AN INTELLIGENT MOBILE APPLICATION TO IMPROVE USERS' SHOOTING MOTION AND COMPARE THEM WITH NBA PLAYERS USING MACHINE LEARNING AND LARGE LANGUAGE MODELS

Yu Chu [1], Garret Washburn [2]

[1] Nanchang University, No. 999, Xuefu Avenue, Honggutan New District, Nanchang, Jiangxi Province, China
[2] California State Polytechnic University, Pomona, CA, 91768

ABSTRACT

Basketball is one of the most popular sports worldwide, with over 610 million people aged 6 to 54 playing the game at least twice a month, according to FIBA. However, access to systematic and professional basketball training remains limited, especially in developing countries, where only 1%–3% of players may receive professional coaching. This lack of access makes it difficult for most basketball enthusiasts to learn and refine proper shooting techniques. To address this issue, we propose Sharp Shooter—a mobile application that helps users improve their shooting form without requiring professional training or expensive equipment.

Our solution combines several cutting-edge technologies: MediaPipe is used to extract key body landmarks from users' uploaded shooting videos; these landmarks are then analyzed by a large language model (LLM) to provide expert-level feedback [10]. Additionally, the app matches users' shooting forms with those of professional NBA players stored in a custom database, allowing users to see which NBA player their form most resembles—further enhancing engagement and motivation.

The project integrates several key components, including a cross-platform front end built with Flutter, a Flask-based backend hosted on AWS, and a machine learning pipeline utilizing YOLOv5 for object detection and Random Forest or LSTM for motion quality assessment [11]. During development, we encountered challenges related to efficient video processing, backend scalability, data security, and precise motion evaluation. These were addressed through asynchronous data handling, load balancing, encrypted communications, and model optimization.

The app was tested in various real-world use cases, including indoor and outdoor shooting scenarios, different lighting conditions, and varied camera positions. It consistently delivered actionable feedback, helping users recognize flaws in their form and track improvement over time.

David C. Wyld et al. (Eds): SIGI, CSTY, AI, NMOCT, BIOS, AIMLNET, MaVaS, BINLP – 2025
pp. 225-238, 2025. CS & IT - CSCP 2025 DOI: 10.5121/csit.2025.151918

Our findings demonstrate that Sharp Shooter is a scalable, accessible, and affordable tool for basketball players at all levels. It offers a novel way for individuals—especially in under-resourced communities—to receive professional-style feedback and engage with the game in a more meaningful and data-driven way.

KEYWORDS

MediaPipe, Random Forest, Mobile Application, Shooting Motion, Large Language Model

1. INTRODUCTION

According to FIBA, over 610 million people aged 6-54 worldwide play basketball at least twice a month. However, for developed countries and some Asian countries, only about 5% -10% of the basketball population can receive more systematic training (such as school teams and club tiers) [1]. For the vast majority of developing countries, only 1%-3% of the basketball population may receive truly professional guidance (such as professional coaching, scientific training systems). (FIBA celebrates more than 610 million players globally on second edition of World Basketball Day | FIBA Basketball) In this case, we want to help more people, especially people in developing countries or those who cannot receive professional basketball education due to poverty. As we all know, without professional training, it is difficult for beginners to learn the correct shooting form. (Uzun, Ahmet & Pulur, Atilla. (2018). The effect of shooting training on the development of the shot hit rate for basketball players. Journal of Human Sciences. 15. 10.14687/jhs.v15i4.5563.) That's why our project primarily focuses on correcting shooting forms, enabling a wider range of basketball enthusiasts to correct their shooting form through our app, even if they lack access to professional basketball training. In addition, media clutch points made statistics on the number of fans of some personal social media accounts of active NBA players and published the ranking list, mainly including ins and X. The results showed that the total number of social media fans of all NBA active and retired stars may be between 1 billion and 3 billion, and the number of independent fans (after removing the duplication) is about 500 million to 1.5 billion [5]. (NBA surpasses one billion likes and followers on social media - NBA.com: NBA Communications) In this case, tens of millions of fans want to show their connections with NBA stars, and our project provides them with this opportunity. After analyzing the users' shooting form, we compare their shooting data with the shooting data of NBA stars stored in the database, thereby identifying the NBA star whose shooting form is the most similar to that of the user.

1.1. First Methodology

The first methodology attempts to send the user's shooting video to a real-world basketball coach, who will analyze the user's shooting posture. The disadvantages of this are: first, it is very expensive, and it costs $97 for a user to do a shooting analysis. Our app is completely free because it uses LLM. Second, the analysis time required for the first methodology is very long, because obviously basketball coaches are not always available, so it takes several days to conduct an analysis. However, our app does not require manual work, so it only takes a few tens of seconds to conduct a shooting analysis.

