Egyptian License Plate Recognition System

Project Documentation Summary

Egyptian License Plate Recognition System

Project Overview

This project presents a complete AI solution for automatic Egyptian license plate recognition. The system uses an ESP32-CAM with ultrasonic sensor to detect vehicles, captures images, processes them through a cloud-based YOLOv10 model, and displays results on a real-time dashboard.

[IMAGE: System Overview]

Caption: High-level system architecture

Problem Statement

Egyptian license plates use Arabic characters and numerals, presenting unique challenges for automated recognition:

- Right-to-left reading direction
- Arabic-Indic numerals (٠١٢٣٤٥٦٧٨٩)
- 16 possible Arabic letters specific to Egyptian plates
- Variable lighting and environmental conditions

Solution Overview

The system addresses these challenges through:

- Custom-trained YOLOv10 models for detection and OCR
- Edge-based triggering with ESP32-CAM
- Cloud processing for ML inference
- Real-time monitoring dashboard

Key Achievements

Metric	Value
Detection Precision	95.88%
Detection Recall	93.98%
mAP@50	97.50%

mAP@50-95	74.94%
Dataset Size	1,800+ images

Technology Stack

Layer	Technology
Hardware	ESP32-CAM, HC-SR04 Ultrasonic
ML Framework	YOLOv10 (Ultralytics)
Backend	FastAPI (Python)
Database	PostgreSQL (Neon)
Storage	Cloudinary CDN
Frontend	React + TypeScript
Deployment	Railway

Live Deployments

- **API**: https://robotics-egyptian-lpr-production.up.railway.app
- **Dataset**: https://universe.roboflow.com/vguard-vxduz/egyptian-license-plate-detection

---

Dataset Preparation

Dataset Overview

Metric	Value
Total Images	1,800+
Annotation Platform	Roboflow
Format	YOLOv10 compatible
Classes	License plate region

Images were manually collected showing Egyptian license plates in various conditions: different lighting, angles, vehicle types, and distances.

Collection Strategy

The dataset includes images from:

- Various vehicle types (cars, trucks, motorcycles)
- Different lighting conditions (daylight, shade, evening)
- Multiple angles and distances
- Urban and highway environments

Annotation Process

Each image was annotated using Roboflow with bounding boxes around the license plate region. The annotations were exported in YOLOv10-compatible format.

Model Training

Model Architecture: YOLOv10n (Nano)

YOLOv10 was selected for this project due to:

- **Real-time Performance**: Optimized for fast inference
- **Accuracy**: State-of-the-art detection accuracy
- **Lightweight**: Nano variant suitable for cloud deployment
- **NMS-free**: End-to-end design reduces complexity

Specification	Value
Parameters	~2.3M
FLOPs	~8.2G
Input Size	640×640

Training Configuration

Parameter	Value
Epochs	150
Batch Size	8
Input Resolution	640×640
Patience	10 (early stopping)
Optimizer	AdamW

Augmentation Strategy

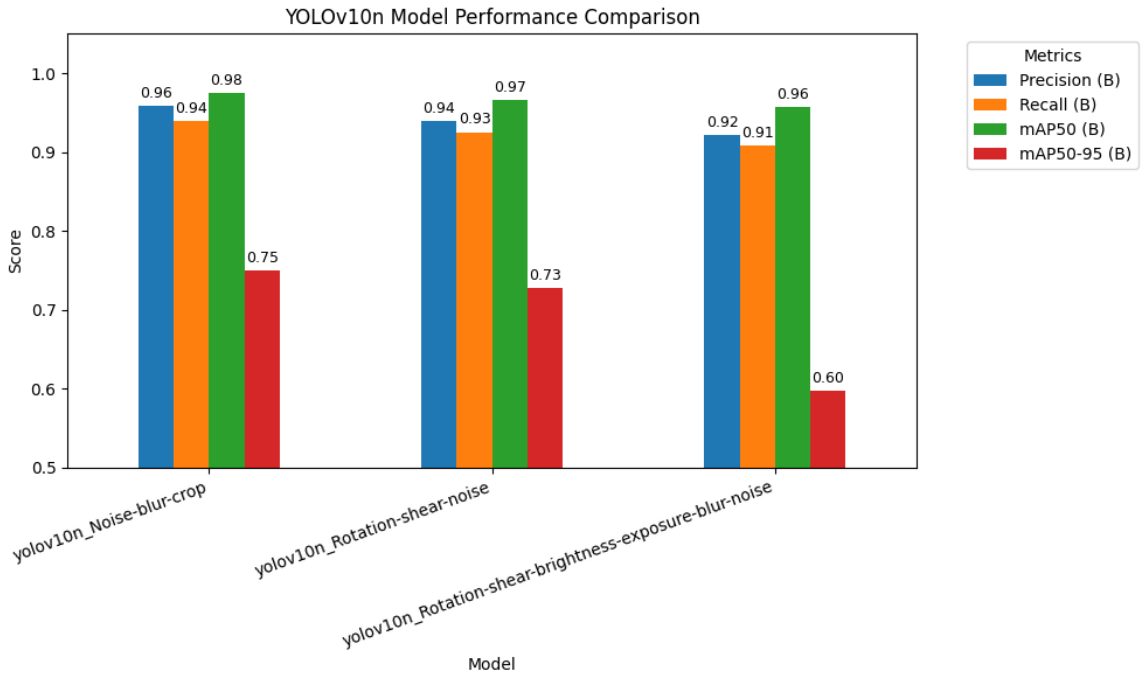
All built-in YOLOv10 augmentations were disabled. Three dataset-level augmentation versions were tested:

