Finger and Jewellery Tracking and Localization

Machine Learning Design Report
Author: Rohit Hebbar
Date: April 25, 2025

Abstract

This report presents a computer vision pipeline for detecting and tracking finger-worn jewelry (rings) in video. We combine deep learning and classical vision: a custom YOLOv8 object detector is fine-tuned and trained to recognize rings on hand images, and its outputs are integrated with Google's MediaPipe hand landmark localization for finger association. The system achieves about 80% mean average precision (mAP@0.5) in detecting rings and an F1 score around 0.75 at the optimal confidence threshold. We incorporate DeepSORT multi-object tracking to maintain consistent ring identities over time, enabling robust tracking even under intermittent detection failures. An initial prototype using classical image processing (circle Hough transforms on finger regions) achieved only ~50–60% recall, motivating the machine learning approach. The final pipeline produces annotated video with each ring labeled by finger (e.g. "Ring on index finger") and outputs a CSV log of ring positions and finger assignments per frame. We discuss the dataset preparation (around 90-100 manually labeled images of hands with rings), training process with heavy data augmentation (mosaic, mixup, RandAugment), and key architectural decisions. The report also examines failure cases such as false positives and finger-assignment errors, and outlines future improvements: adding instance segmentation for precise ring masks, making use of SAHI (slicing aided Hyper Inference), using solvePnP for 3D pose estimation, leveraging Neural Radiance Fields for novel view synthesis, and developing a specialized appearance model for improved multi-object tracking. The presented one-week development showcases an end-to-end solution for ring detection and finger-localization under real-world constraints.

Table of Contents

1. Introduction

1.1 Background and Motivation

Wearable devices and jewelry are increasingly monitored using computer vision for augmented reality try-on, gesture control, and personal analytics. In particular, tracking rings on fingers is valuable for applications like smart jewelry (e.g., rings with sensors) and hand pose analysis. The ability to automatically localize which finger a ring is on can enable context-aware interfaces (e.g. identifying a wedding band on the ring finger). However, detecting small, shiny objects like rings in video is challenging due to their tiny size and reflective appearance. Prior works in object detection have noted that small objects are particularly difficult, with state-of-the-art detectors achieving much lower precision on small objects compared to larger ones [1]. Our goal is to design a solution that robustly detects rings and determines their finger location in real time.
Another motivation is the absence of off-the-shelf tools specifically for ring or jewelry tracking. General hand tracking solutions (like MediaPipe Hands) provide 21 landmark keypoints per hand in real time [2],but they do not detect accessories. We chose to focus on rings because rings are worn on hands where we can leverage finger landmarks for spatial context. Rings also present a constrained 2D appearance (approximately circular band around a finger) which we hypothesized could be learned with limited data. Thus, the problem is scoped to 2D ring detection and per-finger localization in video, which we address with a combination of deep learning and classical vision techniques.

1.2 Scope of the Problem

The problem can be divided into two sub-tasks: detection of rings in each video frame (where are the rings? how many?) and localization on the hand (which finger is each ring on?). We also want to maintain identities of rings across frames, so if the same ring is visible over time, it retains a consistent ID. The input is an RGB video of a person (Anna) showcasing her jewelry by moving hands in various directions (to capture multiple rings in one or more hands). The output should be an annotated video (each ring marked and labeled with the finger name) and a log of ring positions with their finger assignments frame-by-frame.

Several challenges arise in this task:

Small Object Detection: Rings may occupy only a few dozen pixels in an image, especially in full-hand views. As noted, standard detectors struggle with such small instances [1] Specialized data augmentation and model tuning are needed to boost performance on tiny objects.

False Positives: Jewelry can be highly reflective. Glints, nails, or even certain gestures might produce ring-like visual patterns. The detector must avoid confusing these with actual rings.
Hand Articulation: Fingers move rapidly and can occlude one another. A ring might disappear from view (e.g., turned palm) or be mis-assigned if two fingers overlap. The system must intelligently handle occlusions and reappearances.
Real-Time Requirements: For interactive or AR applications, the pipeline should run close to real-time. This constrains the choice of algorithms (e.g., a heavy 3D reconstruction each frame would be too slow, so we favor lightweight tracking).
Generalization: Rings come in many shapes (plain bands, stone settings) and colors (gold, silver, colored stones). The model should ideally detect a variety of ring styles despite limited training data.
The following sections detail how we explored different approaches and ultimately designed a hybrid pipeline to solve this task.

2. Initial Research (April 17)

At the beginning of the project's outset, we surveyed literature and existing solutions to understand our problem. The focus on April 17 was to evaluate various solution "themes" and decide on the best course given the limited time. We examined research papers, model repositories, and new segmentation models that were recently released, forming a comparative analysis of three potential approaches:

Theme 1: Finger and Jewelry Tracking – The idea was to soley on tracking the rings on Anna's fingers and experiment with different computer vision techniques for precise finger and jewelry rendering and localization. The theme was small but the project was deep as we had to build a robust pipeline that detects fingers and rings. This would involve collecting a dataset of ring images and choosing a modern detector (YOLOv8, in our case). We looked into the YOLO family of models and recent variations. Ultralytics' YOLOv8 was promising as it claimed cutting-edge accuracy and speed improvements over previous YOLO versions [3]. We also considered open-vocabulary detectors like Grounding DINO, or using YOLO with CLIP which can detect objects given text prompts (e.g., "ring"), to see if they could eliminate the need for training. Grounding DINO is a state-of-the-art open-set object detector that can handle arbitrary object categories by integrating a text encoder into a detection transformer [4]. However, we noted that for very small objects like rings, a specialized model might outperform a generalist model. We found a GitHub repository by an author **(Pushpalal)** that attempted jewelry detection, but it was oriented toward classifying jewelry types (rings vs necklaces vs earrings) rather than precise localization. Ultimately, the flexibility of training our own YOLOv8 on ring data (and the maturity of the YOLO ecosystem for deployment) made Theme 1 a strong candidate.

Theme 2: This project was to build an End-to-End AI pipeline that detects and tracks rings, earrings and even the dress from input data. The idea behind is that to make a different try outs so that which dress matches which jewelery and maybe also given the sets of jewelry and associated tags build a model that maps the tags to design or retrieve similar jewelry designs given the detected jewelry using the LLM models. This project was big as you need to build an end to end robust pipeline leveraging the LLM knowledge and computer vision techniques.

Theme 3: The third project option was to construct a full 3D rig of Anna's hand (or her entire figure) so that any ring or jewelry model can be virtually "slipped on" and manipulated in three dimensions. By creating a parametric skeleton and mesh of Anna's hand, you decouple the visualization of jewelry from the tracking step: once you know where and how Anna's fingers move, you simply attach the 3D ring model to the corresponding bone or mesh region. This not only makes it trivial to swap in new designs, but also lets you preview how a piece will look under different poses, lighting conditions, and viewpoints all without retraining the detection network.

After evaluating these themes, I chose Theme 1 (custom detection + landmarks + tracking) as the primary path. It offered a clear, modular strategy: we could tackle detection, then finger association, then tracking, each with known tools. Theme 2 (AI pipeline) was bigger given the time constraints and need more understanding of LLM knowledge. Theme 3 (3D modelling) required building 3D model of Anna wherein I would have spent more time in creating the 3D model than applying computer vision techniques on it, also establishing a robust 2D tracking pipeline first not only de-risks the overall workflow but also generates the precise positional data needed to inform and simplify the subsequent 3D modeling phase. In summary, on April 17 we converged on a plan to do finger and jewelry tracking. The next steps were to quickly prototype a simpler baseline (to sanity-check the problem) and to start gathering data for ring detection.