1.2. Second Methodology

The second methodology attempts to obtain the user's arm angle (between the upper arm and the forearm) when shooting through the shooting videos submitted by the user, and then give different correction suggestions to the user based on different arm angles. This is a good solution, but its disadvantages are: first, the arm angle when shooting cannot fully reflect the problem of a person's shooting posture. It should also consider the angle of shoulder rotation, whether the wrist is fully compressed, whether the hip is fully flexed, whether the knee is fully flexed, etc. (Okazaki, V. H. A., Rodacki, A. L. F., & Satern, M. N. (2015). A review on the basketball jump shot. Sports Biomechanics, 14(2), 190–205. https://doi.org/10.1080/14763141.2015.1052541) [6]. However, our project uses mediapipe to comprehensively analyze 33 key points of the human body. This makes our shooting posture analysis more accurate and can also provide users with more targeted suggestions.

1.3. Third Methodology

The third methodology obtains the angle changes of the user's elbow, shoulder, and knee joints when shooting through the videos submitted by the user, and makes them into charts to help users correct their shooting posture. Compared with the second methodology, this method focuses on more points, but still ignores some key points such as the wrist. At the same time, this methodology only provides users with charts but does not give any suggestions. However, our APP can give our users extremely accurate and effective shooting posture correction suggestions through the random forest algorithm plus LLM.

Our project - Sharp Shooter can solve this problem very well. Users can download our App through Apple Store or Google Play. Once they download our app, they can send their shooting videos to our app. After users upload their videos, our App will use MediaPipe to get the landmarks and then send the landmarks to a Large Language Model to receive feedback. To begin with, MediaPipe will analyze the users' shooting videos and get the landmarks. After that, our App will send the landmarks to a Large Language Model [9]. When the Large Language Model receives the landmarks of users, it will expertly inform the user of any issues with their shooting forms and provide guidance on how to correct them. Besides, after getting the users' landmarks, we will compare their shooting data with the shooting data of NBA stars stored in the database, thereby identifying the NBA star whose shooting form is the most similar to that of the user.

The first experiment tested the accuracy of our random forest algorithm in analyzing the user's shooting posture and shooting arc. The second experiment tested the processing time of our video analysis system for videos of different lengths and sizes. In the first experiment, we wrote a helper function to display the confusion matrix. Through the confusion matrix, we can see how many correct and incorrect judgments our model made, and then see the accuracy of our model. In the second experiment, we input multiple videos of different lengths and sizes into the system and recorded their respective processing times. By comparing the processing times, we can see our system's ability to process data. The most significant finding in the first experiment is that our model has a very high accuracy rate in judging whether a user's shooting posture is good or bad. The most significant finding in the second experiment is that our system can process videos in about 40 seconds.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Flutter Frontend Optimization for Video and Real-Time Feedback

One major component is the front end of the app built with Flutter. Implementing this part may raise several potential challenges, including real-time video stream handling, smooth user interface performance, and reliable communication with the backend. To address video stream processing, I could use Flutter's native plugins such as camera or ffmpeg, and apply asynchronous methods to minimize lags. To enhance the interface, I could employ Material Design or Cupertino widgets with customized themes to balance aesthetics and usability. For efficient backend communication, I could adopt WebSocket or gRPC to reduce latency and provide real-time feedback.

2.2. Scaling Flask Backend with Secure and High-Performance AWS Integration

Another major component is the backend server, which uses the Flask architecture and connects to the AWS database. There may be several problems in implementing this part, such as how to ensure the high concurrency processing capability of the server, how to ensure data security, and how to optimize database query performance. To solve the high concurrency problem, I could use Gunicorn or Uvicorn to run with Flask, and combine it with load balancing (such as AWS's ALB) to distribute requests. To ensure data security, HTTPS could be used to encrypt communication, and access control and regular backups could be enabled at the database level. As for query optimization, I could use indexes, caches (such as Redis), and asynchronous task queues (such as Celery) to reduce the pressure on the database and ensure that posture analysis results can be stored and retrieved efficiently.

2.3. Machine Learning for Basketball Detection and Shooting Analysis

Another major component is machine learning logic. I had to think about how to identify the position of the basketball and also the basket, how to identify the user's shooting motion, and how to tell whether the user's shooting form and shooting arc were good or not. I could use yolov5 or Faster R-CNN to identify the basketball and the basket. For identifying the user's shooting motion, I could use MediaPipe or BlazePose or OpenPose To detect key points such as wrists, elbows, shoulders, knees, ankles, etc.. To judge the quality of shooting form and shooting arc, I could use Random Forest or LSTM to analyze the user's shooting action.

