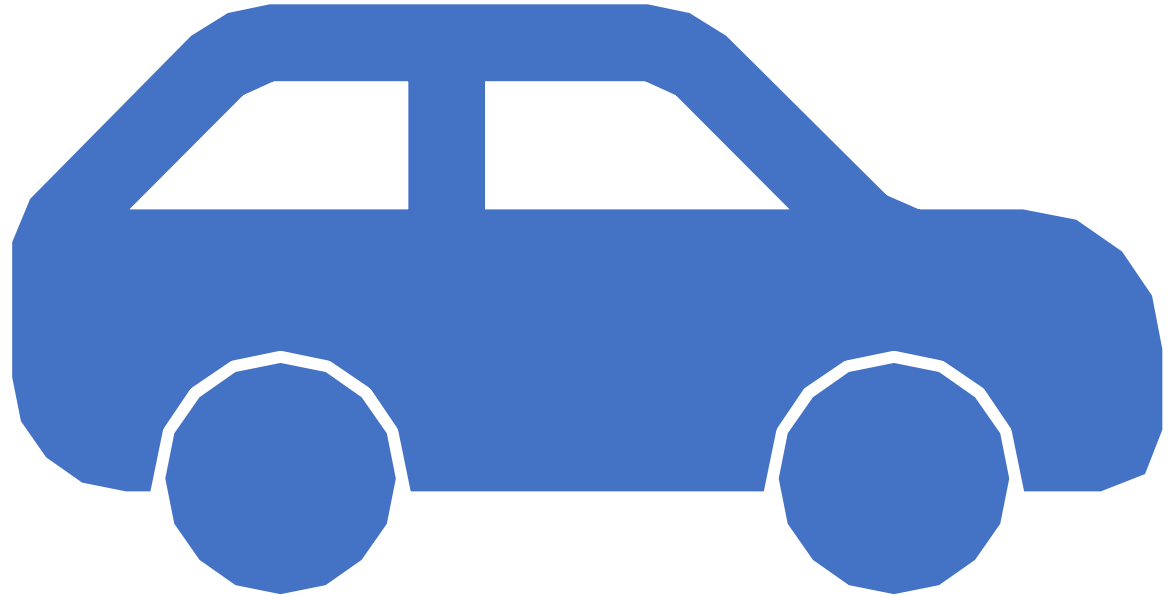


Luke Redmore

Driver and Vehicle
Licensing Agency

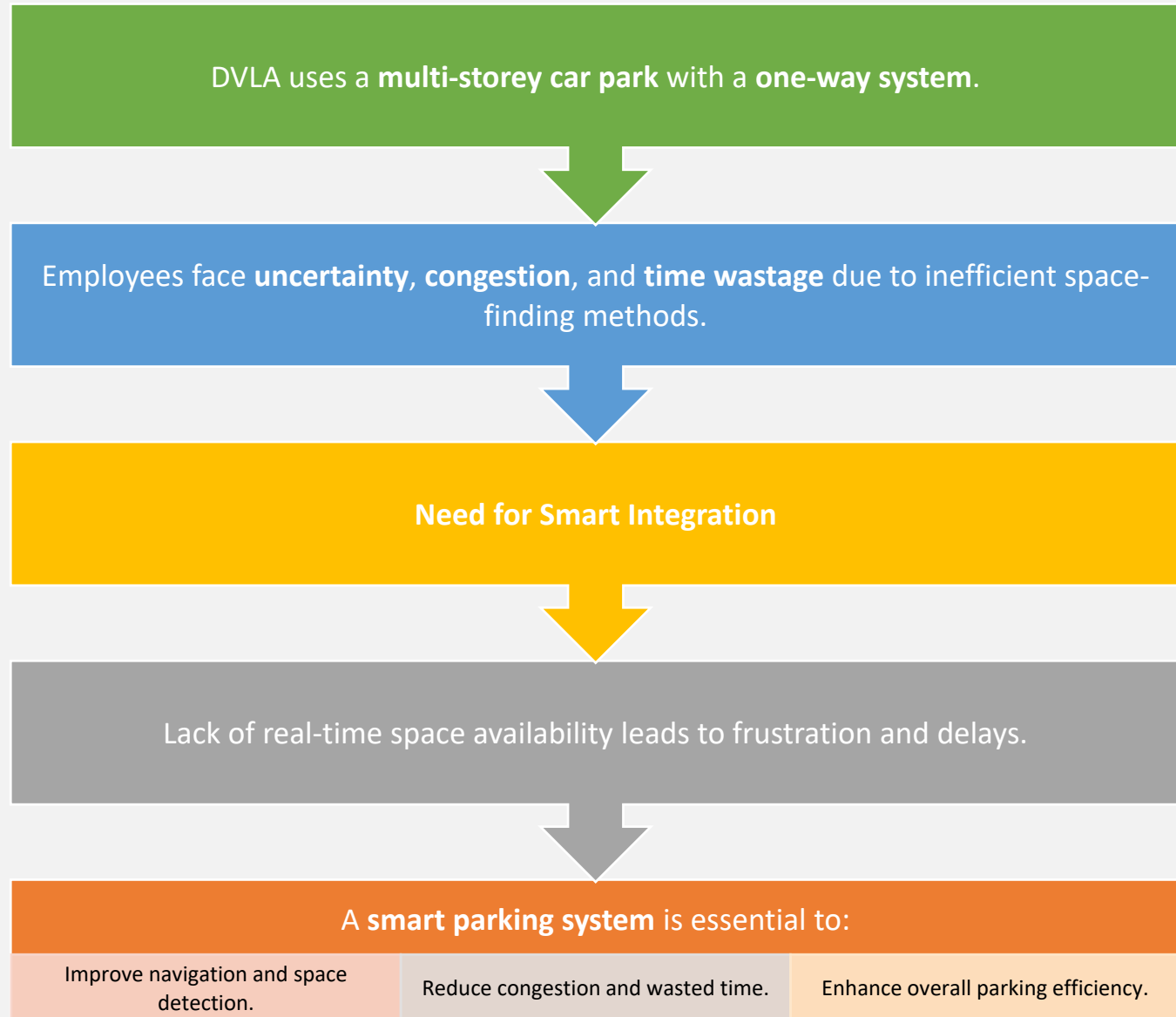
Smart Parking Solution for Multistorey Building