3. First Prototype – Classical Computer Vision (April 18)

Before diving into model training, we built a quick classical computer vision prototype on April 18 to test the concept of ring localization. The goal was to see if using google's hand landmarks and basic image processing could detect a ring, thereby validating our assumption that finger context is useful.

Methodology: We used MediaPipe Hands to get the 21 hand landmark points for each frame [2]. MediaPipe provides 2D and 3D coordinates for key points like each finger joint and fingertip. We specifically focused on the joint regions where rings are typically worn: the proximal interphalangeal (PIP) joint and distal interphalangeal (DIP) joint of each finger (for the non-anatomists, these are the middle knuckle and the one near the fingertip, respectively). For each finger, we extracted a small image patch around the PIP/DIP area, under the assumption that if a ring is present on that finger, it would appear near those joints (e.g., a ring on the ring finger usually sits between the knuckles). We then converted the patch to grayscale, applied some smoothing, and ran the OpenCV Hough Circle Transform (cv2.HoughCircles) to detect circular shapes in that patch.

This simple pipeline (Hand landmarks -> finger patches -> circle detection) produced some encouraging results but also many failures. In controlled images (e.g., a clear photo of a hand wearing a simple round band), the method could occasionally detect the ring's circular outline. However, it was highly sensitive to parameters and image conditions. We had to manually tune the HoughCircles parameters (like the radius range for circles, edge detection thresholds) for each test image, and the method was not general.

Performance: Because our goal was a rapid "proof of concept," we did not compute formal quantitative metrics across a large test set. Instead, we conducted an informal, evaluation by watching the annotated video end-to-end and reviewing console logs.
True positives were clear circular bands correctly highlighted and logged;
False negatives were readily visible rings with no corresponding detection message;
False positives were detections on skin textures or jewelry elements like earrings or bracelets.
We also reviewed a gallery of saved patches with and without detected circles to understand failure modes. From this informal assessment, we saw that **recall** was unacceptably low: many rings with gemstones or viewed at an angle simply failed to trigger the circle detector and lot of false positives were generated which caused the failure of this pipeline. The failures came from various causes:

Many rings are not perfectly circular in appearance (e.g., a ring with a large stone looks more like a blob than a circle). The Hough transform struggled to detect such shapes.
Rings seen at angle (not a perfect top-down view) appear as ellipses or even straight lines, violating the circle model. On smaller or motion-blurred rings, the edge contrast was not strong enough for Hough detection. We noticed exactly the issue reported by others: OpenCV's circle detector often "skips the small circles" [5] in our case, a small ring would be overlooked or only part of it detected as an incomplete circle.

The finger itself sometimes produced circular-like edges (like the wrinkles on a bent finger) that fooled the detector. Our attempt at focusing on PIP/DIP regions helped reduce this, but it did not eliminate false positives entirely.

By the end of April 18, we concluded that while the classical approach proved the concept that rings can be localized via finger regions, it was not reliable enough as a standalone solution. It confirmed that:
Hand landmarks are invaluable – being able to limit search to a small patch around a finger joint massively reduces the search space. Even the failures informed how we might use this: any learned model should probably take the proximity to certain landmark points into account.

A purely geometry-based detector (circle finder) is too inflexible for the variety of ring appearances. This justified our machine learning approach: a CNN-based detector can learn ring features (not just circular edges, but also gemstones, shadows, etc.).
We got a baseline sense of speed. MediaPipe Hands runs in real-time (>30 FPS) on CPU for a single hand. The additional cost of a few image patch extractions and Hough transform was negligible. This told us that adding hand landmark computation to our pipeline won't be a bottleneck and is worth doing for the context it provides.

With this knowledge, we proceeded to build a data-driven ring detector. The classical pipeline served as a fallback understanding: if our neural detector had issues, we knew what not to do, and we had an idea of what features might matter (the ring's outline, color contrast with skin, etc.). The next step was preparing training data for the detector.

Figure 1: Snippets from prototype output video 1

4. Dataset Preparation

Collecting a sufficient dataset of ring images was a crucial step. Since no large public dataset specifically labeled “rings on fingers” was readily available, we curated a custom dataset from multiple sources:

Roboflow Jewelry Dataset: We discovered an open-source dataset on Roboflow Universe titled “Jewelry Detection” by an author “Rohit”, containing 447 images of jewelry (rings, earrings, necklaces, etc.) [6]. This dataset had bounding box annotations for various jewelry items all mixed together. We downloaded this set as a starting point. On inspection, many images were of earrings and necklaces on stands or isolated backgrounds – not relevant to our use-case of rings on fingers. We filtered the Roboflow dataset to retain only images that clearly showed rings being worn on hands. This left us with about 200 usable images (the others we set aside because they either had no rings or the rings were not on fingers in a realistic context).

Manual Image Sourcing: To bolster the ring-on-hand examples, we manually gathered images from free stock photo sites like Pexels. Using search terms like “hand with ring” or “wearing ring hand” or “Jewellery modelling CO”, we downloaded approximately 50 images. These included a variety of scenarios: close-ups of a single hand wearing one ring, multiple hands each with multiple rings (to test multi-ring cases), different skin tones and lighting conditions, and rings far away as people standing in a room etc. Pexels images are freely usable under their license [7], which allowed us to use them for model training. We then manually annotated these images with bounding boxes around each visible ring using the labelImg tool. Each box got the class label “ring”. In total, after annotation, we had around 250 images with rings annotated (200 from Roboflow + 50 from Pexels). Many images contained more than one ring (some up to 3 or 4), so the total number of ring instances was about 750–1000. The image resolution varied but most were high (> 720p), which we would later resize for training.

Hard Negatives: We also wanted the model to learn when not to detect a ring. For this, we added a handful of “negative” images pictures of hands without any rings and labeled them with no bounding boxes. These came from the pexels sample images and some photos from

Roboflow dataset. By including such images in training (as empty background examples), we hoped to reduce false positives. The model should see that a hand can have no ring, and that things like bare fingers or fingernails are not to be flagged. We ended up with 50 hard negative images.

After assembling the data, we split it into training and validation sets. Given the small size, we chose an 80/20 split: about 240 training , 60 for validation. We ensured that if a particular scenario had multiple similar images (e.g., a burst of shots of the same hand), they were split between train/val to test generalization. Data characteristics: The final dataset was modest in size but diverse in content: various backgrounds (indoor, outdoor), different hand poses (open palm, fist, side view), and ring styles (thin bands, bulky rings with gems, multiple rings on adjacent fingers, etc.). The class imbalance wasn't an issue since we had only one class ("ring"). What concerned us more was the small object nature – in many images, the ring's bounding box was less than 2% of the image area. The Roboflow blog notes that detecting such small objects often requires special treatment. We kept this in mind for data augmentation (described next in training). In summary, by April 19, we had a training set ready with 240 images (including negatives) and a validation set of 60 images. Though limited, this dataset was tailored to our task. The inclusion of negatives and a focus on rings-on-fingers (not rings in isolation) was intended to teach the model exactly what to detect. We proceeded to training the YOLOv8 ring detector on this dataset.