3. SOLUTION

The three major components that comprise the SharpShooter project are the Flutter mobile application, the Flask backend server, and lastly the machine learning algorithm to classify good and bad shots. To start, the user is able to open the SharpShooter mobile app and upload videos of themselves shooting a basket. Upon uploading and submitting a video, the video is then sent to the backend server, where it is processed by the computer vision and machine learning algorithms. The computer vision contains the use of Mediapipe for finding the user's landmarks and a Yolov5 model for finding the basketball's landmarks. Once the landmarks have been identified and a dataframe has been created, the data frame is passed to the machine learning model, a trained RandomForestClassifier, where a classification is found for "good" or "bad." This result is then

saved in our Firebase database, where the user can come back later and check to find their results [7]. The trained RandomForestClassifier was trained on good and bad shot data in house as well as data gathered from the internet. In addition to the main flow, the backend server also takes the user's uploaded video and compares it's landmarks to a variety of NBA player's landmarks with the intention of finding the player they shoot the most alike. The video of the player they shoot the most alike is returned back to the user, inspiring them to keep practicing. After each video is broken down by the computer vision and machine learning model, an LLM analyzes the resulting landmark data and classification and provides some feedback and advice for the user to put into practice.

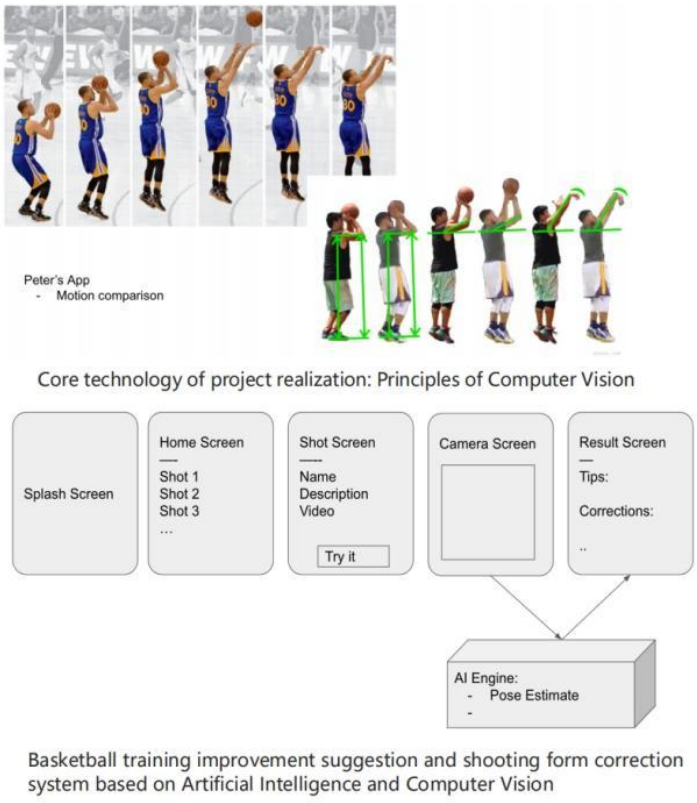


Figure 1. Overview of the solution

The Flutter mobile application for the SharpShooter project serves the purpose of being the user's interface, allowing the user to upload their own videos and see their analysis. The mobile application was created with the Flutter framework, and is available on both Google Play and iOS app stores.

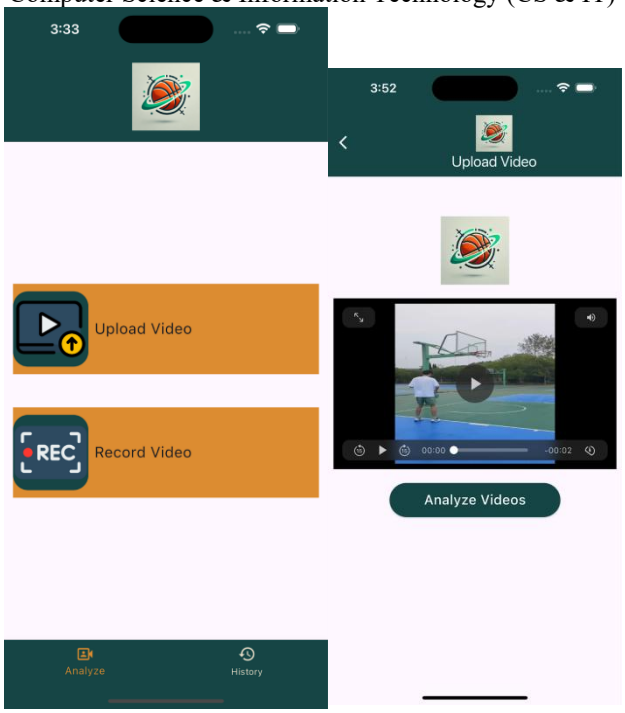


Figure 2. Screenshot of upload and analyze page

```
Future<void> sendVideoToServer(String filePath) async {
  try {
    // Create the custom HTTP client
    var customHttpClient = CustomHttpClient();

    // Create a MultipartRequest using the custom HTTP client
    var request = http.MultipartRequest('POST', Uri.parse('$backend_Url/get_prediction'))
      ..fields['userId'] = user_id!
      ..files.add(await http.MultipartFile.fromPath('video', filePath));

    // Send the request using the custom HTTP client
    var response = await customHttpClient.send(request);

    if (response.statusCode == 200) {
      print('Video uploaded successfully');
    } else {
      print('Failed to upload video');
    }
  } catch (e) {
    print('Error: $e');
  }
}
```