Introduction & Research Context

- **Growing Pressure on Parking Infrastructure**
- UK saw **417,000+ new vehicles** registered from Dec 2023–2024.
- Both **public** and **private** parking facilities are struggling to meet demand.
- **Challenges in Multi-Storey Parking**
- Limited capacity leads to **congestion, stress, and wasted time**.
- Motorists often enter garages **without real-time availability info**, increasing inefficiency.
- **Need for Smart Solutions**
- **IoT-based systems** (Lubna et al.): Reliable, low-cost vacancy tracking.
- **Machine Vision & Algorithms** (Arsam Abhigna, Luiz Eduardo): Efficient space detection using simple sorting and IoT middleware.
- **Object Detection Models** (R. Srinivasan, Shilpa Sondkar): Real-time vehicle & licence plate identification with up to **98.9% accuracy**.
- **Occlusion Detection** (Jiageng Ruan): Tracking vehicle parts (e.g., bumpers) as alternative identification method.
- **Conclusion**
- Smart, scalable, and adaptable parking solutions are essential to meet growing demand—especially in **multi-storey environments**.

Problem Statement Current Parking Challenges at DVLA



Research Aim & Objectives



Aim: To develop a smart parking solution for a multi-storey building



Objectives



1. To critically analyse the current literature to understand current techniques for automated parking



2. To design a prototype model for smart parking solution



3. To develop the model based on the designed prototype



4. To evaluate the developed model using well known evaluation metrics

IoT-Based Smart Parking Solutions (Literature Review)

- **Growing Demand & IoT Integration**
 - Rising urban vehicle density has intensified the need for efficient parking systems.
 - IoT-based solutions use microcontrollers, wireless networks, and cloud platforms to streamline parking.
- **Key Implementations**
 - **Ji et al. & Pham et al.:** Cloud-based, scalable real-time sensor networks with cost/distance optimization.
 - **Khana & Anand, Waili et al.:** Real-time tracking using Arduino, Raspberry Pi, GPS, and ultrasonic sensors.
 - **Akshay et al.:** RFID for automated fee calculation—showcasing IoT sensor versatility.
- **Limitations**
 - Most systems are designed for single-level environments.
 - Lack features like **floor-level tracking, vertical navigation, and reservation systems** for multi-storey structures.

Vision & Deep Learning in Parking Systems

(literature review continued)

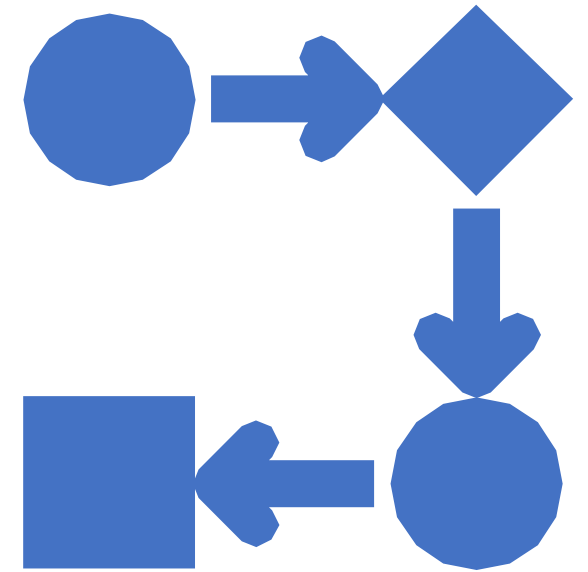
- **Computer Vision Approaches**
- **Wang et al.:** Deep learning with GPFE for robust detection under varied lighting.
- **Luiz Eduardo et al.:** Raspberry Pi + OpenCV + TIP platform for real-time monitoring.
- **Sriramdharnish K et al.:** AI-enhanced vision systems using IoT middleware.
- **YOLOv8 & Deep Learning**
- **Shilpa Sondkar et al.:** YOLOv8 for real-time vehicle detection using COCO dataset.
- **R. Srinivasan et al.:** DEN-YOLO with deformable convolutions, CLAHE, and dark channel preprocessing for low-light accuracy.
- **Occlusion Handling**
- **Zhou & Yuan et al.:** Occlusion pattern recognition improves detection in obstructed views—critical for camera placement in multi-storey setups.

Research Methodology & Evaluation Strategy

- **Philosophical Approach**
- **Positivist philosophy:** Assumes an objective reality measurable through observable data.
- Ideal for evaluating smart parking solutions using **quantitative analysis**.
- **Methodology**
- **Deductive approach:** Begins with existing theories (e.g., user experience, space utilization).
- Hypotheses tested via **empirical data collection** and analysis.
- **Research Strategy**
- Combines **software prototyping** and **simulation** in a controlled environment.
- Follows up with **user feedback** for technical and experiential validation.
- **Mixed Methods**
- **Quantitative:** Time saved, space utilization metrics.
- **Qualitative:** Thematic analysis of user experience.
- **Data Collection**
- **Cross-sectional:** Captures data at a single point in time.
- Includes **statistical analysis** and **graph plotting** to identify patterns.
- **Outcome**
- Ensures research is **objective**, **systematic**, and **user-informed**—supporting practical, scalable smart parking solutions.

Why Agile Development Lifecycle?