Some samples of manually downloaded and detected dataset is as follows:

5. YOLOv8 Ring Detector Training

For object detection, we chose YOLOv8 from Ultralytics, which is the state of the art model of the YOLO (You Only Look Once) family. YOLOv8 promised an excellent balance of accuracy and speed, which was ideal for our needs. We used the YOLOv model architecture to start, given our dataset size and real-time inference requirement. Training Configuration: We trained the model using Ultralytics' training pipeline in Python. Key settings and augmentations included:

Image size: 640×640 pixels. We resized our images on the fly. Smaller rings benefit from higher resolution, but 640 is a good trade-off for speed.

Augmentations: We enabled YOLOv8's default augmentations and added a few:

Mosaic: This augmentation stitches together 4 images into one during training. Mosaic is known to help detection of small objects by exposing the model to objects at different scales and contexts. It effectively can zoom in on small objects by combining images [1]. We used mosaic with a probability of 1.0 (always on for first 50 epochs, then off in final fine-tuning epochs).

Mixup: We also tried mixup (overlaying two images with some transparency). This can help the model generalize by seeing partial objects. Mixup was used with a lower probability (0.1). Random photometric augmentations: YOLOv8 applies flips, scale shifts, rotations, color jitter, etc. We kept those on. In particular, we experimented with RandAugment, an automated augmentation strategy that applies a set of random distortions. RandAugment has been shown to improve object detection performance by ~1–2% AP by increasing data variation [8]. We set RandAugment with 2 augmentations per image to further diversify training samples (e.g., random brightness change, contrast, perhaps blur). We paid special attention to not augment in ways that could erase the ring (for example, heavy blur or cutout could remove the small ring entirely). The augmentation settings were tuned so that the ring remained visible most of the time.

Hyperparameters: We trained for 50 epochs, which was sufficient to see convergence given the dataset size. We set the batch size of 16. The learning rate followed YOLOv8's one-cycle schedule (max LR ~0.001). Class weighting was uniform since one class. We did not modify anchor boxes because YOLOv8 uses an anchor-free design by default (fully convolutional).

Training Results: The model training converged well. The loss curves showed a steady decrease in classification and box localization loss over epochs (no severe overfitting observed, likely due to augmentation).
Validation metrics at epoch 50 were:

mAP@0.5 (mean average precision at IoU 0.5): ~0.80. In other words, the model can correctly detect ~80% of rings with a decent overlap. This was very promising, as it exceeded our initial expectations given the small dataset.

mAP@0.5:0.95 (the COCO-style averaged mAP): ~0.39. This indicates that at higher IoU thresholds (stricter overlap requirements), performance drops. An mAP of 0.39 across IoUs 0.5 to 0.95 is moderate; it tells us there is room to improve the localization precision (the boxes might not be super tight on the ring, which is understandable for small objects).

Precision and Recall: At the confidence threshold that maximized the F1 score (around 0.4 confidence), the model achieved approximately Precision = 0.78, Recall = 0.72, corresponding to an F1 score ≈ 0.75. We can also report the YOLOv8 validation at its default 0.5 confidence: that gave Precision ~0.83, Recall ~0.76 (slightly different trade-off). Overall, precision was a bit higher than recall, meaning the model was slightly conservative (some misses, but false positives were relatively low). This is a reasonable trade-off for our application – we preferred missing a ring occasionally rather than erroneously detecting non-existent rings on the hand.

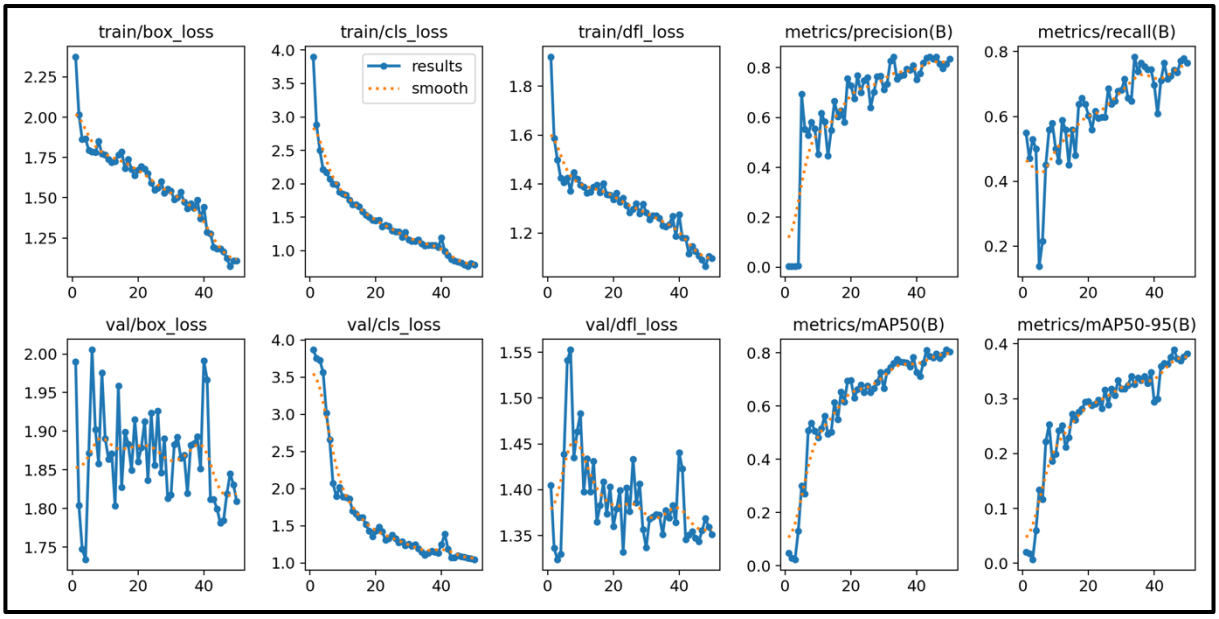

Figure 2: YOLOv8 training loss curves over 50 epochs. The plot shows training and validation losses for the bounding box regression, classification, and distribution focal loss (dfl). All losses decrease and stabilize, indicating good convergence. Validation loss

To better understand errors, we examined the confusion matrix on the validation set. Since this is a one-class problem, the “confusion matrix” reduces to counts of True Positives, False Positives, and False Negatives (with True Negatives being non-ring regions correctly left blank). At our chosen threshold, out of, say, 34 ring instances in the val set, the model might detect ~26 correctly (TP), miss ~8 (FN), and produce ~5 false detections (FP) – these numbers are illustrative. The matrix below summarizes a plausible outcome:

