COMP6030

Hooper Vision Technical Report

Enhancing Basketball Officiating and Analytics through Computer Vision and Deep Learning

Authors

Calvin Princewill (cp697)
Lateef Jawando (lj327)
Samuel Ametefe (sa2090)
Tobi Salau (ooas3)

School of Computing

University of Kent

Word Count: 5940

June 1, 2026

Table of Content

Contents

Abstract

2025 marked the first time in the 147-year history of Wimbledon that automated line-calling was used, with 18 cameras per court replacing human line judges [Fuller, 2024]. This follows a trend of high-level sports leveraging technology to improve officiating, including the Premier League introducing goal-line technology in 2013 and VAR (Video Assistant Referee) in 2019. One sport has been notably absent from this trend, basketball. The NBA, the largest basketball league globally, still depends on manual human stat keeping and has no form of automated officiating.

Hooper Vision leverages Computer Vision and Deep Learning to automate the detection of certain types of rule violations in basketball. We achieved this by training a deep neural network on a labelled dataset of basketball images to predict the action of the player in the frame. Double Dribbles are detected based on the sequence of actions performed by the player in possession of the ball. Hooper Vision can also analyse a user's jump shot, informing them about the arc/angle on their shots, and counting how many shots the user has made. This works great as a practice mode for users to improve their jump shooting.

Hooper Vision runs as a website. The ReactJS framework was used for front-end development. Python was used for back-end development, including a number of libraries such as YOLO, Scikit-Learn, TensorFlow and Numpy.

Introduction

Human refereeing is one of the toughest jobs on Earth. At the highest level of sports, the wrong decision can draw scrutiny from millions of fans. The controversy surrounding Frank Lampard's disallowed goal against Germany in the 2010 FIFA World Cup was enough to push the Premier League to adopt goal-line technology just three years later.

In 2015, the NBA introduced the Last Two Minute Report, a report that breaks down all the refereeing decisions in the last two minutes of close games. The report aims to improve transparency and accountability within refereeing decisions. It cannot alter the outcome of the game, and the decisions cannot be reversed. This has led to some regarding the report as useless, with former NBA player Charles Barkley stating the report "served no purpose" after a 2024 report showed multiple potentially game-altering incorrect decisions after a Philadelphia 76ers playoff game [Gardner, 2024]. The NBA has no automated form of officiating currently.

What if the correct call could always be made in real-time, rather than simply being acknowledged a day later? This question was the foundation of our project. With all four members of the group being basketball players and fans, we decided to create an automated referee for basketball.

We broke down our project into three core aims: double dribble detection, stat keeping and shot feedback, and the website that housed the functionality. We split this amongst the group members. In this report, we will describe how we went from ideation to creation, and the challenges we faced.

Background

Before starting the project, we researched whether people had attempted to automate basketball officiating and stat analysis before and what approaches they took. Below are the most relevant solutions we found.

HomeCourt

HomeCourt is an NBA-partnered app that acts as a personal basketball skills trainer, providing real-time shot metrics such as release time, angle, and shot type from phone footage.
It uses deep learning via TensorFlow and Keras, trained on NVIDIA Tesla P100 GPUs on basketball footage filmed at local schools [Alarcon, 2018]. While not open source, it informed our approach in several ways: we also used TensorFlow and Keras, its GPU requirements pointed us towards the University's HPC cluster Hydra, and it showed that quality insights are achievable from standard phone footage without specialist equipment. HomeCourt is most comparable to our jumpshot feedback feature.

Ayush Pai – AI Basketball Referee

Ayush Pai's AI Basketball Referee detects travelling violations and double dribbles using a YOLO object detection model to identify players, and a YOLO pose estimation model to analyse keypoints on their body. Ayush's referee uses a set of rules based on changes in pose keypoint positions to detect actions and travels. In our project we opted for a different approach and trained a neural network to identify different actions. The AI Basketball Referee is closest in function to our double dribble detection feature.

Basketball datasets

We used HuggingFace and Roboflow to find labelled datasets for basketball. One dataset we used was Simone Francia's SpaceJam dataset, which provided videos from Olympic basketball games with corresponding labels for the action performed in each video. We used this dataset to train our action detection neural network.

Aims

1. Train a custom object detection model capable of identifying players, the ball, and the rim in basketball footage across a variety of environments and camera angles.
2. Build a web application allowing users to upload recorded basketball video and receive an annotated output video with detections rendered on each frame.
3. Implement pose estimation to support violation detection, specifically identifying double dribbles
4. Investigate additional analytical features such as MVP estimation, pass counting and speed estimations
5. Design the backend to handle long-running video processing jobs asynchronously, keeping the interface responsive and providing real-time progress feedback.
6. Explore the feasibility of real-time live stream processing for live game analysis.

Technical Content

Detected Classes

We originally trained our model to detect three classes, players, the ball, and the rim. Tracking player positions over time reveals spacing and formation, ball position relative to players determines possession and shot attempts, and rim detection makes automated scoring possible.

We had to train a custom model because COCO-pretrained detectors don't include basketball or rim as classes. A generic sports ball class exists in COCO but performs poorly in practice, triggering on any round object. Fine-tuning on basketball footage specifically was the only viable option.

As the project expanded into shot feedback and team-based scoring, three classes weren't enough. Knowing a player is near the ball doesn't tell you they are shooting, and knowing the ball is near the rim doesn't confirm it went in.
We expanded to five classes by adding Ball-in-Basket and Player Shooting. Ball-in-Basket fires when the ball is visibly inside the net, giving a signal for a scored basket. Player Shooting fires when a player is in the act of shooting, which is essential for knowing who to attribute a basket to in team mode. Both models are used in the final system. The three-class model is used for accurate player, ball and rim tracking, and the five-class model handles player shooting and ball in basket. Training details are available in the shot analysis section.

Technology Choices

YOLO was chosen as the detection backbone due to its speed, the straightforward Ultralytics fine-tuning API, and strong adoption in sports analytics specifically, meaning there is prior work to reference and existing datasets to build from. It is also one of the fastest single-stage detectors available, which matters for processing long video clips without hardware acceleration.

FastAPI was chosen for the backend over Flask or Django for its native async support. Video processing is long-running and CPU and GPU-bound, so each job runs in a thread pool executor to keep the API responsive during processing.

OpenCV handles all video I/O, reading frames from uploaded files and writing annotated output. It is the standard library for this in Python and integrates directly with NumPy arrays, which is what the model expects as input. The frontend is built with React and Vite.

Optimising Inference Speed via Frame Subsampling

Motivation

Running YOLOv8 on every frame of a video is slow and inefficient, especially on a CPU-only server. At 30 fps, the model needs to process thirty frames per second of footage, but on a single CPU core each forward pass takes roughly 80–300ms (depending on the specific model), so a one-minute clip could take 5 minutes to process**.** This high inference latency makes the system's overall throughput inadequate for practical application. A significant form of inference reduction was needed to increase the processing speed.

Approach and its benefits

The adopted approach is temporal subsampling: the YOLO model is invoked only on every third frame, and the bounding box detections from that inference frame are reused without modification for the two intermediate frames before the next inference call. In parallel, frames are downscaled to 640 pixels wide prior to inference. matching YOLO's native operating resolution, with the resulting bounding coordinates scaled back to the original frame dimensions before annotation. Together, these two measures reduce the theoretical inference workload by approximately a factor of three and eliminate the redundant scaling that occurs when the model internally resizes down to 640 anyway. The implementation is non-destructive, the output video is still written at the original resolution and frame rate: only the frequency of model invocations is reduced, not the output fidelity.

Drawbacks

The most visible limitation of this approach is that bounding box positions are frozen between inference frames. Because the detections drawn on frames one and two of each triplet are identical to those of frame zero, players and the ball continue to move within the frame while their bounding boxes remain stationary. In fast-paced footage where players are accelerating or changing direction or where the ball is in flight, this produces a visually intermittent effect: the annotation appears correctly placed on the inference frame, then lags or misaligns on the subsequent two frames before snapping to the updated position. At the default skip rate of two frames between inferences, this artifact is not noticeable in slow or moderate-paced sequences but becomes noticeable during rapid action. The severity of this effect also impacted by the frame rate of the video, because videos with a lower frame rate will have more things happening between frames. Additionally, a player or ball that

enters the frame between two inference windows will not receive a bounding box until the next inference frame, creating momentary gaps in detection.

After testing with different gameplay footage and because the system is designed to analyse all types of basketball gameplay, the issue of intermittent frames only in fast sequences was deemed to be acceptable.

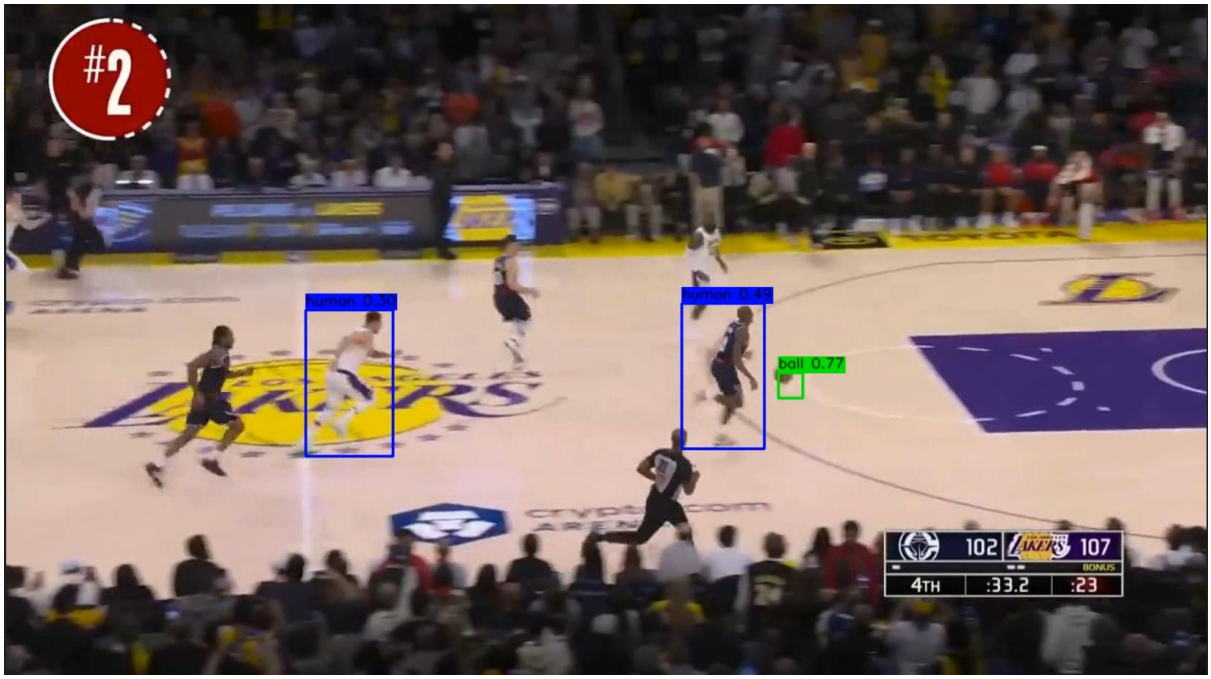


Figure 1: Example of players entering frame between 2 inference windows not receiving a bounding box

Simplicity

A key advantage of temporal subsampling over more sophisticated alternatives is its implementation simplicity. The entire change amounts to a conditional check on the frame index modulo the skip constant, a resize operation, and a coordinate rescaling function of four lines. There is no state machine, no object identity tracking, and no tuning of motion model parameters. The implementation is stateless between runs and introduces no new failure modes: if inference fails on a given frame, the previous detections are silently retained rather than propagating an error. This makes the approach straightforward to reason about, easy to adjust (is a one-line change), and robust to the kinds of challenging edge cases (empty frames, scene cuts, and objects entering and leaving) that more complex tracking pipelines must explicitly handle.

