COLORADO SCHOOL OF MINES

CSCI 370 Final Report

ClubCasters

Ethan Kerstiens
Daniel Longtine
Samuel Wasserman
Thomas Glenzinski

Revised June 14, 2025

CSCI 370 Summer 2025

Dr. Jeffrey Paone

Revision	Date	Comments
New	05/18/25	Submitted rough draft for sections 1-5
Rev - 2	05/23/25	Included new section 6, System Architecture. Added description paragraphs for each section. Reformatted existing sections including grammatical errors and improved clarity. Added another Appendix for tables and Figures. Reformatted our risks into tabular format instead of bullets. Reformatted existing diagrams and included new diagrams to demonstrate important features of our architecture.
Rev – 3	5/30/25	Made formatting changes for spacing between the sections. Added new diagrams and tables and updated paragraphs to reference these new Figures. Added project ethical considerations and software testing and quality sections.
Rev – 4	6/08/25	Added content to new sections for the project completion status, future work, and acknowledgements. Updated the results of the tests. Updated some minor formatting issues.
Rev - 5	6/14/25	Made updates according to peer review, including: formatting changes with certain spacing and font sizes, changing the tense of paragraphs, adding references, changing diagrams, improving clarity on our model architecture and performance, and finally adding details to completion status, future work, lessons learned, acknowledgements.

Table 1: Revision history

Table of Contents

I. Introduction

Our client for this project is ClubCast, a startup founded by Colorado School of Mines graduates. ClubCast focuses on enhancing the experience of watching and sharing club and intramural sports by building a platform that allows games to be recorded, processed, and viewed through their website. They already use AI tools and services to record matches, as well as platforms like Roboflow, a data labeling and cleaning software, to assist in training and managing computer vision models. ClubCast is looking to expand their capabilities by automating the creation of highlight reels, enabling users to quickly access key moments from games without manual editing. This improvement is part of their broader goal to make intramural sports more accessible, watchable, and shareable.

Our project supports this mission by using computer vision to automatically detect and extract highlights from recorded sports matches. We are developing a system that takes in a video-on-demand (VOD) of a game and outputs a structured list of highlight segments, including start and end timestamps and a classification of the type of highlight (e.g., goal, corner, set play). The final deliverable is a deployable pipeline that accepts a trained model and a video as inputs, processes the video efficiently, and returns a JSON file containing detailed highlight information.

Our training data was labeled manually and organized using custom scripts and local tools to support model development. Our primary data sources include pre-recorded videos from ClubCast's existing archives, as well as publicly available footage from similar sports events. In the short term, the ClubCast team is the primary user of the system. They streamline their content creation workflow using various AI and data organization tools like Roboflow and are also responsible for maintaining as it integrates into their existing infrastructure. This project aims to not only meet their current needs but also lay a foundation for scalable improvements in the future.

II. Functional Requirements

The functional requirements define the technical criteria that our minimum viable product (MVP) must meet. Our system is an API-driven platform. Users can upload a sports video, which is then processed by a video classification model that detects highlights. The API returns structured highlight data, including types and timestamps. This includes the type of highlight events, their timestamps, and the current status of the job. The system processes videos faster than their actual duration to ensure scalability and responsiveness.

In addition to meeting the minimum requirements for our MVP, we identify several stretch goals to expand the usefulness and scalability of our system. These goals increase the number of highlights by identifying specific event types such as free kicks, corners, and more. We also plan to generalize our model to additional sports, increasing its commercial and practical value. Finally, we containerize our API using Docker, making it more portable and easier to deploy across different environments [1]. We pursue these enhancements as time and resources allow, after the core functionality is stable and verified.

- Minimum Requirements
    - Users can upload a sports video to the system via a REST API.
    - Users can check the status of their video processing job (e.g., Awaiting, Running, Failed, or Successful).
    - Once processing is complete, users can retrieve a list of detected highlight types and their corresponding timestamps.
    - The API returns a structured JSON response containing job metadata (job ID, status, highlight types, and timestamps).
    - Create a working AI model for highlight detection.