- **Flexibility & Adaptability**
- Agile allows for **iterative development**, enabling quick adjustments based on feedback or changing requirements.
- **User-Centric Approach**
- Frequent testing and feedback loops ensure the solution aligns with **real user needs**, especially important for usability in parking systems.
- **Rapid Prototyping**
- Agile supports **early and continuous delivery** of working software, ideal for demonstrating smart parking features in controlled environments.
- **Risk Reduction**
- Regular reviews and incremental updates help **identify issues early**, reducing the risk of major failures.
- **Collaboration & Transparency**
- Agile promotes **team collaboration**, clear communication, and visibility throughout the development process.
- **Supports Mixed Methods Research**
- Agile complements your **technical validation and user experience evaluation**, aligning well with your research strategy.



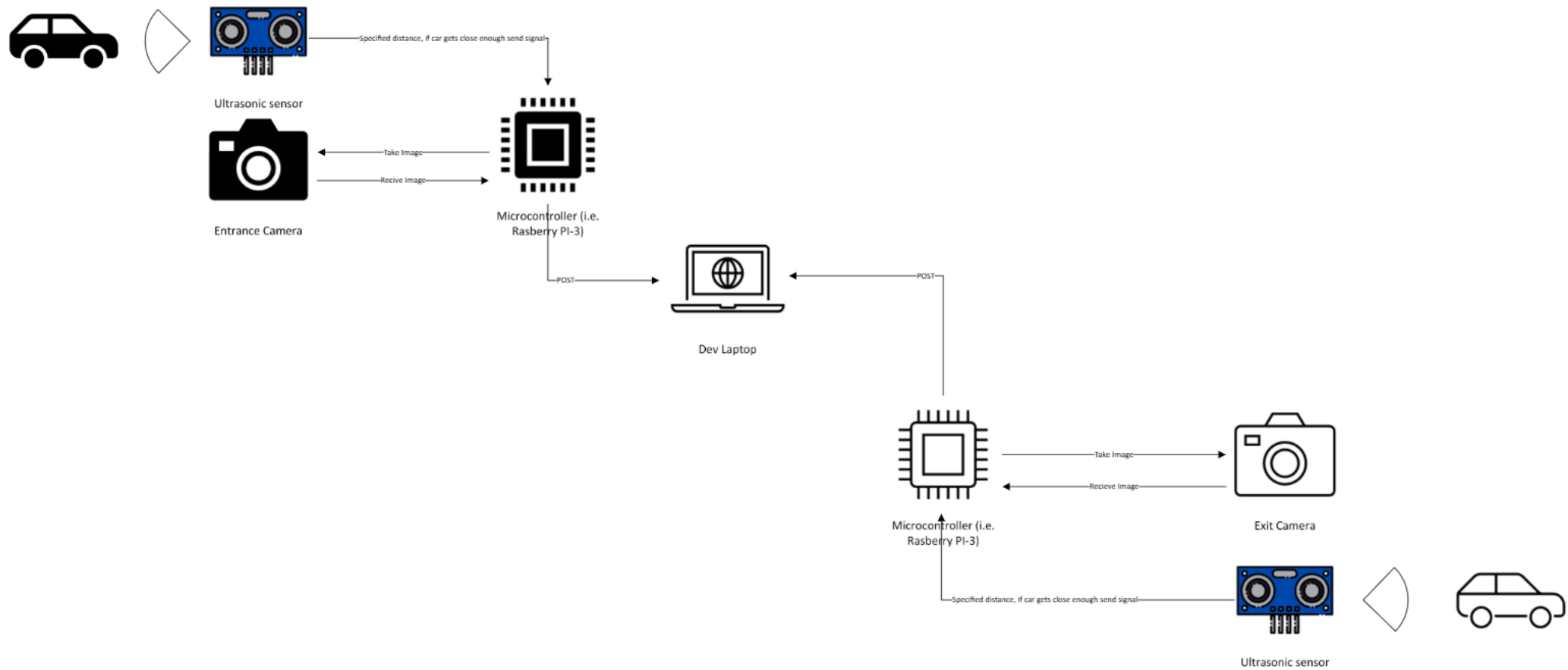
Technical Resources & Project Support

- **Purpose**
- To ensure successful **development and implementation** of the smart parking solution.
- **Key Focus Areas**
- **Hardware & Software Interoperability:** Understanding how components work together.
- **Programming Techniques:** Foundational knowledge for model development and integration.
- **Model Usage Guidance:** Documentation supporting effective deployment of smart technologies.
- **Project Support Strategy**
- Technical materials provide:
 - Practical guidance for system design.
 - Insight into programming and simulation.
 - Support for evaluating and refining the solution.
- **Outcome**
- A well-informed, technically sound foundation for building a **scalable and efficient smart parking system**.

Project Plan

- **Purpose**
- Support successful development of a smart parking solution through technical resources and documentation.
- **System Architecture**
- **Option 1:** **USB COM** port connection between edge devices and laptop.
- **Option 2:** Laptop acts as a **web server** receiving image payloads via HTTP POST.
- **Key Components**
- **Ultrasonic Sensor (HC-SR04):** Measures vehicle proximity.
- **Raspberry Pi Camera Module V2:** 8MP image capture.
- **Raspberry Pi 4B/5:** Handles sensor input, image transmission, and connectivity.
- **Laptop:** Runs image processing stack (OpenCV + YOLOv8).