Player Speed Estimation

An experimental implementation of player speed estimation was tested, with Figure 2 showing the speed overlay on a detected player. The approach used the height of each player's bounding box as an estimate for real-world scale. Knowing that the average basketball player is approximately 1.95 meters, the ratio between that known height and the player's pixel height in the frame gives a rough pixel-to-meter conversion, which can then be applied to the movement of each player's centre point between frames to produce a speed in km/h.

The implementation was not carried forward into the main system because the results would be unreliable without proper camera calibration. The main issue is that the system does not map image coordinates to real-world court coordinates. A player's position in a 2D video frame is affected by camera angle, perspective distortion, distance from the camera, and lens position. As a result, the same real-world movement can appear very different depending on where the player is on the court. Without a homography or another calibration method based on known court markings, pixel movement cannot be accurately converted into metres.

A second issue is player IDs are assigned by detection order within each frame, so player zero in one frame is simply the first bounding box the model returns, not necessarily the same person as player zero in the previous frame. When detections are reordered between frames, which happens regularly, the speed calculation measures the distance between two different players rather than the same player moving, producing readings that are incorrect.

Overall, the speed estimation feature was excluded because the current approach could not produce reliable measurements. Addressing these issues properly would require persistent player tracking with stable IDs across frames and camera calibration using known court markings as reference points, both of which represent significant additional work and are noted as future extensions.

Figure 2: Shows testing of player speed estimation

Double Dribble Detection

A key job of a referee is to call violations such as travels and double dribbles. In basketball, a player is allowed one series of consecutive dribbles each time they are in possession of the ball. If the player gathers the ball after dribbling and dribbles again, they have committed a double dribble, and the other team gets the ball.

The first stage of our detection was determining which player was in possession of the basketball at any given frame, since violation logic should only apply to that player. By identifying the player in possession first, we ensure our detection logic solely out on the relevant player, eliminating unnecessary work and improving performance.

Our work to determine possession started with our custom YOLOv8 object detection model. We searched HuggingFace and Roboflow for labelled basketball datasets and trained the model on an 11,000 image dataset that places bounding boxes around players, the basketball and the rim. The size of the dataset meant we had to train on the university's high performance computing cluster, Hydra. After training, we tested the model on basketball footage by splitting the video into frames, running each frame through the model and stitching the annotated frames back into a video. The model performed well at detecting players, the rim and the ball.

We then considered several approaches to determine which player had possession. One approach was to find the player whose bounding box fully contained the ball. This didn't work well since bounding boxes overlap when players are close together. Another was to train a dedicated possession model, but no suitable labelled dataset existed and creating one ourselves was not feasible. The approach we settled on was to use the distance between each player and the ball, finding the closest one by comparing bounding box centres each frame. This performed well in testing and is annotated in the output video with a distinct bounding box, as shown in Figure 3.

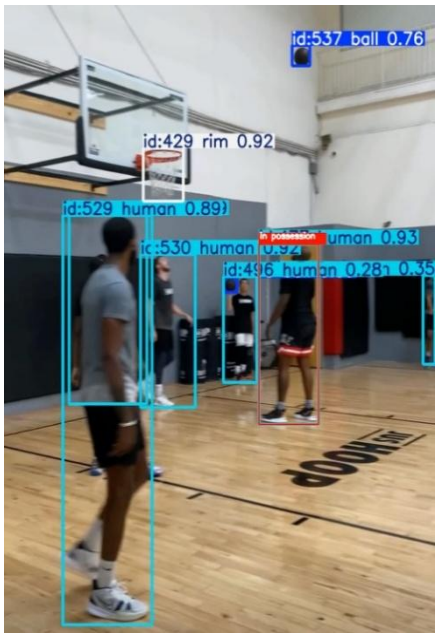


Figure 3:
Player in possession highlighted with a distinct bounding box.

The approach did have edge cases. In amateur pickup sessions, multiple basketballs are often visible in the frame as players on the sideline hold their own balls. We resolved this by using the confidence score of each ball detection. The game ball is generally larger and more prominent, so it receives a higher confidence score, and we always use the highest scoring ball. For shooting, the ball moves away from the shooter and towards other players while in flight, which would incorrectly reassign possession mid-play. To fix this, we added a proximity threshold for how close a player must be to the ball to be classed as in possession. Rather than a fixed pixel distance, the threshold scales with the size of the bounding boxes in the frame, making the system work consistently across videos of different resolutions and camera distances.

Double dribble detection relies on tracking the wrist keypoints of the player in possession. A dribble is characterised by the wrist moving downward toward the ball repeatedly, and a double dribble occurs when that motion stops and then resumes. It was clear early on that object detection alone was not granular enough for this. This kind of temporal reasoning over joint positions requires pose data and cannot be inferred from bounding boxes alone.

Pose Estimation

YOLOv8 supports pose estimation using the COCO 17-keypoint convention, outputting body keypoints including wrists, elbows, shoulders, hips, knees and ankles.

With keypoints and possession tracking in place, we had to decide how to use them to detect the double dribble. A rule-based approach was considered, for example checking whether the wrist keypoints were close enough to detect gathering, or whether the wrist oscillated up and down to detect dribbling. We rejected this approach because modern basketball players gather the ball in an almost infinite number of ways: with two hands, one hand, above the head, against the body, and more. Capturing all of this with fixed rules would be incredibly difficult. Instead, we investigated using machine learning for this task.

Double Dribble Neural Network

To train an action classification model we needed a labelled dataset with actions like dribbling and gathering. After searching, we found SpaceJam: a Dataset for Basketball Action Recognition, which consists of around 37,000 short clips of Olympic basketball footage, each labelled with one of 10 action classes: block, pass, run, dribble, shoot, ball in hand, defence, pick, no action, walk and discard.

Each clip was 16 frames long, with 17 keypoints generated per player per frame. Each keypoint has an x and y coordinate, giving 34 values per frame. We structured the input to the network as a 16 x 34 two-dimensional array, where each row represents the keypoints at that frame, maintaining temporal order. After filtering out clips labelled discard and those where keypoints couldn’t be generated for all 16 frames, the dataset reduced by 57% to around 16,000 clips.

We considered using transformers, RNNs and LSTMs for the architecture. We chose an LSTM because the problem is inherently sequential. Detecting a double dribble requires understanding the player's movement across time, not just at a single frame. LSTMs are well suited to this since they retain memory of earlier frames when processing later ones. We felt a transformer would also be difficult to train well given our dataset size. We used the Python Keras library to build the network. The input layer had the shape 16 x 34, followed by two LSTM layers to capture temporal patterns at different granularities. These feed into a fully connected layer of 64 neurons, which forces the model to retain only the most important information, before the softmax output layer assigns a probability to each of the 10 classes.

We only achieved 39% accuracy with our initial model (Figure 4). Analysing the confusion matrix made the problem clear. The walk class made up 35% of the filtered dataset, nearly double the next most frequent class, and the model was heavily biased towards predicting it. To address this, we dropped 7 of the 10 classes, keeping only dribble, ball in hand and walk, since these were the only ones relevant to detecting a double dribble. We retrained with two variants: one for 50 epochs and one for 200 epochs. The 50 epoch version performed best, achieving 61% accuracy. With random selection giving 33% on a 3-class problem, the model was beginning to learn meaningful patterns.

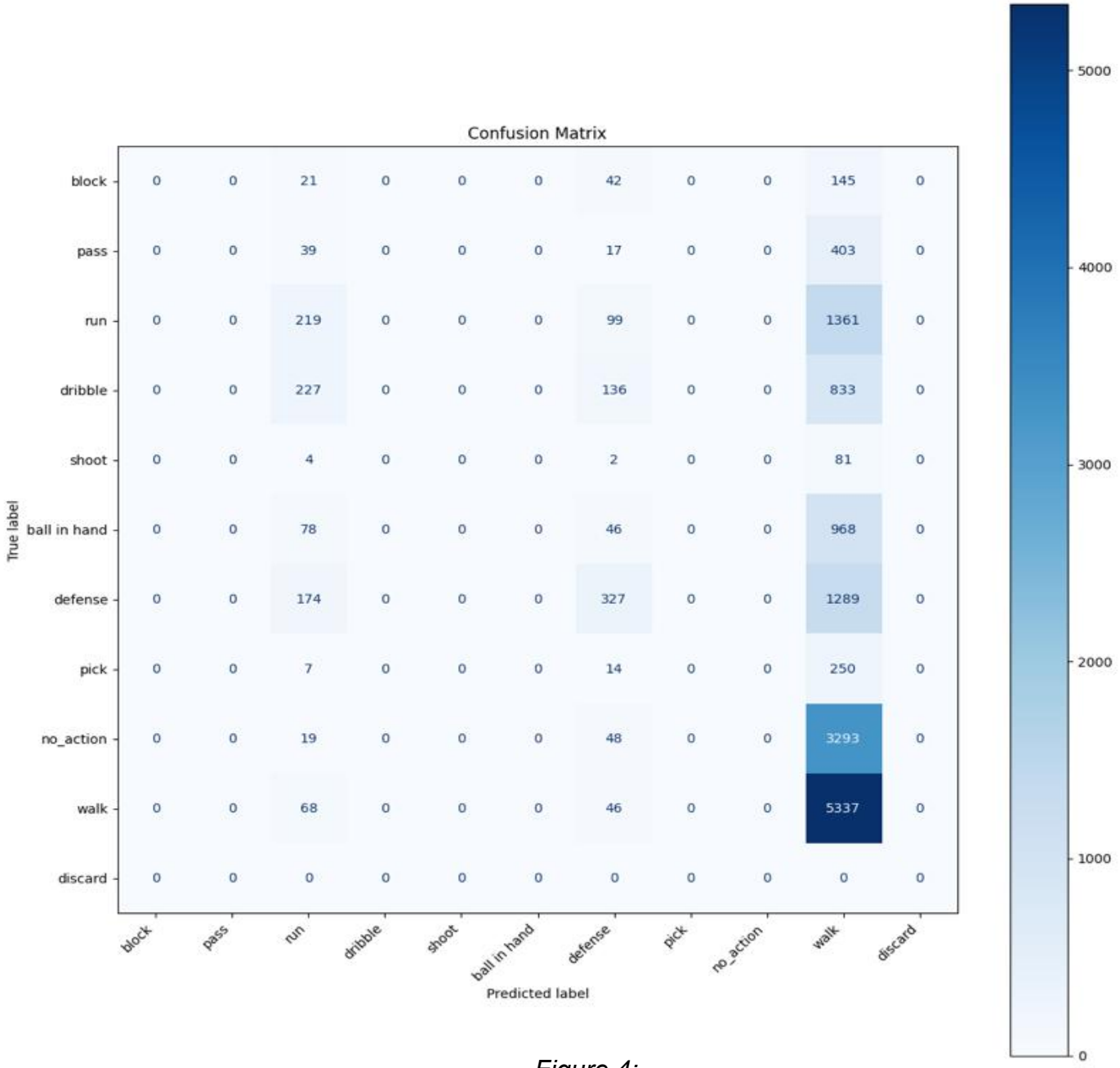


Figure 4:
Confusion matrix for the initial 10-class model (39% accuracy). The walk class dominated predictions

