A Live Foosball Commentary System Using Computer Vision, Markov Models, and Large Language Models

Crasemann, Christoph Daum, Julius Horlbeck, Carolina
Kara, Atakan Krüger, Joost Kujath, Falk Mayer, Luca
Nissen, Mads Schirp, Jonathan Wieck, Philipp

March 31, 2025

1 Introduction

Sports commentary is an essential part of modern sports entertainment, enhancing audience engagement by providing real-time insights, expert analysis, and dynamic storytelling. In popular sports such as football or basketball, advanced analytics enable automated, near-instant commentary based on in-game data. These systems rely on sophisticated tracking technologies and machine learning models to extract relevant information from ongoing matches. While mainstream sports benefit from these advancements, smaller-scale games like foosball have received significantly less attention. Despite the existence of game-tracking systems for foosball, to the best of our knowledge, no previous work has attempted to generate autonomous commentary based on real-time game events.

This project proposes a novel approach to bridging this gap by developing an autonomous foosball commentary system. The proposed workflow integrates multiple technologies to process game data and generate meaningful commentary. At its core, a digital twin of the foosball game is created using computer vision techniques, capturing the game's state in real-time. To extract structured knowledge from the raw digital twin data, a Markov model is applied, identifying key patterns and events that define the game's dynamics. This extracted knowledge is then processed by a large language model (LLM), which translates it into natural-language commentary, making the game more engaging and accessible to spectators.

To support the development and deployment of this system, several additional components were implemented. A customized video player was developed to facilitate game analysis and testing, providing a reliable tool for reviewing and optimizing the tracking pipeline. The entire project is deployed as a Docker-based system, ensuring portability and simplifying the deployment process across different environments. Furthermore, a dedicated frontend was designed to display key performance indicators (KPIs) that provide deeper insights into the game. To enhance the user experience, the frontend also leverages the browser's internal text-to-speech functionality, allowing users to receive real-time auditory commentary as the game unfolds.

A major focus of this project is ensuring that evaluations and commentary generation occur with minimal latency. The system processes game data and produces output almost instantaneously, providing a near-real-time experience for users. Additionally, novel computer vision techniques were integrated to improve data extraction, including leveraging a U-Net model for precise foosball player and ball detection. These contributions not only enable autonomous foosball commentary but also introduce innovative approaches for real-time sports analytics, laying the foundation for future advancements in AI-driven sports broadcasting.

2 Related Work

2.1 Computer Vision

Two principal works stand out for applying computer vision (CV) principles to the rapid, confined environment of foosball: the Foosballytics project by Tooploox [Too] and Bambach et al.'s Real-Time Foosball Game State Tracking [BL]. Although each addresses different facets of the problem, both emphasize the importance of robust, high-speed tracking methods and the handling of frequent occlusions.

Foosballytics offers a concept-centered approach that leverages color-based cues to differentiate critical elements in the foosball environment—most notably the ball, table surface, and goal zones. The underlying idea is that maintaining a dedicated color space allows for consistent segmentation under variable lighting conditions, a notable hurdle in real-world deployment. Beyond detecting the ball's rapid movement, the project aims to seamlessly integrate scoring logic and replays into a single real-time interface. This underscores a larger vision of improving viewer engagement by immediately highlighting pivotal in-game moments—such as goals or near misses—and archiving them for later review. Consequently, Foosballytics illustrates how even simple, well-chosen techniques (e.g., color thresholding) can be highly effective if carefully aligned with the sport's unique speed and spatial constraints.
In contrast, Bambach et al. adopt a more expansive view by conceptualizing the foosball table as a system of both static and dynamic entities, each of which must be tracked simultaneously. From the outset, their framework envisions a "calibrated" foosball table, ensuring that the physical layout—rod positions, player figures, and boundaries—is fully registered before any real-time analysis begins. Once this reference is established, the system monitors ongoing changes, including the rod orientations that drive player positions and the ball's trajectory. The team proposes a predictive layer aimed at anticipating ball motion and partially offsetting motion blur or sudden occlusions. Conceptually, this allows the pipeline to maintain a cohesive picture of the game state even in scenarios where the ball moves too fast or briefly disappears from view. By capturing rod angles in conjunction with ball location, the approach provides a richer representation of each in-progress play.

2.2 Markov Model

Although there is no apparent research on table foosball in combination with Markov chains yet, the general use of Markov chains has found application in other sports-related games.
In the research by Damour et al. [DL15], Markov models have been used to model set pieces in soccer such as throw-ins, free kicks, goal kicks and corners. They employ regression analysis to estimate the transition probabilities between different states and identify factors influencing these transitions. This is due to their motivation, which was mainly risk management in betting. In addition, the resolution of their game model is rather low as they model the soccer field as only three slices.
A second paper by Campos-Chavez et al. [CCTDH22] uses Markov models for predictive modeling in ice hockey. The motivation here is to determine the probability that a team will win a game at a given time in the game and in a particular state of the game. In order to achieve that they didn't model the games as it is but only the parts they think are significant for winning the game. Therefore, their models focus on the shots on the goal transition rates and manpower transition rates.
In the research by Rothe et al. [RL22] they use Markov models to describe tennis game-plays and the relationship between sports behavior and outcomes. They state that Markov chains can accumulate information about the relevance of different actions.
While most Markov implementations in sports do not meet our motivation and therefore cannot be used for our use case, Rothe et al. [RL22] describe an interesting side product. That is, the description of a game through Markov chains, and especially the determination of relevance of different moves in a game.

2.3 Large Language Model

Recent research has demonstrated the growing potential of large language models (LLMs) in the field of automated commentary generation and event prediction. Cook and Karaku [CK24] introduced *LLM-Commentator*, a system that leverages fine-tuning strategies on open-source LLMs to generate real-time football match commentary. Their study highlights the feasibility of using consumer-grade hardware for training and deploying LLMs in live sports scenarios. This is particularly relevant for table foosball, a similarly fast-paced game, where lightweight and efficient commentary systems are required.
Expanding the modeling perspective, Mendes-Neves et al. [MNMMM24] treat soccer event sequences as a form of language and introduce a Large Event Model (LEM) inspired by LLM architectures. This approach enables accurate forecasting of match events by modeling them as tokens in a language sequence. The idea of applying natural language processing techniques to structured sports data provides valuable insights for designing commentary systems capable of understanding and describing the flow of foosball games.

While not directly focused on sports, Geng et al. [GWD+24] explore how LLMs can generate comments with varying developer intents using in-context learning. Their findings demonstrate that LLMs can adapt their output based on prompting and context, which has implications for generating customized commentary styles in foosball—ranging from analytical breakdowns to humorous or emotionally engaging narrations.

