Individual Project Report

Kalyani Vinaygam Sivasundari (G27326652)

An overview of the project

The "DFL - Bundesliga Data Shootout" project is a Kaggle competition that aimed to harness the power of computer vision and deep learning models to automate the detection and classification of specific football events—passes, throw-ins, crosses, and challenges—from Bundesliga match footage. The objective was to develop a model that could not only identify these events with high accuracy but also classify them within predefined scoring intervals. This would enable more detailed analysis and insights into game dynamics, enhancing strategic planning and player performance evaluation.

The motivation behind this project stems from the increasing demand in sports analytics for precise, automated tools that can provide real-time insights into game events without the need for manual tagging. By automating the process, teams and broadcasters can gain a competitive edge, optimizing strategies and improving viewer engagement with detailed, data-driven analyses.

My Part of Work:

1. Exploratory Data Analysis

2. Using a Custom Trained Yolov5x model and perform Object Detection and Tracking, Player's Team Assignment and Ball Possession Analysis

Exploratory Data Analysis

I analyzed the train.csv which had the details of events annotations for the videos in the training folder. Doing the analysis, I was able to visualize and get useful insights about the distribution of event-attributes, events and video-ids.

Object Detection and Tracking, Team Assignment and Ball Possession Analysis using a Custom Trained Yolov5x model

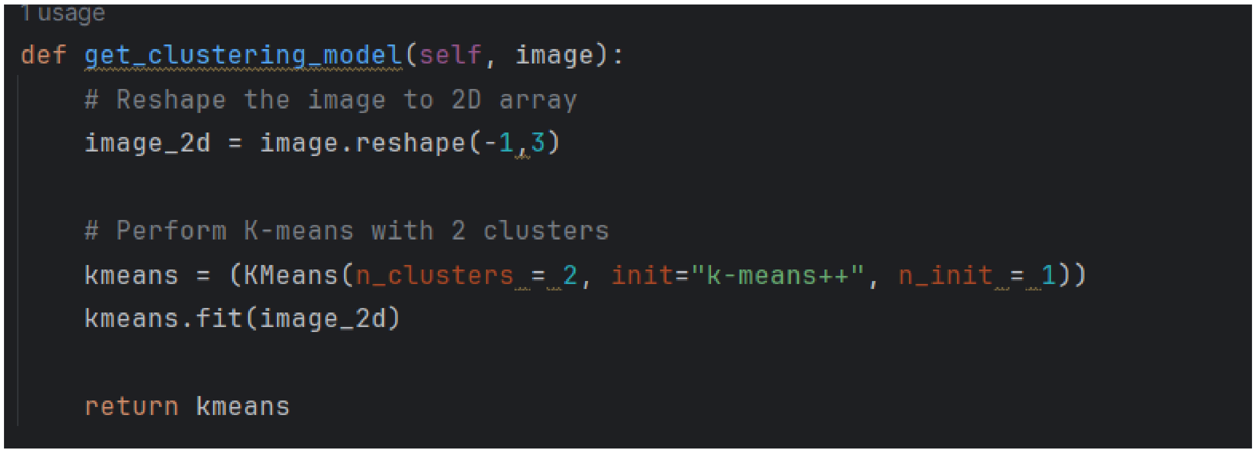
Custom YOLOv5x Model Training

I utilized the Roboflow YOLOv5 Football-Player-Detection Image Dataset, which is specifically designed for detecting football players in various frames of a football game. The dataset was used to train a custom model based on the YOLOv5x architecture. This choice was motivated by YOLOv5x's capabilities in handling complex object detection tasks with high accuracy and speed. Training was conducted and the best and last weights were generated. The best weights were used for subsequent video annotation tasks, as they provided the most accurate detection results.

