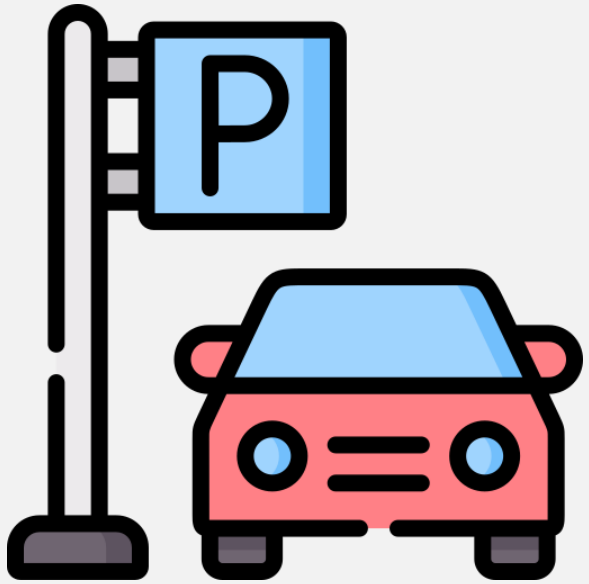
Computer Vision-Based License Plate Detection System for ASU Lot 59

CIS 515 | 04/28/2025

Team 005 - Edselmo Biondi, Ibtehaj U Deen, Kavya Murugan, Piyush Gautam, Vineeth Kalyanaraman

Content to be Covered

- Problem Overview and Current Challenges
- Scope, Stakeholders, and Value
- Solution Approach and Role of CV
- CV Models and System Design
- Model Training and Validation
- Challenges, Risks, and Mitigations
- Concept Demo and Success Metrics
- Deployment Setup and Cost Analysis
- Scalability and Future Outlook
- Team Contributions



The Challenge of Manual Car Registration Checks at Lot 59

Problem Statement

- Parking enforcement at Lot 59 currently relies on staff to manually check each license plate, making the process slow, costly, and prone to errors.
- As vehicle volume grows, this method becomes increasingly inefficient, leading to missed violations and inconsistent enforcement.
- Fun fact: “Lot 59 accommodates over **6,000 vehicles** daily.”

How Can We Quickly Spot Unregistered Cars Without Patrolling the Lot?

Resource: https://galvanizeit.org/project-gallery/arizona-state-university-lot-59?.com

The Challenge of Manual Car Registration Checks at Lot 59

Why We Chose This Problem

- Manual checks at Lot 59 are **slow, costly, error-prone, and unscalable**.
- Computer Vision offers a faster, scalable, and reliable solution.
- Helps create a fairer system by catching unregistered vehicles consistently.

Current Approach

- Staff manually walk or drive through Lot 59 to check plates.
- Manual enforcement is not scalable, leading to frequent errors.

Who Are the Relevant Stakeholders & Potential Value We Are Bringing in?

Scope, Stakeholders, and Value

Scope

Detects **registered vs unregistered** cars

Stakeholders

- ASU Parking Services
- ASU Admin

Values

- **Faster and more consistent** enforcement
- **Cost savings** by reducing manual labor
- Fairer treatment for all drivers
- Environmental benefit: Less patrolling = **less vehicle emissions**
- **Reduced human errors** in violation detection

How Can We Move from Manual Checks to Smart, Automated Enforcement?

Methodology We Applied to Solve the Underlying Problem

1. Data Setup	2. License Plate Detection	3. Character Recognition	4. Verification	5. Classification
				
Loaded registered license plates from CSV file.	Detected license plates in frames using **YOLOv8** and **OpenCV.**	Extracted plate numbers using **PaddleOCR** after preprocessing.	Matched extracted plate numbers against the registered list.	Classified vehicles as **"Registered"** or **"Unregistered"** and logged results.

Now, let's dig deeper into how the Computer Vision model made it possible.

Where CV Fits and Why It's Needed

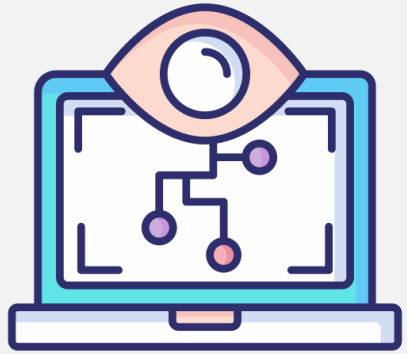
Role of Computer Vision in Our Solution

Need for Smarter Enforcement

- Manual license plate checks are slow, labor-intensive, and cannot keep up with Lot 59's traffic volume.