Visuals for design



Workflow

Detection → Image capture → Processing → Command execution

Programming, Tooling & Project Setup

- **Programming Resources**
- **Language:** Python 3.x
- **IDE:** JetBrains PyCharm
- **Libraries:** OpenCV, NumPy, Pandas, OS
- **YOLO Framework:** Pre-trained & custom model support (YOLOv5/YOLOv8)
- **Image Processing Techniques**
- Feature detection, color space conversion (BGR → HSV/RGB)
- Preprocessing: Histogram equalisation, deionising for low-light accuracy
- **Software & Tooling**
- **Flask:** Handles HTTP GET/POST for web-based platform
- **SSH Access:** Headless setup for Raspberry Pi via terminal
- **Code Deployment:** SCP/rsync for remote file transfer
- **Project Hardware**
- Intel i7 Laptop (32GB RAM, 1TB SSD)
- 2× Raspberry Pi 4B/5
- 2× Pi Camera Modules
- 2× Ultrasonic Sensors
- Development in **Ubuntu VM** via Oracle VirtualBox

References

1. UK Government, *Vehicle Licensing Statistics: 2024*, Department for Transport, 2024.
2. Lubna et al., *IoT-Enabled Vacant Parking Slot Detection*, IEEE Sensors Journal, Vol. 23, No. 7, 2023.
3. R. Srinivasan et al., *Real-Time Classification and License Plate Recognition via YOLOv8*, IEEE Sensors Journal, Vol. 24, No. 23, 2024.
4. *IEEE IoT for Smart Parking System*, ICAIQSA Conference, 2024.
5. Jiageng Ruan et al., *Occluded Object Detection for Autonomous Driving*, Green Energy & Intelligent Transportation, Vol. 2, No. 3, 2023.
6. C. Zhang & J. Yuan, *Occlusion Pattern Discovery*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 30, No. 7, 2020.
7. Meshari Aljohani et al., *Survey of Parking Solutions for Smart Cities*, IEEE Transactions on Intelligent Transportation Systems, Vol. 23, No. 8, 2022.
8. Z. Ji et al., *Cloud-Based Parking Middleware for IoT*, Sensors, Vol. 14, No. 12, 2014.
9. T. Pham et al., *Cloud-Based Smart Parking System*, Vol. 3, 2015.
10. R. Khanna & A. Anand, *IoT-Based Smart Parking*, IOTA Conference, 2016.
11. Al Waili et al., *IoT Parking System Using Android & Google Maps*, Int. J. Appl. Eng. Res, Vol. 13, No. 20, 2018.
12. Akshay et al., *Automated Paid Parking Using RFID*, ICCS Conference, 2019.
13. Li Wang et al., *Robust Parking Space Detection Using Low-Cost Camera*, IEEE Transactions on Intelligent Vehicles, Vol. 8, No. 2, 2023.
14. Luiz Eduardo Giampaoli, *Parking Space Monitoring Using CV & IoT*, PUCRS, 2021.
15. Sriramdharnish K et al., *Vision Park – Next Gen Parking Monitoring*, ICERCS Conference, 2024.
16. Shilpa Sondkar, *IoT for Smart Parking System*, ICAIQSA Conference, Pune, 2024.
17. R. Srinivasan et al., *YOLOv8 for Vehicle & Plate Detection*, IEEE Sensors Journal, Vol. 24, No. 23, 2024.
18. C. Zhang & J. Yuan, *Occlusion Reasoning for Object Detection*, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 30, No. 7, 2020.
19. Raspberry Pi, *Raspberry Pi 4B Datasheet*, [Online].
20. OpenCV, *OpenCV-Python Tutorials*, [Online].
21. Ultralytics, *YOLO Python Docs*, [Online].
22. Flask, *Quickstart Guide*, [Online].
23. Raspberry Pi, *Remote Access Documentation*, [Online].

Conclusion

Summary of Findings

Multi-storey parking facilities face increasing pressure due to rising vehicle numbers and inefficient space-finding methods.

IoT and computer vision technologies—especially YOLOv8 and OpenCV—offer scalable, low-cost, and accurate solutions for real-time parking space detection and vehicle tracking.

Project Impact

The proposed smart parking system improves navigation, reduces congestion, and enhances user experience.

Integration of object detection and sensor-based inputs enables dynamic space allocation and potential for automated access control.

Future Potential

System can be expanded to include reservation features, floor-level tracking, and license plate-based access.

Further testing and refinement could lead to deployment in both public and private multi-storey facilities.

Technical demonstration handover

*Web-based vehicle
identification application*

Over to you me...

This demonstrates globally loading the model as opposed to calling it into an instance with each image, this is far more performant and improves interoperability with web framework

Brief overview of codebase

1. Model Loading (Performance Optimization)

```
from ultralytics import YOLO

# Load YOLO model once for performance reasons
model = YOLO("yolov8s.pt")
```

Key Point: Global loading prevents reloading on every detection call

This is where the code
hands over to Yolo via
OpenCV

Detection inside images

2. Core Detection Logic

```
def detect_cars(image_path, show_image=True):  
    # Load image  
    frame = cv2.imread(image_path)  
  
    # Run YOLO detection  
    results = model(frame)  
  
    # Process each detection  
    for result in results:  
        boxes = result.boxes  
        if boxes is not None:  
            for box in boxes:  
                class_id = int(box.cls[0])  
                confidence = float(box.conf[0])  
  
                # Filter for cars with good confidence  
                if class_id == 2 and confidence > 0.5:  
                    # Draw bounding box  
                    x1, y1, x2, y2 = map(int, box.xyxy[0])  
                    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)  
                    cv2.putText(frame, f"Car: {confidence:.2f}",  
                                (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX,  
                                0.5, (0, 255, 0), 2)
```

Key Points: Class ID 2 = Cars, 0.5 confidence threshold, green bounding boxes

For demonstration's sake,
examples of simple error
handling for image
fetching, makes demo
more fluid

Getting a random image

3. Smart Image Selection

```
def get_random_image(folder_path):  
    try:  
        image_extensions = ['.jpg', '.jpeg', '.png', '.bmp']  
        images = [f for f in os.listdir(folder_path)  
                  if any(f.lower().endswith(ext) for ext in image_extensions)]  
  
        if not images:  
            print(f"No image files found in: {folder_path}")  
            return None  
  
        return os.path.join(folder_path, random.choice(images))  
    except FileNotFoundError:  
        print(f"Folder not found: {folder_path}")  
        return None
```