Figure 3. Screenshot of code 1

The screenshot above depicts the sendVideoToServer function, the function utilized by the SharpShooter app to take a video file at a given file path and send it to the backend server. The function starts by declaring a custom http client, which is necessary because the backend server does not have a registered ssl. Afterwards, the multipart request is created to contain the user_id and the video file itself. The request is being made to the '/get_prediction' route on the server, as this is where the server accepts the video files for analysis. After the request is created, the request is sent and the response is awaited [8]. The response is then checked to see if the status code was 200, in which case, a print statement declaring the video was uploaded successfully

executed. Once the back end server receives the video, it will begin processing it and will eventually save it in the database, where the mobile app will eventually check it.

The Flask back-end server serves the purpose of being the processing center for all of the user's videos. Every user is able to use the mobile app to upload their own videos for the server to use the machine learning algorithm to process them.

```python
@server.route( rule: "/get_prediction", methods=["POST"])  just garret
def get_prediction():
    try:
        # Parse request
        video_file = request.files.get("video")
        if not video_file:
            return jsonify({"error": "You must send a video file."}), 400
        userId = request.form.get("userId")
        if not userId:
            return jsonify({"error": "You must have a valid user ID"}), 400

        # Validate and save video
        if not allowed_file(video_file.filename):
            return jsonify({"error": "Invalid file type"}), 400

        # Save the video to a local file
        video_filename = f"{int(time.time())}_{video_file.filename}"
        video_path = os.path.join("uploads", video_filename)  # Ensure 'uploads' directory exists
        video_file.save(video_path)

        # Start the processing in a separate thread
        processing_thread = threading.Thread(target=prediction_process, args=(userId, video_path))
        processing_thread.start()

        return jsonify({"status": "Processing video file now..."}), 200

    except Exception as error:
        print(f"Error in /get_prediction: {error}")
        return jsonify({"error": f"Error: {error}"}), 500
```

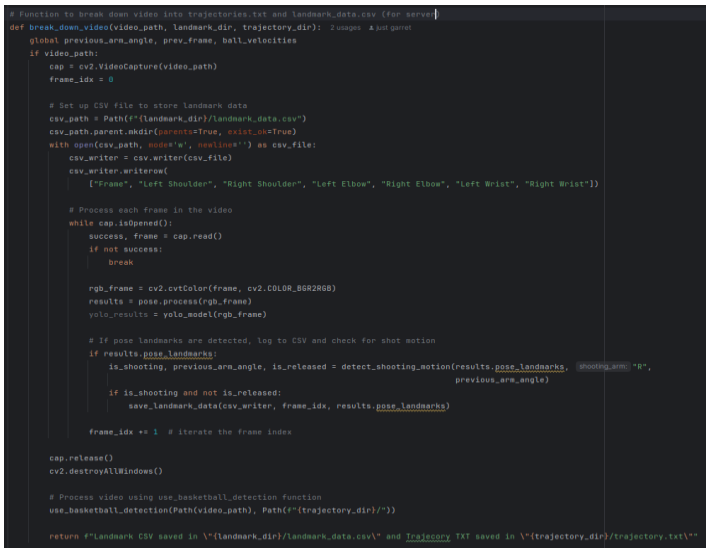
```python
def prediction_process(userId, video_path):  1 usage  just garret
    try:
        # Analyze the video
        print(f"Processing video at path: {video_path}")
        break_down_video(video_path,  landmark_dir: "server_data/",  trajectory_dir: "server_data/")

        # Load models
        landmark_model = load_landmark_model("data/landmark_data/basketball_shot_model.pkl")
        trajectory_model = load_trajectory_model("data/trajectory_data/trajectory_model.pkl")

        # get data paths
        landmark_file_path = "server_data/landmark_data.csv"
        video_split = video_path.split("/")
        video_filename = video_split[len(video_split) - 1]
        video_name = video_filename[:-4]
        trajectory_file_path = f"server_data/{video_name}/trajectory.txt"
        avi_file_path = f"server_data/{video_name}/output_{video_name}.avi"

        # convert avi to mp4
        mp4_file_path = f"server_data/{video_name}/output_{video_name}.mp4"
        convert_avi_to_mp4(avi_file_path, mp4_file_path)

        if not os.path.exists(trajectory_file_path) or not os.path.exists(mp4_file_path):
            print(f"Trajectory and AVI file path don't exist:\n{trajectory_file_path}\n{mp4_file_path}")

        print(f"Trajectory and AVI file path exist:\n{trajectory_file_path}\n{mp4_file_path}")

        # Load features and predict
        landmark_features = load_single_landmark_file(landmark_file_path,  label: 0).drop(columns=["Label"])
        trajectory_features = load_trajectory(trajectory_file_path).drop(columns=["Label"])

        print(f"Loaded landmark features: {landmark_features}")
        print(f"Loaded trajectory features: {trajectory_features}")

        # get predictions
        landmark_result = landmark_predict(landmark_model, landmark_features)
        trajectory_result = trajectory_predict(trajectory_model, trajectory_features)
        print(f"landmark result: {landmark_result}")
        print(f"trajectory result: {trajectory_result}")

        form_classification = "good" if landmark_result[0] == 1 else "bad"
        trajectory_classification = "good" if trajectory_result[0] == 1 else "bad"
```