Augmentation	Parameters
HSV/Rotation/Scale	0 (disabled)
Flip/Mosaic/Mixup	0 (disabled)

Model Versions Comparison

Model Version	Precision	Recall	mAP@50	mAP@50-95
Noise-Blur-Crop	**0.9588**	**0.9398**	**0.9750**	**0.7494**
Rotation-Shear-Noise	0.9396	0.9253	0.9664	0.7273
Full Augmentation	0.9214	0.9081	0.9565	0.5966

Best Model: Noise-Blur-Crop achieved highest scores across all metrics. The simpler augmentation strategy outperformed aggressive augmentations, suggesting excessive transformations introduced too much variation for the dataset size.



Caption: Performance metrics comparison across model versions

---

Arabic OCR Model

Character Classes

The OCR model recognizes 26 classes (16 Arabic letters + 10 Arabic numerals):

Arabic Letters Used on Egyptian Plates

Class	Arabic	Name
a	أ	Alef
b	ب	Ba
g	ج	Jeem
d	د	Dal
r	ر	Ra
s	س	Seen
ss	ص	Sad
tt	ط	Ta
aa	ع	Ain
f	ف	Fa
kk	ق	Qaf
l	ل	Lam
m	م	Meem
n	ن	Noon
00	ه	Ha
w	و	Waw
y	ى	Ya

Arabic-Indic Numerals

Class	Arabic	Western
0	٠	0
1	١	1
2	٢	2
3	٣	3
4	٤	4
5	٥	5
6	٦	6
7	٧	7
8	٨	8
9	٩	9

Character Extraction Process

1. Crop license plate region from detection model
2. Resize to 640×640 for OCR model input
3. Detect individual character bounding boxes
4. Sort characters right-to-left (Arabic reading direction)
5. Map Latin predictions to Arabic Unicode
6. Validate 6-7 character Egyptian plate format

Egyptian plates follow a standard format with letters on the left side and numbers on the right side.

Backend API (FastAPI)

Architecture

The backend is built with FastAPI, a modern Python web framework. It handles image processing, license plate recognition, database storage, and serves data to the dashboard.

Production URL: `https://robotics-egyptian-lpr-production.up.railway.app`

Core Components

Component	Responsibility
FastAPI App	HTTP request handling, routing
LPR Service	License plate detection and OCR
Database Module	PostgreSQL async operations
Cloudinary Helper	Image upload to CDN

Detection Pipeline

The backend processes images through a three-stage pipeline:

```
Image → Plate Detection → Crop & Resize → Arabic OCR → Result
```

Stage 1: Plate Detection

- YOLOv10 model detects plate bounding box

- Model: yolov10_license_plate_detection.pt

Stage 2: Crop & Preprocess

- Extract plate region from image
- Resize to 640×640 for OCR input

Stage 3: Arabic OCR

- Detect individual characters
- Map to Arabic Unicode
- Sort right-to-left

API Endpoints

Endpoint	Method	Purpose
`/`	GET	Health check
`/api/recognize`	POST	Process image, return plate text
`/api/dashboard`	GET	Retrieve all detection records

Recognition Endpoint

Request: Multipart form data with image file

Response:

```
{
  "success": true,
  "plate": "٣ ٢ ١ ج ب أ",
  "confidence": 0.95,
  "image_url": "https://cloudinary.com/...",
  "error": null
}
```