3 Approach

3.1 Architecture

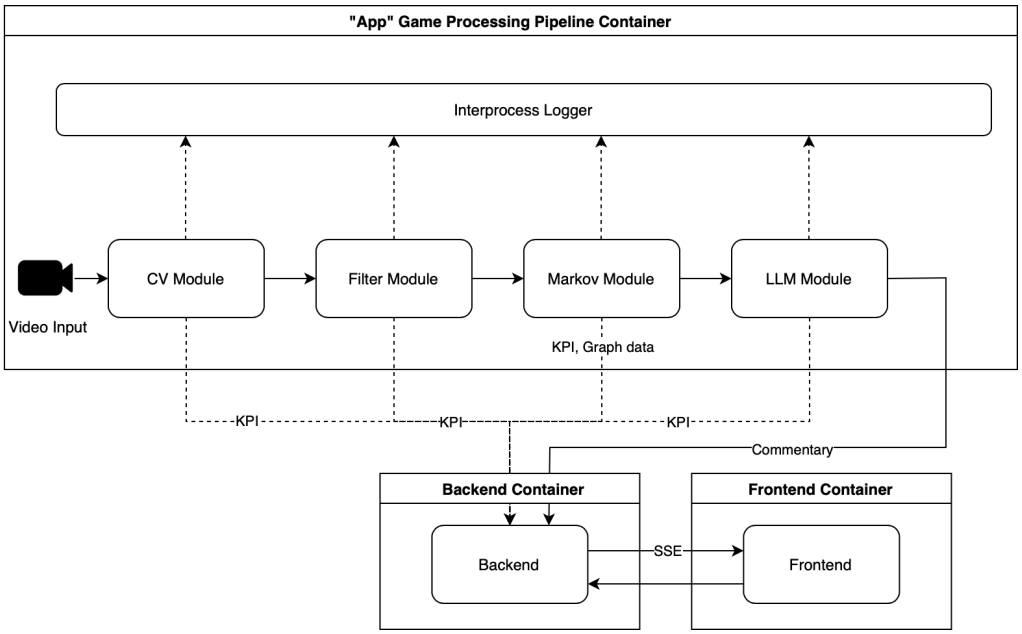


Figure 1: Architecture overview of the project, illustrating the division into multiple Docker containers. The primary application container hosts a full Python code base, where individual components communicate using native Python inter-process queues. Each component runs in its own dedicated process, ensuring efficient parallel execution and modularization of the system.

The project runs in a Docker-based environment [Inc25a] consisting of three containers as shown in figure 1. The app container runs the game processing pipeline, referred to as pipeline, consisting of several modules. The frontend container provides a live dashboard displaying key performance indicators (KPIs), graph visualization and commentary output with text-to-speech. The backend container exposes an API interface that aggregates KPIs, graph data, and commentary from the pipeline and communicates them to the frontend.
Containers are managed by Docker Compose [Inc25b], which allows us to start and coordinate all containers simultaneously. For the machine learning aspects, such as player detection in computer vision and Llama (Meta's large language model) [MP25], we leverage the NVIDIA CUDA Toolkit [Cor25] to optimize performance by utilizing GPU acceleration.
The pipeline, as seen in the top container in figure 1, is structured into four modules. The computer vision module processes video frames to create a digital twin of the foosball game, capturing every movement in near real-time. The filter module filters and de-noises data coming from the previous module. The Markov module uses a Markov model to extract and accumulate knowledge about the game, providing structured insights. Data is aggregated and converted into foosball moves. The large language model generates comments based on extracted moves.
To enable seamless communication between these modules, we utilize Python's multiprocessing queues, with each module running as an independent process, leading to efficient data exchange and parallel execution.
This pipeline-based approach ensures an efficient, scalable and high-performance solution for near real-time foosball game analysis and commentary generation.

3.2 Computer Vision

The main task of the first module of the project's pipeline is to extract the game events happening in the video, from file or live-camera, using a structured Computer Vision (CV) Pipeline to detect players, track the ball, identify key game events like goals and throw-ins and translate this gamestate into 'touches' that can be processed by the next module.

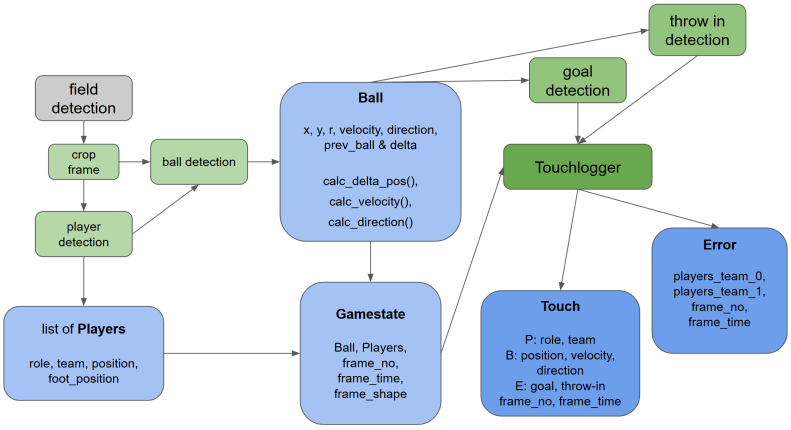


Figure 2: CV Pipeline Overview

As shown in the Figure 2, the pipeline starts with field detection, where the field's boundaries are identified, and the frame is cropped accordingly. Then, player detection extracts player positions, roles, and teams, forming a list of players with their attributes, including foot positions. The found rod positions and player mask calculated for player detection are also used to better detect the ball position underneath them. Overall the ball detection tracks the ball's position, velocity, and direction. The Ball object continuously updates its movement data referencing previous frames for calculating the changes.
All detected entities—players and the ball—are combined into the gamestate, which maintains essential match information such as frame number, frame time, and frame shape and represents the current frame.
The touchlogger module monitors ball interactions and game events. It determines whether a touch has occurred by analyzing player-ball interactions and identifying key events like goals and throw-ins. Each recorded touch includes the player's role, team, ball position, velocity, and event type, along with the frame number and time.
Errors, such as incorrect player-team assignments, are logged separately in the Error module, storing details about affected teams and frames.
This efficient pipeline enables real-time analysis and post-match evaluations, making it a powerful tool for foosball analytics.

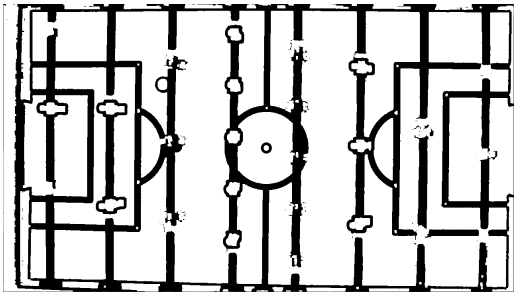
3.2.1 Field Detection

Field detection is a foundational step for tasks like player or ball tracking, as it isolates the playing area to reduce computational noise. Due to environmental complexity (e.g., lighting, camera angles), this system uses hard-coded field coordinates, manually set to predefined boundaries. While efficient, this approach lacks adaptability to new perspectives or venues, requiring recalibration. It remains a practical compromise, prioritizing speed and reliability over dynamic flexibility in real-time applications.

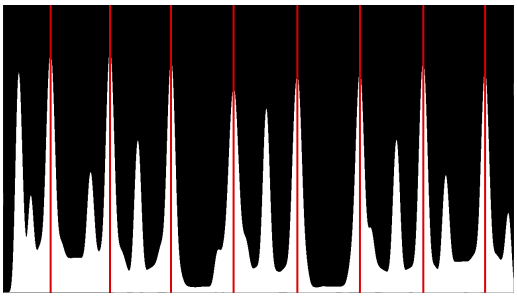
3.2.2 Player Detection

For the detection of foosball players, a multi-step approach is employed. The primary goal is to determine the positions of all players on each rod and assign them to their respective locations in a structured manner.
First, an approach based on primitive image processing operations is used to generate training data for training data generation. This approach has limitations with respect to low light, changes in lighting, and camera orientation during runtime, it also needs an initialization phase to get started. Therefore this approach is is only used to generate training data for the second approach which is based on a UNET. As the first approach has limitations with regard to low light, it is used to generate training data on a single video with good lighting conditions. The UNET is then used on

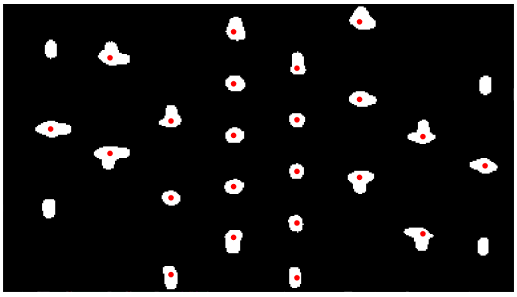
videos with different lighting conditions. Frames where the UNET output is suboptimal are then hand-labeled for model fine-tuning.



