

## Smart Parking Detector - AI app

The idea is to provide users the info on vacant parking spots or potentially available soon.

The app takes visual info from several cameras pointing to the on-street parked cars. For example, here the app can tell what spots are available and point those locations on a map, suggesting the user where to go to park a car.



img1.png

Here app can identify gaps between car and suggest that spot for the user (the space between red and dark blue SUVs):



img2.png

Cameras should be mounted at a quite high position so they can view more cars and capture certain activity better:



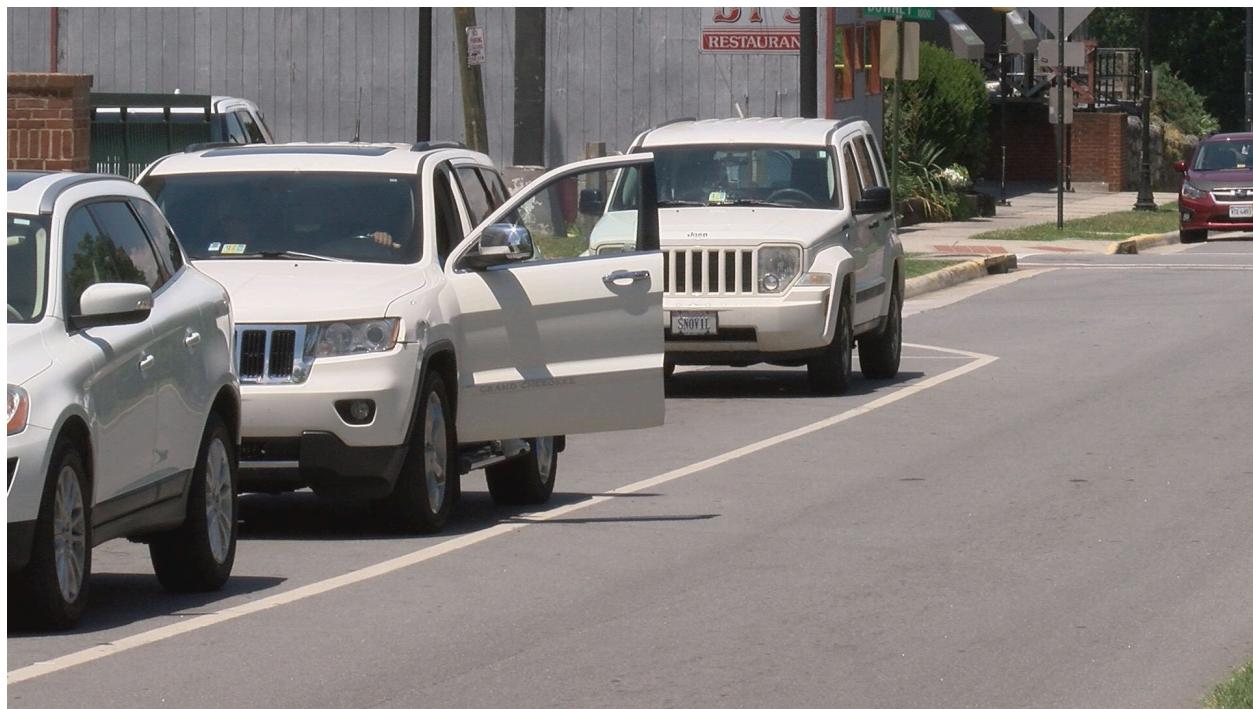
img3.png

For instance, here is the car leaving its spot so the app could immediately report a vacant spot here.