    - The highlight detection model completes inference faster than the actual length of the video to support scalability.
- Stretch Goals
    - Add support for detecting additional highlight types (e.g., free kicks, corners, penalty kicks).
    - Extend model functionality to support other sports (e.g., basketball, rugby) by adapting training data and inference logic.
    - Package the system as a Docker container for consistent deployment across different environments.

III. Non-Functional Requirements

The non-functional requirements define qualities and conditions that support the performance, usability, and maintainability of our product, even though they do not directly affect its core functionality. These requirements help ensure that the system operates efficiently, can be easily used and maintained, and meets the expectations of our stakeholders beyond just delivering technical features. While these factors might not prevent the system from running, neglecting them could lead to user dissatisfaction or long-term maintenance issues. They play a vital role in the overall success of our minimum viable product (MVP) and in meeting the client's expectations for usability, scalability, and adaptability.

- The inference model must be able to handle different camera angles for the same sport to ensure consistent highlight detection across varied footage.
    - Our model should work across footage from multiple camera perspectives for example, sideline, aerial, or broadcast views. This ensures users can upload a wide range of videos without needing standardized filming setups.
- Project management tools must support frequent client feedback and issue tracking.
    - We use Linear for structured task and milestone tracking, and Slack for real-time discussion. This keeps the client (ClubCast) informed, allows us to receive feedback quickly, and ensures transparency throughout the development process.
- Make model training cost-effective.
    - Training AI models, especially with video data, can be computationally expensive. To counteract this we train locally and use our own free software.
- Train a model - locally or on Roboflow.
    - The model should be trained using either local compute or cloud environments like Roboflow, depending on which environment offers the best balance of control, cost, and ease of use during development.

IV. Risks

Risk	Solution	Likelihood	Impact
Not having enough testing points	Creating as many tests as we see fit. Make sure it is clipping the correct times/length and not making too many clips that are unrelated.	Likely / Very Likely	Moderate / Major
Not being able to understand how to use Roboflow	Spend time working with Roboflow and getting comfortable with the features it has to offer	Likely	Minor
Not being able to train models properly	If this occurs we will train models locally potentially using the school GPUs that are provided.	Unlikely	Major
Not having enough data or enough clean data	We would have to find data elsewhere and potentially clean/label it to train our model on.	Likely	Moderate

Table 2: Risk Evaluation Table

The risks are crucial information that we identify as possible issues and prepare solutions for before they occur. This helps us avoid delays and overcome hurdles more efficiently. Risks can range from catastrophic to minor but still warrant discussion. We outline these risks in Table 2, covering issues in training the model and potential problems testing various aspects of our codebase. Most of our risks turn out to be less likely than we initially expect, so the overall impact remains manageable.

V. Definition of Done

Our project is considered complete when we have a functioning REST API that allows a user to upload a sports video, processes the video using a trained computer vision model, and returns a structured JSON file containing the job ID, sport type, highlight types, and their corresponding start and end timestamps. The system also includes the ability to check the status of a processing job, with clear states such as awaiting, running, failed, or successful. For our minimum viable product, the model is trained to detect goals in soccer matches and completes inference faster than the length of the video. We test our model using an 80-20 train-test split to validate its performance on previously unseen footage and ensure its outputs align with our client's expectations. The system is robust enough to handle different video angles and runs cost-effectively, using either Roboflow or local training depending on efficiency. ClubCast is the primary user and maintainer of the system, and we ensure they have the necessary documentation to operate it independently.

In addition to the MVP, our stretch goals include expanding the system's capabilities to detect a wider range of soccer highlights such as free kicks, assists, and set pieces. We also aim to adapt the model to work with 1-2 additional sports, like basketball or rugby, and to package the full API into a Docker container for easy deployment across different environments. These goals are pursued as time allows, following the successful implementation and validation of our core functionality. The project is scheduled for delivery by approximately June 11th to allow time for final testing, review, and presentation.