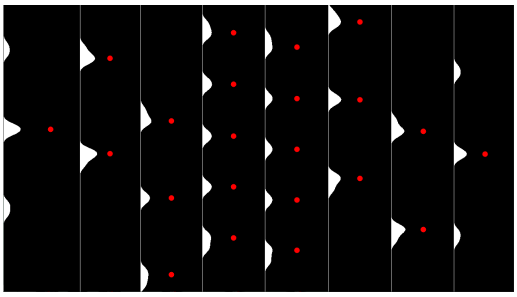
(a) Binary mask of the playing field used for rod detection. A gradient filter, thresholding, and morphological operations (dilation/erosion) are applied.



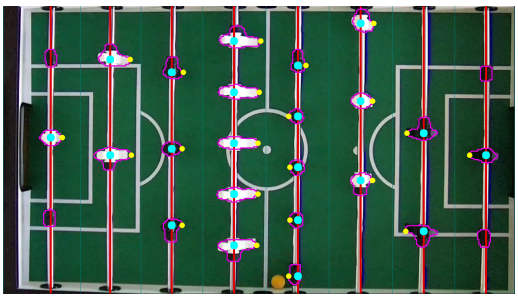
(b) Histogram of the field mask with detected rod positions marked in red.



(c) Output mask generated by the U-Net model, with detected player centers marked as red circles.



(d) Stacked histograms per rod, with player positions indicated by red circles.



(e) Final visualization overlay: rods (red lines), player centers (turquoise circles), foot approximations (yellow circles), and player contours (pink).

Figure 3: Visualization of the player and rod detection process. The pipeline consists of field segmentation, rod localization, and player position estimation, with the final frame displaying all extracted information.

Rod detection Both approaches require the positions of the rods. Rods are detected by first creating a gradient image. Then a threshold operation as well as dilate and erode operations are applied. This process creates an image where the rods are mainly black while the background is white, excluding the markings on the playing field. There are no markings that go from one side of the playing field to the other, whereas the rods completely cross the field. When counting the black pixels, which are predominantly rod pixels, the result is a histogram in which, after applying smoothing, the peaks are the rod positions as seen in 3a and 3b. Note that this process only works when the image of the playing field is close to normalized with no stark rotation.

Motion-Based Player Detection The first approach is based on movement. The difference between three succeeding images is used to detect players through their movements. areas where the difference is relatively large contain either a player or the ball. This information is used to create a long exposure image by only adding the sections between the players as well as keeping

track of how many images were added at the given position. This way a long exposure image is created, without any players visible. Once no gaps are visible in the long-exposure image, it can be used to detect the players in a single frame. By taking the difference between the long exposure and standard frames, only the players are visible.

UNET-Based Player Detection The runtime player detection is based on a UNET that is trained on 1200 images. 1050 images are from the first approach. These images were mainly generated automatically, but were cleaned by hand to remove noise and the ball, which could not be detected and removed by the first approach. Because the first approach only worked under specific lighting conditions, the 1050 images are from a single video. In addition, 50 images from videos with different lighting conditions were labeled by hand. The images were resized to 384 x 224 pixels, which is relatively close to the aspect ratio of the playing field. In addition, augmentations are applied to the dataset. The hand labeled images are added twice, once augmented, and once without augmentations. Training is split into two phases, once with the complete dataset and once only with hand-labeled images to improve the generalization of the model.
The model returns a segmentation mask that contains players and spacers as seen in 3c. To detect the exact positions of the players, the mask is split between the rods. The result is eight vertical images, each containing the player segments of one rod. For each of the eight images, we calculate per row how many pixels are white. When stacking these histograms, the result is 3d. The peaks of the smoothed histograms are the player positions on the respective rod. The first and last rods have a goal keeper in the center as well as spacers on the sides, so only the center peak is a player. Furthermore, we calculate which point of a player's outline is farthest from the respective rod. This point is expected to be the position of the players feet. The feet positions are used to improve the touch detection between the players and the ball. All extracted information is shown as an overlay in 3e.

3.2.3 Ball Detection

The ball detection in this system is conceived as a multi-stage process that starts with an approximate identification of the ball's location and progressively refines that position. Initially, a color-based method locates the approximate region where the ball is most likely to appear. Once this rough position is determined, a small region-of-interest (ROI) around the ball is extracted and then upscaled—an approach that boosts resolution and makes it easier to discern the ball from its surroundings.
Within this ROI, the method generates a binary mask and identifies the largest relevant contour. To handle irregularities or outliers in the contour, the system filters and approximates its shape, discarding points that deviate significantly from the average form. Using these refined points, multiple circle fits are attempted from fixed radii and different initial guesses, providing resilience if the ball is partially occluded or if lighting conditions cause subtle variations in the contour.
After circle fitting, the system evaluates each candidate against additional cues. The system first checks whether all fits converge to a single consensus; if so, it adopts that consensus. Otherwise, it looks for a candidate that lies within a detected player region and is also close to a rod. If that fails, it selects a candidate that is near a rod; and if that also fails, it selects a candidate that is close to a player figure; if none condition matches the ball will not be found. This ordered sequence ensures that the final ball position satisfies the most relevant physical and contextual constraints. The end result is a final circle center and radius that together pinpoint the ball's position at the original resolution. By chaining color-based initialization, precise contour analysis, multiple circle fits, and contextual checks, the approach ensures robust ball detection even when rapid table movements, lighting fluctuations, or partial occlusions complicate the scene.

3.2.4 Goal Detection

To identify when a goal has been scored, the system defines rectangular "goal zones" on both the left and right edges of the foosball table. Each zone is positioned so that any ball crossing its boundary is likely entering a goal area. While the ball remains within a zone, the system registers its presence. If the ball then disappears from tracking after being observed inside a goal zone—and remains missing for a certain number of frames—a goal is signaled.
A key aspect is that the horizontal width of these goal zones can dynamically adjust in response to the ball's velocity. When the ball is traveling quickly, the zone is scaled accordingly, providing a slightly larger detection area and making it more resilient to fast movements or minor tracking errors. If the ball's velocity is lower, the zone narrows to reduce false positives.

By combining the notion of "ball present, then abruptly absent" within a calibrated zone, this approach successfully flags real goals without extensive post-processing. It is also robust to momentary occlusions: the system only concludes a goal if the ball is definitively lost after having been inside the goal zone. This method thus balances sensitivity to genuine scoring events with the necessary safeguards against transient detection lapses.

3.2.5 Throw-In Detection

The system pinpoints narrow "throw-in zones" along the top and bottom edges of the foosball table—those regions where the ball is most likely to be reintroduced into play. Whenever the ball enters one of these zones, its position is tracked until it leaves again. Crucially, the zone boundaries and motion thresholds are chosen to capture the defining characteristics of a valid throw-in rather than routine ball movements.
Once the ball exits a throw-in zone, its trajectory is analyzed by comparing the earliest and latest recorded positions. If the resulting path aligns with a plausible upward or downward motion—specific to whether the ball entered from the bottom or top—then the event is flagged as a throw-in. This filtering mechanism helps avoid false detections triggered by near-horizontal movement. Moreover, a brief "grace window" accommodates temporary occlusions: if the ball disappears after having been in a throw-in zone but reappears soon afterward, the system continues its assessment without prematurely resetting. By limiting the detection area to these edge regions and focusing on logical angle thresholds, the approach efficiently isolates genuine throw-ins while remaining robust to short-lived visual obstructions.