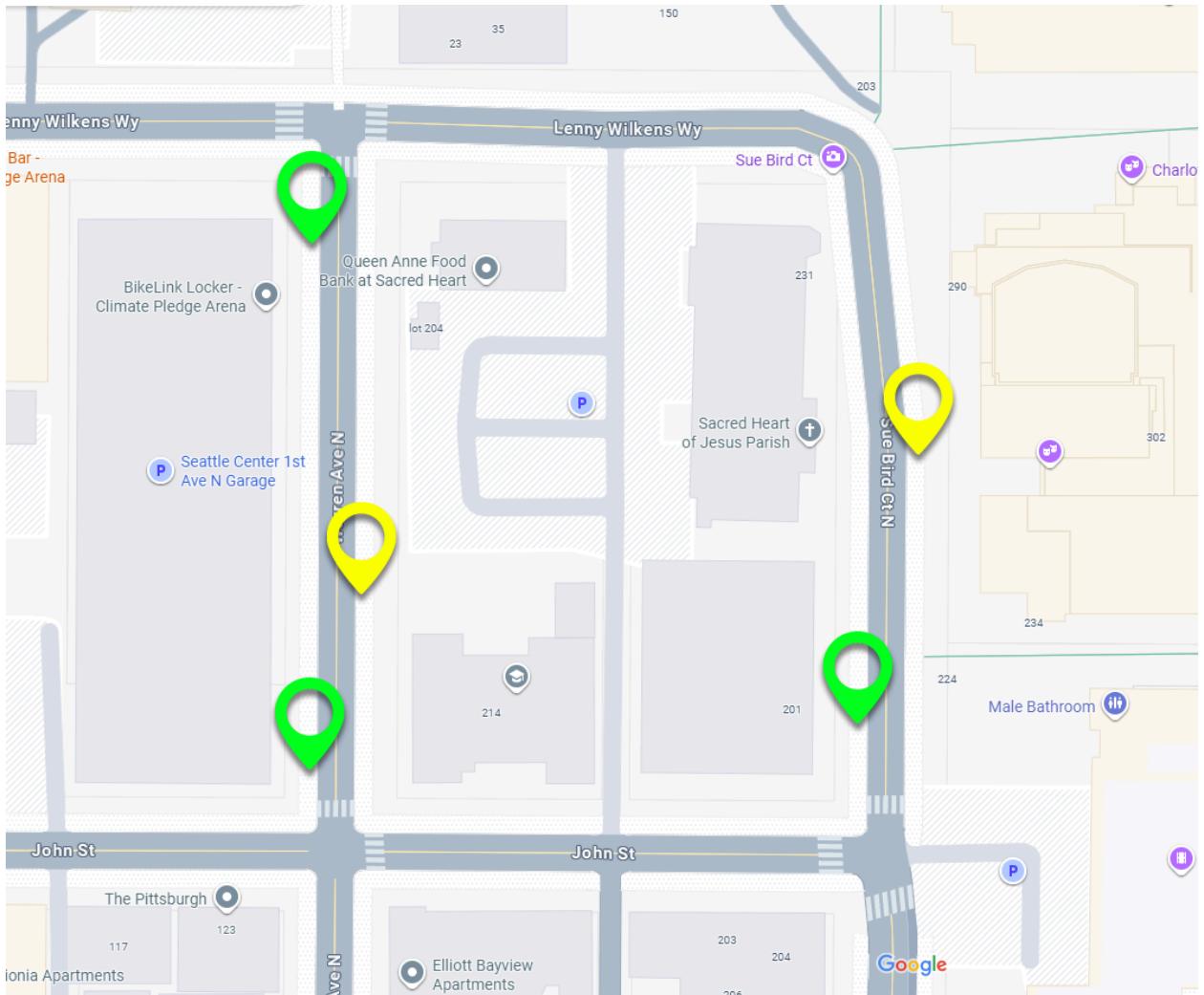
img4.png

Or, if camera sees open door, it could be a sign that car will be leaving soon:



img5.png

The page should show a map like a google map or mapbox or any other available map component, where user could see available spots (green) or potentially available soon (yellow) with high precision:



img6.png

Users can click and go to the specific spot.

I'm open to suggestions on the tech stack.

et's revisit the project requirements, please read this pdf and let me know your thoughts:

ChatGPT said:

I've reviewed your PDF, and here's my take:

---

## 👍 Strengths of Your Requirements

### 1. Clear User Value

- The “vacant” vs “potentially available soon” distinction makes this far more useful than a basic detection system — it adds **predictive intelligence**.
- Visual examples (like gap detection, open door cues, leaving car) are concrete and relatable to stakeholders.