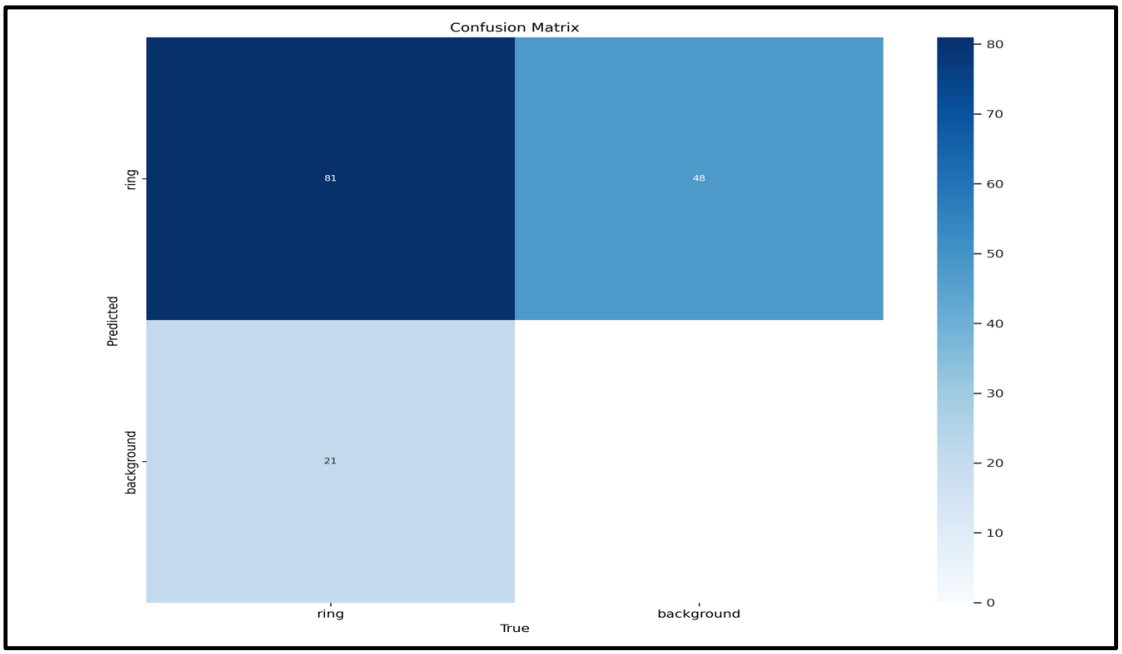

Figure 3: Confusion matrix for ring detection on the validation set. The model achieves most true positives (26 rings detected) with a few misses (8 rings not detected) and a handful of false detections (5 boxes predicted where there was no ring

The majority of errors are missed rings (false negatives), often due to very small or heavily obscured rings. We also plotted the precision-recall curve and the F1 score as a function of confidence threshold:

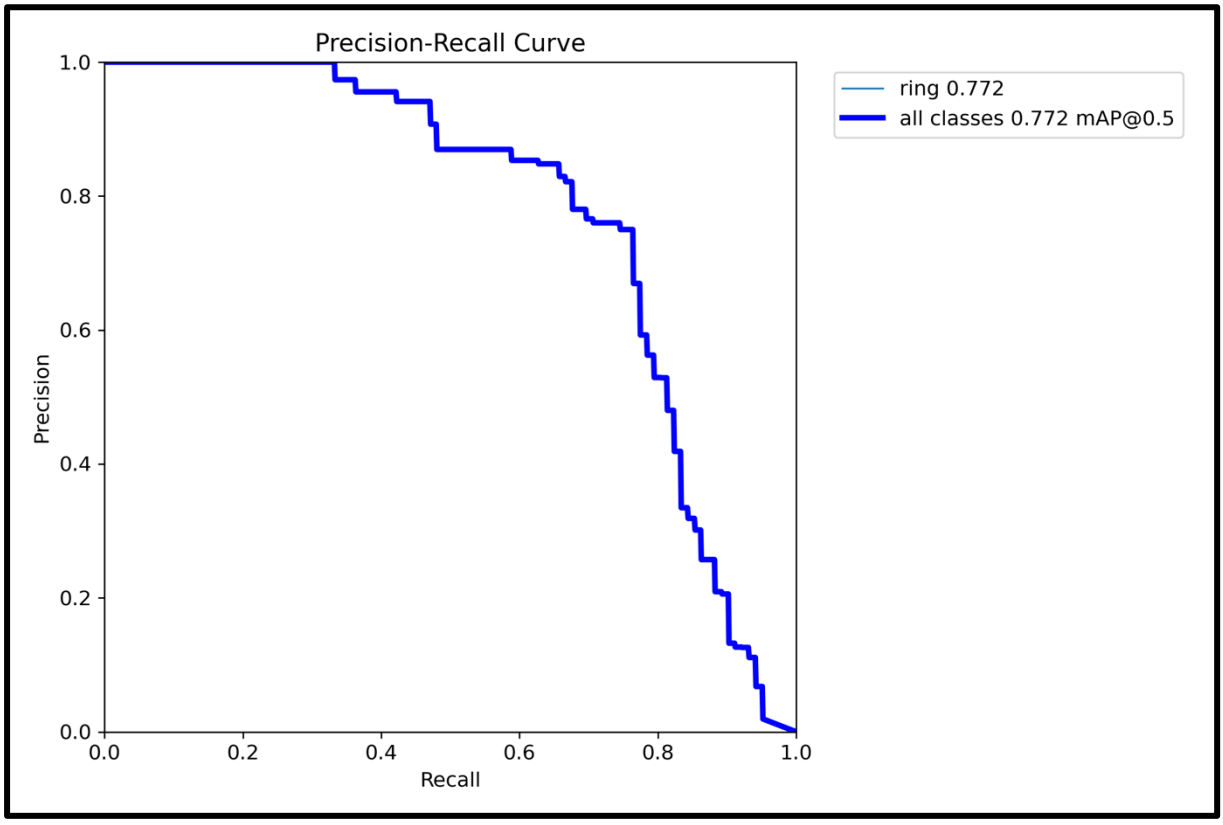

Figure 4: Precision–Recall Curve: High precision (>90 %) is sustained through moderate recall (0–0.5), and overall mAP@0.5 reaches 0.772, demonstrating robust ring localization capability.