3.2.6 Touch Detection

In the context of the computer vision (CV) component of this project, key game actions are identified based on a fundamental assumption: they occur whenever the ball changes speed or direction beyond a set threshold. This change is typically attributed to the ball making contact with a player, the wall, or other objects on the field and are categorized as a touch. Events such as goals and throw-ins are also processed as touches, ensuring that all significant actions are captured before being passed to the backend for further analysis.
To identify the cause of a touch, the positions of players relative to the ball are checked. Only players whose feet are within an acceptable range of the ball are considered. If a player's feet are determined to be too far from their rod position—indicating that they are positioned too high above the field—they are excluded to avoid false positives. Detection thresholds are dynamically adjusted based on the ball's speed, with the detection range being increased for faster movements. If no player is found within range, the wall is checked for potential impact. If neither a player nor the wall is identified as the cause, the event is discarded as a non-touch.
If the ball remains undetected for a period beyond the typical threshold—due to obstruction or interference—an error event is logged, and all detectable players around the ball are noted. This information is then passed to the backend to assist in diagnosing issues when the ball should be in play but cannot be tracked.
In summary, key events are detected based on changes in the ball's speed or direction, with accurate touch detection ensured through dynamic thresholds, player proximity checks, and error logging, even in the presence of obstructions.

3.3 Backend Knowledge Extraction

The general task of the pipeline's backend module is process the incoming touch data stream and capture the gameplay actions in a flexible data model that allows for multi-purpose data usage. Specifically, this model is built to support three main concepts: analysis, reconstruction and aggregation.
For the analysis part, the model records transitions between different gameplay states in a graph structure, tracking transition counts, probabilities and times. By training a knowledge capturing Markov graph model (see 3.3.2) on a multitude of real games, characteristics and patterns of average gameplay can be found. Additionally, during the live stream of a single game, the backend module tracks game-specific statistics, such as the score or ball possession.
The pre-trained knowledge model is then used to introduce a level of robustness to the live processing. For erroneous moments in the touch stream transmission, e.g. due to an obstructed camera view of the ball, the model is able to reconstruct missing gameplay sequences (see 3.3.3) once the ball is visible again, based on the error log and the learned patterns of the knowledge model.

As listed above, the third major concept supported by the model is aggregation. In order to allow commentating on more coarse events instead of singular touches, the model can group sequences of state transitions to detect different types of events in the gameplay (see 3.3.4), which are finally forwarded to the LLM module of the main pipeline.

3.3.1 Backend Pipeline

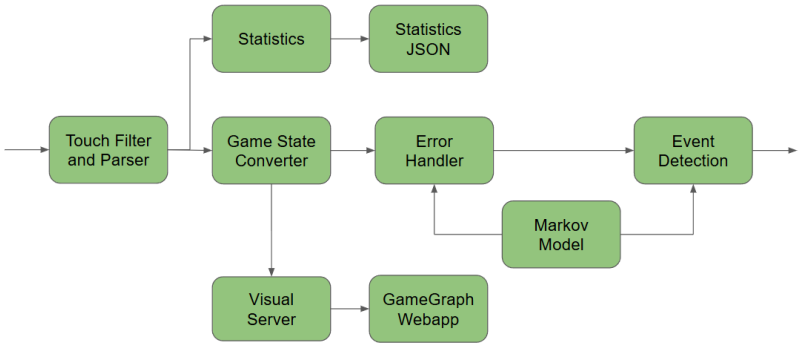


Figure 4: Backend Pipeline Overview

Figure 4 visualizes the general structure of the backend, covering both the small filter and the larger, main processing module.
In the filter module, rules are defined to limit the volume and velocity of the touch or error data stream. The general idea is to cut all messages that do not contain relevant information for the subsequent analysis. In particular, there are two major types of messages that are filtered: subsequent errors that happen so rapidly that they would not contain a sufficient amount of new information to be relevant to the reconstruction process, as well as consecutive touches from the same player in a short time frame. The latter rule is in place to focus on the most relevant touch of a player, the touch with which the ball is delivered to the next position on the foosball table, rather than analyzing multiple touches that do not advance the game in a meaningful way.
In a next step, the received touch data is forwarded to the statistics module, which tracks and updates different statistics for the current game, such as the score, ball possession or touches per player to be able to identify key figures for each team. The statistics are periodically updated in a persistent file, and always after a goal in order to expose it to the LLM module for the commentary.
Additionally, the touch data is converted into one state of the state model, which is described in more detail in 3.3.2. The state representation is then used to draw a visual representation of the gameplay flow in the frontend, as well as forwarded to the error handling and event recognition, which are also described in more detail below (3.3.3, 3.3.4).

3.3.2 State Model

The state model represents the backbone of our knowledge extraction system, translating raw touch data into a structured, probabilistic representation of the game. Each touch detected by the computer vision module carries essential information about the player, rod position, ball coordinates, direction, velocity, and timestamp. Our model systematically classifies these touches into discrete states, enabling us to capture the dynamic patterns of gameplay.
As illustrated in Figure 5, the playing field is segmented into multiple sectors. Each player rod is divided into equally-sized sectors corresponding to potential ball interaction zones. Similarly, the

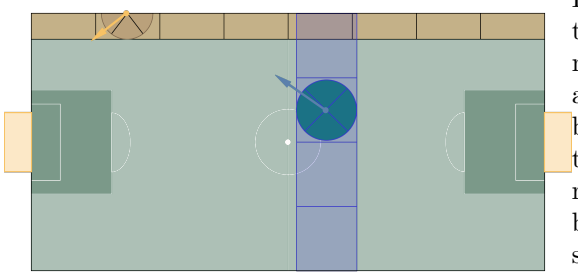


Figure 5: Example representation of 2 ball touches being classified by our proposed state model. The first touch is represented by the blue arrow. It's assigned to a sector along the rod by its coordinates and to a direction sector by the balls new direction. The second touch represented by the yellow arrow models the ball rebounding from the top wall. It's also given a wall sector as well as a direction sector.

walls are partitioned into sectors to account for rebounds. When a touch occurs, it is assigned to the appropriate sector on the basis of its spatial coordinates. Additionally, the ball's post-touch direction is classified into a direction sector, providing a comprehensive representation of the ball's trajectory.
The state model further enhances this spatial classification by incorporating velocity information. Ball speeds are categorized into discrete classes (e.g., slow, medium, fast), allowing a differentiation between gentle passes and powerful shots. The number of position and direction sectors, as well as velocity classes, can be configured during training, allowing for models with different levels of detail. Special game events like goals and throw-ins are represented as distinct states within the model. This approach allows modeling the complete flow of a foosball match, including both continuous play and restart scenarios.
This multidimensional classification creates a rich state space that can capture the nuances of different play styles and strategies. To transform this state classification into a knowledge model, a Markov process framework is employed. By analyzing thousands of touches across multiple games, the system learns the transition probabilities between different states, effectively capturing the statistical patterns of the game. These transition probabilities reveal common strategies, preferred passing routes, and effective shooting positions. The trained model not only provides valuable insight into gameplay patterns, but also serves as a foundation for error handling in subsequent pipeline stages.

3.3.3 Error Handling

As mentioned above, the system is required to be robust against errors in the ball detection, to ensure sensible commentary in all situations. In case of long durations where no gameplay could possibly be detected, e.g. if the capturing camera is heavily obstructed, it is of course not reasonable to invent entire game sequences. However, for short moments of missing ball detection, the model is expected to recover and reconstruct the missing gameplay sequences, in order to still detect potential events.
To meet this requirement, the error handling component of the backend system combines two sources of knowledge in case of error transmissions from the CV component, namely the contents of the errors and the knowledge Markov model. While no errors occur, the error handler constantly updates the latest recorded model state. When a sequence of error messages is transmitted, each error is logged, until the ball is detected again and the next transmitted touch initiates the reconstruction. As all touches, it gets converted into its corresponding state in the model, defining the destination state for the reconstruction. Using the previously updated last known ball state as the start state, the error handler then calculates the most probable gamestate sequence for the in-between time.
This is done by first pruning the search space by cutting all states that correspond to positions on the table that were visible during the time, which are retrievable from the error log. Afterwards, the pre-trained knowledge Markov model is used to determine probabilistic weights for all transmissions between the remaining potential states, which are then used to calculate the most probable path. Since the search space can become large depending on the error transmission duration and the size of undetected space on the foosball table, the reconstruction can of course not guarantee to be correct, but it does calculate the most probable gameplay sequence. However, states along the path are labeled with a decreasing confidence value based on the probabilities along the path, indicating the level of uncertainty. These confidence values are also forwarded to the LLM component in case events are detected in the reconstructed history.