Figure 4. Screenshot of code 2

As seen in the screenshot above, the '/get_prediction' route on the Flask server is responsible for receiving the video files from the users, and then using the prediction_process() function to process the video using the machine learning algorithm [14]. Upon receiving the video in the

get_prediction route, the Flask server saves the video file locally, and passes the path of the video to the prediction_process method in a new thread. Putting the prediction_process in a new thread keeps the request from stalling, as the prediction_process does take upwards of 2 minutes oftentimes. Within the prediction_process method, the video is firstly broken down and the landmarks are collected for the user's shot form as well as the trajectory for the basketball's motion. After these have been found and saved, the trained landmark and trajectory models are loaded up and are ready for predicting [15]. Once the models have been loaded and the data paths and features have been found for the landmark and trajectory data, the resulting predictions for the user's form and arc are calculated using the landmark_features and trajectory_features. Afterwards, corresponding to the prediction value of either 0 for bad or 1 for good, the classification is reassigned to the string value of "good" or "bad".

The machine learning algorithm within the SharpShooter project serves the purpose of analyzing a user's submitted video to determine if their shot form and arc are good or bad. The machine learning algorithm has a lot of moving parts, and relies on technologies like mediapipe, yolov5, and a trained RandomForrest model.



```
# Function to break down video into trajectories.txt and landmark_data.csv (for server)
def break_down_video(video_path, landmark_dir, trajectory_dir):  2 usages  ▲ just garret
    global previous_arm_angle, prev_frame, ball_velocities
    if video_path:
        cap = cv2.VideoCapture(video_path)
        frame_idx = 0

        # Set up CSV file to store landmark data
        csv_path = Path(f"{landmark_dir}/landmark_data.csv")
        csv_path.parent.mkdir(parents=True, exist_ok=True)
        with open(csv_path, mode='a', newline='') as csv_file:
            csv_writer = csv.writer(csv_file)
            csv_writer.writerow(
                ["Frame", "Left Shoulder", "Right Shoulder", "Left Elbow", "Right Elbow", "Left Wrist", "Right Wrist"])

            # Process each frame in the video
            while cap.isOpened():
                success, frame = cap.read()
                if not success:
                    break

                rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                results = pose.process(rgb_frame)
                yolo_results = yolo_model(rgb_frame)

                # If pose landmarks are detected, log to CSV and check for shot motion
                if results.pose_landmarks:
                    is_shooting, previous_arm_angle, is_released = detect_shooting_motion(results.pose_landmarks,  shooting_arm: "R",
                                                                                          previous_arm_angle)
                    if is_shooting and not is_released:
                        save_landmark_data(csv_writer, frame_idx, results.pose_landmarks)

                frame_idx += 1  # iterate the frame index

        cap.release()
        cv2.destroyAllWindows()

        # Process video using use_basketball_detection function
        use_basketball_detection(Path(video_path), Path(f"{trajectory_dir}/"))

        return f"Landmark CSV saved in \"{landmark_dir}/landmark_data.csv\" and Trajecory TXT saved in \"{trajectory_dir}/trajectory.txt\""
```

Figure 5. Screenshot of code 3

In the above screenshot, the break_down_video function that was referenced earlier can be seen. The break_down_video function is used for processing a video and outputs the data frame of MediaPipe landmarks of the shooter as well as the basketball landmarks for the shot arc. The function starts by creating a cv2 capture instance with the video, opening it alongside a csv file to iterate frame by frame, and finding the landmark data and storing it inside the csv file. For each frame in the capture instance, the landmarks and yolo_results are found to determine what the user's landmarks are as well as if they are shooting or not. If the user is shooting and has not released the ball yet, then the landmarks will be collected and saved into the landmark_data.csv file. Once the video has been iterated through for the landmarks, the use_basketball_detection() function will utilize some code from a previous project by github user Stardust87 to find the arc trajectory landmarks of the basketball. Both of these landmarks are the data that the models used for training and what they take as inputs for predicting.