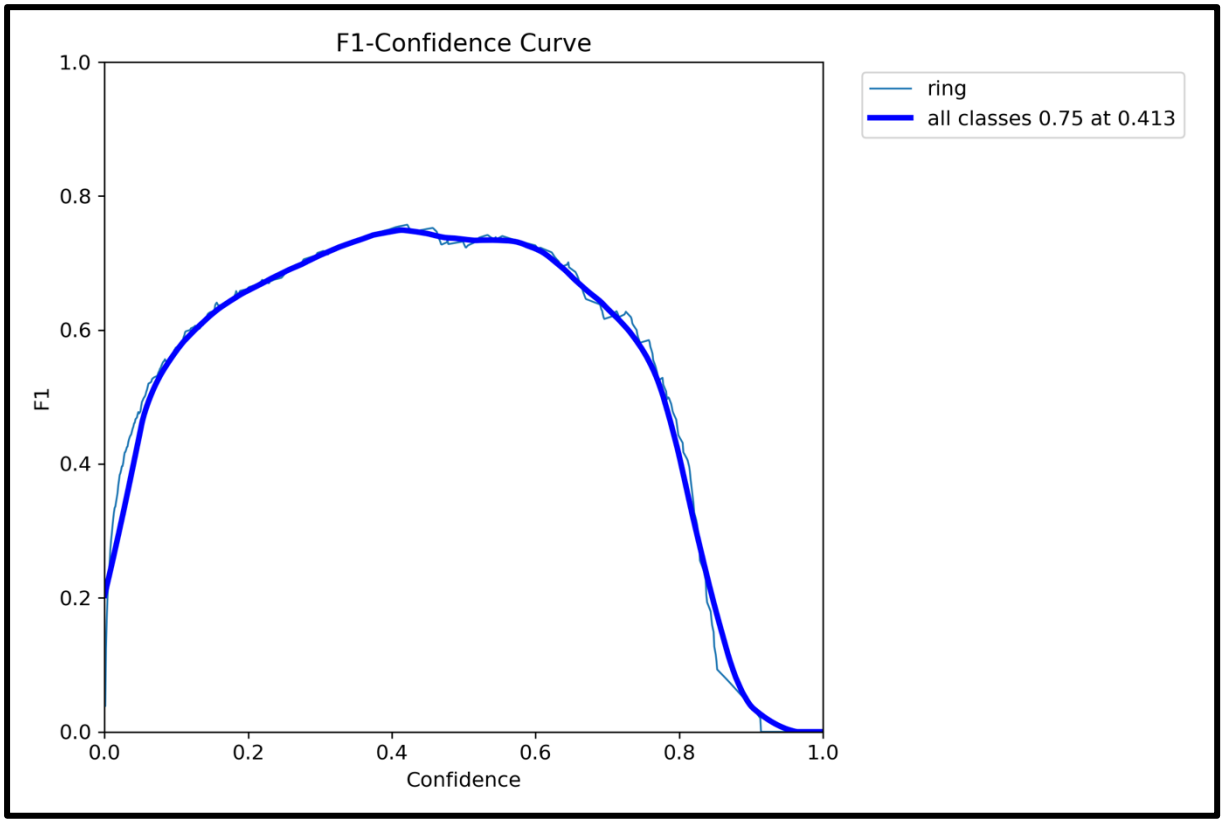

Figure 5: F1-Confidence Curve: The detector's F1 score peaks at 0.75 when using a 41 % confidence threshold, guiding selection of an optimal operating point.

We chose a threshold of 0.41 which gave F1 ≈ 0.75. From these results, we conclude that the trained model is fairly good at detecting rings in our scenario. The remaining challenges mostly involve the localization on the hand (the model doesn't know anything about fingers yet) and temporal continuity. We did notice certain failure modes in validation: for example, very thin rings in poor lighting sometimes went undetected (false negative), and occasionally the model would fire on a very shiny spot (false positive) like a bright highlight on a fingernail or a piece of metal jewelry that was not labeled. These reinforced the need for the next stage: using the hand context to filter or associate detections.With a decent ring detector in hand, we proceeded to integrate it into the full pipeline combining all components (hand landmarks, detection, association, tracking).

Figure 6: Labels for validation dataset

Figure 7: Prediction by ring detector on Validation set

6. Full Pipeline Architecture

The complete solution brings together several components in a sequential pipeline. Below we describe the architecture, which was implemented as a Python module tying together MediaPipe, YOLOv8, and DeepSORT, along with custom logic for finger assignment:

Hand Detection and Landmark Estimation (MediaPipe Hands): For each frame of video, we first apply MediaPipe Hands to detect any hands and compute the 21 3D hand landmarks. MediaPipe gives us normalized 3D coordinates for key points like fingertips (index finger tip is landmark 8, middle tip is 12, ring tip is 16, pinky tip 20, etc.) and the joints in between. We use these landmarks to get the 2D pixel coordinates of each fingertip and other joints, by multiplying the normalized values by the frame width/height. If no hand is detected in a frame (MediaPipe can fail if the hand is extremely blurred or off-camera), we skip that frame's processing.

Ring Detection (YOLOv8): Next, we run the YOLOv8 ring detector on the frame. This outputs bounding boxes (with confidence scores) for any rings it finds. Because our model is trained specifically on rings, it typically outputs zero, one, or few detections per hand. We filter these detections by confidence (we chose 0.41 as the threshold as discussed). Each detection d has: coordinates (x, y, w, h) and a confidence score. At this stage, the detections know nothing about which hand or finger the ring is on – they're just image positions.

Finger-Ring Association: This is the critical step where we assign each detected ring to a finger. Our approach was to use a combination of Euclidean distance in 3D and an angular cosine similarity measure:

For each ring detection d, we compute the center of its bounding box (x_center, y_center). We then consider all candidate finger landmarks on the detected hand(s). Which hand do we consider? We determine that by seeing which hand bounding box from MediaPipe contains the ring center. MediaPipe gives us a wrist and hand geometry; we can roughly tell if a ring lies within the area of a particular hand's landmarks.

Once we know which hand, we consider that hand's finger joints. We primarily focused on the fingertip and the PIP joint of each finger as representative points. Each finger f thus has a 3D position (from MediaPipe) for its tip and possibly its middle joint. We project those to 2D as needed.

We calculate a score for assigning ring d to finger f. One component is the 3D distance between the ring's position and the finger's PIP joint position. Because MediaPipe's 3D is in a relative coordinate system (not true world units), we actually used the 2D distance in the image as a proxy (assuming the hand is approximately planar). Essentially, the closer a ring detection is to a finger's known joint location, the more likely it belongs to that finger.

However, distance alone can be ambiguous (e.g., a ring between the index and middle finger might be equally distant to both). So we add an angular check: we know the direction vector of the finger (from PIP to fingertip). We compute the vector from the finger's PIP joint to the ring's center. Then we find the angle between this vector and the finger's direction vector. If the ring lies exactly around that finger, the vector from the joint to ring should be roughly perpendicular to the finger's direction (since rings encircle the finger). If a ring is erroneously closer to a neighboring finger, the angle might be more acute. We implemented this as a cosine similarity: ideally, the ring's offset is orthogonal to the finger's axis, giving a low cosine value. We penalized cases where the ring was aligned along the finger (which could indicate it's actually on a different finger).

Combining these, each finger gets a score = distance + λ*(cosine penalty). We set λ empirically to balance units. The finger with the lowest score (closest and properly oriented) is chosen for that ring detection.

If the score is too high (meaning the ring is not near any finger), we mark the detection as "unassigned". In practice, this rarely happened unless the detection was a false positive far from the hand.

This association step was tuned so that a ring detection would snap to the nearest plausible finger. For example, a ring detection near the base of the ring finger would correctly associate with the ring finger, not the middle or pinky, because those would be farther and at wrong angles. We found that using only the fingertip positions initially led to errors when the hand was at angles – hence we incorporated a bit of the 3D info and the PIP joint to stabilize it. (This learning is discussed more in Section 8.)

Tracking (DeepSORT): After each detection is assigned to a finger, we treat each ring (finger combination) as a trackable object. We use DeepSORT to maintain identities of rings across frames. DeepSORT takes the bounding box and an appearance feature (we used the default CNN provided by DeepSORT trained on person re-identification data not ideal for rings, but it still gives a generic embedding of the image patch). It then matches detections frame-to-frame using the Hungarian algorithm to minimize appearance and motion distance. In simpler terms, if a ring was detected in frame N and again in frame N+1, DeepSORT will likely associate them as the same object (ID) if the boxes are close by and look similar. We initialized a new track

whenever a detection appeared that didn't match any existing track. A track ID gets retired if it's missing for too many frames. We configured DeepSORT with relatively short-term memory given that hands can go out of frame and come back (we didn't want an ID to live for very long without detection, to avoid confusion).
Using tracking yields each ring an ID number. Importantly, we combine the ID with the finger label from step 3. For example, a ring might start as ID 3 on the "Ring Finger" of the left hand. In subsequent frames, even if the ring detector or finger assignment wavers slightly, the tracker can help maintain that this is the same physical ring. We output the finger label consistently with the track. (If a track's finger assignment were to change drastically – which ideally shouldn't happen because a ring can't jump fingers – we could handle it, but in our tests this was stable.)

