and box regression; anchor-free predictions (objectness + class scores + box offsets).
- **Why it fits:** Small and fast, which helps when processing many IDs. It handles rectangular text regions on cards well and works reliably with the standardized 1000×631 images produced by the rectification step.

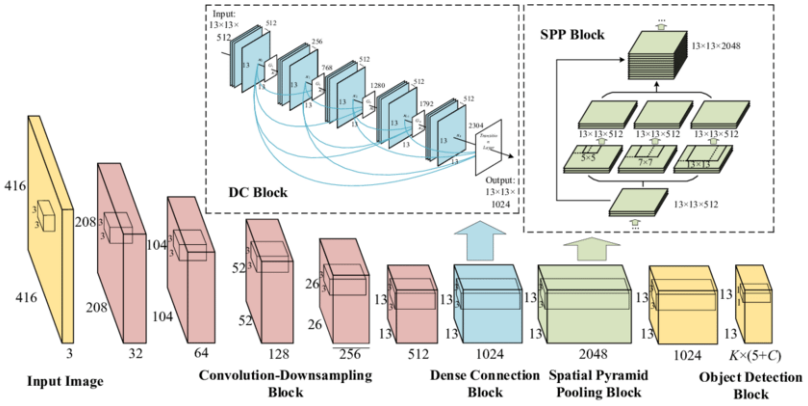


Figure 8. YOLOv8 Model Architecture Overview.

2) RT-DETR-01 - Object Detection (DETR family)

- **Backbone & transformer:** CNN feature extractor followed by a transformer encoder–decoder that uses learned queries to localize objects.
- **Assignment:** One-to-one Hungarian matching during training pairs each prediction with at most one ground truth.

- **Why it fits:** Good at separating adjacent or slightly overlapping text lines and reducing duplicate boxes. In this project.

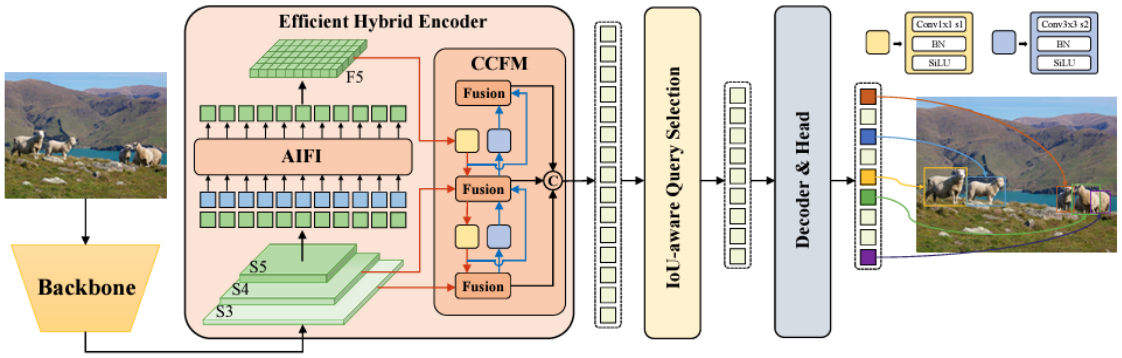


Figure 9. RT-DETR Architecture Overview

3) YOLOv8s-SEG - Instance Segmentation

- **Architecture:** The YOLOv8 detection core plus a prototype mask branch with per-instance coefficients.
- **Outputs in this project:** The training learns masks, but the inference scripts save rectangular crops for OCR.
- **Why it fits:** The mask branch captures the shape of text areas, which aids visual inspection and can support tighter crops later if needed.

Table 2. Model Size, Compute, and Training Settings

Method	Layers (train → fused)	Params	GFLOPs (train → fused)	Peak GPU mem	Image size	Epochs	Train time (hh:mm)
Classical (fixed ROIs)	N/A	N/A	N/A	N/A (CPU ok)	N/A	N/A	N/A
YOLOv8s-det	129	11.14M	28.7	5.05 GB	640	50	~0:06
RT-DETR-01	457	32.82M	108.0	13.7 GB	1024	50	~0:38
YOLOv8s-seg	151	11.79M	40.2	7.11 GB	1024	100	~0:30

Loss Functions

1) YOLOv8 - Detection

- **Box regression:**
  - IoU-based loss to improve geometric overlap between predicted and target boxes.
  - IoU-based box loss directly improves how well each rectangle covers a text line/region.
- **Objectness & classification:**

- Binary cross-entropy (with logits) to learn which locations contain a target and which class they belong to.
- BCE on objectness/class is a stable choice for one-stage detectors and helps balance recall with precision.