VI. System Architecture

Our overall system architecture is quite straightforward, as depicted in Figure 1. It begins with the user sending a video input via a Command Line Interface (CLI) to our REST API, which, in turn, runs the footage through some preprocessing steps to prepare it for our trained model [2]. In this preprocessing step, the original video is converted to eight frames per second using FFmpeg. This new video is then passed to our model for inference, which returns a list of frames where events are detected. A final post processing step is applied, getting rid of overlapping detections, and converting the unique detection events into time stamps. The list of timestamps is then returned to the API. Finally, the API takes that list of timestamps and relays it to the user via JSON file.

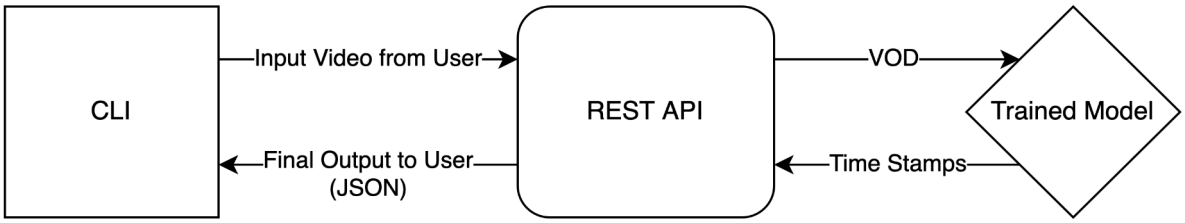


Figure 1: Pipeline Architecture Diagram

While this description allows for an overarching understanding of our project architecture, it is important to go a bit deeper into certain parts in order to fully convey its structure and functionality. Beginning with the custom trained model as seen in Figure 2, we use Roboflow, a computer vision software, to annotate raw training data sourced from full game footage. After preparing the dataset, we export and use it to train a custom model based on the pretrained 3D ResNet-18 model, a 3D convolutional neural network that extends the standard ResNet-18 to better handle event detection in video. It was trained with an 80/20 training validation split, using 4 second clips at 8 frames per second, roughly 50 positives (resets at mid), 50 negatives (open play - not resets at mid), and 50 hard negatives (play that resembles resets at mid, but is not). The Adam optimizer was utilized during training, as well as cross-entropy loss over 10 epochs. The performance of the model was tracked across each epoch, saving the best performing version. The actual training loop shuffles batches and runs on GPU when available, in an attempt to ensure efficient learning.

Figure 2: Model Training Workflow

Now that these models are trained, they are able to process video input for inference, as seen in Figure 3. During preprocessing, FFmpeg is used to cut the video down to 8 fps. This not only cuts down on the number of frames inference needs to be run on, but it is also the format that our model expects. The preprocessing function then resizes, normalizes, and groups the frames into batches of 32. At this point, the data passes onto our model for inference. During this inference step, our model utilizes a sliding window, with a 50% overlap from batch to batch. The model evaluates each window for a detection of a reset-at-mid event. A confidence score is calculated for each frame, and ones exceeding the minimum threshold are marked as containing a reset. In post processing, detections that are deemed too close to one another are filtered out and condensed into one detection event. These unique detections are converted into estimated timestamps of goal events by offsetting them by a fixed number of seconds. A list of estimated goal event times is then finally passed off to the API.

Figure 3: Model Inference

How our product interfaces with the API and the functionalities it provides are important parts of our system architecture that warrant further examination. These functionalities are on display in Figure 4. Before submitting a job, a user may want to know the capabilities and limitations of our product. They can find out by making a simple API request, which returns a list of supported sports as well as the specific highlights the product is designed to detect. If this list fits the needs of the user, they may then submit a video for a job. This is again achieved through a simple API request in which the user inputs the video they wish to be processed, and the API returns the job id. Using this job id, the user can request the job's status and results from the API. The results request returns a JSON file containing the timestamps of highlights if the job was successful, or an error if the job either failed or is not yet complete. As for the job's status, the API returns one of the four states a job can be in: Awaiting, Running, Successful, or Failed.

To implement this functionality, we used FastAPI, a modern web framework for building APIs with Python. FastAPI was chosen for its speed, intuitive design, and great support for asynchronous operations, features that are especially beneficial for handling video processing jobs, which can be time-consuming [3].