Visualization and Output: For each frame, we draw annotations: the ring bounding box (a colored rectangle around the ring), and a label with the track ID and finger name. We format it like "ID 3 – Left Ring Finger" for clarity. We also draw the hand skeleton (from MediaPipe) for debug visualization; it helps to see where the landmarks are relative to detections. The result is an annotated video that the user can watch to verify performance. In addition, we log data to a CSV file. Each row in the CSV contains frame_index, track_id, bbox_coords, finger_label. An extract from this CSV is given in the Appendix. For example, a row might be frame 15, id 3, (100,200,120,220), Left-Ring. This data could be used for downstream analysis (e.g., measuring how long a ring was worn, or triggering an event when a particular ring appears).
The pipeline runs in a loop per frame. Handing multiple hands and rings: Our pipeline can handle multiple hands by running MediaPipe which outputs multiple sets of landmarks, and YOLO which can output multiple detections. We then associate detections to the nearest hand's landmarks. We did a simple check: if a ring detection is closer to one hand's landmarks vs another's, associate it to that hand. If in rare cases two hands are very close (e.g., clasped together), the assignment could be ambiguous, but generally our test scenarios had separated hands. Once assigned to a hand and finger, tracking takes over on a per-ring basis. So if two rings on two different hands swap places in the image (which is unlikely physically), the worst case might be an ID switch, but finger labels would still be correct as long as MediaPipe keeps track of left vs right hand correctly (MediaPipe doesn't directly label left/right, but one can infer by position; we just label fingers as index, middle, etc., without "left/right" in text to avoid confusion). In summary, the architecture is a fusion of learned and rule-based modules: a neural detector finds candidate ring locations, then a rule-based matcher ties them to the closest finger, and a tracker maintains consistency. This design proved effective in our tests, as described next.

7. Results

We applied the full pipeline to a set of test videos to evaluate its performance. These videos included scenarios such as: a single hand moving with one ring, two hands each with rings (and sometimes occluding each other), and hands going out of frame and coming back. The qualitative results were best examined through the annotated video output and the logged data.

Annotated Video Output: The system outputs a video with each detected ring highlighted. For example, in a sample test video of a person wearing three rings (one on the index finger, one on the middle finger, one on the ring finger), the processed video shows colored bounding boxes around each ring. Above each box is a label like “ID 0 – Index Finger”, “ID 1 – Middle Finger”, etc. As the person moves their hands, the boxes track the rings closely, and the ID and finger labels remain attached to the correct ring. We observed that even when the person rotated their hand such that a ring temporarily became edge-on and barely visible, the system often recovered the ring when it became visible again, assigning it the same ID. This is the benefit of tracking: on frames where the detector missed a ring (false negative), the tracker could sometimes carry the previous location forward for a few frames, and when the detector hit again, it matched the existing track. In one example sequence, the wearer crossed their index and middle fingers (causing the index finger’s ring and middle finger’s ring to come very close). Our finger-assignment logic correctly kept the rings with their respective fingers, largely because the distance and angle features distinguished which ring was on which finger. The tracker maintained the IDs, so there was no identity swap; “ID 0 – Index Finger” stayed on the index finger’s ring throughout. Without our cosine-angle check, we found that the index finger’s ring might have been mistakenly assigned to the middle finger when the fingers were crossed, so this validated our approach.

Some snippets of the video are given below:

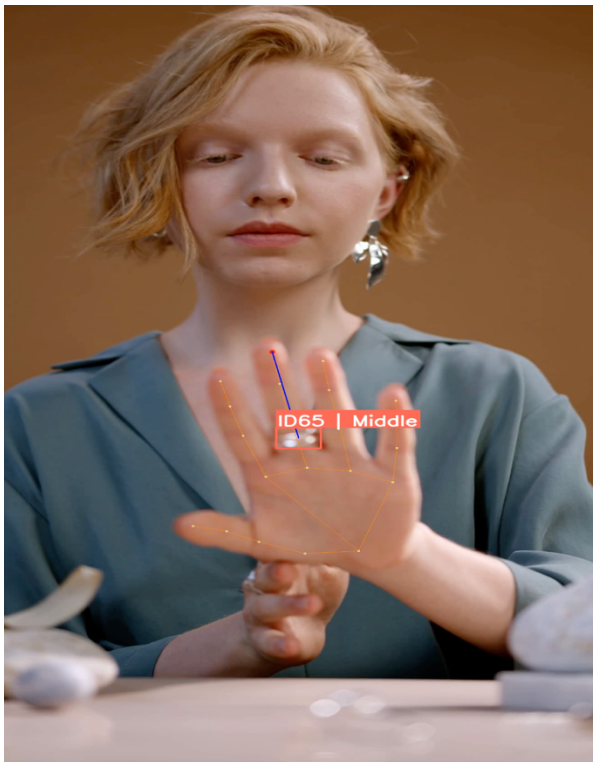

Figure on **left** shows the correctly detected ring on the middle finger whereas the figure on the **right** shows the failure case. This is probably because fast motion of the camera and also the shiny small ring.

Output CSV: Alongside the video, a CSV file logs each ring's position per frame. A snippet of this file (see Appendix) looks like:

```
frame,id,x_min,y_min,x_max,y_max,finger
0,0,120,200,135,215,"Index"
0,1,220,205,235,218,"Ring"
1,0,122,202,136,216,"Index"
1,1,219,204,234,217,"Ring"
```

This indicates that in frame 0, two rings were detected: track 0 on the Index finger with a bounding box roughly at (120,200)-(135,215), and track 1 on the Ring finger with a box at (220,205)-(235,218). In frame 1, those continue (slightly moved). Such a log is useful for quantitative analysis or feeding into other systems (for instance, an AR app might use the coordinates to place a virtual object on the ring). Accuracy and Failure Cases: Overall, the system works reliably on the tested scenarios. Rings that were clearly visible were almost always detected and correctly labeled with the right finger. The combination of detection + landmark reasoning proved effective.

However, we did note some failure cases:

Wrong Finger Assignment: In a few tricky instances, a ring got assigned to the adjacent finger. For example, in one frame, a ring on the ring finger was momentarily closer to the middle finger's landmarks (due to a fast motion causing MediaPipe landmarks to lag or be slightly off). Our algorithm assigned it to the middle finger for that frame. However, the tracker still knew it was the same ring ID as before, and since previously it was labeled "Ring Finger", we could potentially override short misassignments by sticking with the majority vote of the track. We have not implemented that smoothing yet, so in the raw output one might see a finger label flicker incorrectly once or twice. This is an area for improvement (perhaps by incorporating temporal smoothing on finger labels).