Dashboard Endpoint

Response:

```
{
  "vehicles_today": 42,
  "last_detection": "2024-12-24T15:30:45.123Z",
  "detections": [
    {
      "id": 1,
      "car_image": "https://cloudinary.com/car.jpg",
      "lp_image": "https://cloudinary.com/plate.jpg",
      "lp_number": "٣ ٢ ١ ج ب أ",
      "confidence": 0.95,
      "created_at": "2024-12-24T15:30:45.123Z"
    }
  ]
}
```

Dependencies

Package	Purpose
fastapi	Web framework
uvicorn	ASGI server
ultralytics	YOLOv10 models
opencv-python-headless	Image processing
asyncpg	Async PostgreSQL driver
cloudinary	Image upload SDK

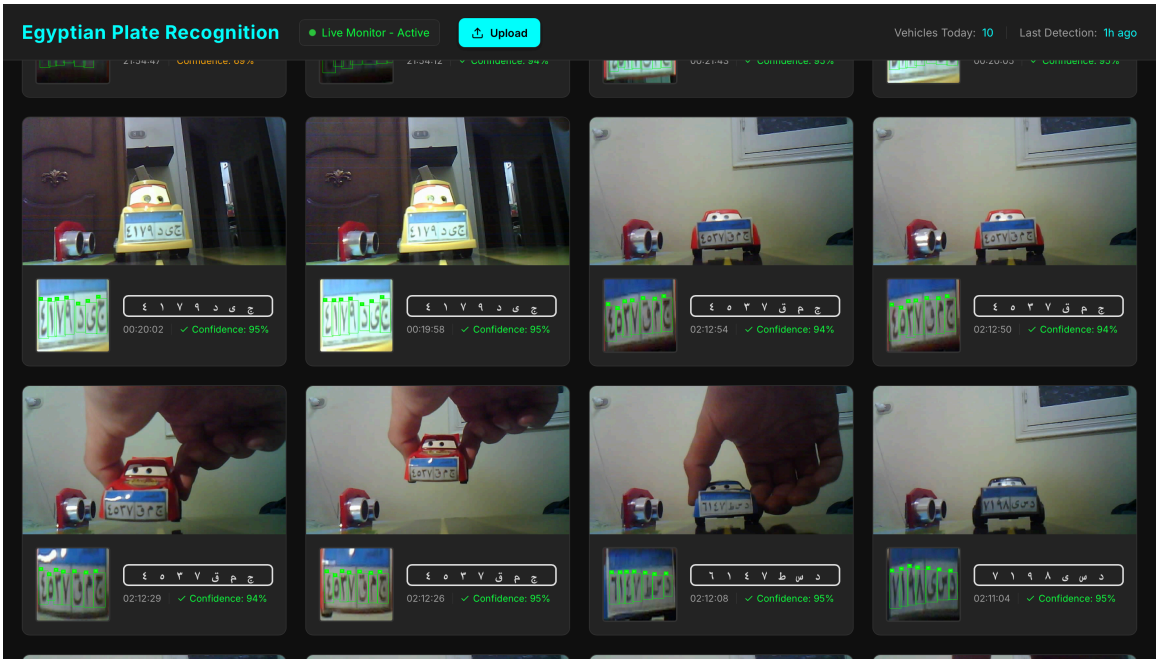
Dashboard (React)

Technology Stack

- **Framework**: React with TypeScript
- **Build Tool**: Vite
- **Styling**: Tailwind CSS
- **Theme**: Dark cyberpunk design

Features

- Real-time monitoring with auto-refresh every 30 seconds
- Detection cards showing:
- Vehicle image / Cropped plate image
- Arabic plate number
- Confidence percentage
- Timestamp
- Today's vehicle count display



Caption: Dashboard showing detection results

Responsive Design

The dashboard uses a responsive grid layout that adapts to different screen sizes, from mobile to desktop.

Hardware Setup

Components List

Component	Specification	Quantity
Microcontroller	ESP32-CAM (AI-Thinker)	1
Camera Module	OV2640	1
Distance Sensor	HC-SR04 Ultrasonic	1
Voltage Regulator	7805 (5V output)	1
Batteries	18650 Li-ion (3.7V)	2
Resistors	1kΩ (1/4W)	3
FTDI Programmer	USB-to-Serial	1

GPIO Pin Assignments

GPIO	Function	Connection
GPIO 13	TRIG Output	HC-SR04 trigger pin
GPIO 12	ECHO Input	HC-SR04 echo (via voltage divider)
GPIO 4	Flash LED	Built-in LED (PWM controlled)
GPIO 0	Boot Mode	GND for programming

Power Supply

Two 18650 batteries connected in series provide approximately 7.4V. The 7805 voltage regulator steps this down to stable 5V for the ESP32-CAM and HC-SR04.