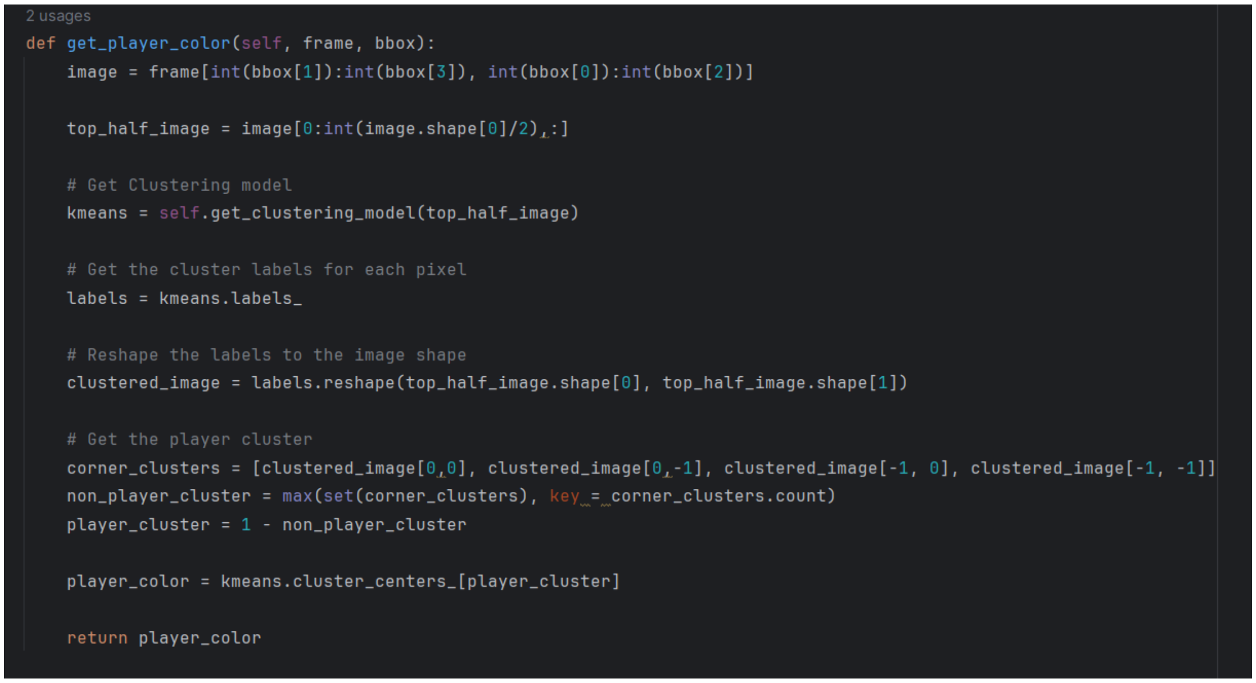
Flow of the modules:

At first the system developed initiates the video analysis process by receiving a football match video file as input. It then breaks down this video into individual frames for detailed scrutiny. Each frame is processed through a custom-trained YOLOv5x model to detect key objects, specifically players and the ball. If the ball becomes obscured or leaves the frame, its position is interpolated based on its last known trajectory and speed to maintain uninterrupted tracking. Early in the analysis, a K-Means clustering algorithm is employed to assign team colors to players based on the predominant colors in their detected bounding boxes. This team color assignment is dynamically monitored and updated throughout the video to accommodate any changes in player appearance due to shifts in lighting or perspective. For each frame, the system calculates the distance between each player and the ball, assigning possession to the player closest to the ball within a predefined threshold. The frames are annotated with tracking information, such as player paths, ball position, team colors, and possession status. Finally, these annotated frames are reassembled into a video format, providing a comprehensive visual representation of the game.