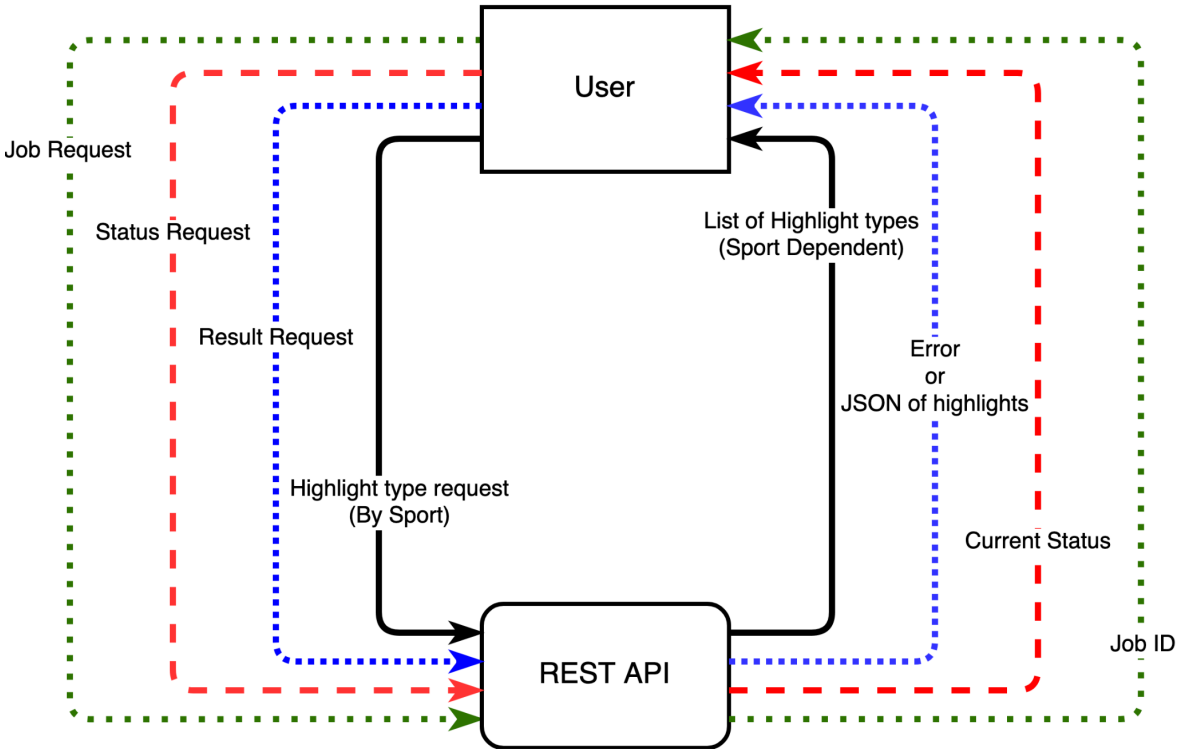


Figure 4: State Diagram Request Output

CLI User Input	Parameters	Output
upload <filepath> <sport>	Path to file of video, Sport type	Job ID
status <job_id>	Job ID	Current Status
result <job_id>	Job ID	Error, Job still processing, or formatted JSON file
highlights <sport>	Sport type	List of highlight type (sport dependent)

Table 3: API Input/Output

The process for updating a job's current status is integral to providing an intuitive and useful product. This process is laid out in Figure 5. It begins when a job is submitted, if the video fails to load in properly, the job status is updated to Failed. Otherwise, the job is placed in a queue, and its status is updated to Awaiting. Once the job is popped out of the queue and sent to the model for event detection, its status is updated to Running. If there is an issue while the model is processing the video, an error may be thrown, resulting in the job's status being updated to Failed. If not, and the model successfully processes the video, the job's status is updated to Successful.