7805 Pin	Connection
Input	Battery positive (~7.4V)
Ground	Common ground
Output	5V to ESP32-CAM and HC-SR04

Voltage Divider Circuit

Critical: The HC-SR04 outputs 5V on ECHO, but ESP32 GPIO is 3.3V tolerant. A voltage divider is required.

```
ECHO (5V) ---[1kΩ]---+---[2kΩ]--- GND
                     |
                GPIO 12 (3.3V)
```

Calculation:

```
V_out = V_in × (R2 / (R1 + R2))
V_out = 5V × (2kΩ / 3kΩ)
V_out = 3.33V ✓ (safe for ESP32)
```

Detection Parameters

Parameter	Value	Purpose
Trigger Distance	15 cm	Maximum distance to trigger capture
Minimum Distance	2 cm	Ignore noise (very close readings)
Debounce Delay	3000 ms	Wait between captures
Camera Resolution	SVGA (800×600)	With PSRAM

Distance Calculation

The ultrasonic sensor measures time-of-flight:

```
Distance = (Duration × Speed of Sound) / 2
Distance = (Duration × 0.034 cm/µs) / 2
```

Three readings are taken and averaged for stability.

Camera Configuration

Condition	Resolution	Quality	Frame Buffers
With PSRAM	SVGA (800×600)	10	2
Without PSRAM	VGA (640×480)	12	1

Flash LED Control

Brightness	PWM Value
Off	0
25%	64
50%	128
75%	192
100%	255

Deployment Architecture

Cloud Services

Service	Platform	Purpose
API Backend	Railway	Image processing, ML inference
Dashboard	Railway	Web interface
Database	Neon	PostgreSQL storage
Image Storage	Cloudinary	CDN hosting

Railway Platform

Railway provides container-based hosting with automatic deployments from Git.

API Container Configuration:

```
FROM python:3.11-slim
RUN apt-get update && apt-get install -y libgl1 libglib2.0-0
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY api/ ./api/
COPY *.pt .
CMD uvicorn main:app --host 0.0.0.0 --port $PORT
```

Features:

- Auto-scaling for variable traffic
- Automatic HTTPS/SSL certificates
- Environment variable management
- Built-in logging and monitoring

Neon PostgreSQL

Serverless PostgreSQL with:

- Auto-suspend when inactive
- Auto-scaling compute
- Connection pooling
- SSL/TLS encryption

Database Schema

Field	Type	Description
id	Integer	Primary key
car_image	String	Vehicle image URL
lp_image	String	Plate crop URL
lp_number	String	Arabic plate text
confidence	Float	Detection confidence
created_at	DateTime	Timestamp (Egypt TZ)

Cloudinary CDN

Folder	Content
lpr/cars	Full vehicle images
lpr/plates	Annotated plate crops

Benefits:

- Global CDN distribution
- Automatic image optimization
- Secure HTTPS URLs

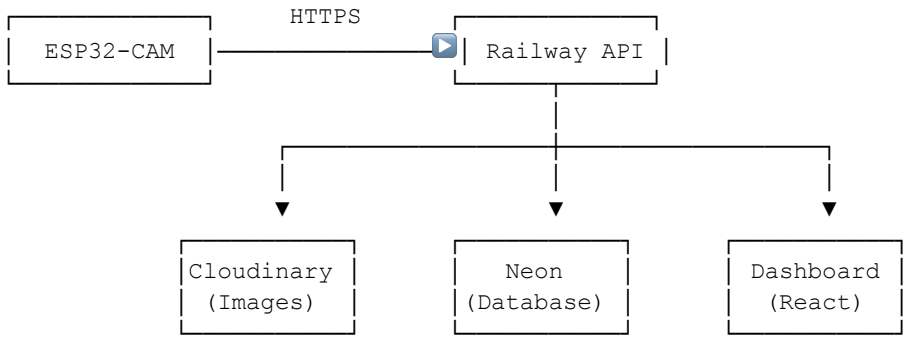
Environment Variables

Variable	Service	Description
DATABASE_URL	Neon	PostgreSQL connection string
CLOUDINARY_CLOUD_NAME	Cloudinary	Account identifier
CLOUDINARY_API_KEY	Cloudinary	API authentication