3.3.4 Event Recognition

The core challenge in providing meaningful commentary is identifying significant gameplay events from the continuous stream of state transitions. To address this challenge, a dual-approach event recognition system was implemented that combines rule-based detection for well-defined events with a machine learning approach for more complex patterns.
For clearly defined events with distinctive state transition patterns, a straightforward rule-based approach was employed. This method efficiently identifies events such as goals, shots, dribblings, throw-ins, and blocked shots. Goals are detected when the ball transitions to a goal state, triggering immediate event notification with the scoring team and player information. Shots are recognized by analyzing the ball's velocity and trajectory, particularly when the ball moves from an offensive rod toward the opponent's goal with high velocity. Dribblings are conscecutive ball touches by the same figure. Throw-ins are identified by a throw-in state, providing information about which

team entered the ball into play. Blocked shots are detected when a ball moving toward a goal is suddenly redirected by an opposing player, indicating a defensive action. These rule-based events are processed immediately upon detection and forwarded to the LLM module for commentary generation, ensuring minimal latency for these critical gameplay moments.
To identify more complex events that are difficult to define through explicit rules, an autoencoder architecture was developed that learns to recognize patterns in state sequences. This approach operates on sequences of three consecutive states. These are extracted from the state transition stream and normalized to create consistent input vectors. Normalized sequences are passed through an autoencoder neural network that compresses state information into a dense embedding space. New state sequences are compared to pre-labeled exemplars in the embedding space, with similarity scores determining the closeness of a match.
This methodology currently allows for the detection of subtle and complex patterns such as wall passes and through passes. Wall passes are deliberate bounces off the side walls to bypass opposing players. Through passes are sequences of passes between players on the same team skipping entire rows of players of that team. The setup also allows for adding new labeled sequences after training to recognize other event types.
The integration of these two approaches creates a comprehensive event recognition system that can identify both fundamental and nuanced gameplay elements. As the system processes more games, the autoencoder's ability to recognize complex patterns improves, enabling increasingly sophisticated and context-aware commentary generation.

3.4 Large Language Model

The third component of the pipeline, hereafter referred to as the LLM-Pipeline, is responsible for generating commentary based on the data provided by the preceding pipeline component. This data includes details such as the event that occurred, the timestamp of the event, the involved players, a likelihood score, and a confidence score. These inputs are processed by the LLM-Component to generate structured prompts for the Large Language Model, Llama 3.2 - 3B - Instruct, which ultimately produces the desired commentary output in a reasonable time.

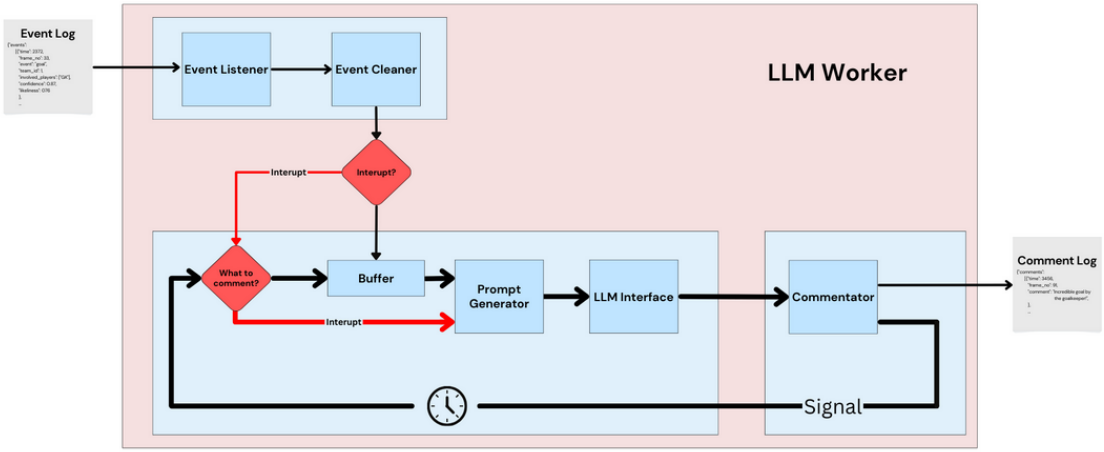


Figure 6: LLM Pipeline Overview

As illustrated in Figure 6, the pipeline comprises three primary threads: the *Listener Thread*, the *Commentator Thread*, and the *Speaker Thread*.
The *Listener Thread* is responsible for receiving event data from the preceding pipeline component and preprocessing it to ensure data quality before further processing. The *Commentator Thread* manages event storage and prompt generation. Events are stored in a buffer and assigned an interest score to prioritize them based on relevance. Additionally, this thread initializes and interfaces with the LLM, utilizing a prompt generator and an LLM interface module for processing. The *Speaker Thread* is responsible for formatting the generated commentary for output. It consists of a single module that interacts with the LLM interface and manages interrupt events by signaling back to the buffer when necessary. The implementation of efficient multithreading enables near real-time processing, ensuring that the generated commentary aligns accurately with live game events.

3.4.1 Event-Listener and Event-Cleaner

The first thread comprises two key components: the *Event Listener* and the *Event Cleaner*, which serve as the entry point of the pipeline. The process begins with the *Event Listener* module receiving event-log data from the backend component.
Once the event-log is received, the data is forwarded to the *Event Cleaner* module, which is responsible for preprocessing and evaluating the information before it is stored in the buffer. This module assigns priority levels to various event types and standardizes player position data to optimize the subsequent prompt generation for the LLM. Additionally, it ensures that unknown events and unrecognized players are filtered out, preventing potential disruptions within the pipeline.
Furthermore, the *Event Cleaner* identifies high-priority events, e.g. goals, that require immediate attention and triggers interrupts when such events are detected. Thereby, the seamless and efficient processing workflow can be guaranteed, while ensuring prioritization of processing extraordinary events.

3.4.2 Buffer

The *Buffer* module is essential for managing game events in real-time commentary within the table foosball system. It employs a dynamic scoring system to prioritize events based on factors such as priority, likeliness, confidence, and time decay, ensuring timely commentary on the most relevant and interesting events. The event management calculates interest scores, which are used to order events regarding their individual importance and automatically cleans up stale events. Additionally, it prevents buffer overflow by removing the least interesting events when capacity is reached. This system ensures that the commentary remains responsive and focused on the most engaging events, enhancing the overall commentary experience.

3.4.3 Interest Score

The *Interest Score* is an integral component of the buffer, designed to prioritize incoming events and identify the most significant ones for commentary. This score is computed for each initial event added to the buffer, leveraging all available event data to assess its relevance as accurately as possible.
A key aspect of this scoring mechanism is the incorporation of a time decay factor, which accounts for the diminishing relevance of an event over time. In this way, events are removed from the *Buffer* after a certain amount of time to ensure only relevant events remain. This guarantees that older events gradually lose priority, reducing the likelihood of generating outdated commentary. The *Interest Score* is calculated using the following formula:

$$\textbf{Interest Score} = \frac{1}{\text{priority}} \times (1 - \text{likeliness}) \times \text{confidence} \times 100 \times \text{Time-Decay}$$

In this formula, the *time decay* component is defined as:

$$\textbf{Time-Decay} = \frac{(Current\ Time - EventTime)}{Time\ Offset}$$

The *Time Offset* establishes the threshold for event relevance, while the *Current Time* serves as the reference point for updating the *Interest Score*. By refining the prioritization of events, the *Interest Score* enhances the effectiveness of the buffer, ensuring that only the most relevant and timely events are selected for commentary.

