Video Processor

After examining the video data and the accompanying JSON files, it seems the most reliable way to extract player information is to process the videos frame by frame. This approach minimizes errors caused by motion blur, low resolution, or fast movements, ensuring that no critical data is missed.

Approach:

Frame Extraction:

First, split the video into individual frames using a tool like FFmpeg, OCR

We'll use a frame rate of around 30 FPS to maintain accuracy while managing processing speed.

Frames will be saved in a structured format, such as numbered image files in a designated folder.

Manual Data Tagging:

Once frames are extracted, we'll manually tag each frame with the relevant player information (e.g., jersey numbers and names).

Use a simple tagging tool or a custom script that allows adding metadata to each frame.

Store the tagged data in a structured JSON file for further processing.

Data Aggregation:

After tagging, aggregate the results to calculate how long each player was visible.

Count the number of frames in which each player appears, and convert this count to time using the frame rate.

Handle edge cases where players may be partially visible or where there is overlap between players.

Data Validation:

Double-check the manually tagged frames for consistency.

Handle potential errors such as duplicate entries or mismatches between frame numbers and player data.

Observations:

Videos are around 30 seconds long, which means roughly 900 frames at 30 FPS.

Typically, one or two players are the main focus of the video at any given time.

JSON files contain metadata and identifiers for all players relevant to the sport.

Technical Specifications:

1. Frame Extraction and Tagging:
Tools:

Use FFmpeg to split videos into frames

Manual Tagging:

Create a basic UI using React.js that allows users to view frames and enter player information.

Save the tagged data as JSON files, linking each frame to a player ID and name.

Backend Processing:

Node.js:

Create a backend service to manage frame processing and data aggregation.

Use Express.js for API endpoints to upload videos and retrieve processed results.

Batch Processing:

Implement asynchronous processing using worker threads to manage multiple videos simultaneously.

Track progress and allow for resuming if the process is interrupted.

Caching and Storage:

Redis:

Cache intermediate results and manage the status of each processing job.

Use Redis as a task queue to handle frame extraction and tagging jobs.

PostgreSQL:

Store structured data with clear relationships between frames and player visibility.

Use a table structure that links player IDs to video frame timestamps.

Frontend Interface:
React.js:

Provide a user-friendly dashboard for uploading videos and viewing processing results.

Include options to manually correct player data if errors are detected during the tagging phase.

Deployment and Environment:
Docker:

Containerize the application to ensure consistent execution across environments.

Use Docker Compose to manage the multi-service setup (Node.js, Redis, PostgreSQL).

Firebase:

Store processed results and raw video data for analysis.

Host the web interface for managing uploads and results.

Challenges and Considerations:
Manual Tagging Effort:

Manually tagging frames is time-consuming, but it ensures accuracy, especially when AI is not reliable.

Consider developing a basic keyboard shortcut system to speed up the tagging process.

Handling Data Volume:

Storing and processing thousands of frames requires efficient data management.

Use compressed formats (like JPEG) to save space without significantly impacting quality.

Optimization:

Batch processing frames to reduce I/O operations.

Minimize memory usage by processing frames in chunks rather than loading an entire video at once.

Possible Future Enhancements:

Semi-Automated Tagging:

Implement a basic heuristic to pre-tag frames, allowing manual correction afterward.

Enhanced UI for Tagging:

Add features like zoom, frame-by-frame navigation, and undo/redo functionality.

Data Visualization:

Show a timeline of player appearances based on the frame data, making it easier to analyze patterns.

User Collaboration:

Allow multiple users to tag frames simultaneously and merge the results for improved efficiency.