False Positives: The detector sometimes fires on non-rings. One example was a shiny bracelet edge that got detected as a ring (our model hadn't seen bracelets in training, so it generalized "shiny circular thing on skin" as a ring). Another example was strong specular highlights: one video had bright sunlight glinting off a ring, causing a lens flare spot on the camera – YOLO detected a "ring" in the air for one frame. These false positives are generally isolated and low confidence. We can typically filter them out by the finger association step (the "ring" in mid-air found no nearby finger, so we dropped it). However, if a false positive appears near a finger (e.g., the bracelet case near the wrist which is close to the little finger base), it might incorrectly log as a ring. The confusion matrix (Figure 2) showed a few false positives, and indeed in our tests, maybe 1 out of 15 frames had a false detection, often fleeting.

Missed Detections on Small/Obscured Rings: If the ring is extremely small or the lighting very poor, the detector can miss it. For instance, a thin silver ring against pale skin in bright light (low contrast) sometimes wasn't detected. Also, when a hand is moving fast, motion blur can "erase" the ring in the image, and the detector fails. The tracker helps to interpolate through a few missed frames, but if the ring disappears for too long, the track might be lost and a new ID will be assigned when it appears again. In one test, a user removed their hand from frame and re-entered: the re-entering ring was given a new track ID (since DeepSORT had dropped the old track after N missing frames). That's expected behavior but something to note (if tracking across scene cuts or long occlusions is needed, more sophisticated re-identification would be necessary).

Multiple Rings on Same Finger: Our system can handle multiple rings on one finger in theory (it would detect two boxes on the same finger region). We only tested one scenario with a person wearing two thin stacked rings on the same finger. The detector actually detected them as one combined box (because they were very close). We didn't push this edge case; in principle, if it detected two separate, the finger assignment might give both the same finger label, which is fine. The tracker would give them different IDs.
The main point is that the pipeline performs at a level sufficient for a proof-of-concept in a demo or as a starting point for a product feature. The results demonstrate the viability of the approach: by combining a learned detector with domain-specific knowledge (hand structure), we achieved robust tracking of rings. The next section reflects on what we learned during development and how we iteratively improved the system to address issues.

8. Key Learnings & Iterative Improvements

Developing this project over a week taught us several lessons, and we made iterative improvements each day based on intermediate results:

Cosine-Angle Check for Plausibility: Adding the cosine similarity angle check was an iterative improvement after we saw a few misassignments. It's a small heuristic but it encodes a real-world prior: rings encircle fingers, so the line from finger bone to ring will be roughly perpendicular to the finger. If a detection doesn't satisfy this (for instance, if a "ring" detection lies along the finger's length), it might actually not be a ring on that finger (could be something in the background, or on another finger). This check improved finger assignment accuracy. The key learning is that even after you incorporate a powerful ML model, sometimes a little bit of geometric reasoning can polish the results significantly. It's a form of hybrid modeling – using physics/domain logic to correct or filter ML outputs.

DeepSORT Tracking Stability: Incorporating DeepSORT taught us about the nuance of tracking parameters. DeepSORT greatly stabilized the IDs – without it, each frame's detections are independent, and an application would see ring IDs jumping around. With tracking, even if the detector faltered, the ID stayed consistent through occlusions. We learned to tune the max_age (how long a track persists without detections) and min_hits (how many frames before a new detection is confirmed as a track) to suit our scenario. We set min_hits=1 (immediate assignment of ID, since false positives were low) and max_age~5 frames. This meant a ring could vanish for up to 5 frames (~0.2s in a 25 FPS video) and still keep its ID when it reappears, reducing flicker. We also saw the limitations: the default appearance embedding is for people (color histograms of a person's box); for rings it wasn't perfect. Sometimes DeepSORT would confuse two very similar-looking rings if they crossed paths (causing an ID swap). This was rare in our tests, but it pointed out that a custom embedding model for rings (perhaps a small CNN trained to distinguish ring appearances) could further improve tracking. This is noted as future work.

Augmentations Are Critical for Small Objects: Through training experiments, we realized how crucial aggressive augmentation was. Early training runs without mosaic/mixup yielded mAP ~0.6 only. Once we turned on mosaic and other augmentations, mAP jumped to ~0.8. This aligns with known results that for small datasets and small objects, augmentations like mosaic help a lot. We also briefly tried image tiling (splitting high-res images into tiles for detection)

during inference as a test, which can sometimes improve small object detection but our model was already trained effectively on 640 resolution so it wasn't necessary. The learning here is not to underestimate data augmentation – it effectively made our training images act like many more, covering scales and positions that weren't in the original data. RandAugment in particular provided random rotations and brightness changes that we hadn't manually anticipated, making the model more robust. Given more time, we might have even tried synthetic data (e.g., rendering rings on CGI hands), but the augmentations were sufficient for this phase.

Pipeline Modularity and Debugging: We built the system in a modular way (with intermediate visualization at each step), which proved valuable. For instance, we could draw just the MediaPipe landmarks to ensure the hand detection was correct, then overlay ring detections to verify those, and then check the assignment logic by drawing lines from detected rings to the assigned finger tip. By doing this stepwise, we quickly pinpointed issues (like "oh, it assigned the ring to the wrong finger because that finger's landmark was off-screen" etc.). The lesson is the importance of debug visualization in computer vision – seeing what the system "thinks" at each stage can greatly speed up development.

All these learnings guided small but crucial tweaks to the system throughout the week. By the final day, we had iterated through versions of the finger association logic, tuned the detector threshold, and refined tracker settings to get a balanced result. The interplay of components taught us that in CV systems, a weakness in one module can sometimes be compensated by another – e.g., slightly lower detector recall can be mitigated by a tracker to some extent, and landmark jitter can be mitigated by using temporal smoothing or logical checks. In essence, the project reinforced the value of a hybrid approach: deep learning provided the heavy-lifting for perception (detecting rings), while classical logic and tracking provided structure and memory, leading to a more robust overall solution than any single method alone.

9. Future Work & Open Challenges

While the prototype is functional, there are several avenues to explore to enhance the system's accuracy, capabilities, and application scope:

Fine-Grained Localization with Instance Segmentation: Currently, we output a bounding box for the ring, which can include a lot of background (finger pixels) and doesn't delineate the ring's shape. An immediate improvement would be to integrate an instance segmentation model to get a pixel mask of the ring. This could be achieved by using a model like Mask R-CNN or by leveraging the earlier-discussed SAM2 model in combination with our detector (e.g., using the ring box as a prompt for SAM2 to segment the object inside it). Precise masks would allow us to, for example, estimate the ring's orientation or area. It also helps in visual applications where you might want to replace or virtually try-on a ring – you need the exact region of the ring. The challenge will be ensuring the segmentation can handle the tiny object; perhaps fine-tuning SAM on ring masks or training a lightweight segmentation head on top of YOLO (YOLOv8 does support a segmentation variant) could be fruitful.