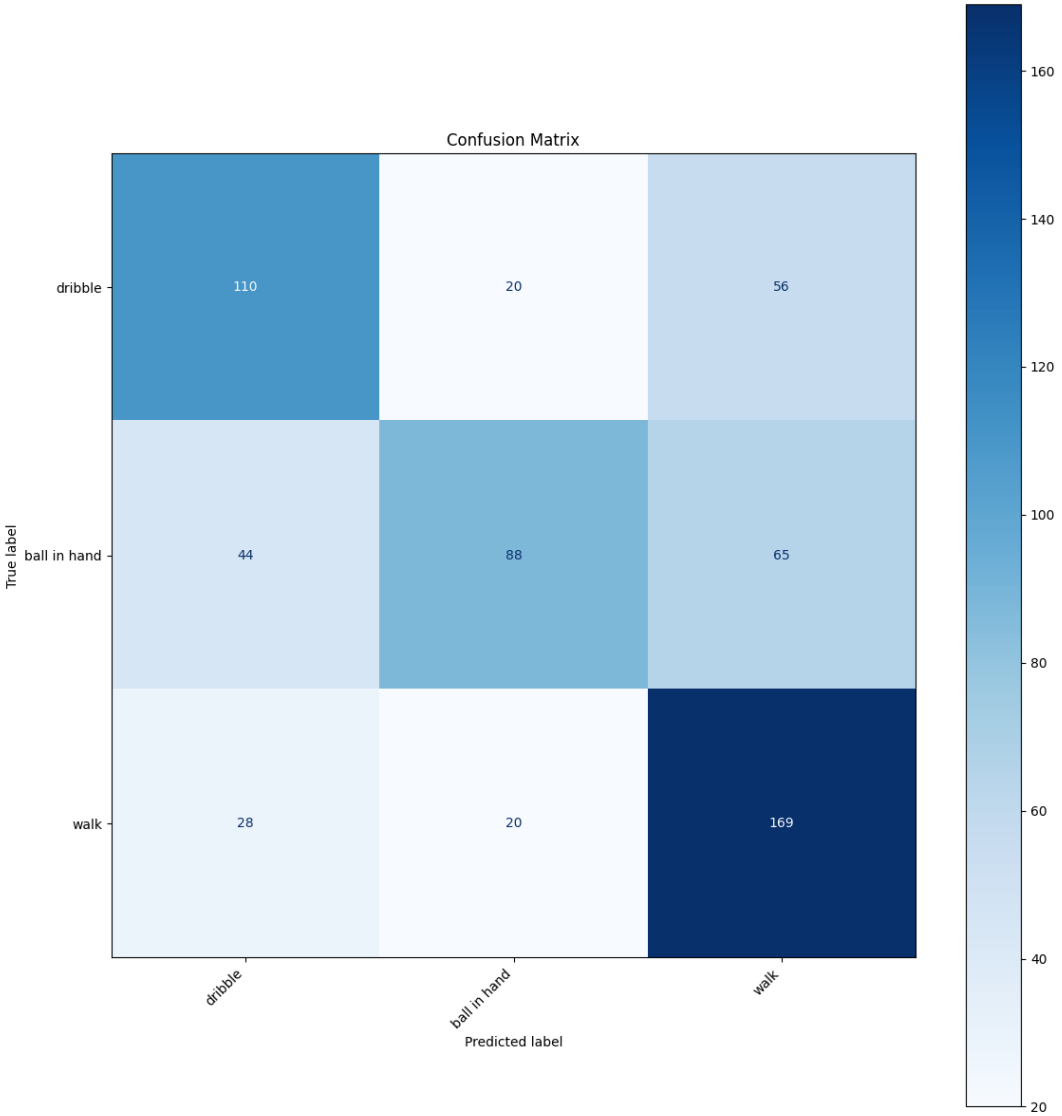


Figure 5:
Confusion matrix after reducing to 3 classes (61% accuracy).

The model was still experiencing significant confusion between walk and ball in hand, which was unsurprising given how similar those actions look. In a bid to improve accuracy, we applied z-score normalisation to the keypoint coordinates. This centred the coordinates around a mean of zero, removing any bias from absolute position values and ensuring the model focused on the relative movement of keypoints rather than where on screen they appeared. This single change improved accuracy to 75%, a 14% increase.

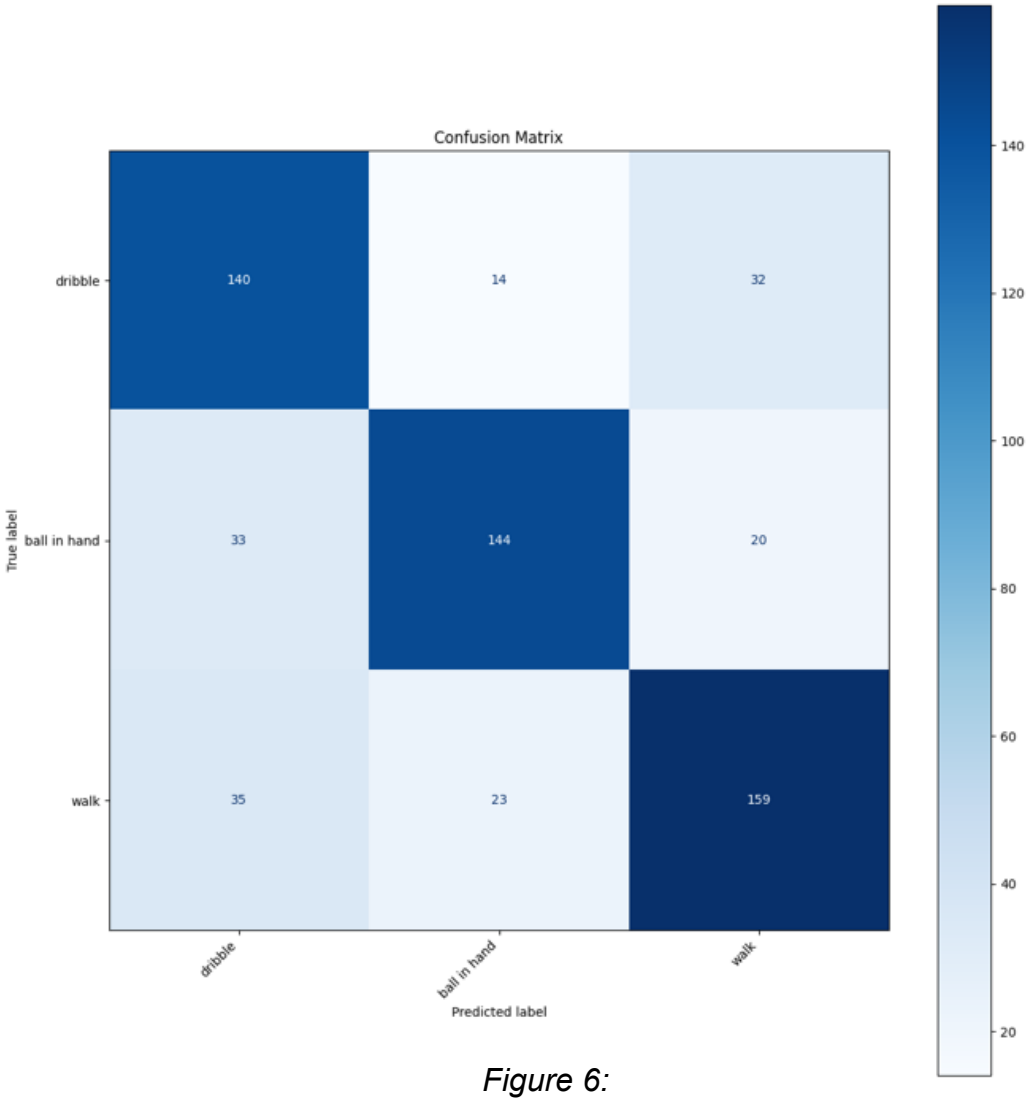


Figure 6:
Confusion matrix after z-score normalisation (75% accuracy, 3 classes)

We continued trying to improve the model. Increasing dropout in the LSTM layers slightly and adjusting patience improved performance on the dribble class specifically, though overall accuracy did not increase significantly. We also halved the learning rate to 0.0005 to slow down training and find better trends in the data. This did not significantly improve overall accuracy either but produced a more balanced F1 score across all three classes (0.741, 0.759, 0.745), meaning the model was no longer overly favouring one class.

After further hyperparameter tuning it became clear we had reached a ceiling with this architecture and class set. The model still wasn't reliably detecting double dribbles on real footage. We investigated simplifying to binary classification, since detecting a double dribble only requires knowing whether the player has the ball in their hands or is moving. We explored two approaches: merging dribble and walk into a single moving class or dropping walk entirely and keeping dribble. Dropping walk gave better results. Since the model only ever runs on the player in possession of the ball, walk is less relevant in practice anyway. The 2-class model achieved 87% accuracy.

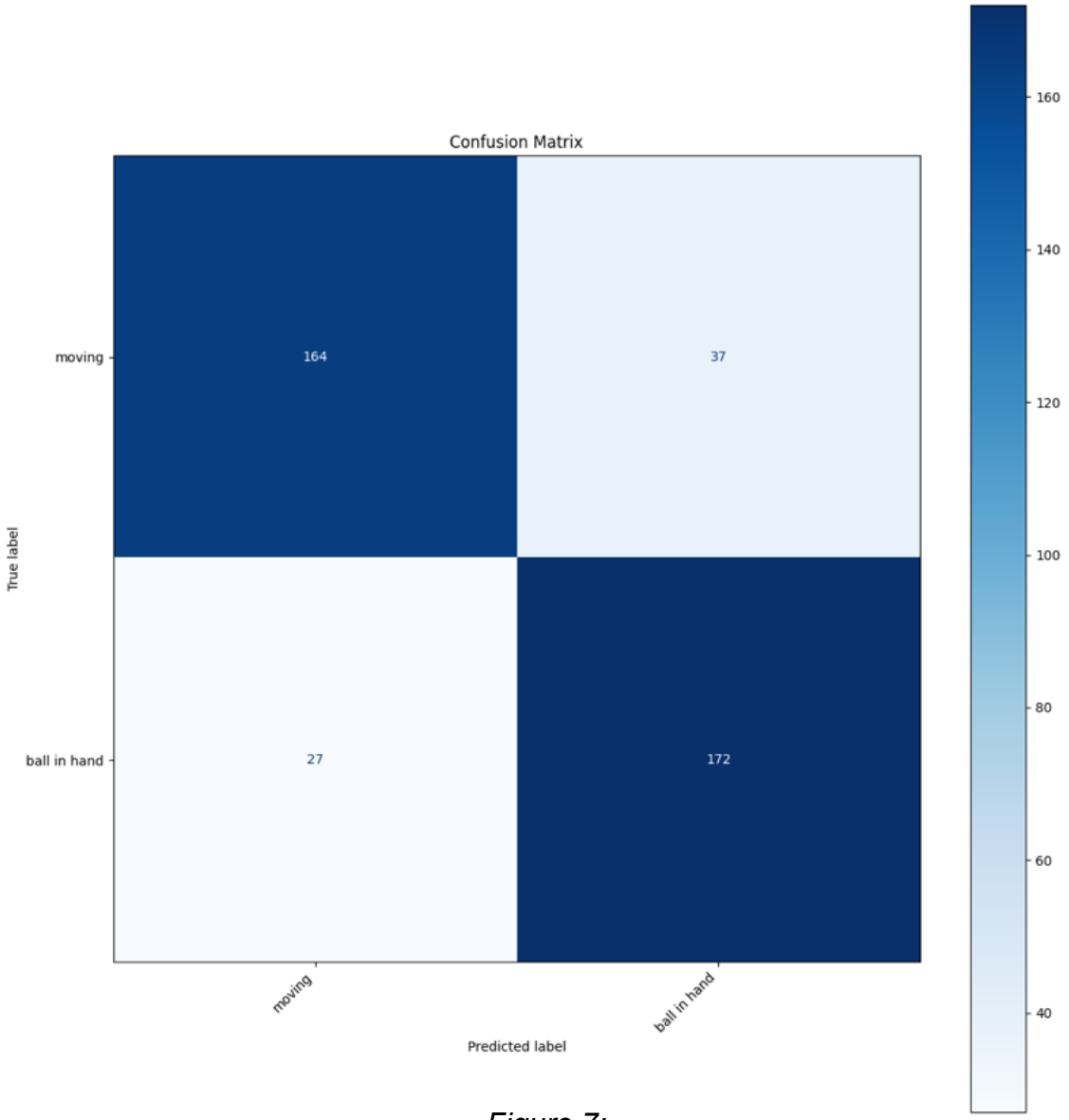


Figure 7:
Confusion matrix for the merge approach, combining dribble and walk into a single moving class.