Key Points: Error handling, file extension filtering, random selection

Quick glance at how I got
some metrics

Benchmarking my software

4. Performance Benchmarking

```
def benchmark_detection(folder_path, runs=100):  
    times = []  
    car_counts = []  
  
    for i in range(runs):  
        image_path = get_random_image(folder_path)  
        if image_path:  
            car_count, processing_time = detect_cars(image_path, show_image=False)  
            times.append(processing_time)  
            car_counts.append(car_count)  
  
    # Calculate metrics  
    avg_time = statistics.mean(times)  
    fps = 1 / avg_time  
    total_cars = sum(car_counts)  
  
    print(f"Average FPS: {fps:.1f}")  
    print(f"Total cars detected: {total_cars}")
```

Key Points: No visual output for speed, comprehensive metrics collection

How to run 100's of iterations with no visuals versus a full visual output

Running various aspects

5. Command Line Interface

```
if __name__ == "__main__":  
    import sys  
    folder_path = folder_path = "CarsInGarages"  
  
    if len(sys.argv) > 1 and sys.argv[1] == "--benchmark":  
        benchmark_detection(folder_path)  
    else:  
        image_path = get_random_image(folder_path)  
        print(f"Processing: {image_path}")  
        detect_cars(image_path)
```

Usage:

- `python main.py` → Demo mode
- `python main.py --benchmark` → Performance testing

Technical specifications of software

6. Key Technical Specs

Model Details:

- **YOLOv8s:** 11M parameters, 22MB file size
- **Input:** 640x640 pixels
- **Classes:** 80 objects (Class 2 = Cars)
- **Performance:** 30+ FPS on modern hardware

Detection Pipeline:

1. **Preprocessing:** Image resize & normalization
2. **Inference:** Neural network processing
3. **Postprocessing:** NMS, confidence filtering, visualization

Confidence Thresholds:

- **0.9-1.0:** Extremely confident
- **0.7-0.9:** Very confident
- **0.5-0.7:** Moderately confident ← Our threshold
- **Below 0.5:** Filtered out

Performance metrics YoloV8s

*normal car
images*

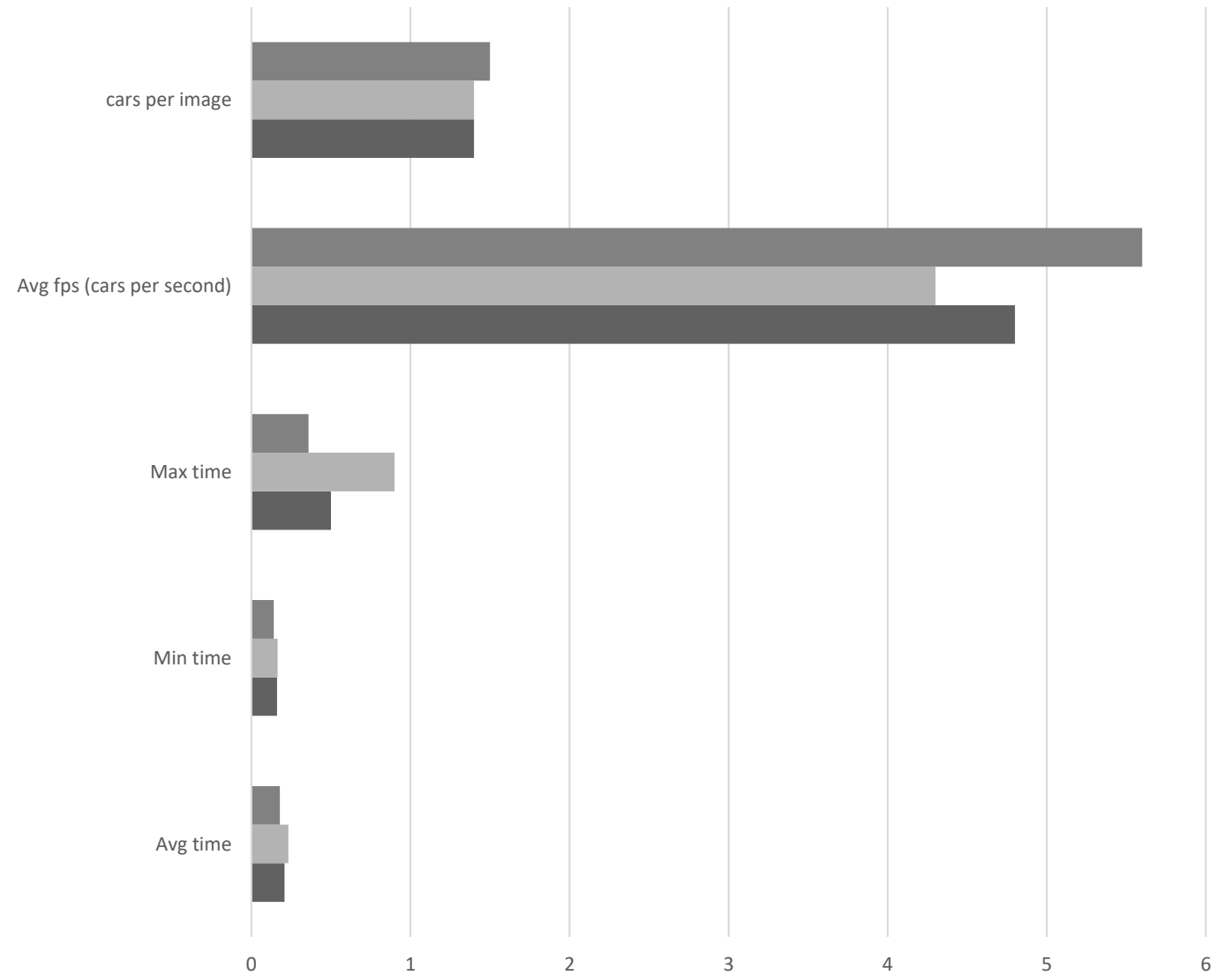
Avg time	Min time	Max time	Avg fps (cars per second)	cars per image	Iteration
0.208	0.161	0.5	4.8	1.4	1
0.232	0.165	0.9	4.3	1.4	2
0.179	0.141	0.359	5.6	1.5	3