3.4.4 Interrupt Handling

The handling of high-priority events, as identified by the *Event Cleaner*, requires an efficient *interrupt management* process to ensure accurate and timely commentary generation. When an interrupt is detected, the system first clears the entire buffer to remove all events preceding the detected interrupt, preventing outdated or incorrect commentary. Following this, the active speaker thread is terminated to ensure that no prior events continue to be processed or commented on.
Once the buffer is cleared and the previous speaker thread is terminated, the interrupt event is transmitted to the *Prompt Generator*, which formulates a new prompt specifically for the high-priority event. A new *Speaker Thread* is then initiated, ensuring a seamless transition from the previous commentary cycle. With the new *Speaker Thread*, a characteristic sentence for the interrupt is generated, connecting the previous comment to the interrupting event-comment. Within

this whole process, the system continuously monitors incoming information and repeats the interrupt handling procedure whenever a new high-priority event is detected.
By implementing this structured approach, the pipeline maintains a consistent and reliable commentary output while preventing conflicts within the buffer or subsequent processing modules. The independent interrupt-handling mechanism ensures that events of high interest are properly handled. This also improves error resilience and overall pipeline stability.

3.4.5 Prompt Generator and LLM Interface

The final two modules of the *Commentator Thread* are responsible for generating and formatting prompts, as well as managing the accessibility interface for the Large Language Model. These components work in tandem to ensure the production of contextually relevant commentary.
The *Prompt Generator* is a highly adaptable system that constructs event-specific prompts using predefined templates tailored to various types of in-game events. Beyond template management, this module validates the contextual data of incoming events, processes team-wide plays, handles multiple-player interactions, and accounts for individual actions. Additionally, it integrates game statistics into the prompt generation process, ensuring that comments are produced at key moments, such as after each goal or when a certain period of time has elapsed without commentary.

Working in close coordination with the *Prompt Generator*, the *LLM Interface* manages the initialization and execution of the Llama 3.2-3B-Instruct model from Hugging Face. This module oversees token management, configures text generation parameters, and ensures efficient interaction with the model. Together, these two modules produce structured, event-driven commentary adapted to various event-driven situations.

3.4.6 Commentator

The third thread, referred to as the *Speaker Thread*, consists of the *Commentator Module*, which is responsible for both the visualization and voice output of the generated commentary. This module manages the presentation of LLM-generated commentary for the table foosball system, ensuring a smooth presentation and visualization of the commentary.
By controlling the timing, speed, and flow of text output, the module effectively simulates natural speech patterns. Its flexible output control enables dynamic speech adjustments based on the amount and significance of incoming events. As a result, in less eventful matches, the system generates commentary at a slower pace, wherever in more dynamic and eventful games, it accelerates both comment generation and speech output to match the intensity of the events.
To manage accurate event commentary, as previously mentioned in the interrupt handling chapter, the module continuously monitors for interrupt events from the *Event Cleaner* module. In the absence of an interrupt, it retrieves and comments on the most relevant event from the *Buffer*.
Through this approach, the final commentary is presented in such a manner that sensibility of the output as well as grammatical correctness are preserved, while maintaining high error resilience and timeliness.

4 Evaluation

4.1 Pipeline

To complement the following component-wise evaluations of the system, we conducted an end-to-end performance test focusing on the overall pipeline latency—from the moment a game event is detected by the computer vision module to the point a comment is generated by the LLM pipeline. Notably, this evaluation excludes the time taken for actual speaking the comment. The goal of this evaluation was to quantify the system's responsiveness and ensure the integration of components does not introduce significant delay. During testing on a Windows 10 machine equipped with an NVIDIA RTX 4070TI GPU and an AMD Ryzen 9 7900X 12x 5.6GHz CPU (in the component-wise evaluations only named Test Computer), the average processing time per 'goal'-event across the entire pipeline was measured at 0.41 seconds, with a maximum observed latency of 0.64 seconds. These results indicate that the pipeline maintains near real-time performance even under peak load conditions, meeting the design goal of fast and responsive commentary generation.

4.2 Computer Vision

The evaluation uses a set of foosball match videos recorded at approximately 60 FPS with a resolution of 1280 × 720, featuring significant noise and uneven lighting—brighter on the left, which casts pronounced shadows from players and rods. The dataset encompasses both short and long versions, providing varied durations for a comprehensive assessment. Specifically, the short videos include *test_009* (31 seconds, 1 goal, best lighting), *test_010* (66 seconds, 1 goal, medium lighting), and *test_011* (29 seconds, 2 goals, worst lighting), with their long counterparts extending to 9 minutes and 34 seconds, 7 minutes and 30 seconds, and 8 minutes and 18 seconds, respectively. This range in video length, lighting conditions, and dynamic gameplay offers a robust basis for evaluating the performance and reliability of our ball detection method.

4.2.1 Ball Detection

In the evaluation using the short foosball videos (selected due to the exhaustive annotation process), detection results were aggregated over all frames to obtain meaningful performance metrics. Table 1 summarizes the detection performance: for each video, true positives (TP) indicate the frames where the ball was correctly detected when present, true negatives (TN) represent the frames where the ball was physically absent from the field as expected, and false negatives (FN) capture instances where the ball was missed—often due to occlusion by players, rods, or walls. Notably, no false positives (FP) were observed. Derived metrics such as precision, accuracy, recall, and F1 score provide further insight into detection reliability.

Table 1: Ball Detection Performance Metrics

Video	Total Frames	TP	TN	FP	FN	Precision	Accuracy	Recall	F1 Score
`test_009`	1817	1044	709	0	66	100%	96.4%	94.1%	97.0%
`test_010`	3796	3376	147	0	273	100%	92.8%	92.5%	96.0%
`test_011`	1731	1121	449	0	161	100%	90.7%	87.5%	93.3%

Additionally, Table 2 presents the localization performance for frames in which the ball was present, reporting the mean error, root mean square error, standard deviation, maximum error, median error, and the third quartile of the error distribution. These tables collectively illustrate the strengths and challenges of our ball detection method under varying conditions.

Table 2: Ball Localization Performance Metrics (in pixels)

Video	Mean Error	RMSE	Std. Deviation	Max Error	Median Error	3rd Quartile
`test_009`	1.26	2.49	2.15	29.43	1.00	1.41
`test_010`	2.33	3.30	2.34	27.66	1.41	3.00
`test_011`	2.46	3.38	2.33	21.26	1.41	3.16

The experiments reveal that improved lighting enhances ball detection and results in a more precise approximation of the center point, thanks to clearer contrast and reduced ambiguity. However, this benefit is accompanied by a significantly higher maximum error, indicating that occasional outlier cases—likely due to glare or reflections—can lead to large deviations.

4.2.2 Goal and Throw-In Detection

Using the long versions of the videos to capture a comprehensive range of dynamic events—including goals and throw-ins—we evaluated the detection performance of the system. The results, summarized in Table 3, indicate perfect detection for both goal and throw-in events. For instance, *test_009* recorded 37 true positive detections, *test_010* 27, and *test_011* 32, with no false positives or false negatives across all cases. This consistent performance, reflected in 100% precision, recall, F1-score, and accuracy for each video, demonstrates the robustness of the detection method under challenging, real-world conditions.

4.2.3 Player Detection

During training, the player detection achieves a training loss of 0.03 and a validation loss of 0.029. At the end of fine-tuning the training loss is 0.019 and the validation loss is 0.038. Although the loss increases, the generalization of the model improves. The performance of the player detection was evaluated on different hardware configurations 4.