Limitations of Alternative Solutions

- Technologies like RFID tags or manual plate entry are expensive, rigid, and still prone to operational errors.
- They also require specialized equipment or vehicle modifications, adding hidden costs.



Advantages of Computer Vision

- CV automates detection and reading of plates using existing vehicle features - no special hardware needed.
- It provides a scalable, flexible, and low-cost solution suitable for large, open parking lots like Lot 59.

CV is essential to make license plate checks fast, scalable, and hands-free.

Overview of Our CV Models

CV Models Used in Our Solution

1) YOLOv8 (You Only Look Once Version 8)

- ⇨ Pre-trained object detection model
- ⇨ Used to locate the license plate on the car

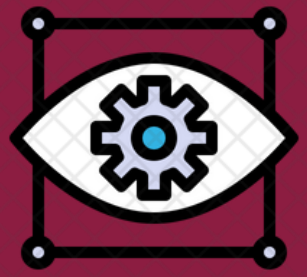
2) PaddleOCR

- ⇨ Open-source OCR (Optical Character Recognition) engine
- ⇨ Used to read the text from the detected license plate

We combined YOLOv8 for detection and PaddleOCR for reading to handle both detection and text extraction effectively.

Why These Models?

- Fast and lightweight models
- High accuracy on small plates
- Effective on noisy text
- Strong open-source support



Model Training, Validation, and Monitoring

Training	• **YOLOv8** pre-trained model: Pre-trained on general license plate datasets (e.g., OpenALPR) • **PaddleOCR**: Pre-trained on English alphanumeric text
Validation	• Internal Testing: Applied models on Lot 59 entrance/exit video clips (static images extracted) • Success Metrics: Plate detection success rate, OCR reading accuracy, matching success
External Validation	• Field testing with different lighting conditions and video frames per second (fps) • Manual sampling and cross-checking
Post-Deployment Monitoring	• Track false positives (flagged registered cars) and false negatives (missed unregistered cars) • Update OCR tuning or retrain if drift occurs

We plan continuous learning from real-world feedback

YOLO Reference: https://github.com/Muhammad-Zeerak-Khan/Automatic-License-Plate-Recognition-using-YOLOv8/blob/main/util.py

Model Outcomes, Interpretation, and Actions

Model Outcome	What It Means	Action Taken
True Positive (TP)	Unregistered car correctly flagged	Staff alerted
False Positive (FP)	Registered car wrongly flagged	Staff manually verify
False Negative (FN)	Unregistered car missed	Small risk accepted
True Negative (TN)	Registered car correctly ignored	No action needed

Clear outcome-action mapping speeds up enforcement and reduces mistakes, making the system faster and fairer.