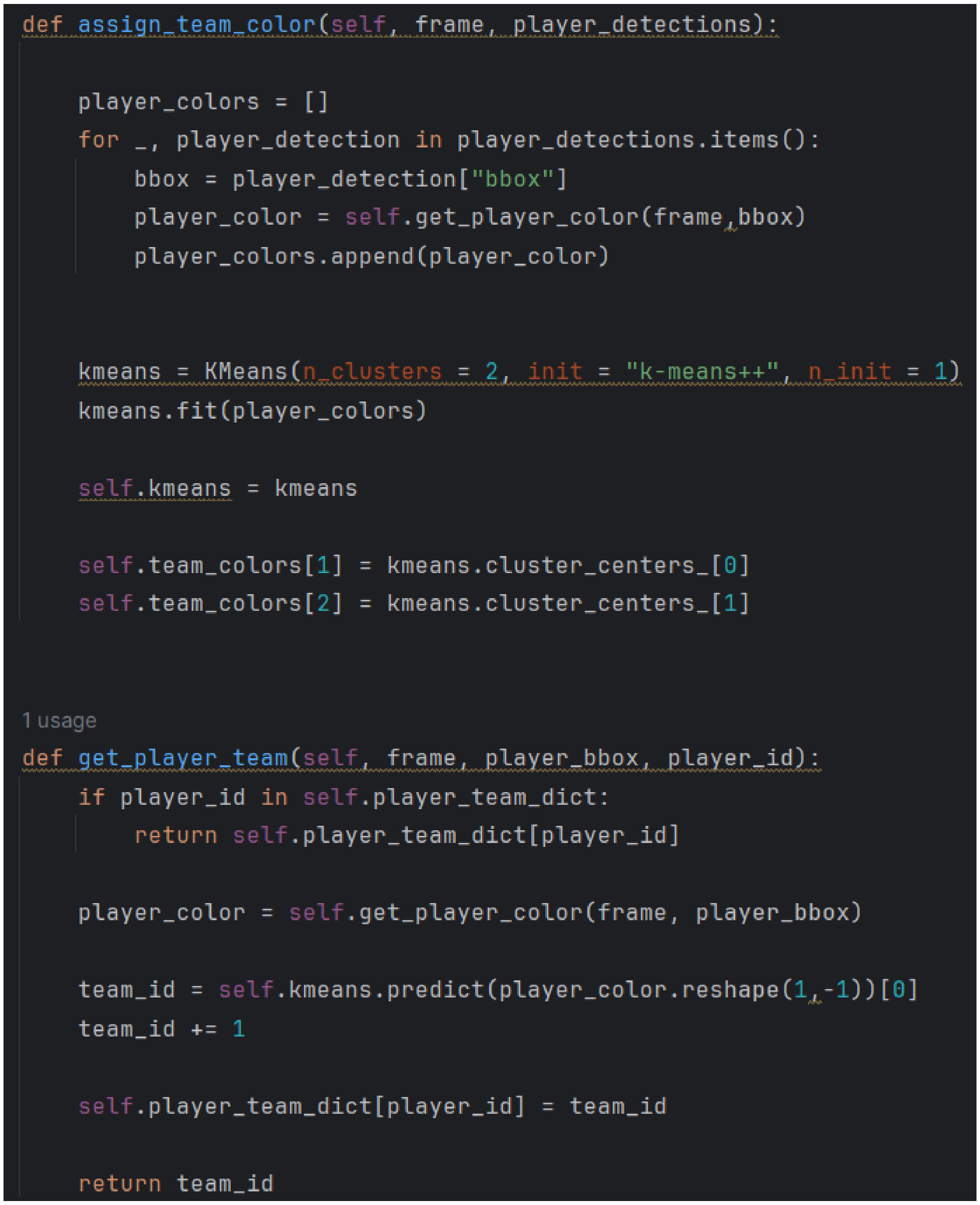
Code Snippets:



```
1 usage
def get_clustering_model(self, image):
    # Reshape the image to 2D array
    image_2d = image.reshape(-1,3)

    # Perform K-means with 2 clusters
    kmeans = (KMeans(n_clusters = 2, init="k-means++", n_init = 1))
    kmeans.fit(image_2d)

    return kmeans
```



```
2 usages
def get_player_color(self, frame, bbox):
    image = frame[int(bbox[1]):int(bbox[3]), int(bbox[0]):int(bbox[2])]

    top_half_image = image[0:int(image.shape[0]/2),:]

    # Get Clustering model
    kmeans = self.get_clustering_model(top_half_image)

    # Get the cluster labels for each pixel
    labels = kmeans.labels_

    # Reshape the labels to the image shape
    clustered_image = labels.reshape(top_half_image.shape[0], top_half_image.shape[1])

    # Get the player cluster
    corner_clusters = [clustered_image[0,0], clustered_image[0,-1], clustered_image[-1, 0], clustered_image[-1, -1]]
    non_player_cluster = max(set(corner_clusters), key = corner_clusters.count)
    player_cluster = 1 - non_player_cluster

    player_color = kmeans.cluster_centers_[player_cluster]

    return player_color
```