### 2. Practical Hardware Considerations

- The requirement for high-mounted cameras is a good call — better angle → fewer occlusions → higher detection accuracy.
- Multi-camera input makes it more scalable for real-world deployment.

### 3. Strong UX Concept

- Google Maps / Mapbox integration for visualizing spots is perfect.
  - Green/yellow markers are intuitive for quick decision-making.
  - Click-to-navigate is a nice touch for user engagement.
- 

## ⚠️ Things to Refine / Clarify

### 1. Spot Definition

- Will spots be **fixed polygons** predefined in the system (best for consistent monitoring) or dynamically inferred from the scene?

- Fixed is easier, but dynamic is more flexible.

## 2. Multi-Camera Coordination

- If multiple cameras overlap, how will we prevent **double-counting** the same spot?
- Need a camera calibration step to map detections to GPS/map coordinates.

## 3. Prediction Logic

- "Open door" → likely to leave soon is a good heuristic, but might require:
  - Additional training dataset for such events.
  - Thresholds to avoid false positives (e.g., someone just opening trunk).

## 4. Latency & Real-Time Processing

- Need to define the acceptable delay from detection to map update (e.g.,  $\leq$  1–2 seconds).
- This affects hardware (GPU vs CPU) and backend choice.

## 5. Privacy Concerns

- If street-level cameras record people, license plates, etc., you may need **privacy masking** in processing pipeline.



# Suggested Technical Approach

## Detection & Tracking

- **YOLOv8** for vehicle detection.
- **DeepSORT** or **ByteTrack** for ID persistence across frames.

## Prediction

- Simple rule-based first:
  - Car moves → mark spot as vacant.
  - Open door detected → mark spot as “yellow” (potentially soon).
- Later upgrade to LSTM/CNN hybrid for more nuanced predictions.

## Mapping

- **Mapbox GL JS** or **Leaflet.js** for interactive frontend.
- FastAPI backend serving JSON with current spot states.

## Infrastructure

- Camera feeds → RTSP → processing service (Docker container with GPU support).
  - WebSocket or MQTT for real-time updates to clients.
- 

If you want, I can now create a **full development roadmap** breaking this into milestones, including:

- Dataset creation
- Model training/integration
- Map API integration
- Multi-camera calibration  
...so it's ready to pitch or start building.

Do you want me to prepare that?

You said:

yes

ChatGPT said:

Alright — here's a **full development roadmap** for the **Smart Parking Assistant** based on your PDF and our earlier discussions.

---

## Development Roadmap

### Phase 0 — Planning & Setup

**Goals:** Define system boundaries, tech stack, and resources.

**Tasks:**

#### 1. Confirm Project Scope

- Target: On-street parking only, or parking lots too?
- Number of cameras & their placement.

#### 2. Choose Tech Stack

- Detection: **YOLOv8**
- Tracking: **DeepSORT** or **ByteTrack**
- Backend: **FastAPI** (real-time API + WebSocket updates)
- Frontend: **React + Mapbox GL JS**
- Processing: **Docker** container with GPU support (NVIDIA Jetson / server GPU).

#### 3. Define Spot Geometry

- Manually annotate parking spot polygons for each camera view.
- Store as JSON or GeoJSON.

#### 4. Setup Dev Environment

- Install YOLOv8 (`pip install ultralytics`), OpenCV, Mapbox API key, FastAPI scaffold.
-

## Phase 1 — Detection & Spot Mapping

**Goals:** Get basic “occupied/free” status working from a single camera.

**Tasks:**

1. Load RTSP/local video feed.
2. Run YOLOv8 on frames, detect cars.
3. Check if a detection overlaps a spot polygon (intersection over area threshold).
4. Display a simple OpenCV overlay (red = occupied, green = free).
5. Store state changes in memory or simple DB.