4. EXPERIMENT

4.1. Experiment 1

The major components whose accuracy is of the utmost importance for the proper function of the SharpShooter mobile application are the machine learning classification models for both the shot trajectory and shot form. It is critical that these models work well, as they are the main service of the app and provide the user with analysis on their basketball shot.

To test the accuracy of the shot trajectory and form models, we will utilize a helper function we have built to show a confusion matrix using the Seaborn and Matplotlib python libraries. The initial cufusion_matrix function originates from the Scikit-learn library, as do the RandomForestClassifier models we use as a base for both models. Once the confusion matrix data has been gathered, the Matplotlib figure is created, and the data is displayed demonstrating the Predicted values compared to the Actual values. This confusion matrix setup will allow us to clearly see how many accurate predictions the model makes immediately next to how many it gets incorrect.

```python
# Helper function for making a confusion matrix
def show_confusion_matrix(y_test, y_pred, title):  3 usages  new *
    # Compute the confusion matrix
    cm = confusion_matrix(y_test, y_pred)

    # Display the confusion matrix using a heatmap
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["bad", "good"], yticklabels=["bad", "good"])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(title)
    plt.show()
```

```python
def main(trajectories_list_csv):  A just garret *
    combined_df = load_from_list(trajectories_list_csv)  # Load the combined DataFrame
    print("Combined Data Preview:")
    print(combined_df.head())

    X = combined_df[["x", "y"]]  # Features
    y = combined_df["Label"]  # Labels

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)

    # Create the model
    model = create_model()

    # Train the model
    model = train_model(model, X_train, X_test, y_train, y_test)

    # Save the trained model
    model_path = "/media/user/Storage/Documents/Coding_Minds/peters_project/BackendServer/data/trajectory_data/trajectory_model.pkl"
    save_trajectory_model(model, model_path)

    # Predict and evaluate the model
    predictions = trajectory_predict(model, X_test)
    print("Model Accuracy:", accuracy_score(y_test, predictions))
    print("Classification Report:")
    print(classification_report(y_test, predictions))

    # show the confusion matrix
    show_confusion_matrix(y_test, predictions,  title: "Trajectory Confusion Matrix")
```

Figure 6. Code of experiment 1

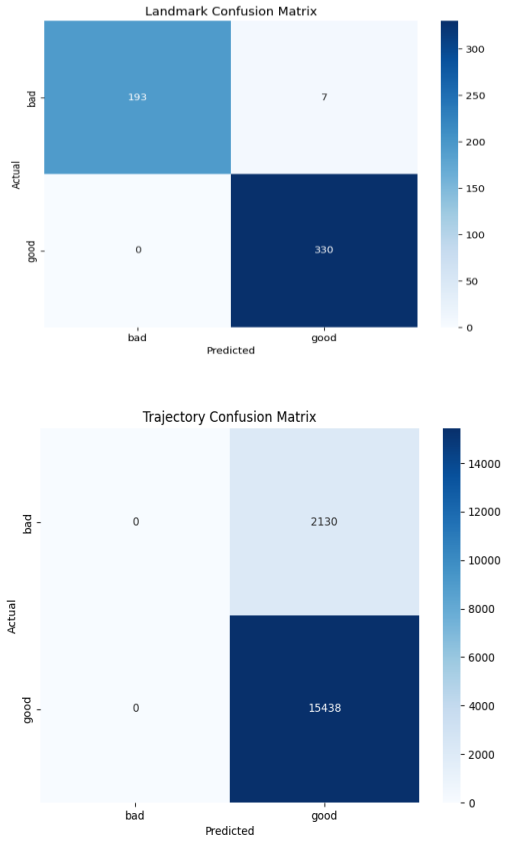


Figure 7. Figure of experiment 1

After running the code and performing the collection of our graphs, we have found that both the landmark and trajectory classification models are quite accurate. During the collection of our data, we were expecting a few inaccuracies due to the difference in data size for good and bad samples. Due to this difference in data size, we were expecting the model accuracy to suffer, in that it would not be as capable of accurately classifying an input. However, after performing the experiment, we have found that the landmark classification model was quite capable of accurately classifying data, as it only made three incorrect predictions out of 230 total points. We believe the reason for this consistency is due to the simplicity of the training data, as both models are trained on their own landmark-based data as opposed to complete image sets.