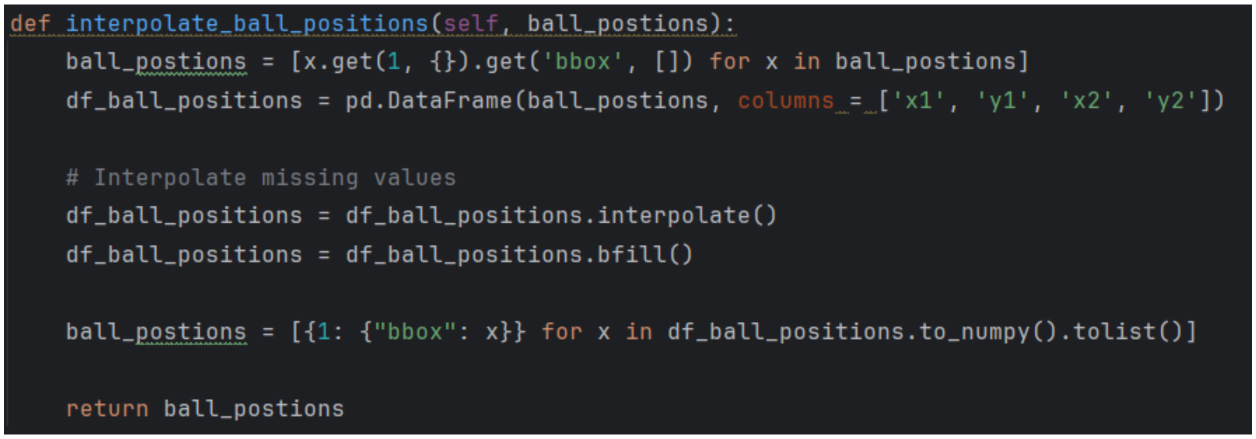
```
def assign_team_color(self, frame, player_detections):

    player_colors = []
    for _, player_detection in player_detections.items():
        bbox = player_detection["bbox"]
        player_color = self.get_player_color(frame,bbox)
        player_colors.append(player_color)


    kmeans = KMeans(n_clusters = 2, init = "k-means++", n_init = 1)
    kmeans.fit(player_colors)

    self.kmeans = kmeans

    self.team_colors[1] = kmeans.cluster_centers_[0]
    self.team_colors[2] = kmeans.cluster_centers_[1]


1 usage
def get_player_team(self, frame, player_bbox, player_id):
    if player_id in self.player_team_dict:
        return self.player_team_dict[player_id]

    player_color = self.get_player_color(frame, player_bbox)

    team_id = self.kmeans.predict(player_color.reshape(1,-1))[0]
    team_id += 1

    self.player_team_dict[player_id] = team_id

    return team_id
```

Clustering for Team Assignment

To differentiate between two teams based on jersey colors, I developed a clustering-based method. For each detected player, I extracted the relevant section of their jersey from the frame, focusing on the top half to capture the most distinct color data. I applied the K-means clustering algorithm to these color data points, isolating the player's jersey color from the background. By analyzing the clusters identified in the corners of the image, I determined the predominant non-jersey colors, thereby identifying the actual jersey color as the other cluster. This approach was extended to group all players' jersey colors into two distinct team colors using another round of K-means clustering.