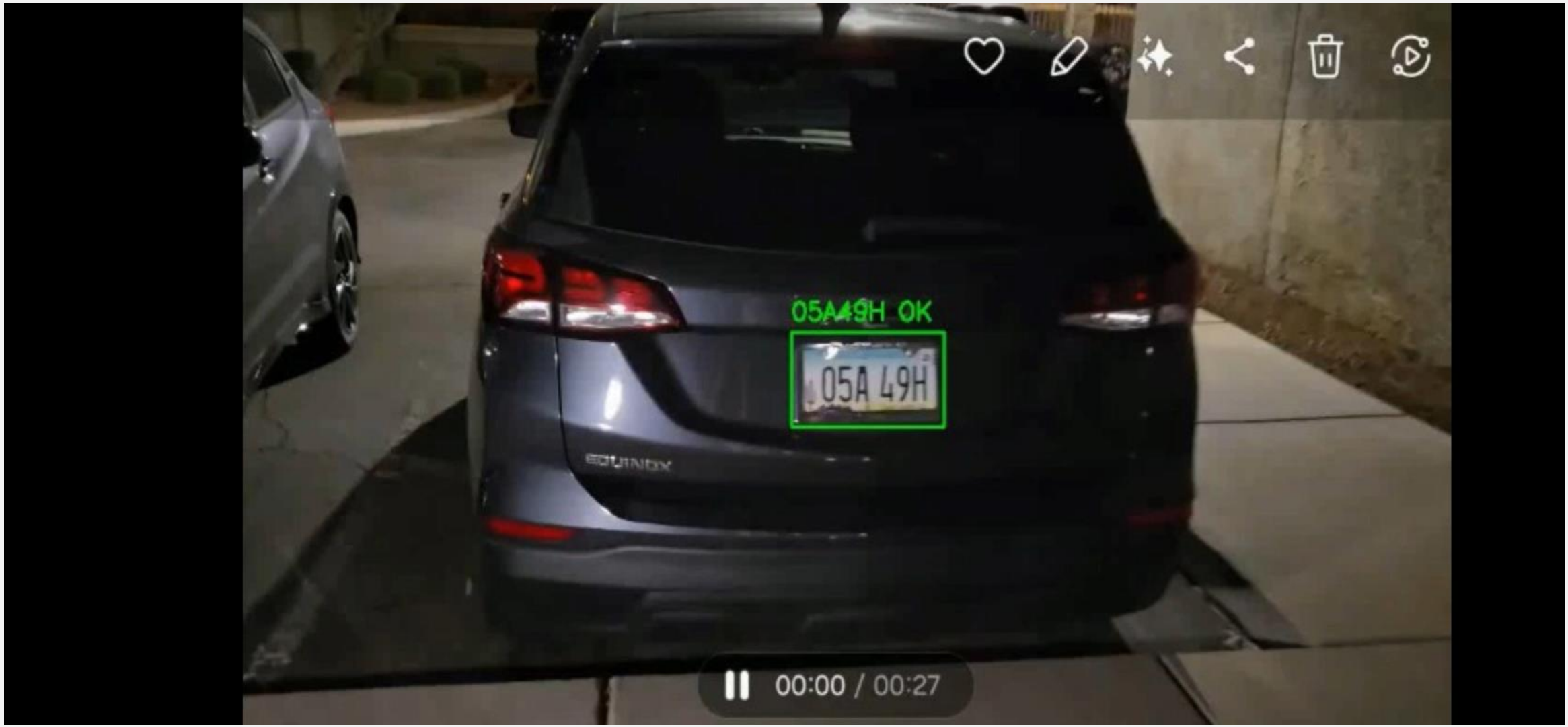
Biases, Challenges, and Mitigation Strategies

Challenge	Risk
Plate not visible (**angled, dirty**, faded)	Missed or **wrong OCR readings**
Temporary **paper tags**	OCR model struggles to read
Lighting changes (sun glare, night)	**Poor detection/reading quality**
Different plate designs (state variations)	OCR confusion, mismatches

Observation	Mitigation
Clean preprocessing improves OCR	Plan for **model re-training and live database updates** over time
Plate visibility matters most	Strongly recommend **head-in parking rules**
Video quality affects results	Use HD cameras and consistent camera angles
System needs flexibility	Upscaling and thresholding plates before reading

Real-world conditions introduce noise, but smart preprocessing and setup choices make the system reliable.

Proof of Concept Demo Video



Behind the Scenes

Debug Plate Crop

Debug: Grayscale Plate

Debug: Otsu Threshold

raw PaddleOCR results: [[[[220.0, 103.0], [541.0, 138.0], [536.0, 183.0], [215.0, 148.0]], ('ARiZONA', 0.6954571604728699)], [[162.0, 163.0], [635.0, 224.0], [612.0, 410.0], [140.0, 350.0]], ('TNA 49T', 0.9918074607849121)]….

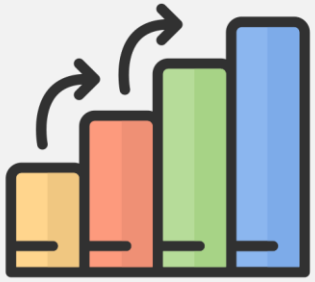
OCR raw concatenation:
ARIZONA**TNA49T**26ALTERNATVEPUEL

→ **matched plate:** TNA49T

Success Metrics for License Plate Detection System

Metric	Target
Plate Detection Accuracy	≥ 95% (plates correctly detected in frames)
OCR Reading Accuracy	≥ 92% (plate numbers correctly read by OCR)
Registered vs Unregistered Matching Accuracy	≥ 90% (correct flagging of vehicles)
Reduction in Manual Patrol Time	≥ 70% reduction compared to current manual system
Staff Efficiency Improvement	≥ 50% increase (monitor more cars in less time)
Real-Time Alert Latency	≤ 5 seconds (from capture to alert generation)