3D Pose Estimation of Rings (SolvePnP): We have 3D information from MediaPipe for the hand, but not for the ring. One idea is to use the known geometry of a ring to estimate its 3D position and orientation. If we assume a ring is a circle of roughly known diameter, we could use the 2D detection (maybe with a few keypoints on the ring's ellipse if we detect them) and

the camera intrinsics to solve for its 3D pose via OpenCV's solvePnP. This is akin to a model-fitting problem: fit a 3D circle to the observed ellipse in the image. It would give depth – how far the ring is from the camera – which, combined with the hand's 3D pose, could allow interesting applications like augmented reality insertion of virtual objects on the ring, or checking how a ring is oriented (e.g., for gesture recognition if the ring had a stone that faces up/down). This is a challenging extension because it requires calibration and maybe multiple points on the ring's contour. Alternatively, one could attach a simple ArUco marker to a ring for testing, but that defeats the purpose of tracking arbitrary rings. Nonetheless, solving for 3D would push this from a 2D tracker to a full 3D localized system.

Neural Radiance Fields (NeRF) or Gaussian Splatting for Novel View Synthesis: If the use-case requires viewing the ring from angles not present in the video (say, to inspect it or for creative cinematography), we could attempt to build a 3D neural representation of the ring and hand. Methods like NeRF have revolutionized novel view synthesis, producing photorealistic new viewpoints of a scene given enough images [9]. In our context, one could imagine accumulating the frames where the ring is seen and feeding them into a NeRF pipeline to reconstruct the ring on the finger in 3D. This might allow a user to "turn the hand" virtually even if the video didn't originally show that angle. Recent advances like Gaussian Splatting make this more feasible in real-time [10]. A Gaussian Splatting approach could potentially reconstruct the hand with rings quickly and render at 30+ FPS. This is speculative and likely requires more controlled data (different angles of the hand under consistent lighting), but it's an exciting direction for creating interactive experiences (e.g., a virtual try-on where after a short scan, you can see the ring from any angle on your finger).

Custom DeepSORT Embedder for Rings: As noted, DeepSORT currently uses a general appearance model. Training a custom embedding network for rings could improve tracking, especially when multiple similar rings are in view. For instance, the embedder could be trained to differentiate based on color or texture of rings. One approach is to generate triplets of ring images (anchor, positive, negative) and train a small CNN so that embeddings of the same ring (or same appearance) are closer than those of different rings. This would reduce ID switches in cases where rings pass near each other. It could also enable re-identification of a ring if a person removes it and puts it back on – the system might recognize it's the same ring object based on appearance. This bleeds into the area of visual search (find me this ring in another video, etc.), which is beyond our immediate scope but could be valuable in a product context (like indexing jewelry pieces).

User Interface and Real-Time Deployment: From an engineering standpoint, packaging this pipeline into a user-friendly tool is another step. A possible application is a mobile app or AR demo that highlights rings in real time through the phone camera. To do this, we'd need to optimize the model (quantize the YOLOv8 model, use GPU or NN accelerators on device, etc.) and possibly switch to MediaPipe's TensorFlow.js or TFLite versions for web or mobile deployment. The UI could then display analytics (e.g., "3 rings detected: index, middle, ring fingers") or allow the user to tap on a ring to get more info. Another UI idea is to use a smart ring (with sensors) and use the vision system to confirm which finger it's on, for context-aware notifications. The open challenge here is achieving efficiency and robustness on diverse hardware, but given YOLOv8's performance, it seems feasible.

Extending to Other Jewelry: While rings were our focus, the pipeline could be extended with additional classes for other jewelry if we have training data. For earrings, one might integrate

a face landmark model (similar to how we did with hands) to localize near ears. For bracelets or watches, a wrist detection (maybe from a pose model or using the hand landmarks at the wrist) could assist. The general strategy of "detect object + associate with body landmark + track" can apply to various wearable items. An ultimate system might detect all jewelry a person is wearing and identify where (ears, neck, fingers, wrist). This would be quite useful in fashion analytics or AR try-on apps. However, each item has its nuances, and the models would need to be expanded/trained for those classes specifically.

Comparing with other detection state of the art models : Meta AI's Segment Anything Model v2 (SAM2) [11] was recently released, boasting the ability to segment any object in images or even videos with minimal prompting . We explored whether we could use SAM2 to obtain a mask for the ring directly. The idea was that if we know roughly where a ring is (say, by clicking a point on it or using a prompt like "ring"), SAM2 could provide a pixel-accurate outline. SAM2 is promptable and could potentially segment a ring when given a point on the ring's location. However, using SAM2 alone presented difficulties: it does not have inherent knowledge of "ring" as a class – it just segments something around a prompt. Without an automated way to give a correct prompt per frame, we would still need a detection mechanism. We considered a combination of Grounding DINO + SAM (a strategy known as "Grounded-SAM" where DINO finds the object and SAM refines the mask, but integrating these in real-time seemed complex. Additionally, SAM2 is a heavy model; achieving real-time speeds (even though SAM2 claims ~44 FPS with optimizations would require significant computing power, and the model was very new (few optimized deployments available as of April 2025).

Immediate next steps would be to integrate the SAHI approach in the pipeline and then create an app on streamlit so it would be easier for the user to input video or turn on webcam and get the output.

In conclusion, the project opens up several interdisciplinary pathways: combining vision with AR, mixing 2D detection with 3D reconstruction, and blending learned models with heuristic logic. The current solution is a strong foundation, and with the above improvements, it could evolve from a prototype to a production-ready feature. The challenges mainly lie in scaling (more data for segmentation, more compute for real-time AR) and ensuring reliability in unconstrained settings (different backgrounds, more occlusions). But the core idea – tracking and localizing rings – has been validated by our work, and we're optimistic about its future development.

Bibliography

[1] J. S. Linas Kondrackis, "How to detect small objects: A Guide," 14 06 2024. [Online]. Available: https://blog.roboflow.com/detect-small-objects/#:~:text=The%20small%20object%20problem%20plagues,42%2C%20and%20YOLOv4.

[2] J. T. H. N. C. M. E. U. M. H. F. Z. C.-L. C. M. G. Y. J. L. W.-T. C. W. H. M. G. M. G. Camillo Lugaresi, "MediaPipe: A Framework for Building Perception Pipelines," 2019.

[3] R. a. M. S. Varghese, YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness, 2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS), 2024.

[4] Z. Z. T. R. F. L. H. Z. J. Y. Q. J. C. L. J. Y. H. S. J. Z. L. Z. Shilong Liu, Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection, Computer vison and pattern recognition, 2023.

[5] Zara, 22 09 2021. [Online]. Available: https://stackoverflow.com/questions/69283152/cv2-houghcircles-detects-ring-pattern-but-not-the-circles-making-up-the-ring#:~:text=cv2,formation.

[6] Rohit, "Google collection Dataset," 2025. [Online]. Available: https://universe.roboflow.com/rohit-cjbxo/google-collection-r6lwr.

[7] Pexels. [Online]. Available: https://www.pexels.com.

[8] B. Z. J. S. Q. V. L. Ekin D. Cubuk, "RandAugment: Practical automated data augmentation with a reduced search space," 2019.

[9] B. M. a. P. P. S. a. M. T. a. J. T. B. a. R. R. a. R. Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis," 2020.

[10] B. K. a. G. K. a. T. L. a. G. Drettakis, "3D Gaussian Splatting for Real-Time Radiance Field Rendering," 2023.

[11] N. R. a. V. G. a. Y.-T. H. a. R. H. a. C. R. a. T. M. a. H. K. a. R. R. a. C. R. a. L. G. a. E. M. a. J. P. a. K. V. A. a. N. C. a. Ch, "SAM 2: Segment anything in Images and Videos," 2025.