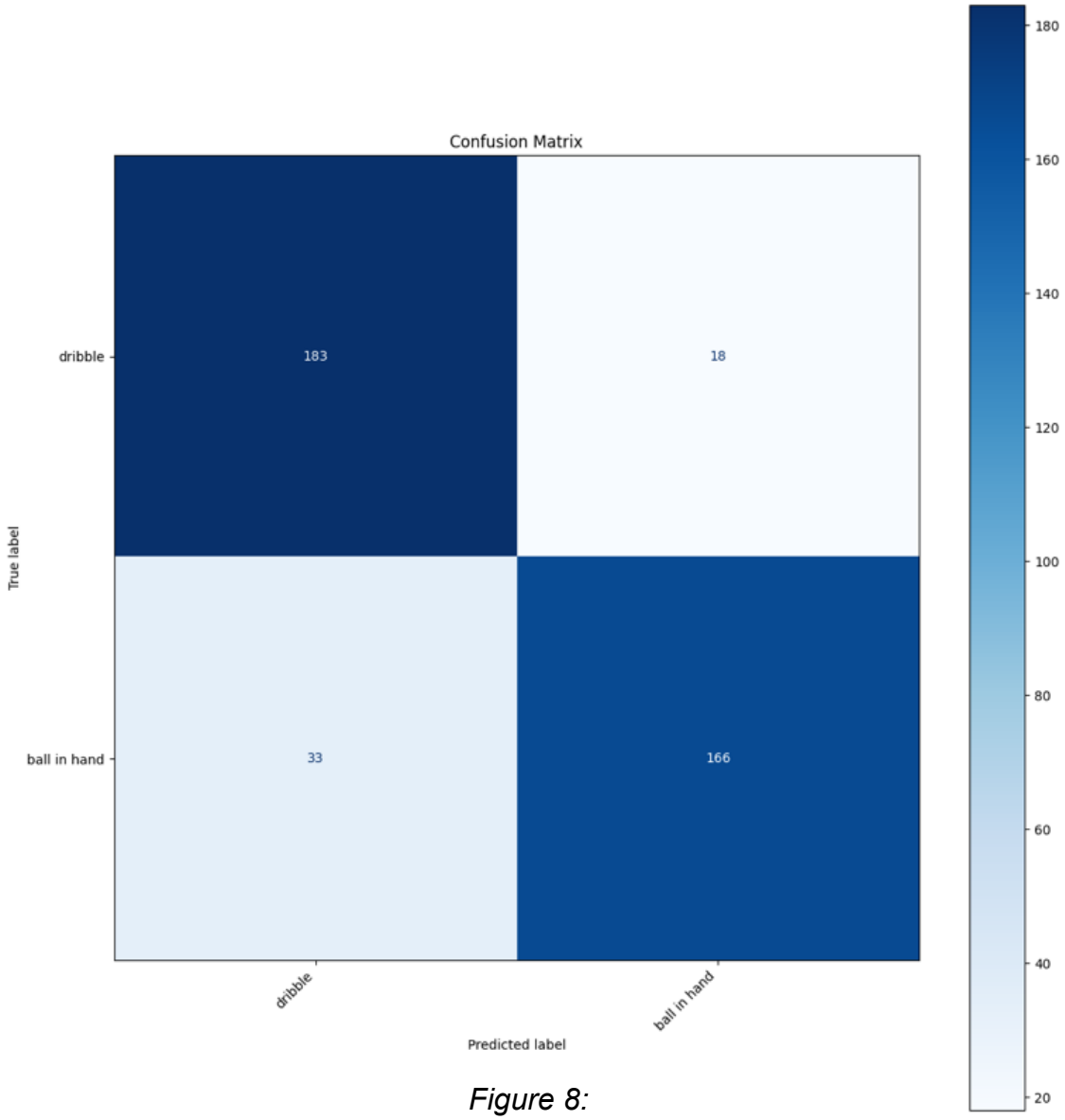


Figure 8:
Confusion matrix for the final 2-class model, dropping walk entirely (87% accuracy)

Integrating the Neural Network

Now that the model was performing well, we integrated it into the pipeline. This is how a double dribble is detected:

- The user uploads a video
- The video is split into frames, and we iterate over each frame
- At each frame, the player in possession of the ball is identified
- Their keypoint coordinates are added to a rolling buffer
- Once the buffer holds 16 frames worth of keypoints, the data is passed to the neural network to predict whether the player is dribbling or has the ball in hand
- The predicted action is stored in an actions buffer
- If the sequence dribble, ball in hand, dribble appears in succession, a double dribble has been committed
- A double dribble label is displayed on the frame and persists for the next 30 frames, so it does not disappear instantly

This workflow yielded great results. We tested it on footage of ourselves committing double dribbles and they were flagged correctly, as shown in Figure 7. Our decision to use a neural network rather than fixed rules had paid off.

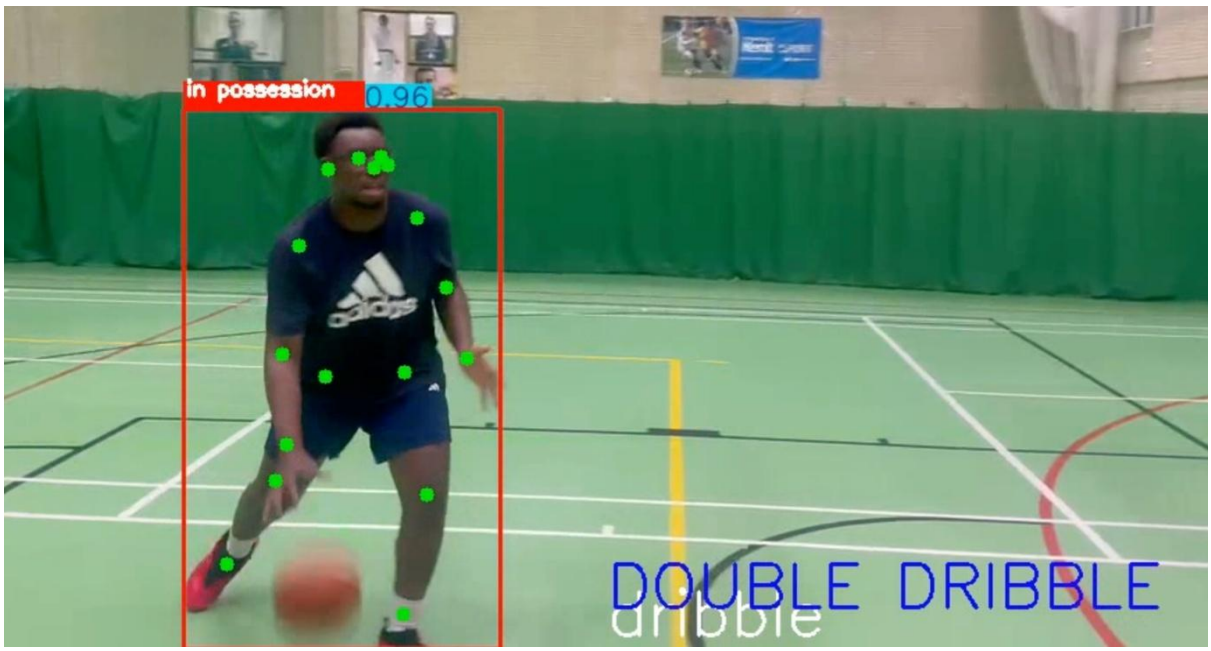


Figure 9:
Final system output showing a detected double dribble with pose keypoints and action label

Shot Analysis and Team Tracking

Five-Class Detection Model

As discussed in the Detected Classes section, we expanded the model to five classes, Ball-in-Basket and Player Shooting, to support shot analysis and team scoring. This section covers how we trained it.

We sourced and merged five basketball datasets from Roboflow, giving us 39,485 labelled images. We used YOLO26l as the backbone and trained on Hydra at 1280px resolution.

Training wasn't smooth. Automatic mixed precision caused NaN losses from epoch 29 onwards, visible in Figure 9. We disabled it and resumed from the last clean checkpoint. The final model achieved mAP50 0.827 overall. Basket (0.992), Player (0.928) and Ball (0.855) performed well. BIB (0.340) and PS (0.179) were weaker, as those moments are rare in training footage, so the model sees far fewer examples of them.

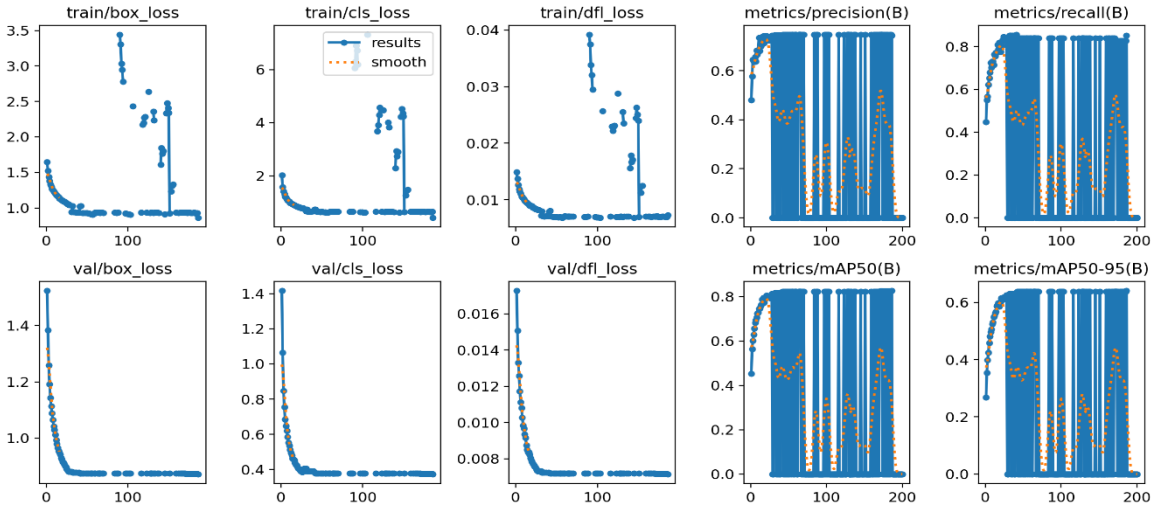


Figure 10

Training loss and metric curves over 200 epochs. NaN spikes from epoch 29 are caused by AMP instability.

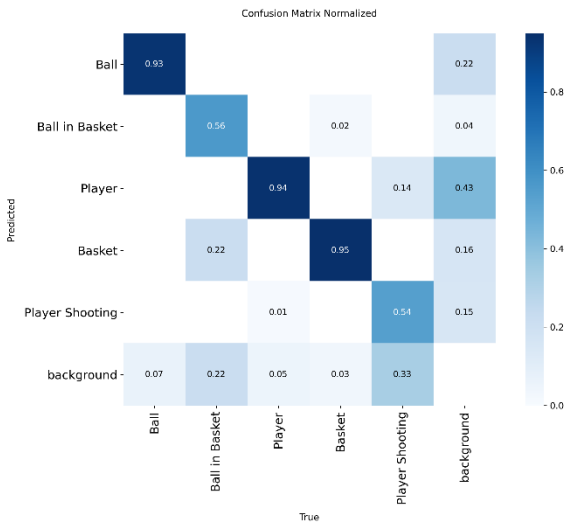


Figure 11

Normalised confusion matrix for the five-class model. Ball, Player and Basket perform well. BIB and Player Shooting are weaker due to class imbalance.