Player Team Identification

Each player's team affiliation was determined by checking if they had been previously assigned a team to avoid redundant processing. If not, I extracted their jersey color using the previously mentioned method and matched it to the closest team color using our K-means model. This allowed me to assign players to their respective teams accurately and update our system's records for ongoing analysis.



```python
def interpolate_ball_positions(self, ball_postions):
    ball_postions = [x.get(1, {}).get('bbox', []) for x in ball_postions]
    df_ball_positions = pd.DataFrame(ball_postions, columns = ['x1', 'y1', 'x2', 'y2'])

    # Interpolate missing values
    df_ball_positions = df_ball_positions.interpolate()
    df_ball_positions = df_ball_positions.bfill()

    ball_postions = [{1: {"bbox": x}} for x in df_ball_positions.to_numpy().tolist()]

    return ball_postions
```

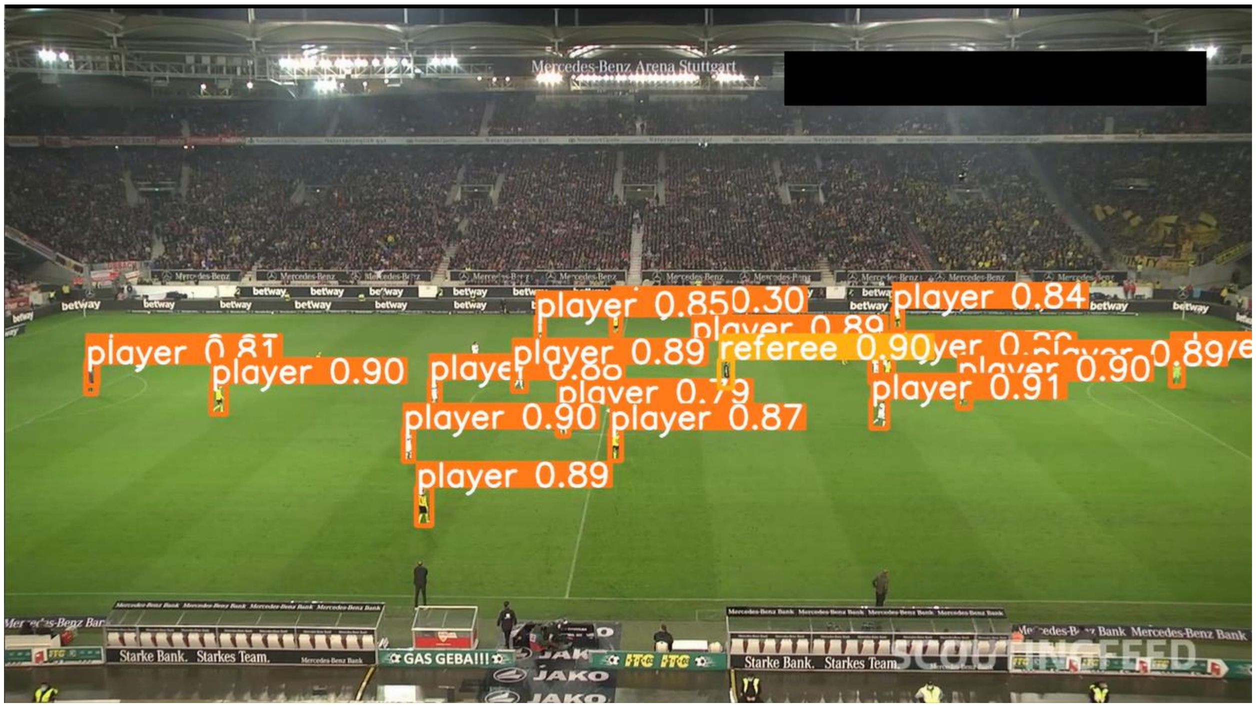
Interpolation of Ball Positions

Recognizing the challenge of tracking the ball when it is temporarily obscured or out of view, I implemented a method to estimate its position during these periods. I started by extracting bounding box coordinates of the ball from each frame's detection results, which I organized into a pandas DataFrame for ease of manipulation. I then employed pandas' interpolation function to estimate missing positions, ensuring continuous tracking by backfilling any remaining gaps. The final step involved converting the DataFrame back into a format compatible with our system, allowing seamless integration into the tracking process.