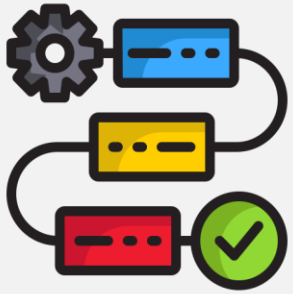
High accuracy, fast alerting, and reduced labor will validate the system's success at Lot 59.



Workflow Updates

Aspect	Old Workflow (Manual Checks)	New Workflow (Cart-Based CV System)
Plate Capture	Manually check parked vehicles	Staff patrols Lot 59 using a cart-mounted camera system
Plate Reading	Staff visually read and note plates	YOLOv8 **detects plate** → PaddleOCR **reads text** from captured images
Registration Check	Manual lookup and entry into system	Automatic matching against the registered database on cart computer/cloud
Violation Detection	Staff manually issue citations	System flags unregistered plates in real-time
Staff Effort	High walking/patrolling effort	Low physical effort (just drive cart slowly)
Scalability	Limited by human effort	Scalable with minimal additional resources

The new cart-based system reduces manual errors, saves staff effort, and improves speed while keeping mobility flexible.



Cart-Based Camera Setup for License Plate Detection

- **Axis P1445-LE-3** and **Hikvision DS-2CD4A26FWD-IZS/P** are potential camera models selected for high-accuracy license plate capture.
- Cart is human-driven initially, with potential for future automation.
- Real-time alerts are generated instantly when unregistered vehicles are detected, enabling quick staff action.
- Existing ASU carts can be retrofitted for license plate detection; Club Car Villager 2 and Carryall 500 are potential models if new carts are needed.

Costing & Resources

Item	Qty	Unit Cost	Total Cost	Comments
Carts (Club Car Villager 2 or Carryall 500)	2	~$10,000 each	$20,000	New electric carts
Cameras (Axis P1445-LE-3 or Hikvision)	2	~$700 each	$1,400	License plate optimized outdoor cameras
DC-to-AC Inverters (for powering devices)	2	~$150 each	$300	Connects electronics to cart battery
Computers (Laptop or Mini-PC)	2	~$900 each	$1,800	For **YOLOv8 + PaddleOCR** processing
Solar Panel (ECO-Worthy 390W)	2	~$160 each	$320	Day time power-savings
Misc. (Mounting kits, wiring, signage)	-	~$500	$500	Mounts, wiring, installation accessories
Total			**$24,320**	Total estimated hardware investment

Assuming only hardware costs here, potential savings in human labor could be incorporated to calculate the payback period and future savings.

Potential Risks & Mitigation Strategies

Risk	Potential Issue	Mitigation Strategy
Over-Reliance on Automation	Staff may over-rely on system alerts, potentially missing rare errors.	Require periodic manual **random checks** by staff.
Missed Violations	Poor visibility (dust, weather, dirty plates) could cause OCR/detection failures.	Regular camera maintenance and system performance **audits**.
Privacy and Data Security	Collection of license plate data raises personal data protection concerns.	Encrypt stored data, restrict access to authorized ASU Parking Services personnel only.
Lack of Community Transparency	Students and staff may feel uncomfortable with surveillance.	Public notices and clear communication about system use and **data privacy** measures.