4.2. Experiment 2

Another major factor in ensuring a quality user experience is the analysis response time for each of the user video analysis. Ensuring these videos are analyzed in a prompt and consistent manner will display stability and make the service convenient for users.

To find the average video response time, we will conduct an experiment to find it by running multiple video analysis trials. Two experimenters will work together to achieve this, the first experimenter being responsible for the submission of the videos of different length and the second experimenter being responsible for the recording of the time. The first experimenter will

submit a video and inform the second experimenter to start the stopwatch. Finally when the first experimenter notices that the video response is completed they will let the second know to stop the timer and update the record.

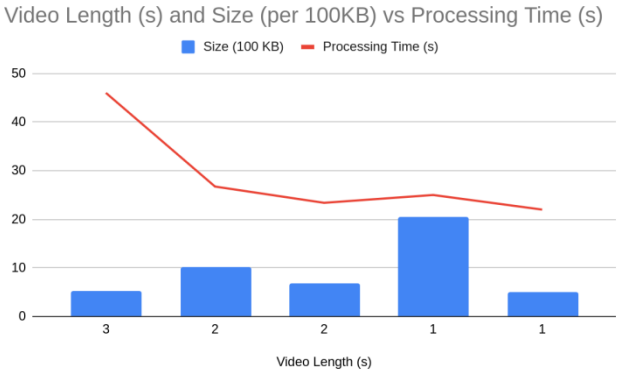


Figure 8. Figure of experiment 2

5. RELATED WORK

Here is the official website of an app that is also trying to provide users with shooting form analysis [2]. Different from our project, it sends users' shooting videos directly to a real world basketball coach for shooting posture analysis, rather than a Large Language Model, which leads to the fact that it is paid (each analysis of a shot costs 97$) while our project is completely free. Obviously, this is a huge gap. In addition, it can not compare the user's shooting form with the NBA stars' shooting form, so as to find the NBA star who is most similar to the user. (Actually, we are the first ones to achieve so.) And also, if its users are not satisfied with the analysis results, they need to pay 97$ for a second analysis, but our users only need to upload the video again to get a brand new answer. Lastly, it is not trying to solve the problem in a "Computer Science" way. It sends its users' shooting videos to a basketball coach. However, how many videos can a single basketball coach see through in a day? Our Large Language Model can respond to thousands of requests in a few minutes.

Here is a github project which designs a flutter application that uses pose estimation to provide personalized instructions on correcting users' basketball shooting form [3]. Different from our project, it gives fixed suggestions to its users based on their arm angles. (As you can see in the picture below) However, our project sends our users' shooting data to a Large Language Model (I'm pretty sure that we are the very first one to do so.) which means they can get different answers based on their own situations. In addition, it can not compare the user's shooting form with the NBA stars' shooting form, so as to find the NBA star who is most similar to the user. (Actually, we are also the first ones to achieve so.)

```
List<Widget> genSum(){
  List<String> bad = List();
  List<Widget> b = <Widget>[];
  if(!pts[0]){
    bad.add('Feet too wide apart');
  }
  if(!pts[1]){
    bad.add('Body bends over too much when bringing the ball down');
  }
  if(!pts[2]){
    bad.add('Hands should be close to chest when bringing the ball down');
  }
  if(!pts[3]){
    bad.add('Elbow should be close to side of body when shooting');
  }
  if(!pts[4]){
    bad.add('Ball should be kept close to body before the shot');
  }
  if(!pts[5]){
    bad.add('Body should be close to vertical to ground when shooting');
  }
  if(!pts[6]){
    bad.add('Extend out your arm upon release and follow through');
  }
  if(!pts[7]){
    bad.add('Ball should be released around your forehead');
  }
  if(!pts[8]){
    bad.add('Shooting arc is too low');
  }
```

Figure 9. Code of the project

Here is another github project which records its users' joint angle and makes it into a chart [4]. (As you can see in the picture below) In this way, its users can interpret it and change their shooting form. It's a really good way to look deep into our shooting form, but it ignores the shooters' wrist and chest. However, using mediapipe, our project can comprehensively analyze 33 key points of the human body. And also, this methodology only makes it into a chart but won't give you any suggestions. In contrast, our Random forest algorithm + Large Language Model can give you detailed suggestions based on your own situations. Lastly, it can not compare the user's shooting form with the NBA stars' shooting form, so as to find the NBA star who is most similar to the user. As I said, we are the first one to achieve so.