**Deliverable:** Local script showing occupancy detection on test video.

---

## Phase 2 — Tracking for Stability

**Goals:** Smooth out flicker, persist car IDs.

**Tasks:**

1. Integrate YOLOv8 with DeepSORT/ByteTrack.
2. Maintain object IDs over time to avoid false transitions.
3. Update status only when:
  - A spot is continuously empty for X seconds.
  - A spot is continuously occupied for X seconds.
4. Optimize for FPS (aim  $\geq$  15 FPS real-time).

**Deliverable:** Stable real-time detection that avoids rapid “flashing” changes.

---

## Phase 3 — Predictive Logic (Potentially Available Soon)

**Goals:** Add yellow status for “likely leaving soon.”

**Tasks:**

1. Train or integrate a model to detect:
  - **Open car door**
  - **Car moving out of spot**
2. Use rule-based logic first:
  - Open door → mark yellow for up to N seconds.
  - Vehicle movement → mark yellow until spot becomes green.
3. Tune thresholds for false positive control.

**Deliverable:** Three-state detection (Green / Yellow / Red) in local test video.

---

## Phase 4 — Backend API & Data Flow

**Goals:** Serve live parking data to frontend.

**Tasks:**

1. Build FastAPI backend:
  - `/spots` → JSON list of spots with status & coordinates.
  - `/ws` → WebSocket for real-time updates.
2. Store current state in Redis or in-memory dict.
3. Add logging for spot history.

**Deliverable:** Backend serving up-to-date spot states.

---

## Phase 5 — Map UI Integration

**Goals:** Show live spot status on map.

**Tasks:**

1. React app with **Mapbox GL JS**.
2. Load spot polygons (GeoJSON).
3. Subscribe to backend updates.
4. Change polygon color dynamically (green/yellow/red).
5. Add click → show “Navigate to” (Google Maps link).

**Deliverable:** Web app showing live parking status.

---

## Phase 6 — Multi-Camera Support

**Goals:** Combine feeds into one unified map.

**Tasks:**

1. Calibrate each camera → map coordinates.
2. Merge overlapping coverage.
3. Prevent double-counting (spot ID is global, not per camera).
4. Test with synthetic multi-camera footage.

**Deliverable:** Unified dashboard for multiple cameras.

---

## Phase 7 — Optimization & Deployment

**Goals:** Prepare for real-world test.

**Tasks:**

1. Optimize YOLO model size (yolov8n/s for speed).
2. Enable GPU acceleration (TensorRT, CUDA).

3. Dockerize backend & detection service.
4. Deploy to local server / edge device (Jetson Xavier / cloud GPU).
5. Add basic privacy masking (blur faces & plates).

**Deliverable:** Deployable containerized system.

---

## Dataset Sources

- **PKLot** (large-scale parking dataset)
  - **CNRPark + EXT** (varied weather, lighting)
  - **Custom labeled footage** (for “open door” & “car leaving”)
- 

## Suggested Timeline

Phase	Time Estimate
0–1	1 week
2	1 week
3	1–2 weeks
4–5	1–2 weeks
6–7	2–3 weeks

**Total:** ~6–9 weeks for MVP to deployment

---

If you want, I can now **draw the full system architecture diagram** so you can see how cameras, AI models, backend, and UI connect.

Do you want me to prepare that next?

**You said:**

just for you note. Please keep you tone/mood as my friend and colleague that I like to collaborate, feel free to use simpler words. I'd like to navigate through this project with joy and easiness like it's less like work and more like an enjoyable process of creating something new and really useful and all the excitement around that.

**ChatGPT said:**

Updated saved memory

Got it — I'll keep it light, friendly, and fun so it feels like we're building something awesome together rather than grinding through a task list.