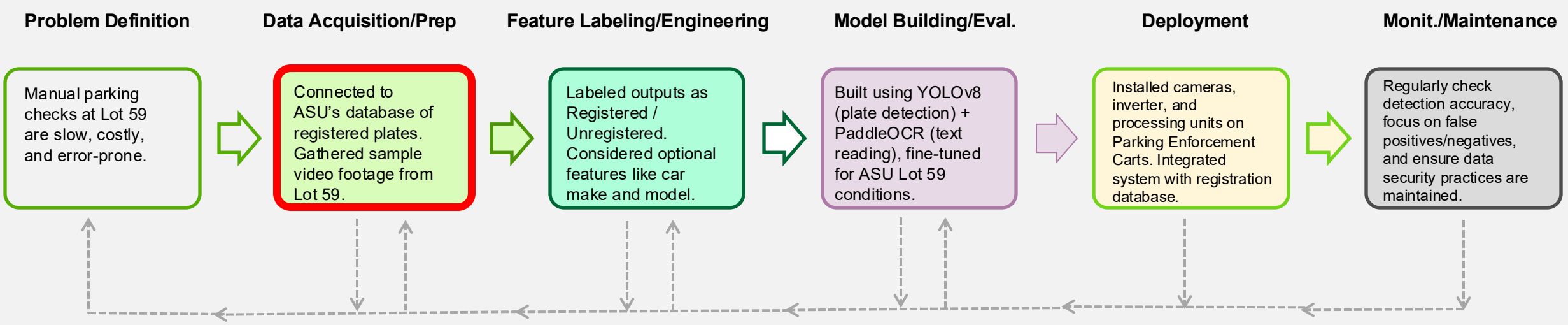
Smart oversight and strong privacy safeguards are essential for successful deployment.

Similar Solutions and Scalability

Examples	Applications
Airports (LAX, JFK, Phoenix Sky Harbor)	Automatic License Plate Recognition (ALPR) systems for parking management and security.
Universities (UC Berkeley, MIT)	Camera-based license plate scanning for campus parking enforcement.
Retail Locations (e.g., Scottsdale Fashion Square)	ALPR systems for ticketless parking and managing parking sessions without human checks.

The solution is simple to deploy and scalable across locations with less incremental investment and effort.

Final Summarized Project Slide



Continuous Feedback Loop with Domain Experts and Stakeholders

Team Ownership Structure?

Name	Nature of Work	Input%
Vineeth Kalyanaraman	Coding	20%
Piyush Gautam	Coding	20%
Edselmo Biondi	Coding	20%
Kavya Murugan	Content	20%
Ibtehaj U Deen	Content	20%

Everyone participated equally in the logic; however, we divided the coding and content (presentation) into two sub-groups, a learning from last quarter's project, and we all aligned on this final deck.

Thank you, have a Great Summer Break!!!

Appendix

```
import cv2
from paddleocr import PaddleOCR
import pandas as pd
from ultralytics import YOLO
import matplotlib.pyplot as plt
```

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
    from google.colab import files
```

```
Mounted at /content/drive
```

```
# 1. Load your "registered" list from CSV

df = pd.read_csv("/content/drive/My Drive/CIS 515/Test-Project/registered_plates.csv")
registered_set = set(df["plate"]
                .str.replace(r"[^A-Za-z0-9]", "", regex=True)
                .str.upper())
registered_set
```

```
{'05A49H',
 '9BA73G',
 'ABC123',
 'AXU4238',
 'HRA3YM',
 'PKA94U',
 'T3A2EZ',
 'TNA49T'}
```

```
[ ] # 2. Initialize YOLOv8 model for plate detection

    model = YOLO("/content/drive/My Drive/CIS 515/Test-Project/license_plate_detector.pt")

    # 2b. Initialize PaddleOCR
    ocr = PaddleOCR(use_angle_cls=True, lang='en')
```

```
download https://paddleocr.bj.bcebos.com/PP-OCRv3/english/en_PP-OCRv3_det_infer.tar to /root/.paddleocr/whl/det/en/en_PP-OCRv3_det_infer/en_PP-OCRv3_det_infer.tar
100%|██████████| 3910/3910 [00:16<00:00, 241.73it/s]
download https://paddleocr.bj.bcebos.com/PP-OCRv4/english/en_PP-OCRv4_rec_infer.tar to /root/.paddleocr/whl/rec/en/en_PP-OCRv4_rec_infer/en_PP-OCRv4_rec_infer.tar
100%|██████████| 10000/10000 [00:17<00:00, 557.22it/s]
download https://paddleocr.bj.bcebos.com/dygraph_v2.0/ch/ch_ppocr_mobile_v2.0_cls_infer.tar to /root/.paddleocr/whl/cls/ch_ppocr_mobile_v2.0_cls_infer/ch_ppocr_mobile_v2.
100%|██████████| 2138/2138 [00:14<00:00, 146.11it/s][2025/04/25 22:52:58] ppocr DEBUG: Namespace(help='==SUPPRESS==', use_gpu=False, use_xpu=False, use_npu=False, use_mlu
```

