**🔹 Complete Architecture for Consistent Card Tracking in Live Blackjack Detection**

Tumhare **YOLO-based card detection system** mein **tracking aur identity consistency** maintain karne ke liye ek **multi-stage pipeline** banani hogi. Neeche **end-to-end architecture** hai jo tumhare WebSocket-based system ke liye **tracking, re-identification, aur occlusion handling** ko improve karega.

**🔹 System Architecture Overview**

🔹 **Input:** Live video frames from WebSocket.  
🔹 **Stage 1:** YOLO detects cards in frames.  
🔹 **Stage 2:** Kalman Filter + SORT/DeepSORT assigns unique IDs to track each card.  
🔹 **Stage 3:** Re-ID Model ensures same card gets same ID across frames.  
🔹 **Stage 4:** Mask R-CNN handles occlusion cases.  
🔹 **Stage 5:** Consistent card states are maintained in memory (to avoid flickering detections).  
🔹 **Output:** Final structured data with consistent card detections.

**🔹 Implementation Breakdown (Step-by-Step)**

**🟢 Step 1: YOLO Object Detection (Already Done ✅)**

* YOLO **detects** cards in each frame.
* **Output:** Bounding boxes, class labels, confidence scores.

**🟢 Step 2: Multi-Object Tracking (SORT/DeepSORT + Kalman Filter)**

🔹 **Why?** YOLO works per-frame, lekin tracking ke bina **same card har frame mein naya card lagta hai**, jo incorrect hai.  
🔹 **Solution:** Tracking se **object ka motion predict** kar ke same ID maintain kar sakte hain.

✅ **Integration in tracker.py:**

1. **SORT or DeepSORT use karo:**
   * SORT = Simple & Fast (Kalman Filter + IoU matching).
   * DeepSORT = Feature-based (Agar ek card temporarily hide ho jaye toh bhi track kar sakta hai).
2. **Kalman Filter ka use:**
   * Predict karta hai ke ek card **next frame mein kaha ho sakta hai** (helps when card is partially hidden).

**📌 Code Example (tracker.py mein add karo):**

python

CopyEdit

from deep\_sort\_realtime.deepsort\_tracker import DeepSort

import numpy as np

class CardTracker:

def \_\_init\_\_(self):

self.tracker = DeepSort(max\_age=30) # Track for 30 frames

def update(self, detections):

"""

detections = [(x1, y1, x2, y2, confidence, class\_id), ...]

Returns: Tracked object IDs with their positions

"""

tracks = self.tracker.update\_tracks(detections, frame=np.zeros((720,1280,3)), embeds=None)

tracked\_objects = []

for track in tracks:

if not track.is\_confirmed():

continue

tracked\_objects.append({

"id": track.track\_id,

"bbox": track.to\_ltwh(),

"label": track.det\_class

})

return tracked\_objects

**🟢 Step 3: Re-Identification Model (Re-ID)**

🔹 **Problem:** Agar ek card **occlude** ho jaye ya temporarily frame se bahar ho jaye, toh tracking ID change ho sakti hai.  
🔹 **Solution:** **Re-ID Model** ko ek **feature extractor** ke tor pe use karo jo cards ke embeddings generate kare aur track kare.

✅ **Integration in tracker.py:**

1. **Ek lightweight Re-ID model use karo (e.g., FastReID ya ResNet50)** jo card ka **unique fingerprint** generate kare.
2. **Agar tracking ID lost ho jaye**, toh Re-ID model card ko **dubara recognize** karke usko **same ID assign** kare.

**📌 Code Example:**

python

CopyEdit

from torchvision import models, transforms

import torch

class ReIDModel:

def \_\_init\_\_(self):

self.model = models.resnet50(pretrained=True)

self.model.fc = torch.nn.Identity() # Remove last classification layer

self.model.eval()

self.transform = transforms.Compose([

transforms.Resize((128, 128)),

transforms.ToTensor(),

])

def extract\_features(self, image):

"""

Takes a cropped card image and returns a unique embedding vector

"""

img\_tensor = self.transform(image).unsqueeze(0)

with torch.no\_grad():

embedding = self.model(img\_tensor)

return embedding.numpy()

✅ **Usage:**

* Har detected card ka **embedding extract karo**.
* **Compare embeddings of new detections with previous embeddings**.
* Agar similarity score **> 0.9 hai**, toh **same ID assign karo**.

**🟢 Step 4: Occlusion Handling with Mask R-CNN**

🔹 **Problem:** Kabhi kabhi ek card dusre ke neeche chhup jata hai aur YOLO use detect nahi kar pata.  
🔹 **Solution:** **Mask R-CNN** se **instance segmentation** use karke overlapping cards ko better detect kar sakte hain.

✅ **Integration in tracker.py:**

1. **Mask R-CNN use karo taake overlapping objects pe masks generate ho sakein**.
2. **Bounding boxes aur mask-based segmentation combine karo**.
3. **Occluded cards ke liye segmentation-based tracking improve karo**.