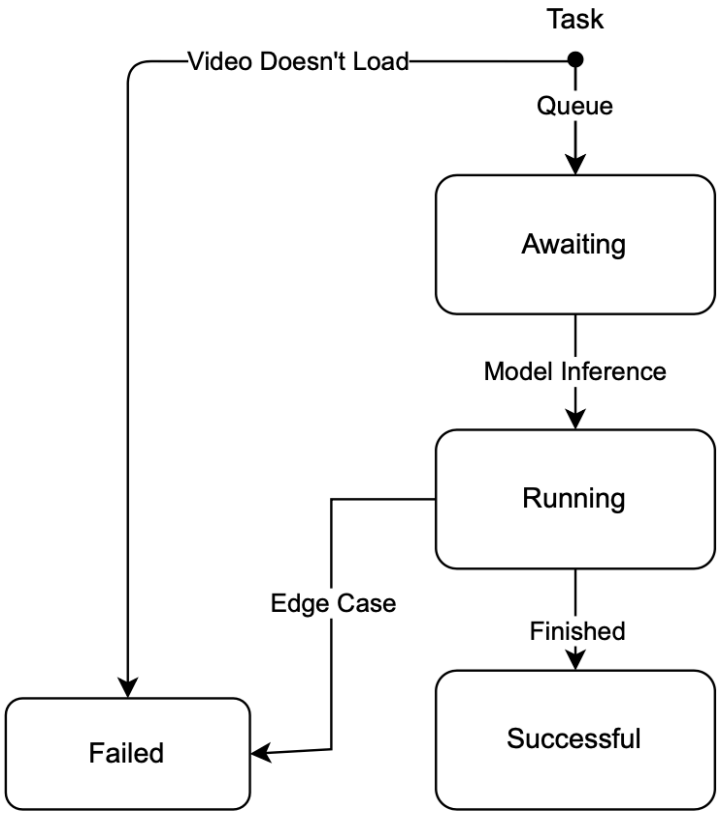


Figure 5: State Diagram for Status State Update

VII. Software Test and Quality

1. **Different API Requests Tests**

Purpose of test:	Ensure all API requests functionality is working properly and the user can make every type of request.
Description of test:	Set up requests commands through the API and ensure the responses are correct for each type of request.
Tools required:	FastAPI (Python library)
Threshold:	100% endpoint reliability with appropriate error handling.
Edge Cases:	File does not exist; Error in uploading; Error processing; Processing not complete;
Results of testing:	All API requests were successfully tested using FastAPI, with each endpoint returning correct and expected responses. The system met the 100% accuracy threshold and handled all edge cases effectively, including missing files, upload errors, processing failures, and incomplete jobs. The API consistently provided appropriate error messages and status updates, confirming reliable and robust functionality across all request types.

2. **Accuracy of Model**

Purpose of test:	Ensure the model is picking up a majority of the correct highlights.
Description of test:	Test the accuracy of the model.
Tools required:	Evaluation functions included in libraries like pandas, NumPy, and scikit-learn.
Threshold:	Accurately predicts over 50% of the highlights. Faster than the length of the video.
Edge Cases:	None
Results of tests:	We found the accuracy of our model to be around 80%, completing inference in about 10% of the original video time. For example, given a 60 minute video with 10 reset-at-mid events in it, our model correctly identifies around 8 events in 6 minutes.

3. **Highlight Type Distinction Accuracy**

Purpose of test:	Ensure the model can differentiate highlights to a certain accuracy.
Description of test:	Test the accuracy of the model regarding the labeling the highlights with the correct type
Tools required:	Evaluation functions included in libraries like pandas, NumPy, and sklearn.
Threshold:	over 50%
Edge Cases:	None
Results of tests:	We only have functionality for detecting one highlight type (soccer goal) at this time.

4. **Sport Type Request (Stretch)**

Purpose of test:	Ensure when the user specifies different sports it will change to the corresponding sport model.
Description of test:	Test the API to make sure when the input has a sport name it will switch to the model with that corresponding sport.
Tools required:	FastAPI
Threshold:	100% endpoint reliability with appropriate error handling.
Edge Cases:	If the user enters a sport that is not available.
Results of tests:	The API successfully switched to the appropriate model when a sport type was specified, demonstrating correct functionality. Although only two theoretical sport

types were tested, the system handled them as expected, achieving 100% accuracy. It also managed the edge case of unsupported sports by returning a clear error message. This confirms the API's readiness for future expansion to additional sports.