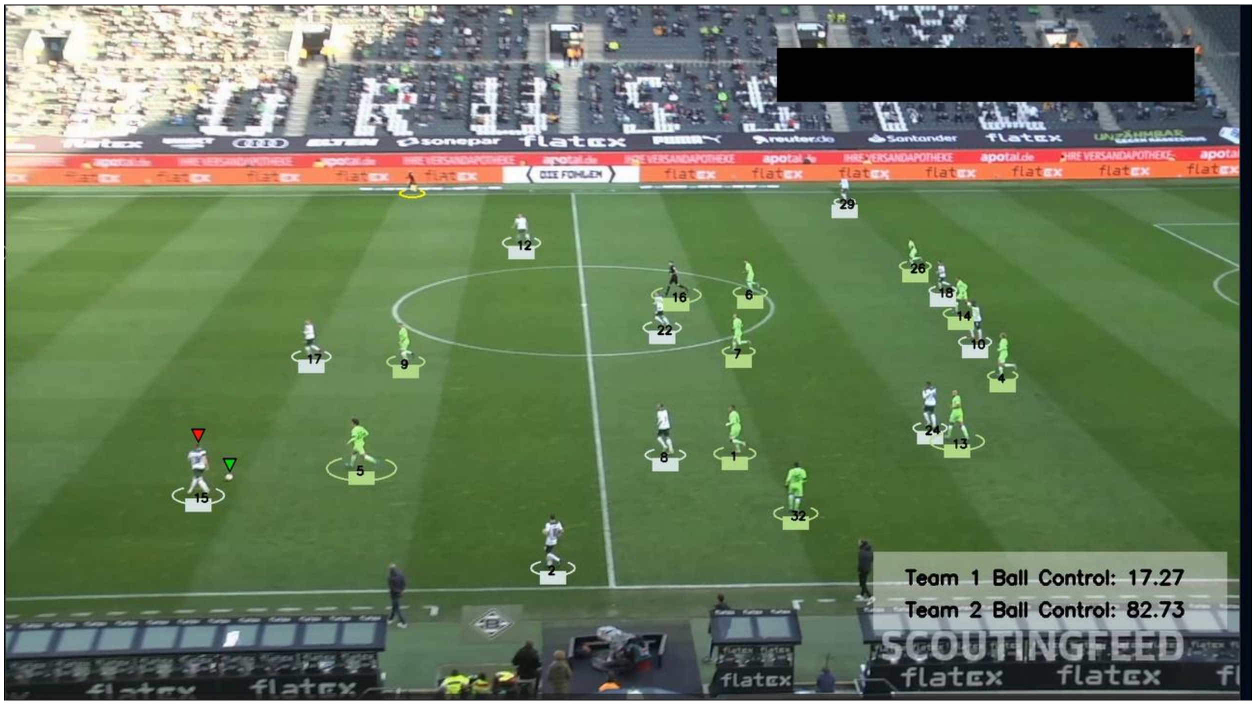
Results

[redacted]

Input Video


[redacted]

Results applying Custom Yolov5x


[redacted]

Team 1 Ball Control: 17.27
Team 2 Ball Control: 82.73

Results applying Custom YOLOv5x with all module implementations

The first picture is a capture of the video where no annotations where made. But the second has annotations but it is doesn't have a clear view of the the details. The picture is the fully processed video capture. It has clear representations of the annotations and shows of how it looks like when all the modules I discussed was applied.

Though I was able to picture the way to annotate the videos, I faced challenges related to computational resources and system compatibility, which have impacted the pace and effectiveness of model development and training to the fullest of reaching the final goal of the project. Issues with AWS instance stability and local system constraints have posed significant hurdles, limiting the ability to conduct extensive real-time video processing and analysis.

Summary

The system effectively annotates the football videos by tracking the ball and the players, determining the team of the player by analysis Jersey color and enhancing the track viewability of the ball by handling the interpolation of ball positions. This all together is a great learning experience starting with how the YOLO model works how to integrate and use it to the model I wanted to develop.

Code Precentage Copied From Internet: 66.66%

References

1. https://sh-tsang.medium.com/brief-review-yolov5-for-object-detection-84cc6c6a0e3a
2. https://github.com/ultralytics/yolov5/releases
3. https://universe.roboflow.com/roboflow-jvuqo/football-players-detection-3zvbc
4. https://medium.com/@amritangshu.mukherjee/tracking-football-players-with-yolov5-bytetrack-efa317c9aaa4
5. https://eudl.eu/pdf/10.4108/eai.23-11-2023.2343216