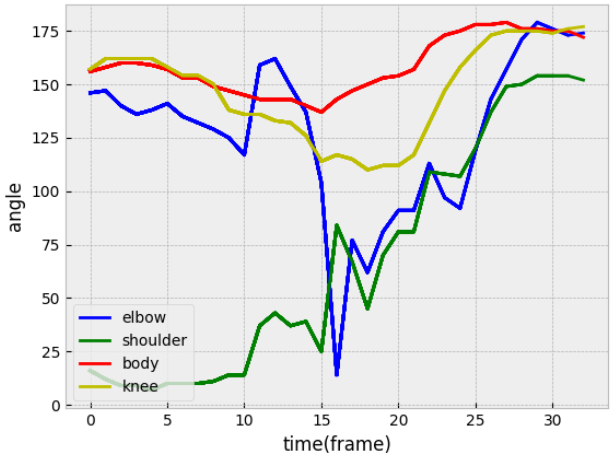


Figure 10. Chart of the project

6. CONCLUSIONS

Regarding the limitations of our project, first of all, I think ChatGPT is part of it [12]. As we all know, ChatGPT has its own shortcomings in some aspects, such as the formatting of replies and the accuracy of data analysis results. Secondly, I think our project's requirements for user-submitted videos are also limited to a certain extent. For example, we require that the videos submitted by users need to keep the camera angle as fixed as possible, otherwise it will affect the final analysis results. But in real life, people are used to shooting moving shots, such as moving the camera with the movement of the characters, which leads to the fact that if users want to use our APP for shooting analysis, they may need to record a separate video that meets our requirements. Lastly, Our random forest algorithm tends to judge whether the user's shooting arc is good or bad, and tends to judge all arcs as good arcs. If I have enough time, the first problem is still difficult to solve, because this is a common problem of all the large language models we can see on the market. For the second problem, I think the mediapipe currently used in our project is difficult to solve, because it is difficult to find a fixed coordinate system in a moving shot, which is crucial for analyzing the user's shooting action [13]. For the third question, I think our shortcoming is that there are too few samples of bad shooting arcs, or the definition of good and bad shooting arcs is not rigorous enough. We can add more samples of bad shooting arcs to our algorithm and define the quality of shooting arcs more accurately.

As a computer science student who loves basketball, I have seen many people who love basketball. Many of them do not have access to professional basketball training. They can only learn basketball by themselves bit by bit. They often feel very struggling and painful because basketball is a competitive sport and no one wants to lose to others on the court. Therefore, I sincerely hope that my project can help other people in the world who love basketball like me but cannot access professional basketball training.

REFERENCES

[1] Cosentino, Valerio, Javier Luis, and Jordi Cabot. "Findings from GitHub: methods, datasets and limitations." Proceedings of the 13th international conference on mining software repositories. 2016.
[2] Yan, Wenlin, Xianxin Jiang, and Ping Liu. "A review of basketball shooting analysis based on artificial intelligence." IEEE Access 11 (2023): 87344-87365.
[3] Miller, Stuart, and Roger Bartlett. "The relationship between basketball shooting kinematics, distance and playing position." Journal of sports sciences 14.3 (1996): 243-253.
[4] Li, Chenxuan, et al. "A Self-Correcting Vision-Language-Action Model for Fast and Slow System Manipulation." arXiv preprint arXiv:2405.17418 (2024).
[5] Okazaki, Victor HA, André LF Rodacki, and Miriam N. Satern. "A review on the basketball jump shot." Sports biomechanics 14.2 (2015): 190-205.
[6] Komić, Jasmin, et al. "The Influence of Game-Related Statistics on the Final Results in FIBA Global and Continental Competitions." Applied Sciences 14.12 (2024): 5357.
[7] Uzun, Ahmet, and Atilla Pulur. "The effect of shooting training on the development of the shot hit rate for basketball players." Journal of Human Sciences 15.4 (2018).
[8] Klock, M., Mateusz Tomanek, and Wojciech Cieśliński. "Social media and the value of contracts based on the example of the NBA." Journal of Physical Education and Sport 20.5 (2020): 3063-3069.
[9] Trevarthen, Colwyn. Children with autism: Diagnosis and interventions to meet their needs. Jessica Kingsley Publishers, 1998.
[10] Chang, Yupeng, et al. "A survey on evaluation of large language models." ACM transactions on intelligent systems and technology 15.3 (2024): 1-45.
[11] Zhang, Yu, et al. "Real-time vehicle detection based on improved yolo v5." Sustainability 14.19 (2022): 12274.

[12] Welsby, Philip, and Bernard MY Cheung. "ChatGPT." Postgraduate Medical Journal 99.1176 (2023): 1047-1048.
[13] Lugaresi, Camillo, et al. "Mediapipe: A framework for building perception pipelines." arXiv preprint arXiv:1906.08172 (2019).
[14] Ray, Susmita. "A quick review of machine learning algorithms." 2019 International conference on machine learning, big data, cloud and parallel computing (COMITCon). IEEE, 2019.
[15] Barris, Sian, and Chris Button. "A review of vision-based motion analysis in sport." Sports medicine 38.12 (2008): 1025-1043.