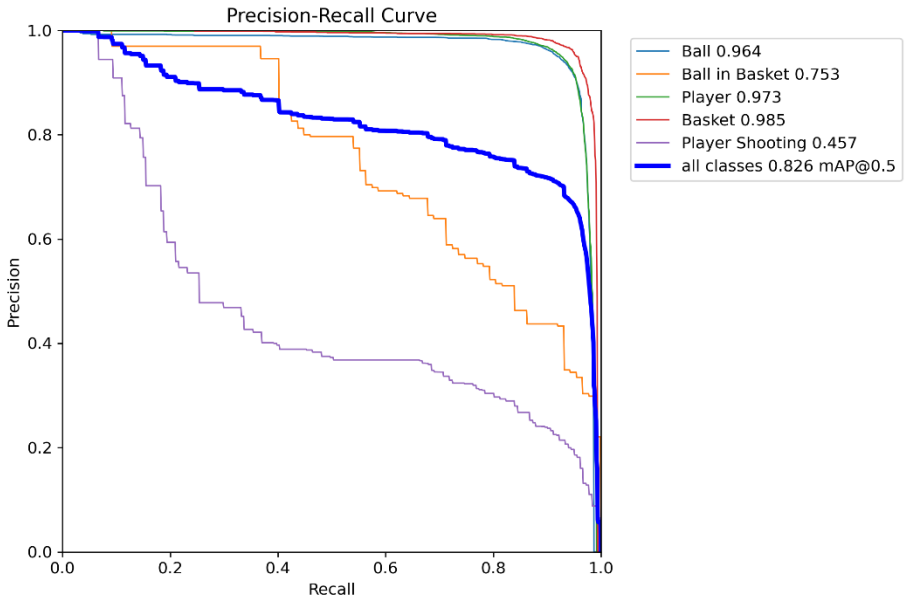


Figure 12

Precision-Recall curves per class. Player Shooting and Ball-in-Basket are the weakest classes

Figure 13

Sample validation predictions showing all five classes detected

We trained a second model with the AMP issue fixed and improved augmentation settings. It scored mAP50 0.892, higher overall, but detected zero BIB and PS events on real pickup footage at any confidence threshold. It had overfit to NBA broadcast angles and didn't generalise to amateur footage. We went back to the first model. A higher validation score does not always mean a better model in practice.

Shot feedback analysis

This feature lets a player upload a video of themselves shooting and receive detailed feedback on their form, going beyond just counting makes to analyse elbow position, release consistency and knee bend.

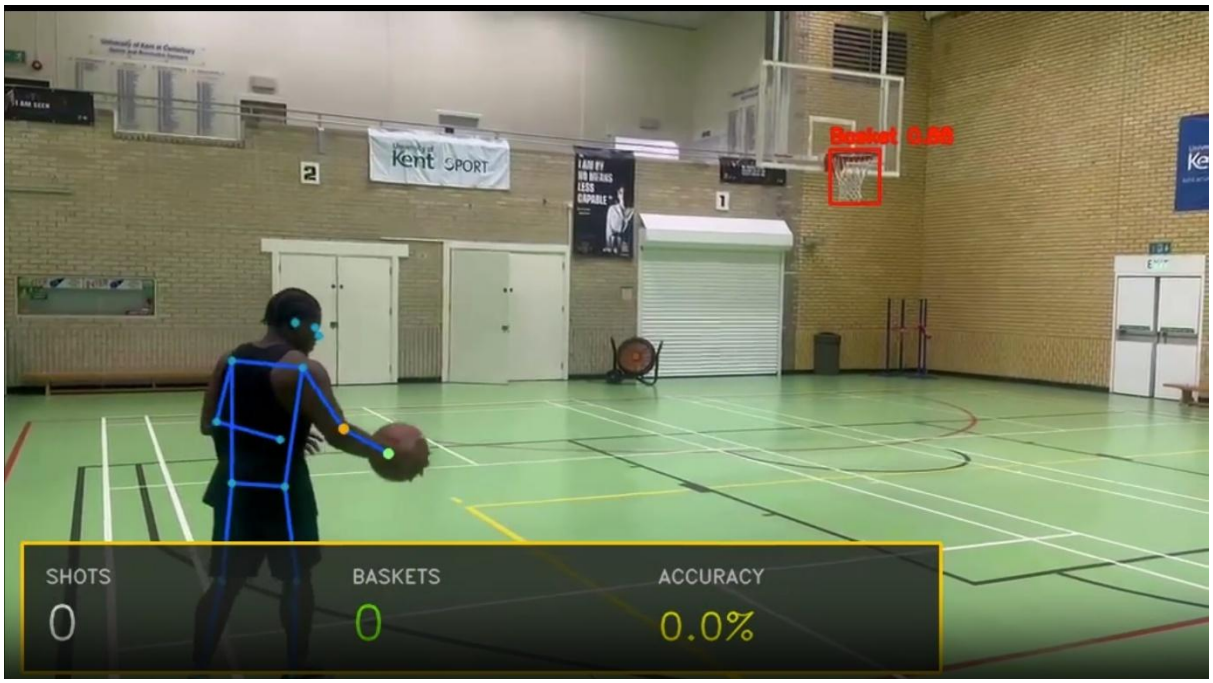


Figure 14
Shot feedback output showing the pose skeleton on the shooter, basket detection and a HUD

The pipeline runs in three passes. The first pass runs BoT-SORT tracking across all frames. Player Shooting detections register a shot attempt, with a 1.5 second cooldown per player so a single jump doesn't count multiple times.

Detecting makes was trickier. BIB only fires at around 56% recall and on amateur footage it can stop firing entirely. We added a geometric fallback, if the ball's centre passes through the padded area around the basket bounding box, we count it as a make. Because the basket class reaches mAP50 0.992, this fallback is reliable even when BIB isn't.

We compute 11 biomechanical metrics per shot. Release angle is measured by tracking the ball's path and calculating the angle at which it leaves the player's hands, rather than relying on body keypoints. Release timing combines two signals, when the ball visibly separates from the player's hands and when the wrist reaches its highest point, averaged where the two agree. The remaining metrics, elbow angle, release height, forearm angle, knee bend, follow-through, lateral drift, landing displacement, head stability and pocket consistency, are calculated from joint

positions at key moments in the shot. Pocket consistency tracks how consistently the player holds the ball in the same position before each shot across a session.
The final pass renders the annotated video and produces per-metric breakdown images for each shot.

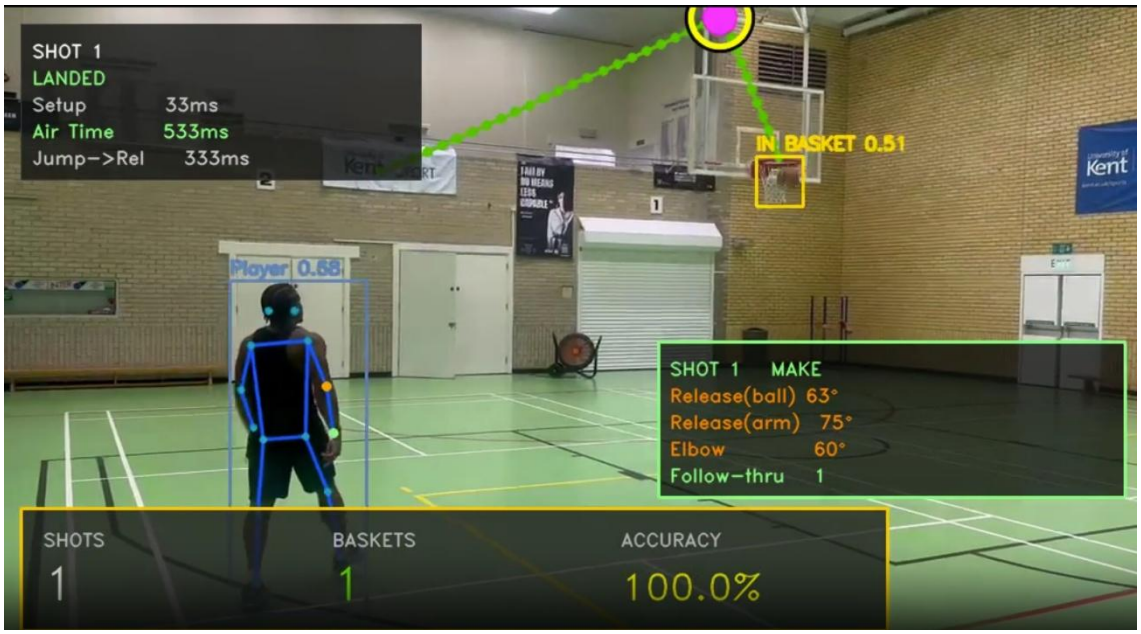


Figure 14

Figure 15

Annotated output showing a made basket with timing data, pose skeleton, make celebration overlay and metric panel

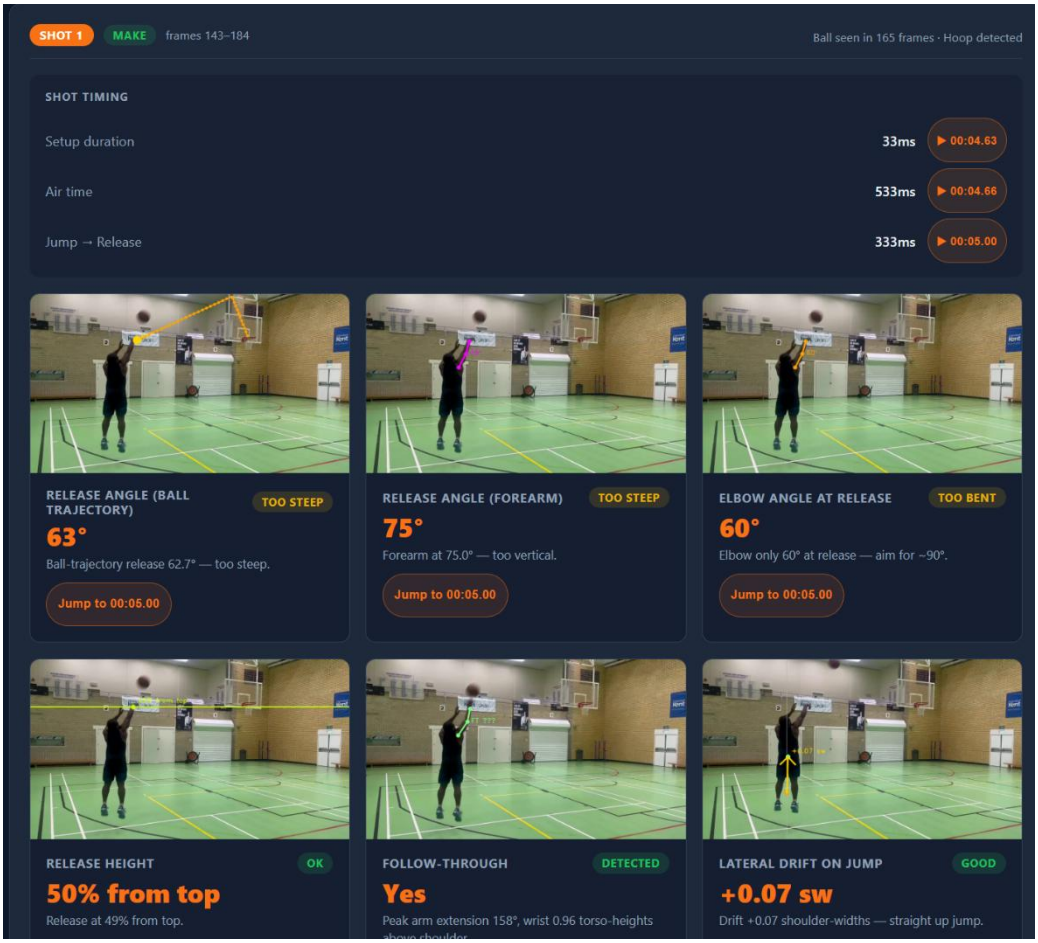


Figure 16

Shot Feedback results page showing all metric cards with per-metric breakdown thumbnails

Team Classification and Scoring

The first problem was team identity. In professional basketball, teams are separated by jersey colour or number. Neither works in pickup basketball where players wear their own clothing and number recognition on consumer video is unreliable. Rather than attempting full automation, the system asks for a small amount of user input. After the first processing pass, the user is shown a freeze frame and clicks to assign each visible player to Team A or Team B. The system handles everything else from there.