Performance
metrics YoloV8s

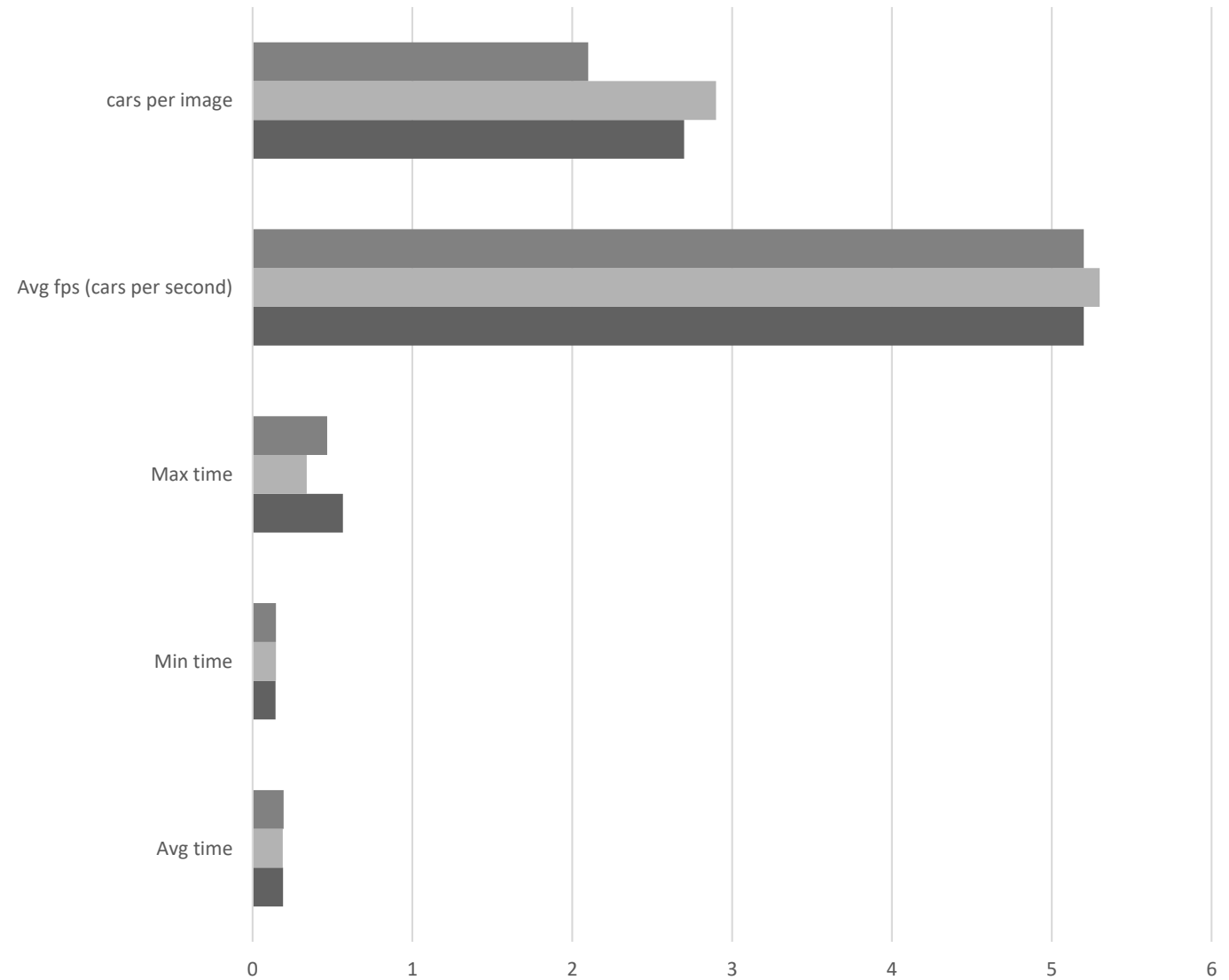
*enclosed space
car images*

Avg time	Min time	Max time	Avg fps (cars per second)	cars per image	Iteration
0.191	0.144	0.565	5.2	2.7	1
0.189	0.146	0.339	5.3	2.9	2
0.194	0.146	0.467	5.2	2.1	3

Visualise
previous metrics
(normal images)