CLOUDINARY_API_SECRET	Cloudinary	API secret key
PORT	Railway	Server port (auto-set)
VITE_API_URL	Dashboard	Backend API URL

System Data Flow

Complete Data Flow Diagram



End-to-End Process

7. **Vehicle Detection**: Ultrasonic sensor detects object within 15cm
8. **Image Capture**: ESP32-CAM captures JPEG image with flash
9. **API Request**: Image sent via HTTPS POST to Railway backend
10. **ML Processing**:

- YOLOv10 detects license plate region
- Plate cropped and resized to 640×640
- Arabic OCR extracts characters
- Characters sorted right-to-left

11. **Storage**: Images uploaded to Cloudinary, record saved to PostgreSQL
12. **Dashboard**: React app fetches and displays results every 30 seconds

Communication Protocols

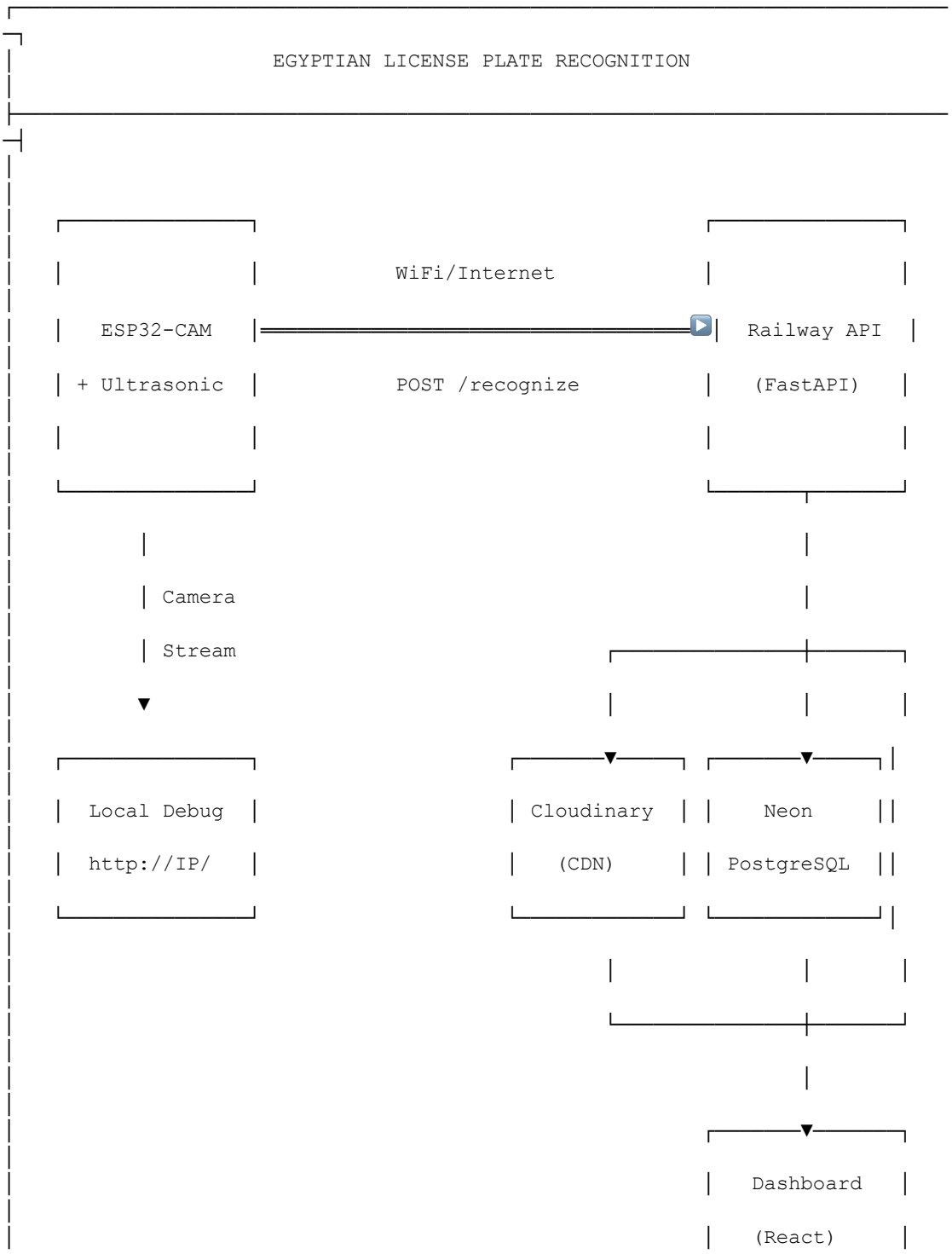
Connection	Protocol	Security
ESP32 → API	HTTPS	TLS 1.3
API → Database	PostgreSQL	SSL
API → Cloudinary	HTTPS	API Key
Dashboard → API	HTTPS	CORS