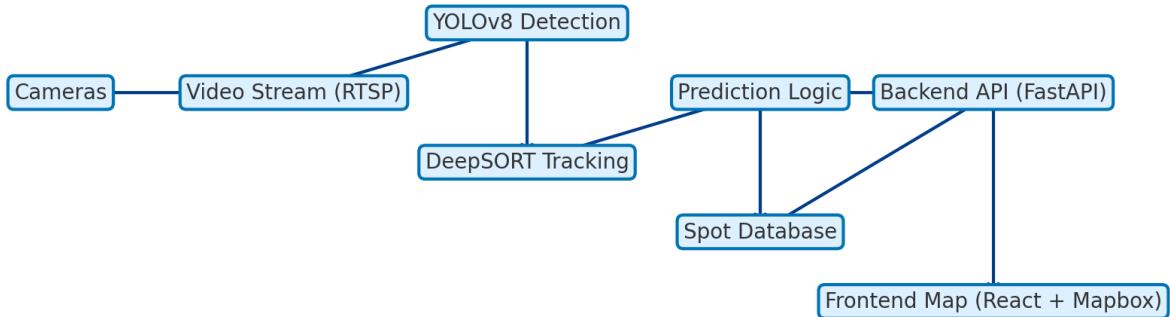
Want me to sketch that **system architecture diagram** next so we can see the whole Smart Parking flow in one picture?

**You said:**

yes, sure

**ChatGPT said:**

### Smart Parking Assistant - System Architecture



img7.png

Here's our Smart Parking Assistant flow in one picture — from cameras all the way to the map on someone's phone.

It's kind of like we're building a little "brain" that sees, thinks, and talks to the user in real time.

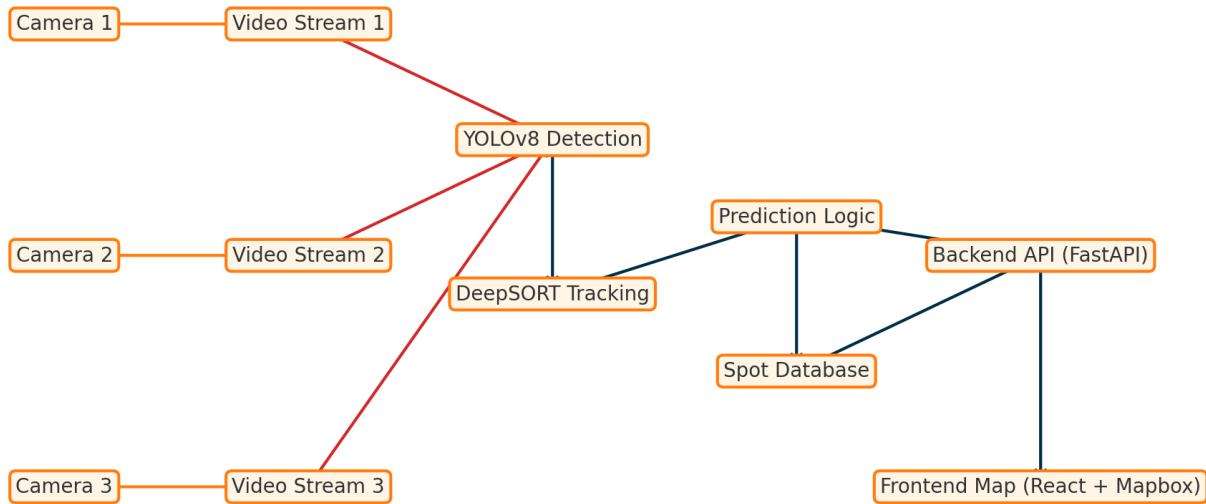
Want me to also add the **multi-camera integration** part to this diagram so it shows how they all connect to the same backend?

You said:

yes, please

ChatGPT said:

### Smart Parking Assistant - Multi-Camera Architecture



img8.png

Here's the updated version — now all three cameras feed into the same detection/tracking brain, and everything funnels into one backend that powers the live map.