```
[ ] # 3. Open video file (or camera stream)
    cap = cv2.VideoCapture("/content/drive/My Drive/CIS 515/Test-Project/My movie 2.mp4")
    fps    = cap.get(cv2.CAP_PROP_FPS) or 20.0  # fallback to 20 if unreadable
    width  = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fourcc = cv2.VideoWriter_fourcc(*"mp4v")
    out = cv2.VideoWriter("output.mp4", fourcc, fps, (width, height))
```

Appendix

```
# 4 Setup PaddleOCR to Read the Text to Video
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    results = model(frame, conf=0.5, iou=0.45)
    for box in results[0].boxes.data:
        x1, y1, x2, y2, *_ = box
        x1, y1, x2, y2 = map(int, (x1, y1, x2, y2))

        # Add padding and crop
        pad = 10
        crop = frame[
            max(0, y1 - pad): y2 + pad,
            max(0, x1 - pad): x2 + pad
        ]

        # 4a Upscale for better OCR resolution
        crop = cv2.resize(crop, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)

        # DEBUG: show the crop
        plt.imshow(cv2.cvtColor(crop, cv2.COLOR_BGR2RGB))
        plt.title("Debug Plate Crop")
        plt.axis("off")
        plt.show()

        # 4b Grayscale + simple Otsu threshold (black-on-white)
        gray = cv2.cvtColor(crop, cv2.COLOR_BGR2GRAY)

        # — DEBUG #1: show the gray image —
        plt.figure(figsize=(4,4))
        plt.imshow(gray, cmap="gray")
        plt.title("Debug: Grayscale Plate")
        plt.axis("off")
        plt.show()

        _, thresh = cv2.threshold(
            gray, 0, 255,
            cv2.THRESH_BINARY + cv2.THRESH_OTSU)

        # — DEBUG #2: show the thresholded image —
        plt.figure(figsize=(4,4))
        plt.imshow(thresh, cmap="gray")
        plt.title("Debug: Otsu Threshold")
        plt.axis("off")
        plt.show()
# ——————————————————————————————
        # Ensure black text on white background

        # 5 Remove noise with a small opening
        thresh = 255 - thresh  # ensure black text on white
        kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2,2))
        thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)


        # 6 OCR with PaddleOCR
        # PaddleOCR expects BGR or RGB images
        # Convert the single-channel `thresh` back to BGR:
        thresh_bgr = cv2.cvtColor(thresh, cv2.COLOR_GRAY2BGR)
        ocr_results = ocr.ocr(thresh_bgr, det=True, rec=True, cls=False)

        # DEBUG: print out the raw PaddleOCR results structure
        print("raw PaddleOCR results:", ocr_results)

        raw = ""
        if ocr_results and ocr_results[0]:
            for _, (txt, _) in ocr_results[0]:
                raw += txt

        # 6.x.1) Clean to alphanumeric uppercase
        clean = "".join(ch for ch in raw if ch.isalnum()).upper()
        print("OCR raw concatenation:", clean)

        # 6.x.2) Find the one plate in your registered_set that appears inside
        plate_text = ""
        for reg in registered_set:
            if reg in clean:
                plate_text = reg
                break

        print(" → matched plate:", plate_text or "NONE")
```

```
        print(" → matched plate:", plate_text or "NONE")

        # 7. Registration check & annotate
        is_reg = plate_text in registered_set
        color = (0,255,0) if is_reg else (0,0,255)
        label = f"{plate_text or 'UNK'} {'OK' if is_reg else 'UNREG'}"
        cv2.rectangle(frame, (x1,y1), (x2,y2), color, 2)
        cv2.putText(frame, label, (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)


    out.write(frame)

cap.release()
out.release()

files.download('output.mp4')
```

Debug: Otsu Threshold


```
[2025/04/25 22:55:42] ppocr DEBUG: dt_boxes num : 3, elapsed : 0.16173100471496582
[2025/04/25 22:55:43] ppocr DEBUG: rec_res num  : 3, elapsed : 0.29256296157836914
raw PaddleOCR results: [[[[201.0, 81.0], [421.0, 57.0], [423.0, 78.0], [203.0, 102.0
OCR raw concatenation: FTORAT3A2EZ
 → matched plate: T3A2EZ
```