5. **Load Testing**

Purpose of test:	Ensure that when there are multiple jobs the program can handle it.
Description of test:	Test the API's ability to queue and process multiple jobs sequentially without failure.
Tools required:	FastAPI
Threshold:	100% endpoint reliability with appropriate error handling.
Edge Cases:	If a job fails and then another job is in the queue.
Results of tests:	The API performed well under load, successfully queuing multiple jobs and processing them sequentially. When multiple requests were submitted, the system marked jobs as "awaiting" until previous jobs were completed, meeting the 100% accuracy threshold. Edge cases, such as a job failure followed by a queued job, were handled correctly without disrupting the queue. This confirms the API's reliability in managing concurrent processing scenarios.

VIII. Project Ethical Considerations

Our project uses computer vision to detect highlights in club and intramural sports, aiming to automate the process of identifying key moments in games. However, this approach raises several ethical concerns. One major issue is the potential loss of nuance and subjectivity in highlight selection; what makes a moment truly exciting or meaningful often involves human judgment, which our model may overlook. As a result, the generated videos may lack creativity and feel "soulless," missing elements like a long build-up to a goal or meaningful moments of sportsmanship. There's also a risk of contributing to the oversaturation of AI-generated content online and displacing human video editors, especially in professional contexts. Additionally, the model could unintentionally highlight negative moments, such as a goalkeeper's repeated mistakes, leading to unfair embarrassment or criticism.

To address these concerns, we might explore several approaches in future development. These could include supporting human oversight in the final editing process, adding filters or customizable parameters to guide the model's focus, and building feedback mechanisms to fine-tune results based on user input. Our tool is primarily aimed at club and intramural teams, where highlight coverage is often minimal or nonexistent, so it has the potential to increase access rather than replace existing media workflows. We also recognize the importance of fairness and bias mitigation, and would consider strategies to ensure that the model treats different types of plays and players equitably as we continue refining our dataset and design choices.

IX. Project Completion Status

The project successfully implemented a comprehensive API service with REST endpoints that allow users to submit a video, check the status of a processing job, view available highlight types, and request specific highlight timestamps. The API returns structured job metadata including job ID, job status, detected highlight times, and highlight types, formatted in JSON upon request.

We also created an AI model used for the highlight detection of goals in soccer matches. This allows the system to generate goal-based highlight segments automatically, from only raw game footage. This model is able to reliably identify reset-at-mid events, as a way to indirectly locate goals, with an accuracy of around 80%. In addition to this, the model is able to run inference far faster than the length of the actual video, roughly by a factor of 10. At this moment, goals are the only highlight type the model is able to successfully detect and extract. That being said, this proves a strong "proof of concept", opening the doors for more variation in

highlight detection down the line. One of the biggest limitations of the model is its reduced accuracy when processing night games. This is the result of the absence of low lighting matches in the training data. While there are still improvements to be made, overall, the system functions in accordance with our MVP and demonstrates practical viability. It provides a more efficient method of finding goals in full game footage, and is a solid foundation to build upon for broader highlight detection.

All core API features are fully implemented and functioning as intended. Some stretch functionality, such as theoretical support for switching models by sport type, was tested at a basic level. However, two stretch goals remain incomplete: applying the AI model to additional sports and packaging the API as a Docker image. These are documented as opportunities for future improvement.

X. Future Work

There are several directions we would explore to expand and improve the system. A major opportunity is extending the AI model to support additional sports beyond soccer. This would involve collecting and labeling sport-specific data, and learning how to adapt our models to the unique flow and structure of each game. Each sport may require different input features, highlight definitions, and modeling strategies, which would push us to explore new tools and machine learning techniques.

Due to the late stage completion of a working model, there are still quite a few obvious actions to be taken to help improve the overall viability of our product. First and foremost, increasing the amount of training data. The current model was trained using less than 200 events which is nowhere near the ideal amount of data needed to achieve optimal accuracy and performance. In addition to increasing the amount of data for training, it is also prudent to increase the variation. More specifically, night games to increase the model's accuracy in low light settings. Moving past this model in particular, it is also entirely plausible to extrapolate into detecting other pattern based highlights. Given the appropriate data, the model architecture used can easily be translated to detect events like corners, freekicks, or any other play that follows a constant pattern. This flexibility provides a clear path for the potential expansion of the system's overall capabilities.