Table 3: Goal and Throw-In Detection Performance Metrics

Video	TP	FP	FN	Precision	Recall	F1-Score	Accuracy
test_009	37	0	0	100%	100%	100%	100%
test_010	27	0	0	100%	100%	100%	100%
test_011	32	0	0	100%	100%	100%	100%

Hardware	**Total Time**	**UNET Detection Time**	**Rod Detection Time**
AMD Ryzen 7 5800X cpu	39ms	24ms	4ms
Intel(R) Xeon(R) Gold 6242, RTX 2080 Ti	59ms	38ms	13ms
Intel(R) Xeon(R) Gold 6242	75ms	59ms	8ms

Table 4: Player detection times on different hardwares.

4.2.4 Field Detection

As the field detection system utilizes hardcoded parameters, no performance evaluation is required. Coordinates are manually calibrated per input stream to account for video-specific spatial properties, ensuring precision at the expense of automated adaptability.

4.2.5 Touch Detection

The evaluation of the touch detection compares manually annotated touch events for the short version of video *test_011*, determined by human observation, with automatically detected touch events from the system. The goal is to assess the system's performance in identifying relevant touch events under different parameter settings. The following table presents the evaluation results for different configurations:

Vel Thres	**Dir Thres**	**Near Thres**	**Dist Wt**	**Prec**	**Rec**	**F1**
0.2	15	1.2	1.2	0.500	0.211	0.297
0.2	5	2.0	2.0	0.326	0.326	0.326
0.2	5	1.5	1.5	0.414	0.312	0.356

Table 5: Evaluation results under different parameter configurations.

Based on this short, preliminary optimization, the best results were obtained with a velocity threshold of 0.2, a direction threshold of 5, and a near threshold of 1.5, with a distance weight of 1.5, yielding the highest precision of 0.414 and an F1-score of 0.356. These findings provide initial insights into the parameter impact, though further evaluation is needed for more conclusive results.

4.3 Backend Knowledge Extraction

The following part contains different qualitative and quantitative evaluation approaches for the major components of the backend knowledge extraction.
One of the most important aspects to ensure near live commentary is to keep the processing time of each received touch or error low. Evaluated on the above mentioned videos (using the above mentioned Test Computer), this processing averaged at around 5 milliseconds in the standard cases, therefore producing only a non-relevant overhead for the commentary. Noticeably, the error handling process averages at a duration of 297 milliseconds over 17 total error sequences, slightly slowing the backend processing down whenever a reconstruction is necessary.
Additionally, the quality of the error handling was evaluated by averaging the confidence values of the uncertain states in the reconstructed history. This average value evaluates the probability with which the error handling reconstructs gameplay sequences that actually happened on the table during the error transmission duration, based on the assumption that the gameplay is similar to the training data of the Markov model. Over all 17 error reconstructions, the average result was 0.32. However, as the first few states in a reconstructed path generally have higher confidence, the reconstruction yields more confident predictions for short reconstructions.
The performance of the event recognition system was evaluated across both implemented approaches: rule-based detection and autoencoder-based pattern recognition. The evaluation was

conducted on recorded matches.
The rule-based approach demonstrated robust performance for well-defined gameplay events. Goals, throw-ins, and shots were consistently identified and correctly forwarded to the LLM module for commentary generation. The detection of blocked shots also proved reliable in most instances, though occasional misclassifications were observed when rapid consecutive touches occurred. The strength of the rule-based approach lies in its deterministic nature, which allows for high precision in detecting events with clear state transition patterns. While quantitative metrics were not formally collected to support these observations, the qualitative performance was deemed satisfactory for the commentary generation requirements.
The autoencoder architecture for complex event pattern recognition demonstrated significant limitations despite being trained on approximately 2.5 hours of gameplay footage. Two instances of each desired complex event (wall passes, through pass) were used as labeled examples during testing, but this proved insufficient for reliable detection. The primary challenge identified was the dependency of the autoencoder on the quality and completeness of the touch data stream. The performance was substantially hindered by missing touches or incorrectly detected touches within the input sequences. Since the autoencoder operates on three consecutive state transitions, any error in state classification propagated through the embedding space comparison, resulting in poor similarity scores even for events that visually resembled the labeled examples. It was observed that the embedding space created by the autoencoder failed to generalize beyond the specific examples provided, likely due to inconsistencies in the input touch data stream and high variability in how similar events manifest in the state transition model.
The comparison mechanism in the embedding space rarely yielded similarity scores above the established threshold, resulting in very few complex events being detected and forwarded to the commentary system. When events were detected, they often did not correspond to actual gameplay patterns observable by human validators.

4.4 Large Language Model

The evaluation of the large language models is divided into quantitative and qualitative analyses. All tests were conducted on the above mentioned Test Computer. The quantitative evaluation primarily focuses on the computation time for generating comments, serving as a key performance indicator to justify the choice of the current model. The average computation times are presented in Table Table 6, where the Llama 3.2-1B-Instruct model performs almost as fast as the first model but significantly faster than the third. Given the fast-paced nature of table foosball, we considered the first and second models as viable options for comment generation, while the third model was deemed too slow, thus ranking last in terms of time efficiency.

Table 6: Model runtime evaluation

Llama-Model	Average Computationtime	Model Size
`Llama 3.2-1B-Instruct`	0.21 s	1.24 GB
`Llama 3.2-3B-Instruct`	0.48 s	3.21 GB
`Llama 3.2-8B-Instruct`	5.82 s	16-20 GB

In the qualitative evaluation, we analysed the outputs to assess interpretability and relevance. The Llama 3.2-1B-Instruct model struggled even with simple questions, sometimes responding "What is 1+1? What is 2+2? What is 3+3?" to the question "What is 1+1?". This indicated its inadequacy for the task. Conversely, the Llama 3.2-8B-Instruct model, while superior in generating varied and grammatically sound comments, was too slow, making it impractical for real-time applications. The Llama 3.2-3B-Instruct model emerged as the optimal choice, balancing computational efficiency, acceptable interpretability, and manageable storage requirements and was thereby chosen for it's overall performance across all evaluated dimensions, despite some challenges.
The model sometimes struggled with generating accurate and relevant commentary for table foosball, as it occasionally produced comments related to other sports. This indicates a challenge in consistently recognizing the specific context of table foosball, which is essential for delivering precise commentary. Additionally, the model often generated meta-comments, such as "GOAL! is a good answer, but GOAL FROM JHONSON! would not fit the requirements." These were not actual game commentary but rather reflections on possible responses, which detracted from the immersive experience intended for the audience. Another issue observed with the model was its tendency to introduce information that was not provided, such as player names or events that

did not occur. This problem highlights a challenge in maintaining factual accuracy, as the model sometimes fabricates details that can confuse the audience and undermine the credibility of the commentary.
To address these challenges, hyperparameter tuning and prompt engineering aimed to improve its ability to generate accurate and contextually relevant commentary and was utilized to refine the input prompts, guiding the model to produce more precise and factually correct outputs. These efforts were focused on enhancing the model's generation time while mitigating issues related to incorrect sport commentary, meta-comments, and the inclusion of fabricated information. Despite these adjustments, some problems persisted in the output comments, indicating the complexity of achieving perfect accuracy in close to real-time commentary generation.