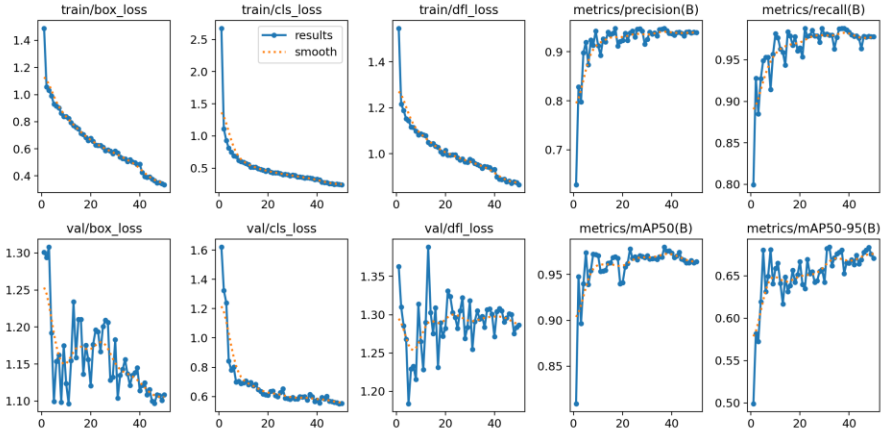


Figure 10. Loss plots of YOLOv8 Detector

2) RT-DETR-01 - Detection (DETR family)

- **Assignment:**
  - One-to-one Hungarian matching pairs each prediction with at most one ground-truth box.
  - The matching step reduces duplicate boxes and helps the model pick a single box per line, which matters when lines are adjacent or slightly overlapping.
- **Box regression:**
  - Combination of L1 (center/size) and IoU (overlap).
- **Classification:**
  - Cross-entropy on the matched queries.
  - The L1 + IoU mix improves both placement (centers and sizes) and coverage (overlap), giving cleaner crops for OCR.

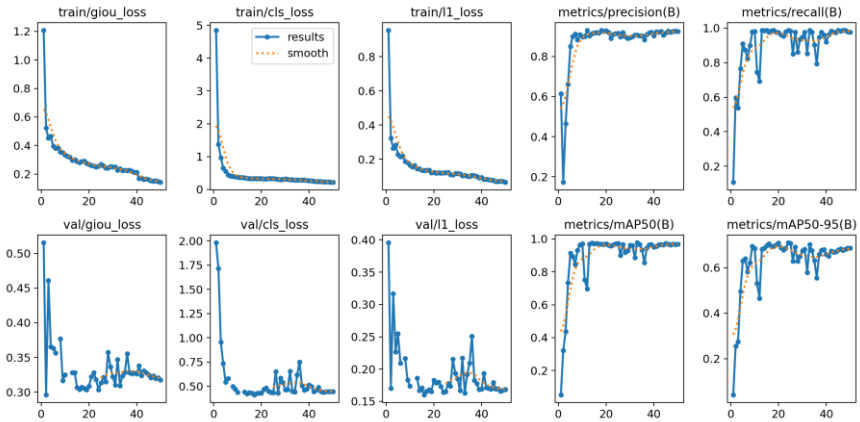


Figure 11. Loss plots of RT-DETR-01 Detector

3) YOLOv8 Segmentation

- **Detection heads:**
  - Same losses as YOLOv8 detection (IoU for boxes, BCE for objectness/class).
- **Mask branch:**
  - Per-pixel mask loss (Ultralytics defaults commonly use BCE and/or Dice on the prototype mask head).
  - Mask supervision teaches the model the shape of text regions. Even though we exports rectangular crops for OCR, learned masks improve what the model focuses on.