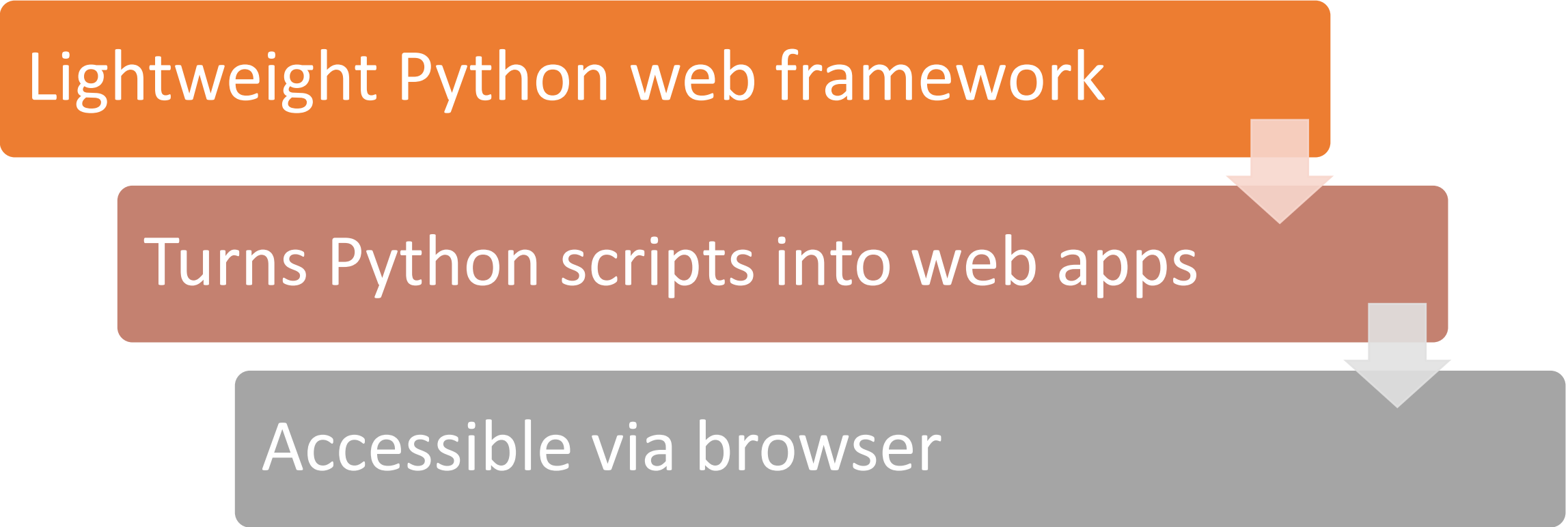


Visualise performance metrics (Enclosed)



Flask, what is it?

Lightweight Python web framework



```
graph TD; A[Lightweight Python web framework] --> B[Turns Python scripts into web apps]; B --> C[Accessible via browser];
```

Turns Python scripts into web apps

Accessible via browser

Project structure as a web-app

```
PythonProject/
```

```
|— flaskerisedYolo.py      # Main app  
|— templates/index.html   # Web interface  
|— yolov8s.pt             # YOLO model  
└— static/                # Optional: CSS, JS, images
```

Flask workflow

App Setup: `app = Flask(__name__)`



Routing: `@app.route('/') → index.html`



HTTP Methods:

GET: Load page

POST: Upload/process data

Car Detection Web Flow

User visits homepage (/)