5 Conclusion

This project presents a novel system that generates live commentary for foosball matches by integrating computer vision, probabilistic modeling, and large language models. The architecture is based on a modular pipeline that processes raw video input, detects and interprets key game events, and transforms them into natural-language commentary with minimal latency.
The computer vision module successfully extracts complex spatial and temporal patterns from challenging video material, including robust detection of the ball, players, goals, and throw-ins. A structured backend translates these low-level events into higher-level gameplay insights using a Markov-based state model, which supports both statistical game analysis and error-resilient reconstructions. Finally, a dedicated LLM pipeline selects and prioritizes the most relevant events, generating coherent and engaging commentary in real-time.
Throughout the development, several key challenges were addressed, including detection robustness under varying lighting conditions, efficient handling of missing or noisy data, and responsive commentary output. Evaluation results confirm the system's viability, particularly in terms of precision for key event detection and runtime performance of the selected LLM.
This project contributes to the broader field of AI-based sports analytics by demonstrating how complex models can be orchestrated to work together in real-time environments. While the domain of foosball offers unique constraints, the core approach and architecture are transferable to other fast-paced sports scenarios. The system lays the groundwork for further improvements and invites future research at the intersection of computer vision, event modeling, and language generation.

5.1 Future Work

Despite the system's achievements, several areas offer potential for further development. In the field of computer vision, ball detection currently depends on specific color assumptions, which limits adaptability. A future version should support color-independent ball recognition to improve robustness across different lighting conditions and table designs. Moreover, the field detection component relies on hardcoded configurations and would benefit from more flexible methods that can handle changes in camera perspective and height. To enhance the system's general applicability, future work should aim to remove foosball-specific assumptions from the detection pipeline, such as fixed rod configurations or player counts. Improving player detection to be invariant to scale and orientation is another promising step toward robustness and transferability. With every improvement in ball and player detection the touchlogger should be optimized as well, to guarantee a good touch detection. In addition future work could include a complete optimization of the touchlog parameters across multiple annotations from various individuals and diverse videos. Ideally, this would guarantee a robust and precise touch detection and could eliminate the need for a filter module altogether.
In terms of backend modeling, future work should focus on enhancing event recognition. While the rule-based system already performs well for clear events like goals and throw-ins, the autoencoder for detecting complex patterns such as dribbling or wall passes currently lacks sufficient generalization due to limited training data. Better training procedures, more representative labeled data, and improved handling of missing touches are key areas for improvement. Furthermore, error handling could be expanded by incorporating temporal constraints to limit the length of reconstructed sequences and improve plausibility. An additional research focus should lie in fine-tuning the meta-parameters of the state model to strike an optimal balance between simplification and detail retention.
The large language model pipeline also offers several avenues for refinement. The event prioritization logic could be improved to better reflect the dynamics of the match, while the interest

scoring formula may be further adapted to include more nuanced temporal and contextual factors. Integrating a short-term memory would enable the system to refer back to previous events when generating commentary, enhancing narrative coherence. The integration of a vector database offers an additional opportunity to enhance both the efficiency and richness of the generated commentary. By storing comments generated during phases of low in-game activity and associating them with semantically similar event representations, the system could retrieve contextually appropriate commentary even faster. This would reduce the need for on-the-fly generation, enabling longer and more elaborate comments without compromising latency, and thereby supporting more sophisticated commentary patterns. Finally, additional prompt engineering and template tuning would help refine the generated language, improving clarity, variety, and alignment with the match's tone and flow.
Taken together, these future directions provide a comprehensive roadmap for extending the system toward a robust, adaptable, and intelligent commentary engine capable of serving a wide range of sports and entertainment applications.

References

[BL] Sven Bambach and Stefan Lee. Real-Time Foosball Game State Tracking. https://computing.ece.vt.edu/~steflee/pdfs/foosball-tracking.pdf. [Online; accessed 20.03.2025].

[CCTDH22] Harvey Campos-Chavez, Soren Thrawl, Anthony DeLegge, and Amanda Harsy. Predictive Modeling and Analysis of Hockey Using Markov Chains. 2022.

[CK24] Alec Cook and Oktay Karakuş. LLM-Commentator: Novel fine-tuning strategies of large language models for automatic commentary generation using football event data. *Knowledge-Based Systems*, 300:112219, 2024.

[Cor25] NVIDIA Corporation. CUDA Toolkit - Free Tools and Training — NVIDIA Developer, 2025.

[DL15] Gabriel Damour and Philip Lang. Modelling Football as a Markov Process : Estimating transition probabilities through regression analysis and investigating it's application to live betting markets, 2015.

[GWD+24] Mingyang Geng, Shangwen Wang, Dezun Dong, Haotian Wang, Ge Li, Zhi Jin, Xiaoguang Mao, and Xiangke Liao. Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–13, 2024.

[Inc25a] Docker Inc. Docker: Accelerated Container Application Development, 2025.

[Inc25b] Docker Inc. Docker Compose — Docker Docs, 2025.

[MNMMM24] Tiago Mendes-Neves, Luís Meireles, and João Mendes-Moreira. Forecasting Events in Soccer Matches Through Language. *arXiv preprint arXiv:2402.06820*, 2024.

[MP25] Inc. Meta Platforms. Meta — Social Metaverse Company, 2025.

[RL22] F. Rothe and M. Lames. Simulation of Tennis Behaviour Using Finite Markov Chains. 55(20):606–611, 2022.

[Too] Tooploox. Foosballytics: Real-Time Computer Vision for Foosball. https://tooploox.com/foosballytics-real-time-computer-vision. [Online; accessed 20.03.2025].

A List of Contributions

- **Crasemann, Christoph**
  - Prompt Engineering, Hyperparameter tuning
  - Event-Cleaner and Event-Preprocessing in Listener-Thread
  - LLM-Pipeline, Multithreading
  - LLM Model Testing, (Early) VM Running Tests
- **Daum, Julius**
  - Conducted exploratory computer vision analysis for ball and player detection systems during the project's inception phase.
  - Established baseline logic for entity representation and enumeration.

  - Conducted initial field detection analysis.
  - Contributed to pipeline architecture planning and development as part of the Pipeline Task Force.
  - Served as an animator in internal (and occasionally external) project meetings.

- **Horlbeck, Carolina**
  - Calculation of ball movement
  - CV Pipeline in cv.main
  - Touchlogger
  - Annotation, Tests and Evaluation of Touchlogger

- **Kara, Atakan**
  - Development of Debug Video Player
  - Ball Detection, Goal Detection, Throw-In Detection
  - Code for Annotating, Testing, Evaluating Ball-, Goal-, Throw-In Detection
  - Demo Preparation for 2. Milestone

- **Krüger, Joost**
  - Buffer Module
  - Interest Score
  - Prompt Engineering/Hyperparameter Tuning
  - LLM Pipeline, Multithreading
  - Model Testing

- **Kujath, Falk**
  - Modularization of the LLM Pipeline (e.g. config, modules)
  - Interrupt Management
  - Speaker Management
  - Statistics Comments Integration
  - Multithreading for the LLM Pipeline
  - Prompt Engineering, Hyperparameter Tuning
  - Contributed to pipeline integration and testing as part of the Pipeline Task Force
  - Presentation Video Recording Guy

- **Mayer, Luca**
  - Development of the markov model
  - Model graph, KPI and commentary visualization in webapp
  - KPI integration into pipeline
  - Output data aggregation and distribution service (refered to as 'backend container')
  - Containerization of the project
  - Part of the "pipeline task force"

- **Nissen, Mads**
  - Development of error handling component
  - Development of most of the class structure (Touch, State, Pipeline classes)
  - Refinement of bidirectional mapping of states and their indices (used in the transition matrix)
  - Development of large parts of the main LivePipeline structure and logic
  - Initial integration and development of the touch filter component
  - Development of the statistic tracking component

- **Schirp, Jonathan**
  - Development and implementation of state model concept
  - Development of touch to state parsing
  - Development of both event recognition approaches
  - Development of the training pipeline
  - Training of the autoencoder and markov model
  - Refinement of the touch filter component
- **Wieck, Philipp**
  - Development of the rod detection
  - Development of the player detection including player center positions, player contours and player feet positions