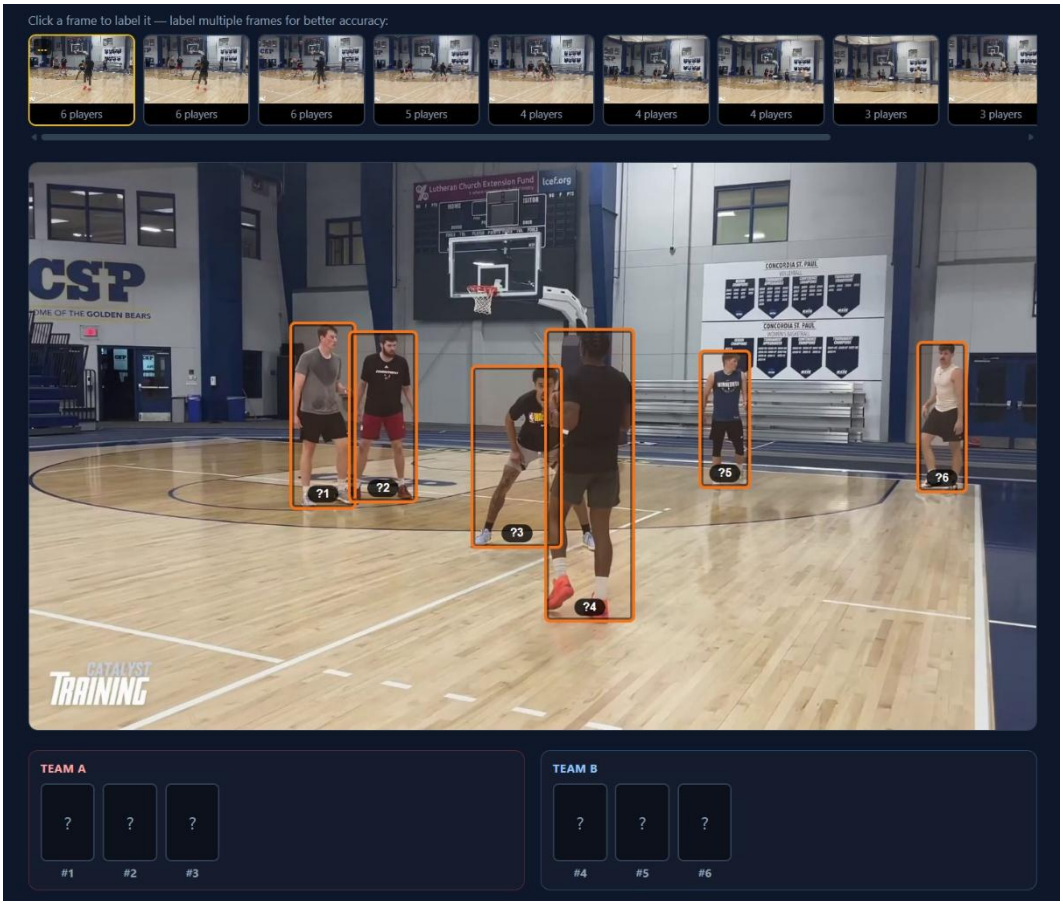


Figure 17
Team setup page where the user assigns visible players to Team A or Team B.

The first pass uses BoT-SORT to track all players and build an appearance profile for each one. For re-identification we used OSNet, a network pretrained on pedestrian datasets that produces compact appearance embeddings, generalising better across poses and clothing than colour histograms.
Rather than averaging all crops into one embedding, which loses pose variation, we pick 8 representative crops per player using a greedy diversity algorithm. Matching takes the best score across all 8, which proved significantly more reliable than a single mean embedding for players not in matching kit.

Team assignment runs in priority order: seeded players are locked to their team, remaining players are matched by appearance to the seeded ones, and if appearance is too ambiguous, we assign the nearest known player's team spatially. BoT-SORT occasionally loses and reacquires tracks during occlusions, giving the same player two IDs. We merge these tracklets and apply majority voting to correct frame-by-frame identity errors.

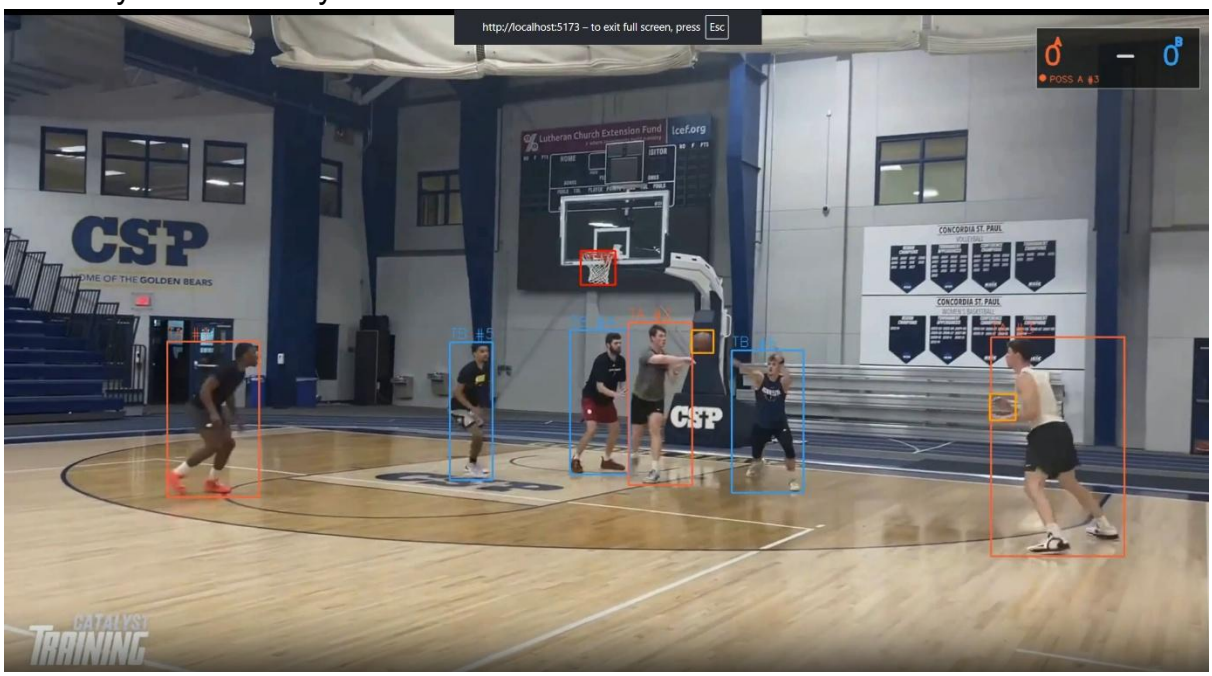


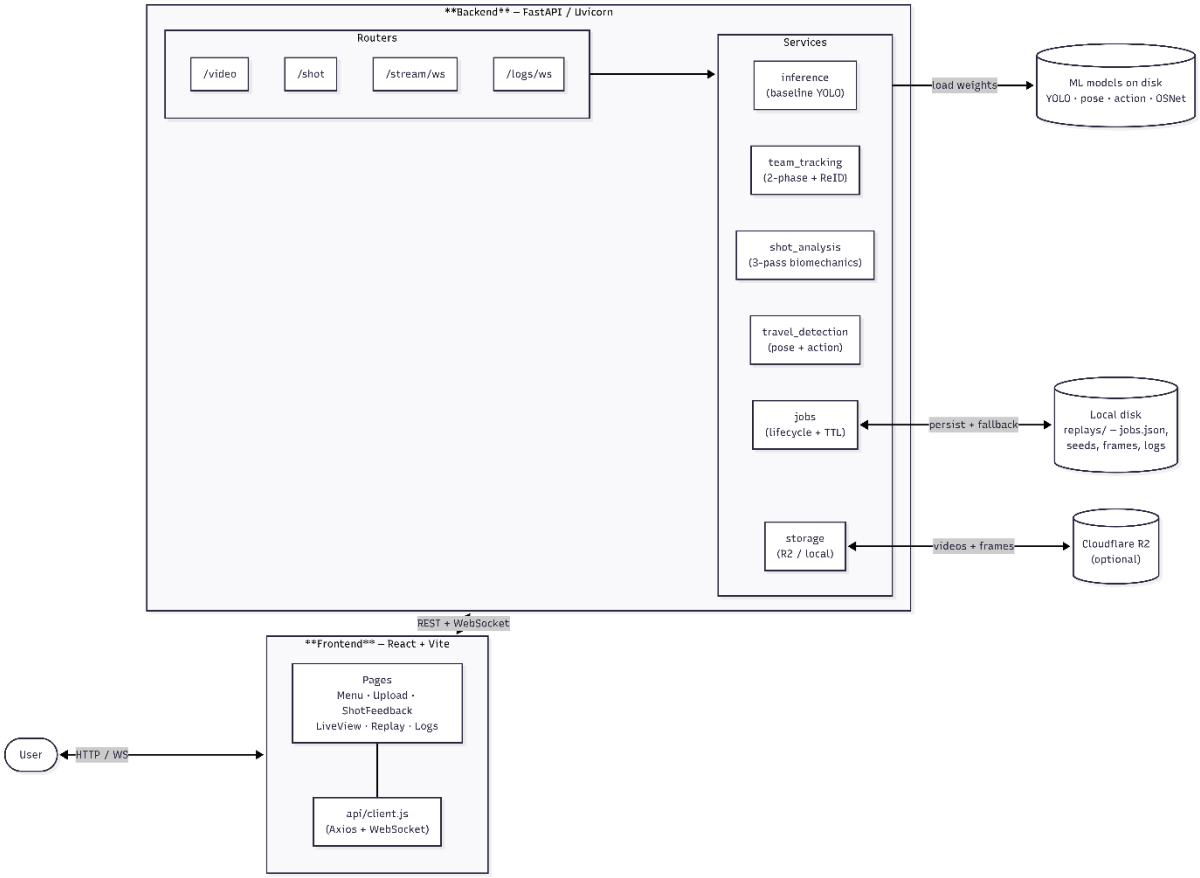
Figure 18
Annotated output with team-coloured bounding boxes and live scoreboard HUD

For scoring, the same BIB-plus-geometric-fallback approach detects makes. The scoreboard HUD also tracks and displays which team currently has possession of the ball. Currently the system does not distinguish between two and three point baskets, so all made shots are counted as two points. This would need to be verified manually. In future, a court detection model could identify the three-point line automatically and handle this distinction.

Figure 19

Team tracking results page showing the scoreboard, Game Timeline, per-player

System Architecture Diagram



Web Application and Frontend

The web application is built with React and Vite across four primary pages: Upload Video, Shot Feedback, Replay, and Logs, all sharing a dark navy and orange colour scheme via CSS custom properties. The Navbar renders persistently across all pages, displaying the logo as a home link and collapsing to an animated hamburger drawer on screens narrower than 640 pixels.