Uploads image → JavaScript → POST /detect

Flask processes → JSON response → Display results

Key Flask Components

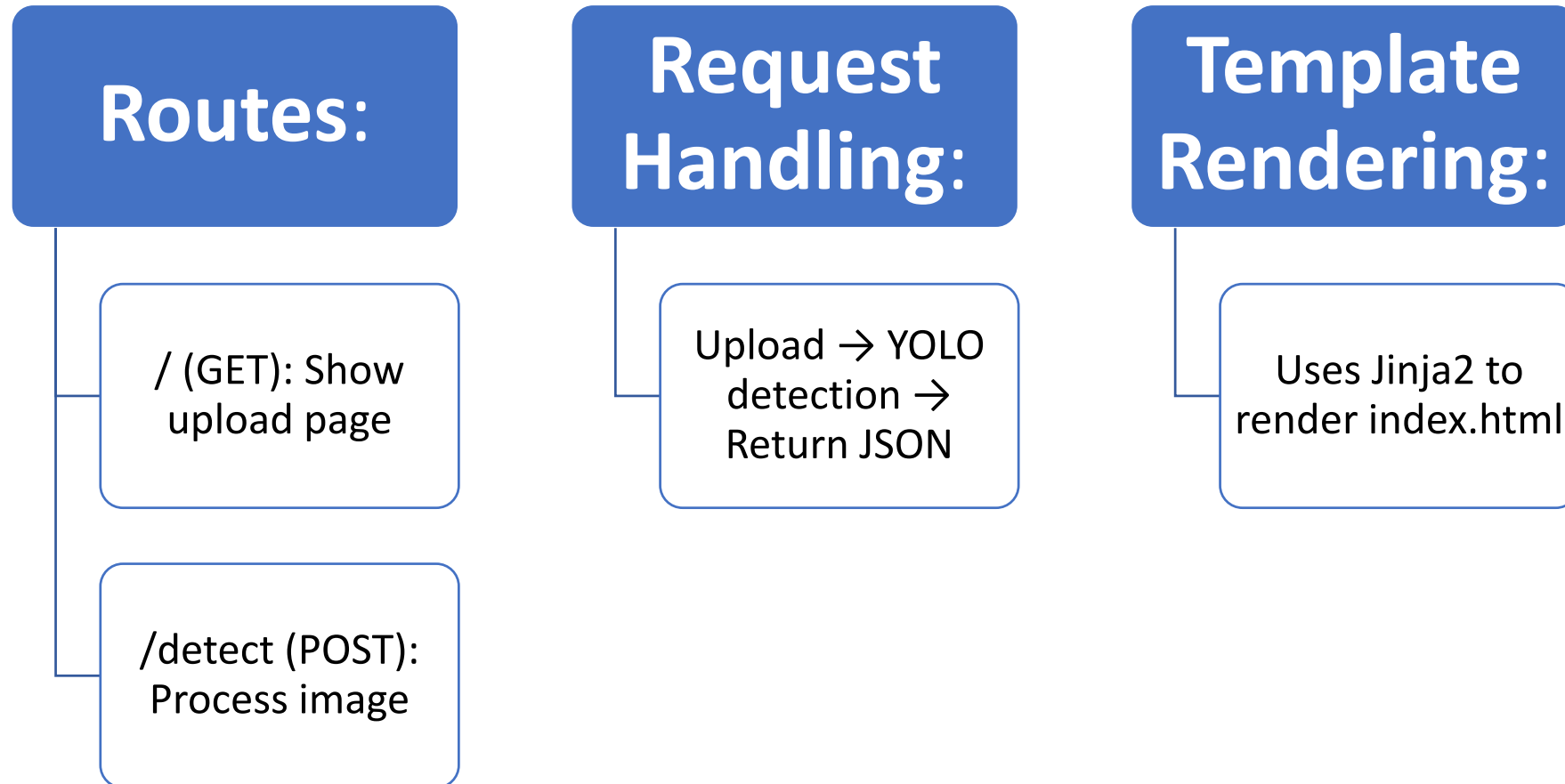
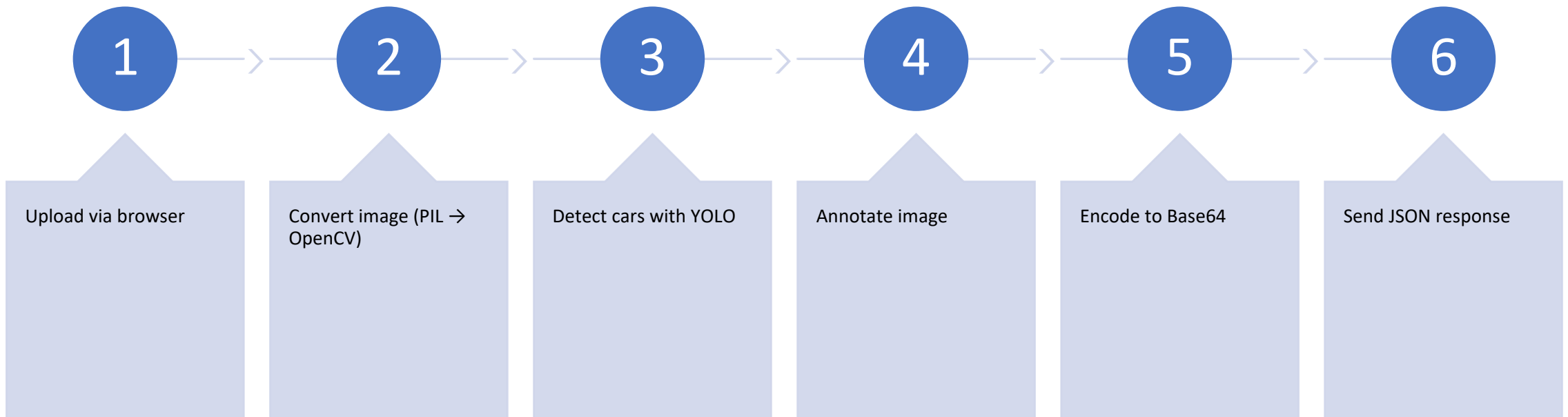
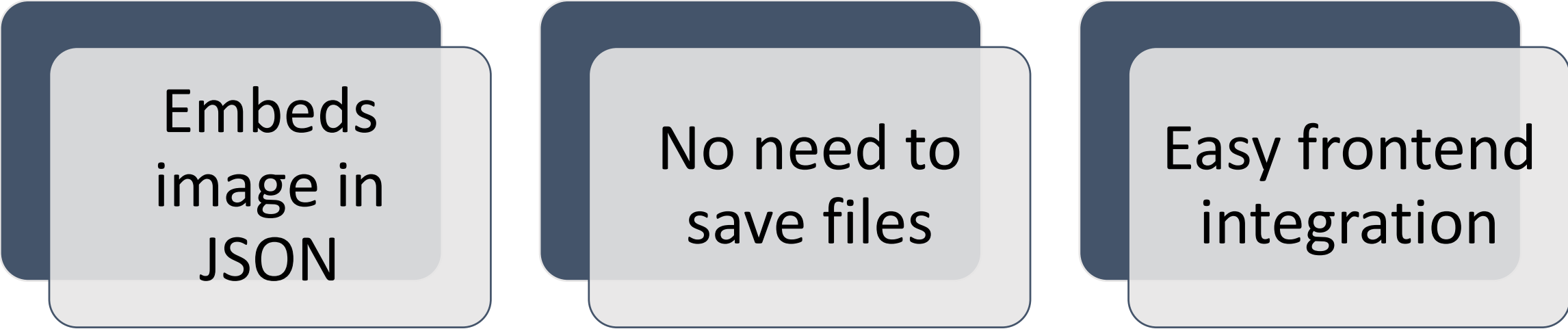


Image processing pipeline



Base64 encoding



Embeds
image in
JSON

No need to
save files

Easy frontend
integration

Client-Server Communication

Frontend (JS): Sends
image
via `fetch('/detect')`

Backend (Flask):
Processes and returns
JSON

Full Request Cycle



User visits /



Flask routes to index()



Renders HTML



User uploads image



JS sends POST to /detect



YOLO detects cars



Flask returns JSON



JS updates UI

Advantages of Using Flask

For Development:

Easy testing

Fast debugging

Scalable

Shareable

```
1
2  ✓ thank_you_slide:
3     title: "★ Thank You! ★"
4     subtitle: "We appreciate you!"
5  ✓ message: |
6     Thank you for listening!
7     And being awesome etc
8  ✓ background:
9     color: "pastel yellow"
10    image: "cute-stars-and-hearts.png"
11  ✓ font:
12     title: "Comic Sans MS"
13     subtitle: "Arial Rounded MT Bold"
14     message: "Segoe UI"
15  ✓ decorations:
16  ✓   - type: "emoji"
17     | content: "💖 ✨ 😊"
18  ✓   - type: "border"
19     | style: "dotted"
20     | color: "light pink"
21  ✓ footer:
22     text: "Made with love 💖"
23     alignment: "center"
24
25
```