**📌 Code Example:**

python

CopyEdit

import torchvision

import torch

class MaskRCNNDetector:

def \_\_init\_\_(self):

self.model = torchvision.models.detection.maskrcnn\_resnet50\_fpn(pretrained=True)

self.model.eval()

def detect(self, image):

"""

Returns bounding boxes and segmentation masks

"""

transform = transforms.Compose([transforms.ToTensor()])

img\_tensor = transform(image).unsqueeze(0)

with torch.no\_grad():

predictions = self.model(img\_tensor)[0]

boxes = predictions['boxes'].cpu().numpy()

masks = predictions['masks'].cpu().numpy()

scores = predictions['scores'].cpu().numpy()

return boxes, masks, scores

✅ **Usage:**

* **YOLO detections + Mask R-CNN detections combine karo.**
* **Agar YOLO kisi overlapping card ko detect nahi kar raha, toh Mask R-CNN ka mask check karo.**
* **Final detection ko tracker.py mein pass karo.**

**🔹 Final Workflow Implementation**

1. **WebSocket se frame lo**
2. **YOLO se detections lo**
3. **Tracking model (DeepSORT + Kalman Filter) apply karo**
4. **Re-ID model se embeddings extract karo (to ensure ID consistency)**
5. **Mask R-CNN use karo (for occluded cards)**
6. **Final cleaned output send karo**

**📌 WebSocket Script Update (websocket\_worker mein ye integrate karo):**

python

CopyEdit

from tracker import CardTracker, ReIDModel, MaskRCNNDetector

tracker = CardTracker()

reid\_model = ReIDModel()

mask\_rcnn = MaskRCNNDetector()

def process\_frame(frame):

results = model(frame, conf=0.05)

detections = []

for result in results:

for box in result.boxes:

x1, y1, x2, y2 = map(int, box.xyxy[0])

label = result.names[int(box.cls[0])]

confidence = box.conf[0].item()

detections.append((x1, y1, x2, y2, confidence, label))

# Mask R-CNN for occlusions

boxes, masks, scores = mask\_rcnn.detect(frame)

# Update tracker

tracked\_objects = tracker.update(detections)

# Re-ID model to maintain consistency

for obj in tracked\_objects:

x1, y1, x2, y2 = obj["bbox"]

card\_crop = frame[y1:y2, x1:x2]

embedding = reid\_model.extract\_features(card\_crop)

# Compare embeddings and assign consistent ID

return tracked\_objects

**🔹 Expected Results**

✅ **More stable tracking (less flickering IDs)**  
✅ **Better occlusion handling**  
✅ **Cards keep the same ID even after temporary disappearance**  
✅ **Overlapping cards are correctly separated**

Ye **modular implementation** tumhare **real-time Blackjack Card Detection System ko kaafi stable bana degi**. Tum pehle sirf SORT/DeepSORT implement kar lo, phir gradually Mask R-CNN aur Re-ID model add kar sakte ho. 🔥

### 🔧 ****Solutions: How to Improve Live Inference?****

#### ✅ **1. Use a Tracking Algorithm (SORT, DeepSORT, or ByteTrack)**

Since your model detects **frame-by-frame**, you need a **tracking algorithm** that:

* **Assigns an ID** to each card (so the same card is recognized in future frames).
* **Tracks the card’s movement** across multiple frames.
* **Maintains detection even during short occlusions** (if another card covers it for a moment).

🚀 **Best Tracking Methods for Your Case:**  
🔹 **SORT (Simple Online and Realtime Tracker)** – Lightweight, good for basic tracking.  
🔹 **DeepSORT** – Uses deep learning for better tracking but is slower.  
🔹 **ByteTrack** – Best for handling occlusions and multiple objects.

📌 **Steps to Integrate SORT with YOLOv8:**

1. Install SORT:

bash

CopyEdit

pip install filterpy

1. Modify your inference code to integrate SORT:

python

CopyEdit

from sort import Sort

tracker = Sort() # Initialize tracker

# Inside your inference loop:

detections = model.predict(frame) # Run YOLO on frame

track\_objects = tracker.update(detections) # Track detected objects

👉 This **reduces flickering** and **improves stability** by maintaining card IDs across frames.

#### ✅ **2. Implement Confidence Score Smoothing**

Your model fluctuates because confidence scores vary across frames.  
🔹 **Solution:** Use an **Exponential Moving Average (EMA)** to stabilize scores.

📌 **Modify YOLOv8 Output:**

python

CopyEdit

alpha = 0.6 # Adjust between 0-1 (higher = smoother)

smoothed\_conf = alpha \* prev\_confidence + (1 - alpha) \* current\_confidence

This prevents **sharp drops in confidence** due to motion blur.

#### ✅ **3. Use Optical Flow to Predict Missing Detections**

If a card disappears (due to occlusion), **Optical Flow** can predict where it should be in the next frame.

📌 **Example using OpenCV Optical Flow:**

python

CopyEdit

import cv2

prev\_gray = cv2.cvtColor(prev\_frame, cv2.COLOR\_BGR2GRAY)

curr\_gray = cv2.cvtColor(curr\_frame, cv2.COLOR\_BGR2GRAY)

flow = cv2.calcOpticalFlowFarneback(prev\_gray, curr\_gray, None, 0.5, 3, 15, 3, 5, 1.2, 0)

This allows **temporal smoothing**, helping your model **"remember"** the card location.

#### ✅ **4. Improve Model Post-Processing (Non-Maximum Suppression - NMS)**

If your model detects the same card multiple times, adjust **NMS settings**:

* **Lower IoU threshold** (default is 0.7, reduce to 0.4-0.5).
* **Increase confidence threshold** to remove duplicate low-confidence detections.

📌 **Example in YOLOv8:**

python

CopyEdit

model.predict(frame, conf=0.5, iou=0.4)

This **reduces overlapping detections**.

### ****🚀 Final Plan: Apply These Techniques Together****

1️⃣ Use **SORT/DeepSORT/ByteTrack** to track cards across frames.  
2️⃣ Apply **EMA smoothing** to stabilize confidence scores.  
3️⃣ Use **Optical Flow** to predict missing detections.  
4️⃣ Tune **NMS settings** to reduce duplicate detections.