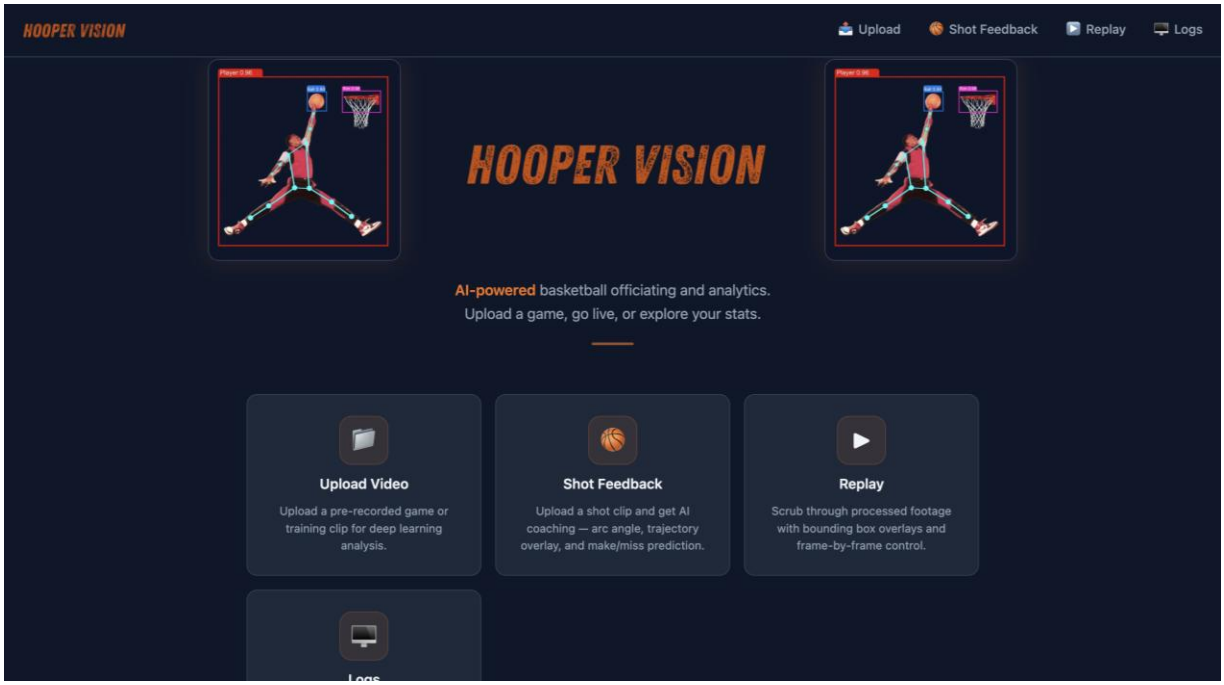


Figure 20
Main Home Page of Hooper Vision on a Desktop

Upload Page and Settings

The Upload page supports file upload and camera recording modes. Camera mode uses getUserMedia to display a live preview with a pulsing REC badge, falling back to a native file input on mobile. Before submission, users configure analysis via the VideoSettings component across four groups: Analysis Mode (Team Mode with game type selector and jersey colour auto-detect, plus Double Dribble Detection), Overlays (Bounding Boxes, Skeleton, Confidence Labels), Tracking (Player ID, Ball Trail), and Performance (Full Accuracy Mode with an info tooltip). All settings are serialised to JSON and sent with the upload form data for the backend to apply.

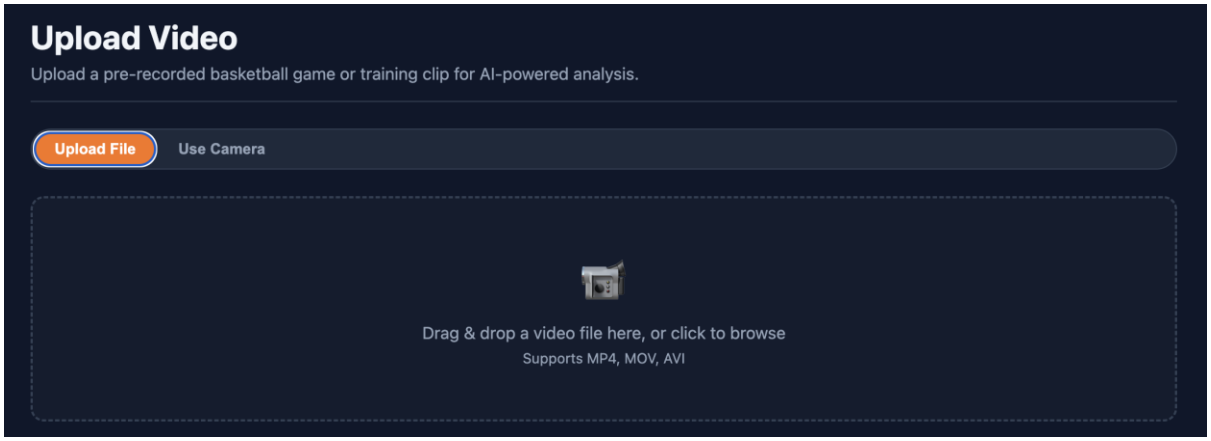


Figure 21

The Upload Video page with drag and drop file input and a camera recording mode toggle.

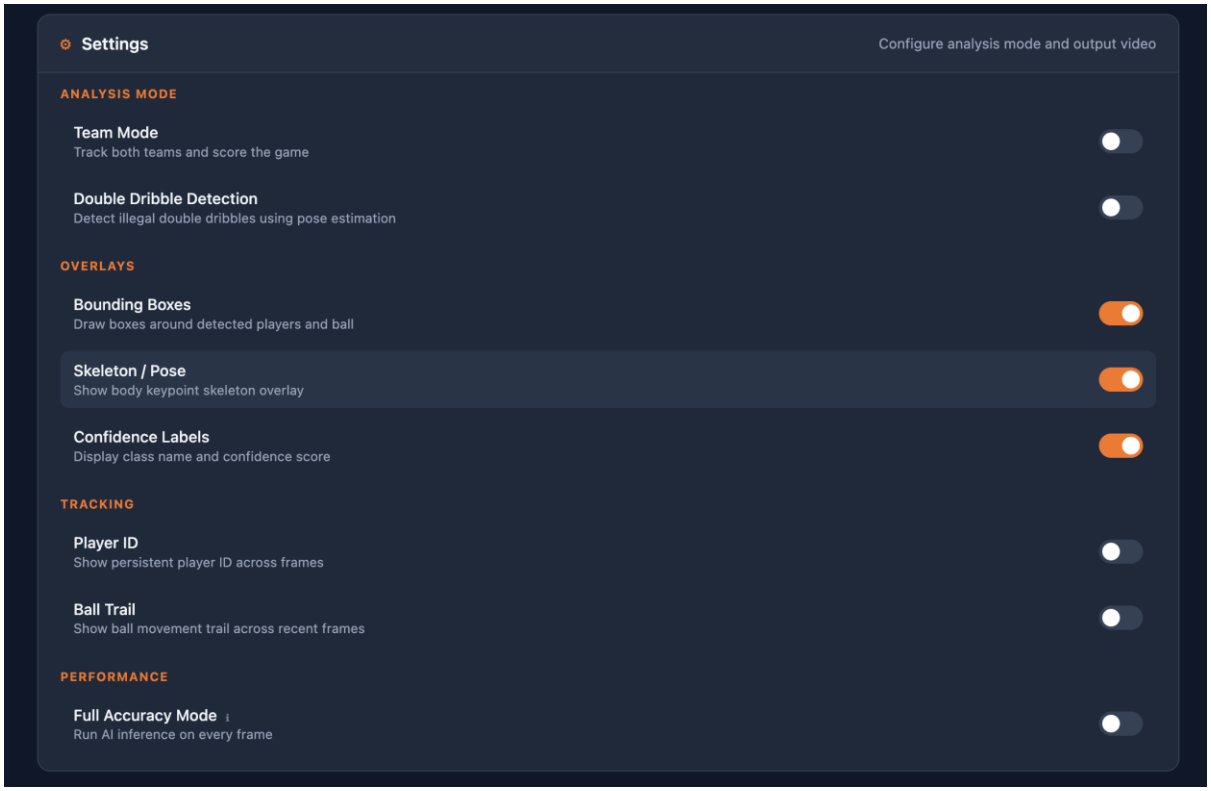


Figure 22

Figure 22

The Settings panel showing overlay, tracking, analysis mode, and performance configuration

When Team Mode is enabled, processing pauses at a seed frame and the TeamSetup component renders. Players are assigned to teams by tapping interactive bounding box overlays on the frame image, with a dropdown list fallback for accessibility. On completion the results include a scoreboard, a per-player stats table with tappable avatar thumbnails that open in a lightbox, a shot log of every detected attempt, and a GameTimeline. This is a two-lane interactive chart plotting each team's shots against the game duration. Dots are filled for makes and outlined

for misses; clicking a dot seeks the video to that shot, and hovering shows a tooltip with team, player, timestamp, outcome, and running score.

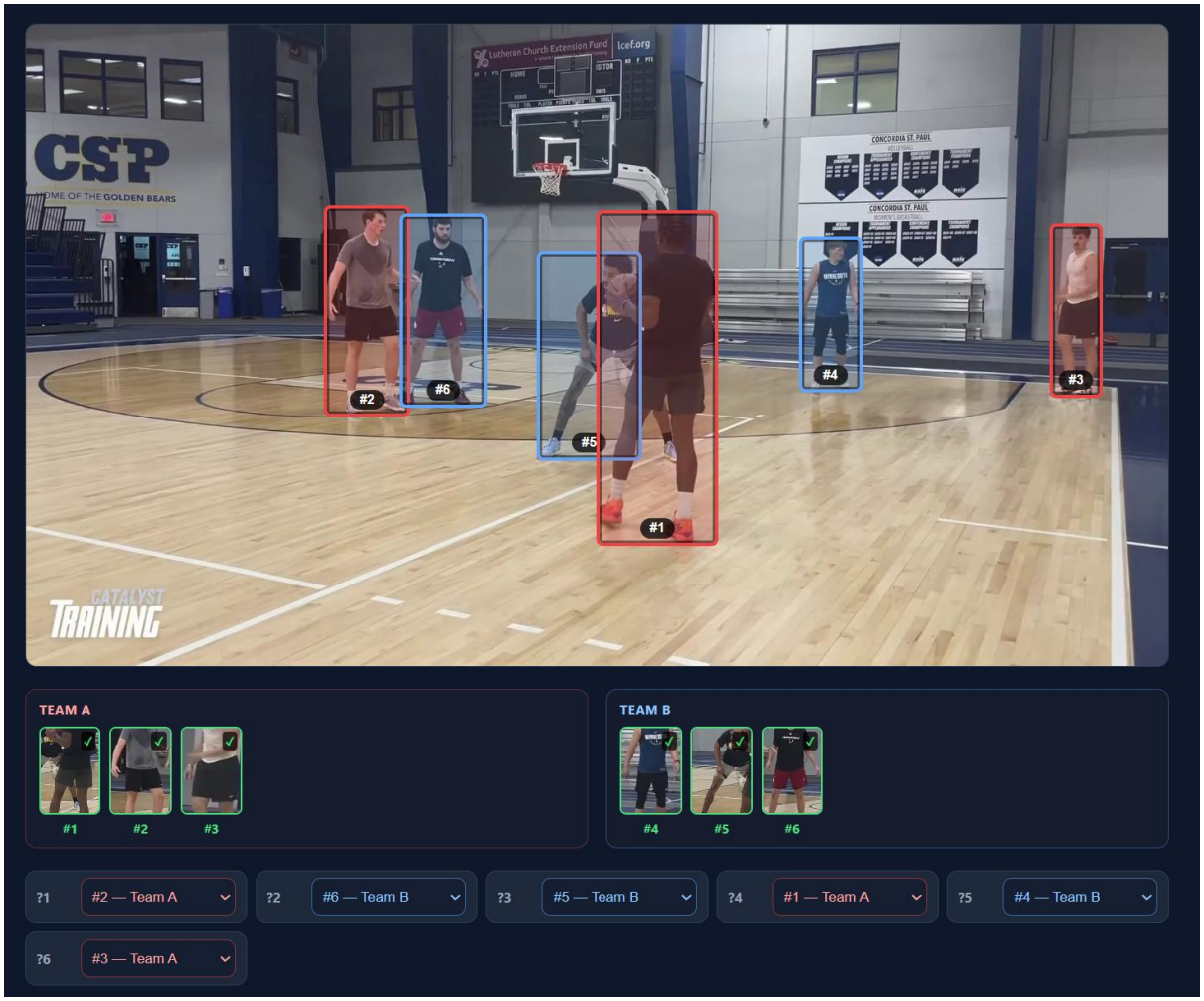


Figure 23
The TeamSetup component showing the seed frame with colour-coded bounding boxes for each detected player, team assignment previews with avatar thumbnails, and the dropdown fallback list for explicit team assignment.