Improving the API is another important area for future work. One priority would be introducing user authentication and permissions so that job statuses and results are only visible to the appropriate users. This would involve creating user IDs, securing endpoints, and possibly managing role-based access. We would also like to add more API functionality, such as the ability to track model performance metrics, upload metadata, or even allow users to manually flag or edit highlights.

To make the system more user-friendly, we also see value in developing a graphical user interface (GUI). A web-based frontend would allow non-technical users, such as coaches or players, to interact with the tool more easily by uploading videos, viewing highlights, and tracking processing status without needing to work through the command line.

Finally, packaging the system with Docker and other deployment tools remains a strong goal for future development. Learning how to deploy software across different environments would make the system easier to scale, share, and maintain long-term.

XI. Lessons Learned

Throughout the course of this project, we gained valuable experience in developing end-to-end AI-powered applications, from building and deploying RESTful APIs to integrating computer vision models for real-time video analysis. One of the most important lessons we learned was the importance of designing modular, scalable systems that support asynchronous processing and can adapt to future changes. We also came to appreciate how critical model performance and inference speed are when working with video data, especially to ensure responsiveness in a production setting. Balancing rigorous software engineering with the iterative nature of AI development proved to be a consistent challenge, particularly as we transitioned between experimentation and deployment. This highlighted the importance of planning for deployment early in the process, aspects like containerization, scalability, and data access had a much larger impact than we expected.

Equally important were the lessons we learned through the struggles. We had to work with tools and platforms we had never used before, such as Roboflow, and quickly adapt to unfamiliar workflows. We learned the value of tracking our progress clearly, both for efficiency and sanity, especially when hitting roadblocks. One of our biggest challenges was getting a highlight detection model to work. After spending many hours researching and trying different model architectures, we realized that getting it "right" was more difficult than expected. We began by exploring object detection models like YOLOv8, which creates bounding boxes of the coordinates of objects in a frame, and training them by annotating hundreds of frames of footage with Roboflow. From there we created the bounding boxes to train the model, extracting features like ball-to-goal distance and player proximity, and used those features to classify events like goals or set plays based on short frame sequences. This approach ultimately failed to deliver meaningful results. We then tried using the object detection model directly to detect moments like midfield resets based on object clustering. Again, this didn't work and highlighted the mismatch between model choice and data complexity.

Still, those failures taught us how to work around data limitations and come up with creative solutions. We eventually circled back to one of the sources that inspired our early direction, the SoccerNet framework, and tried to follow its pipeline more closely [4]. We spent hours labeling hundreds of events across games, hoping this would help us get a more structured training process in place. Unfortunately, due to some missteps in how we followed the pipeline, we were unable to train a working model in time. That experience helped us realize that we could have benefited from taking a bit more time early on to deeply explore which paths were most promising, rather than spending weeks troubleshooting approaches that were unlikely to work with our resources.

In the end, this project taught us the importance of technical adaptability, strategic decision-making under constraints, and resilience in the face of repeated setbacks. Every failed model or false start helped us sharpen our approach, and we walk away from this experience better prepared to tackle future challenges in machine learning, computer vision, and software development.

XII. Acknowledgments

We would like to extend our sincere gratitude to our client, ClubCast, for providing us with a meaningful and engaging project that challenged us to apply both technical and problem-solving skills in a real-world context. Their clear vision and continued support were instrumental in shaping the direction of our work. We are especially grateful for their flexibility and constructive feedback throughout the development process, which helped us refine our goals, troubleshoot issues, and stay focused on creating a solution that aligned with real user needs. This collaboration gave us valuable experience working with a client in a professional setting and helped us grow both as engineers and as communicators. They also helped us understand the importance of thorough documentation and consistently tracking our progress, both of which proved essential in managing a complex, evolving project.