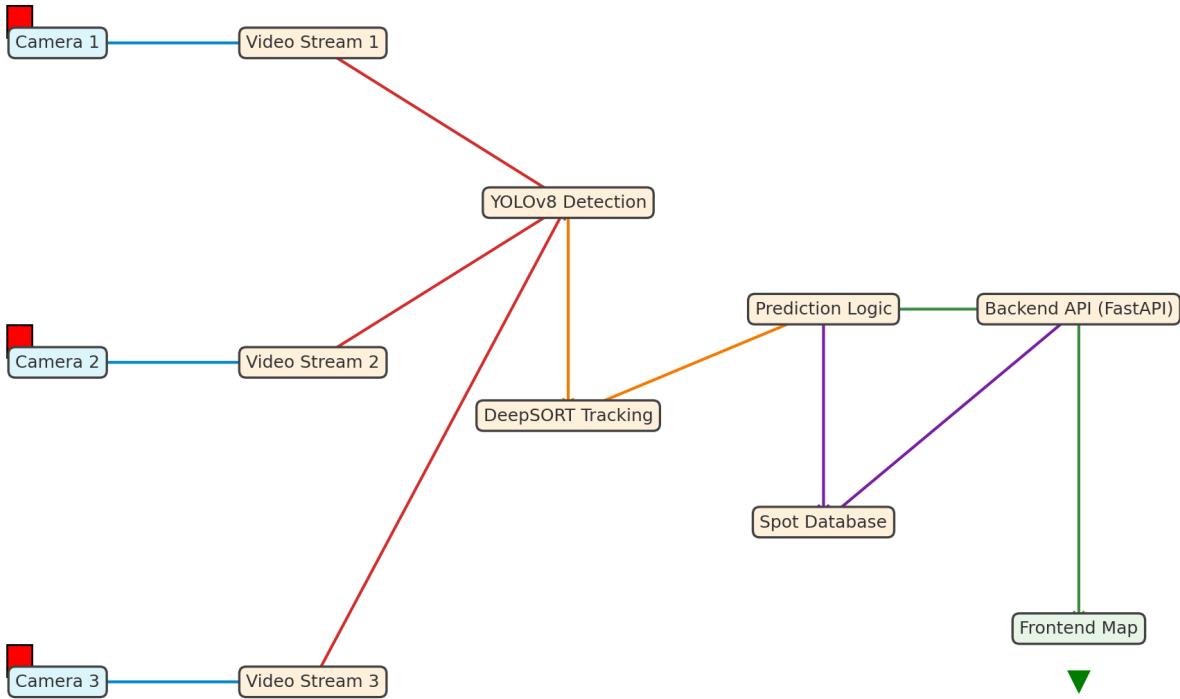
If you want, I can also add a **real-world example flow** with tiny icons of cars, cameras, and map markers so it feels more like we're storyboarding the app. That could make it easier to visualize the user experience.

You said:

yes, please

ChatGPT said:

## Smart Parking Assistant - Fun Storyboard Flow



img9.png

Here's our "fun storyboard" — now we've got little cars feeding into the camera system, the detection brain doing its magic, and the results popping up on a friendly map.

It's already starting to feel like a real product rather than just a diagram.

Next, we could make a **mockup of the actual live map UI** so we can imagine what the user will see when spots turn green or yellow.

Cursor prompt:

Smart Parking Detector - AI app

The idea is to provide users the info on vacant parking spots or potentially available soon.

The app takes visual info from several cameras pointing to the on-street parked cars. For example, here the app can tell what spots are available and point those locations on a map, suggesting the user where to go to park a car: @img1.png

Here app can identify gaps between car and suggest that spot for the user (the space between red and dark blue SUVs)@img2.png

Cameras should be mounted at a quite high position so they can view more cars and capture certain activity better: @img3.png

For instance, here is the car leaving its spot so the app could immediately report a vacant spot here: @img4.png

Or, if camera sees open door, it could be a sign that car will be leaving soon: @img5.png

The page should show a map like a google map or mapbox or any other available map component, where user could see available spots (green) or potentially available soon (yellow) with high precision: @img6.png

Users can click and go to the specific spot.

I'm open to suggestions on the tech stack.

Let's revisit the project requirements, please read this pdf and let me know your thoughts:

ChatGPT said:

I've reviewed your PDF, and here's my take:

👉 Strengths of Your Requirements

Clear User Value

The “vacant” vs “potentially available soon” distinction makes this far more useful than a basic detection system — it adds predictive intelligence.