Replay Page

The Replay page fetches all completed sessions from GET /video/sessions and handles both video analysis and shot analysis session types. Client-side overrides in localStorage allow soft-delete and rename without modifying server state. Each session shows an expiry label computed from the expiresAt field. The stream endpoints return 206 Partial Content for HTTP range requests, enabling in-browser video seeking. Shot sessions load the full shot breakdown, including per-metric cards with frame thumbnails, verdict badges, and Jump buttons that seek the video

to the exact measured frame. These reuse the same components as the Shot Feedback page.

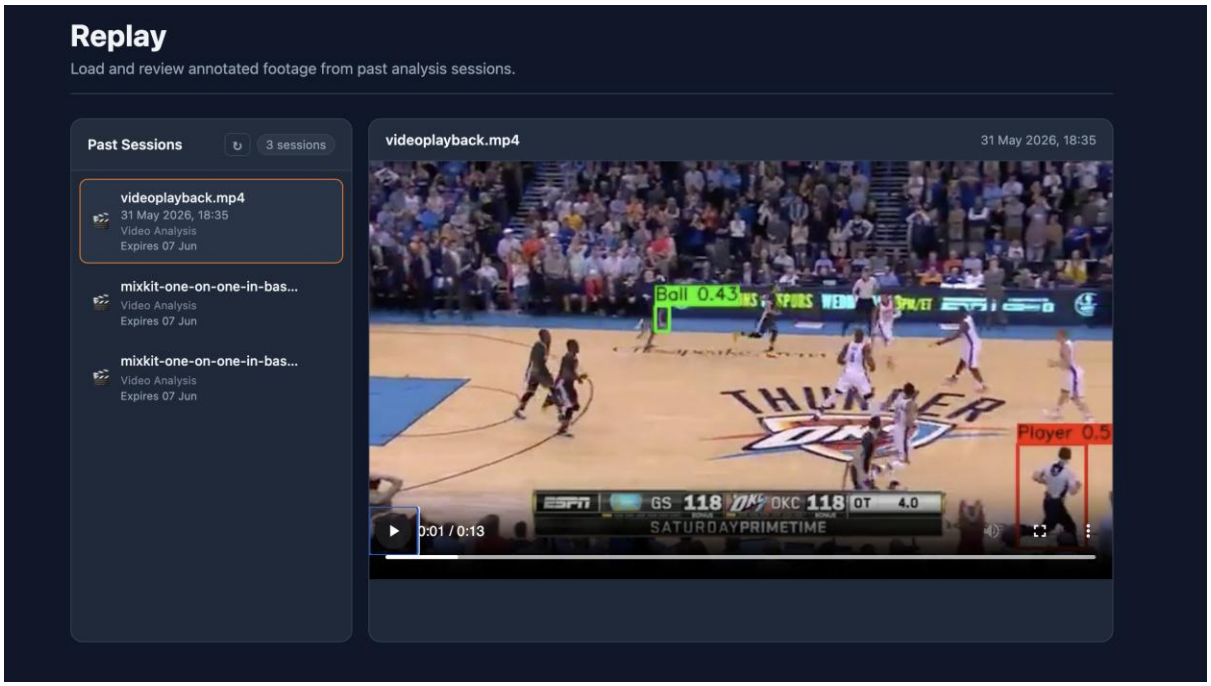


Figure 24

The Replay page showing the session sidebar with expiry labels and a selected annotated video with YOLO bounding box detections visible on the players and ball.

Cross-Platform Compatibility and Mobile Responsiveness

The OpenCV VideoWriter was originally initialised with CAP_MSMF (Windows Media Foundation), which silently produces empty output on macOS and Linux. Switching to CAP_FFMPEG with avc1 H.264 fixed this across all platforms. The same fix was applied to the travel detection service. The API client derives the backend host from window.location.hostname so requests resolve correctly from any device on the network. Vite is bound to all interfaces via host: true and Uvicorn is started with the host 0.0.0.0 flag, making the full application accessible from a phone on the local network.

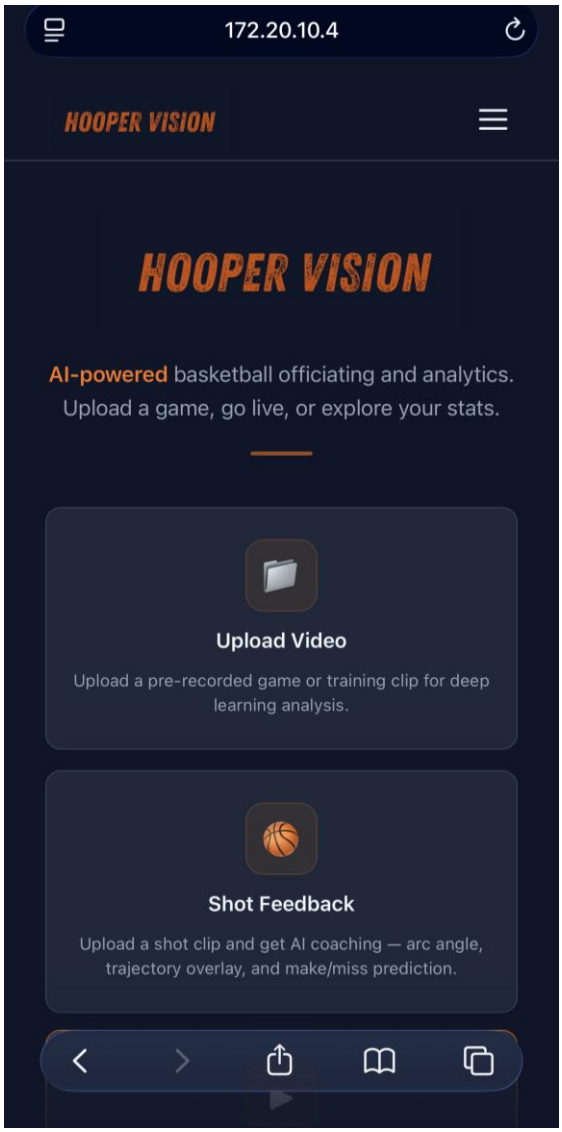


Figure 25

Figure 25

The Menu page viewed on iPhone Safari, showing the responsive single-column layout

Evaluation

Overall, the project successfully achieved most of its aims. A custom object detection model was trained to identify basketball players, the ball, and the rim across different video conditions, and this was integrated into a web application where users can upload footage and receive an annotated output video. Pose estimation was also implemented to support shot feedback. The backend was designed to manage long-running video processing tasks asynchronously. Player speed estimation was investigated as an additional analytical feature, but its practical reliability was limited by the lack of proper camera calibration.

The only aim that was not fully completed was real-time live stream. Initial code stubs were included to support this, but the feature was not implemented within the final system. This is an area for future work

Limitations

The training data used to develop the detection model consists predominantly of NBA broadcast footage. NBA players are not representative of all basketball players in terms of height, build, or playing style, and a model trained on this data may perform less reliably on youth leagues, wheelchair basketball, or other formats where the visual characteristics of the game differ. This is a recognised limitation and would need to be addressed before the system could be responsibly deployed in a broader context.

Ethics

The primary ethical consideration in this project is privacy. The system processes video footage of individuals. Users uploading footage of amateur games should ensure they have permission from all players appearing in the video before submitting it for analysis.
Uploaded videos are stored temporarily on the server during processing and retained for replay. In a production deployment this would require a clear data retention policy and compliance with data protection regulations such as the UK GDPR.

There is also a consideration around automated officiating. A system that flags violations such as double dribbles carries the risk of incorrect calls. Unlike a human referee whose decisions can be contextualised and challenged in the moment, an automated flag may carry an unwarranted sense of objectivity. Any deployment of the violation detection features would need to position them as assistive tools to be reviewed by a human official rather than as authoritative decisions.

Future Work

Future work could improve the system by extending it beyond basic object detection and adding more advanced basketball analysis features. One important area for future development is player speed estimation. An experimental version of this feature was tested, but it was not included in the final system because the results were not reliable enough. A future version would need to use court markings or another known reference point to calculate a homography between the video frame and the real basketball court. This would allow player movement to be measured more accurately.

Since we run the application locally, videos can take a while to process. In future we could run the application on a cloud platform like AWS and provision a powerful computer, to improve performance.

Live stream analysis could also be added as a future extension. A stub for live streaming functionality was originally included in the codebase, but it was not fully implemented due to time constraints. This feature would allow the system to process video directly from a camera feed or live match stream rather than only analysing uploaded video files. The main challenge would be performance, as the system would need to run detection and tracking with low latency so that the output remains close to real time. A CPU-only server would likely struggle with this, so future work may require GPU acceleration.

Overall, future work should focus on making the system more accurate, and practical for all basketball analysis. These additions would require significant extra development, but they would move the project from a proof-of-concept detection system towards a more complete basketball analytics platform.

Conclusion

Hooper Vision demonstrates that a functional basketball analysis platform can be built using open-source computer vision tools and deployed as a web application without requiring specialist hardware. The system successfully detects players, the ball, and the rim in uploaded video, produces an annotated output at the original resolution and frame rate, and provides users with a replay interface for reviewing past sessions. The custom-trained YOLOv8 model performs well on broadcast footage, and the frame subsampling approach makes processing tractable on CPU-only infrastructure at the cost of some box lag during fast action. The project is novel in combining domain-specific model training, pose estimation for violation detection, and a deployable web interface into a single pipeline accessible to non-technical

users. Most comparable academic work focuses on detection accuracy in isolation rather than building an end-to-end system that a coach or analyst could actually use. The web-based architecture also means the system is platform-independent and requires no local software installation.

Bibliography

Kaggle. Basketball Tracking Dataset. Available at:
https://www.kaggle.com/datasets/trainingdatapro/basketball-tracking-dataset

Roboflow Universe. Basketball Dataset. Available at:
https://universe.roboflow.com/cricket-qnb5l/basketball-xil7x

Roboflow Universe. Basketball Dataset. Available at:
https://universe.roboflow.com/test-datset/basketball-bs0zc

YouTube. Basketball video reference. Available at:
How to Train YOLO Object Detection Models in Google Colab (YOLO26, YOLO11, YOLOv8)

YouTube. Basketball video reference. Available at:
Human Pose Estimation Python using YOLOv8 (Revealing Lebron James 3 Pointer Secrets)

YouTube. Basketball video reference. Available at:
Building an Advanced AI Basketball Referee

Homecourt - https://www.homecourt.ai/

Ayush Pai – AI Basketball Referee - https://github.com/ayushpai/AI-Basketball-Referee/tree/main

FarzaTV – Gemini-bball - https://github.com/farzaa/gemini-bball
https://www.youtube.com/watch?v=VZgXUBi_wkM

Simone Francia – SpaceJam Basketball Actions Dataset
https://github.com/simonefrancia/SpaceJam

Appendices

Background
B1.

References

1. Fuller, R (2024). '*Wimbledon brings in electronic line calling for 2025*', BBC Available at: https://www.bbc.co.uk/sport/tennis/articles/ce3zg3y23v7o (Accessed: Friday 29 May 2026).
2. NBA (no date). '*NBA Last Two Minute Reports – Frequently Asked Question*s', NBA. Available at: https://official.nba.com/nba-last-two-minute-reports-frequently-asked-questions/ (Accessed: Friday 29 May 2026).
3. Gardner S (2024). '*Charles Barkley, Shaq weigh in on NBA refereeing controversy, 'dumb' two-minute report*', AOL. Available at: https://www.aol.com/charles-barkley-shaq-weigh-nba-161557772.html (Accessed: Friday 29 May 2026).
4. Alarcon N (2018). '*This AI App Can Help You Improve Your Jump Shot*', NVIDIA Developer. Available at: https://developer.nvidia.com/blog/this-ai-app-can-help-you-improve-your-jump-shot/ (Accessed: Friday 29 May 2026)
5. Bewley, Alex, et al (2016). "Simple online and real-time tracking." *2016 IEEE international conference on image processing (ICIP)*. Ieee.

Acknowledgements

We would like to thank our project supervisor, Dr. Giovanni Masala, for his guidance and support throughout the project.