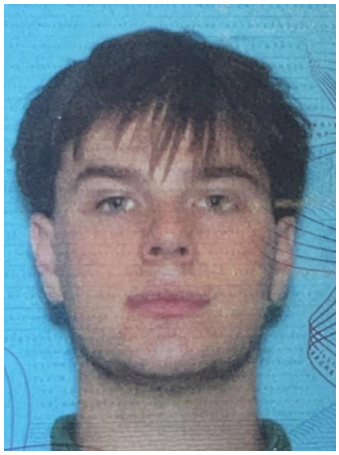
We also want to express our gratitude to our technical advisor, Jeff Paone, for his valuable support, guidance, and insight throughout the project. His expertise helped us stay on track and continuously improve the quality of our work. More importantly, Jeff consistently challenged us with thoughtful, critical questions that pushed us to think deeper and arrive at our own solutions. Rather than simply providing answers, he encouraged us to explore alternatives, consider edge cases, and develop a more rigorous understanding of the technical decisions we were making. That approach helped us grow not only as developers, but also as problem-solvers.

Lastly, we would like to thank Kathleen Kelly, who leads and coordinates the entire field session program. Her dedication to organizing these projects and connecting students with real-world companies made this experience possible. We're especially grateful for the time and effort she puts into managing logistics, guiding the structure of the course, and supporting each team throughout the process. Thanks to her work, we were able to gain hands-on experience not only working with a client, but also learning how to navigate agile development, manage sprints, and deliver functional software under real-world conditions.

XIII. Team Profile

Ethan Kerstiens
Computer Science - Computer Engineering
Arvada, Colorado
Automotive Detailing and Real Estate Sign Runner
I enjoy watching Soccer, Basketball and Football. I love the outdoors.

Daniel Longtine
Computer Science - Data Science
Westminster, Colorado
Work experience ranges from sales to freight operations at various companies
Interests include climbing, AI research, soccer, and other sports.



Thomas Glenzinski
Computer Science
Lombard, Illinois
Manual Labor
Enjoy cooking and intramural sports

Samuel Wasserman
Computer Science - Computer Engineering
Overland Park, Kansas
Experience in digital systems, search engine optimization, and web design.
Enjoys soccer, skiing, hiking, and video production.

References

[1] "What is a container?," Docker Documentation, 2024. [Online]. Available: https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-a-container/

[2] L. Gupta, "What is REST – Learn to create timeless REST APIs," Restfulapi.net, Jun. 13, 2019. [Online]. Available: https://restfulapi.net/

[3] FastAPI, "FastAPI," fastapi.tiangolo.com, 2023. [Online]. Available: https://fastapi.tiangolo.com/

[4] A. Deliège et al., SoccerNet-v2: A Dataset and Benchmarks for Holistic Understanding of Broadcast Soccer Videos. 2021. [Online]. Available: https://arxiv.org/abs/2011.13367

Appendix A – Key Terms

Term	Definition
JSON file	*JSON stands for JavaScript Object Notation and it is just a method of formatting the text in the file.*
Rest API	*An API is an Application Program Interface which means that it allows two programs to interact with each other. REST API's use a specific format protocol such as HTTP for web interfaces.*
Minimum Viable Product (MVP)	*A Minimum Viable Product (MVP) is a stripped-down version of a product or feature that's released to early users with the goal of gathering feedback and validating its core functionality before investing heavily in further development.*
Docker	*Docker is a platform that allows developers to build, run, and manage applications within lightweight, portable containers.*
Command Line Interface (CLI)	*A Command Line Interface is a text-based interface that allows users to interact with a computer by typing commands. Usually simplifies requests.*
Training	*Using data and optimization functions to update the weights and predictive power of a machine learning model.*
Inference	*Using a pretrained model and its trained weights for prediction. In our case, we are using it to predict whether a video is a certain class of event in videos.*
Reset-at-mid	*A soccer event where play restarts from the center circle at midfield, typically occurring after a goal has been scored or at half time. In this project, detecting reset-at-mid events serves as an indirect but reliable method for identifying goals, since every goal in soccer is followed by a kickoff from the center of the field.*

Appendix B – Tables and Figures