Performance Metrics

Metric	Value
Detection Accuracy	97.5% mAP@50
OCR Precision	95.88%
End-to-end Latency	~2-3 seconds

Dashboard Refresh	30 seconds

System Architecture Diagram



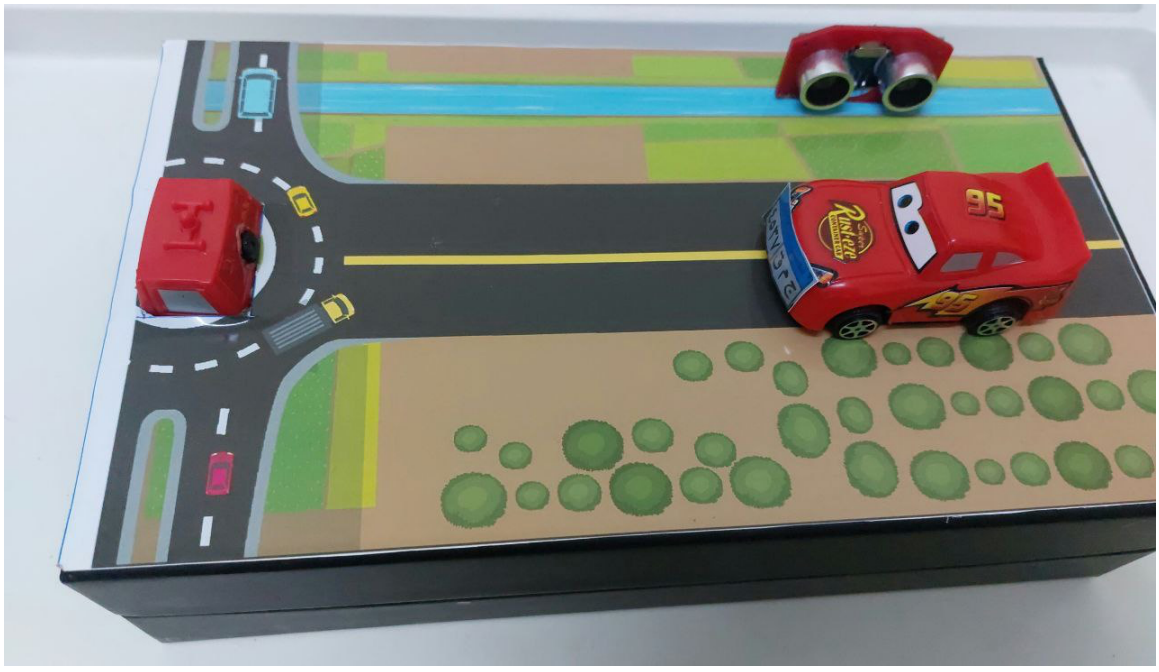
Demonstration

The project is demonstrated using a miniature maket with a toy car displaying an Egyptian license plate. When the car approaches the ultrasonic sensor, the system automatically captures and processes the plate.

Demonstration Components:

- Toy car with printed Egyptian license plate
- ESP32-CAM mounted at plate level
- Ultrasonic sensor positioned to detect approaching vehicles
- Battery-powered portable operation

This allows showcasing the complete detection flow without requiring actual vehicles.



Caption: Maket demonstration showing hardware setup with toy car

Scaling Considerations

Current Resource Limits

Resource	Limit
API Memory	512MB
Request Timeout	60 seconds
Database Connections	100 pooled
Image Size	10MB max

Performance Optimizations

- **Model Lazy Loading**: ML models loaded on first request
- **Connection Pooling**: Reuse database connections
- **Image Compression**: JPEG quality optimization
- **CDN Caching**: Static assets cached globally

Reliability Features

- WiFi auto-reconnection on ESP32
- Database connection pooling
- Image fallback handling
- Error state recovery

Cost Structure

Service	Pricing Model
Railway	Usage-based (memory × time)
Neon	Free tier available, then usage-based
Cloudinary	Free tier (25GB), then storage + bandwidth