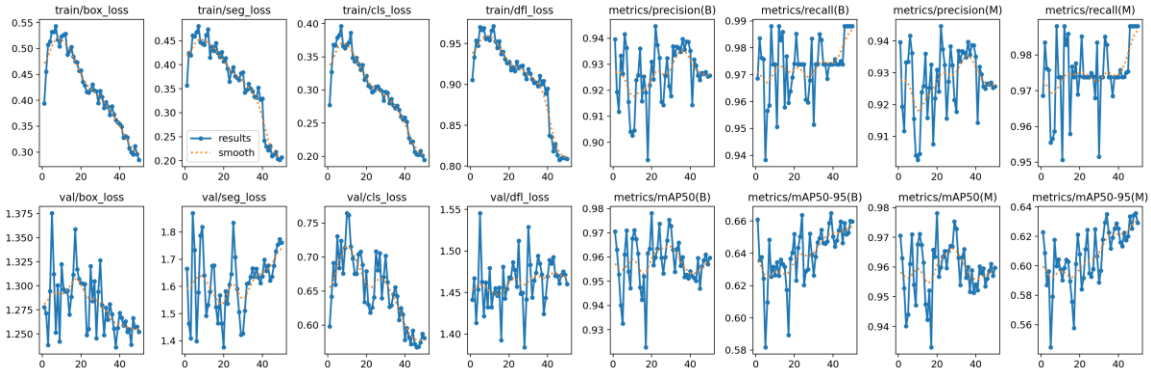


Figure 12. Loss plots of YOLOv8 segmentation model

Performance Analysis and Evaluation Metrics

During training

- Evaluation runs on the validation split defined in data.yaml. Ultralytics reports:
  - **mAP@0.5**: mean Average Precision at IoU 0.5 (how well boxes overlap the ground truth at a single threshold).
  - **mAP@0.5:0.95**: mAP averaged over IoU thresholds from 0.5 to 0.95 in steps of 0.05 (stricter and more informative).
  - **Precision / Recall / F1**: class-wise and overall.
  - For segmentation, mask metrics are also reported (mask **mAP@0.5** and **mAP@0.5:0.95**).
- **Model selection**: the checkpoint named "*best.pt*" is chosen automatically based on validation metrics. This is the weight file used for inference in the repo scripts.

Table 3. Loss Components at Final Epoch

Method	Box loss	Cls loss	DFL loss	L1 loss	Seg loss
Classical (fixed ROIs)	—	—	—	—	—
YOLOv8s-det (epoch 50/50)	0.3343	0.2399	0.8653	—	—
RT-DETR-01 (epoch 50/50)	GIoU 0.1428	0.2187	—	0.06624	—
YOLOv8s-seg (epoch 100/100)	0.3174	0.2141	0.8223	—	0.2047

Table 4. Validation Metrics

Method	Precision (P)	Recall (R)	mAP@0.5	mAP@0.5:0.95	Mask mAP@0.5	Mask mAP@0.5:0.95
Classical (fixed ROIs)	—	—	—	—	—	—
YOLOv8s-det	0.936	0.980	0.967	0.685	—	—
RT-DETR-01	0.924	0.988	0.971	0.705	—	—
YOLOv8s-seg (box)	0.943	0.984	0.981	0.666	—	—
YOLOv8s-seg (mask)	—	—	—	—	0.981	0.639

During inference (repo scripts)

- The pipeline saves 4 crops per ID card image. Filenames include the predicted class and confidence score, which makes it easy to scan results.
- The scripts print the number of output crops per ID card image. It is mainly 4 crops (first name, second name, location, ID number)
- Review each image for typical issues:
  - Missed lines (false negatives),
  - Extra boxes on background (false positives),
  - Box placement that clips characters or includes too much surrounding area.
- This visual check is the intended validation step in the repo's inference stage. If you run on your test images, it will provide a quick way to confirm that the trained model behavior matches the validation metrics reported during training.

Table 5. Inference Speed (per image)

Method	Image size	Inference time / image
Classical (fixed ROIs)	N/A	Very fast (CPU)
YOLOv8s-det	640	32.9 ms
RT-DETR-01	1024	85.6 ms
YOLOv8s-seg	1024	40.3 ms

System Limitations

- Fixed ROIs extraction method (fully classical) assumes one layout; precise and fast but not flexible like deep learning-based method that I used.
- Classical method of rectification may fail if the edges are not clear or even if the card image is already rectified, so I trained another model that localizes the ID card from the input image. I may consider putting it in the rectification script as a fall back if the classical method failed.
- The project works only for the front face of the Egyptian ID card because I did not find suitable labelled dataset that contains the two faces.
- Model generalization is limited by dataset coverage quality, which is not the best data quality, but it worked well when tested on different ID card images.

Appendix: Primary Entrypoints (for reproducibility)

```
Pipeline Orchestration:
python main.py (driven by paths and option in config.py)

Rectify:
python rectify.py <input_dir> <output_dir>

Derotate (180° check):
python derotate.py --ref_dir <derotation_ref_dir> --img_dir
<rectify_output_dir> --save_dir <derotated_dir>

Classical ROI Authoring (once):
python roi_select.py <sample_rectified.png> ROIs.json

Classical ROI Batch Crops:
python roi_extract.py <derotated_dir> ROIs.json --outdir
<classic_out_dir>

Detection Inference:
python infer_det.py --input_dir <derotated_dir> --save_dir
<det_out_dir>

Segmentation Inference:
python infer_seg.py --input_dir <derotated_dir> --save_dir
<seg_out_dir>
```