Visual examples (like gap detection, open door cues, leaving car) are concrete and relatable to stakeholders.

## Practical Hardware Considerations

The requirement for high-mounted cameras is a good call — better angle → fewer occlusions → higher detection accuracy.

Multi-camera input makes it more scalable for real-world deployment.

## Strong UX Concept

Google Maps / Mapbox integration for visualizing spots is perfect.

Green/yellow markers are intuitive for quick decision-making.

Click-to-navigate is a nice touch for user engagement.

## ⚠️ Things to Refine / Clarify

### Spot Definition

Will spots be fixed polygons predefined in the system (best for consistent monitoring) or dynamically inferred from the scene?

Fixed is easier, but dynamic is more flexible.

## Multi-Camera Coordination

If multiple cameras overlap, how will we prevent double-counting the same spot?

Need a camera calibration step to map detections to GPS/map coordinates.

## Prediction Logic

"Open door" → likely to leave soon is a good heuristic, but might require:

Additional training dataset for such events.

Thresholds to avoid false positives (e.g., someone just opening trunk).

## Latency & Real-Time Processing

Need to define the acceptable delay from detection to map update (e.g.,  $\leq$  1–2 seconds).

This affects hardware (GPU vs CPU) and backend choice.

## Privacy Concerns

If street-level cameras record people, license plates, etc., you may need privacy masking in processing pipeline.

### Suggested Technical Approach

Detection & Tracking

YOLOv8 for vehicle detection.

DeepSORT or ByteTrack for ID persistence across frames.

## Prediction

Simple rule-based first:

Car moves → mark spot as vacant.

Open door detected → mark spot as “yellow” (potentially soon).

Later upgrade to LSTM/CNN hybrid for more nuanced predictions.

Mapping

Mapbox GL JS or Leaflet.js for interactive frontend.

FastAPI backend serving JSON with current spot states.

Infrastructure

Camera feeds → RTSP → processing service (Docker container with GPU support).

WebSocket or MQTT for real-time updates to clients.

If you want, I can now create a full development roadmap breaking this into milestones, including:

Dataset creation

Model training/integration

Map API integration

Multi-camera calibration

...so it's ready to pitch or start building.

Do you want me to prepare that?

You said:

yes

ChatGPT said:

Alright — here's a full development roadmap for the Smart Parking Assistant based on your PDF and our earlier discussions.

📍 Development Roadmap

## Phase 0 — Planning & Setup

Goals: Define system boundaries, tech stack, and resources.

Tasks:

Confirm Project Scope

Target: On-street parking only, or parking lots too?

Number of cameras & their placement.

Choose Tech Stack

Detection: YOLOv8

Tracking: DeepSORT or ByteTrack

Backend: FastAPI (real-time API + WebSocket updates)

Frontend: React + Mapbox GL JS

Processing: Docker container with GPU support (NVIDIA Jetson / server GPU).

Define Spot Geometry

Manually annotate parking spot polygons for each camera view.

Store as JSON or GeoJSON.

Setup Dev Environment

Install YOLOv8 (pip install ultralytics), OpenCV, Mapbox API key, FastAPI scaffold.

## Phase 1 — Detection & Spot Mapping

Goals: Get basic “occupied/free” status working from a single camera.

Tasks:

Load RTSP/local video feed.

Run YOLOv8 on frames, detect cars.

Check if a detection overlaps a spot polygon (intersection over area threshold).

Display a simple OpenCV overlay (red = occupied, green = free).

Store state changes in memory or simple DB.

Deliverable: Local script showing occupancy detection on test video.

## Phase 2 — Tracking for Stability

Goals: Smooth out flicker, persist car IDs.

Tasks:

Integrate YOLOv8 with DeepSORT/ByteTrack.

Maintain object IDs over time to avoid false transitions.

Update status only when:

A spot is continuously empty for X seconds.

A spot is continuously occupied for X seconds.

Optimize for FPS (aim  $\geq$  15 FPS real-time).

Deliverable: Stable real-time detection that avoids rapid “flashing” changes.

### Phase 3 — Predictive Logic (Potentially Available Soon)

Goals: Add yellow status for “likely leaving soon.”

Tasks:

Train or integrate a model to detect:

Open car door

Car moving out of spot

Use rule-based logic first:

Open door → mark yellow for up to N seconds.

Vehicle movement → mark yellow until spot becomes green.

Tune thresholds for false positive control.

Deliverable: Three-state detection (Green / Yellow / Red) in local test video.

### Phase 4 — Backend API & Data Flow

Goals: Serve live parking data to frontend.

Tasks:

Build FastAPI backend:

/spots → JSON list of spots with status & coordinates.

/ws → WebSocket for real-time updates.

Store current state in Redis or in-memory dict.

Add logging for spot history.

Deliverable: Backend serving up-to-date spot states.

Phase 5 — Map UI Integration

Goals: Show live spot status on map.

Tasks:

React app with Mapbox GL JS.

Load spot polygons (GeoJSON).

Subscribe to backend updates.

Change polygon color dynamically (green/yellow/red).

Add click → show “Navigate to” (Google Maps link).

Deliverable: Web app showing live parking status.

Phase 6 — Multi-Camera Support

Goals: Combine feeds into one unified map.

Tasks:

Calibrate each camera → map coordinates.

Merge overlapping coverage.

Prevent double-counting (spot ID is global, not per camera).

Test with synthetic multi-camera footage.

Deliverable: Unified dashboard for multiple cameras.

Phase 7 — Optimization & Deployment

Goals: Prepare for real-world test.

Tasks:

Optimize YOLO model size (yolov8n/s for speed).

Enable GPU acceleration (TensorRT, CUDA).

Dockerize backend & detection service.

Deploy to local server / edge device (Jetson Xavier / cloud GPU).

Add basic privacy masking (blur faces & plates).

Deliverable: Deployable containerized system.

 Dataset Sources

PKLot (large-scale parking dataset)

CNRPark + EXT (varied weather, lighting)

Custom labeled footage (for “open door” & “car leaving”)

 Suggested Timeline

Phase

Time Estimate

0–1

1 week

2

1 week

3

1–2 weeks

4–5

1–2 weeks

6–7

2–3 weeks

Total: ~6–9 weeks for MVP to deployment

If you want, I can now draw the full system architecture diagram so you can see how cameras, AI models, backend, and UI connect.

Do you want me to prepare that next?

You said:

just for you note. Please keep you tone/mood as my friend and colleague that I like to collaborate, feel free to use simpler words. I'd like to navigate through this project with joy and easiness like it's less like work and more like an enjoyable process of creating something new and really usefull and all the excitment around that.

ChatGPT said:

Updated saved memory

Got it — I'll keep it light, friendly, and fun so it feels like we're building something awesome together rather than grinding through a task list.

Want me to sketch that system architecture diagram next so we can see the whole Smart Parking flow in one picture?

You said:

yes, sure

ChatGPT said:

@img7.png

Here's our Smart Parking Assistant flow in one picture — from cameras all the way to the map on someone's phone.

It's kind of like we're building a little "brain" that sees, thinks, and talks to the user in real time. Want me to also add the multi-camera integration part to this diagram so it shows how they all connect to the same backend?

You said:

yes, please

ChatGPT said:

@img8.png

Here's the updated version — now all three cameras feed into the same detection/tracking brain, and everything funnels into one backend that powers the live map.

If you want, I can also add a real-world example flow with tiny icons of cars, cameras, and map markers so it feels more like we're storyboarding the app. That could make it easier to visualize the user experience.

You said:

yes, please

ChatGPT said:

@img9.png

Here's our "fun storyboard" — now we've got little cars feeding into the camera system, the detection brain doing its magic, and the results popping up on a friendly map.

It's already starting to feel like a real product rather than just a diagram.

Next, we could make a mockup of the actual live map UI so we can imagine what the user will see when spots turn green or yellow.