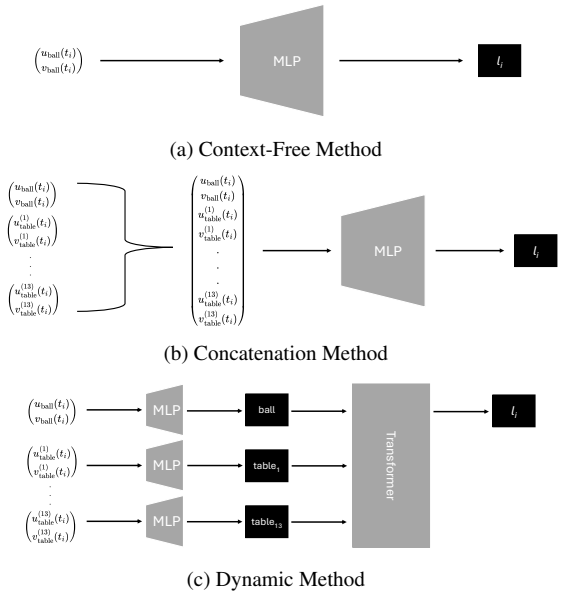
# Towards Ball Spin and Trajectory Analysis in Table Tennis Broadcast Videos via Physically Grounded Synthetic-to-Real Transfer

Supplementary Material

## A. Further Architecture Details




(a) Context-Free Method

(b) Concatenation Method

(c) Dynamic Method

Figure 7. Token embedding methods. Input to the embedding layers are the 2D coordinates of the ball and the 13 table keypoints. The output is the location token $l_i$. The embedding layer is applied for each time step $t_i$ separately.

Figure 7 provides an overview of the different token embedding strategies utilized in our model. Each method processes the 2D coordinates of the ball along with 13 table keypoints to generate the location token $l_i$ at time $t_i$. The embedding operation is applied separately for each time step $t_i$.

- **Context-Free Method:** The context-free method directly embeds the ball coordinates via a multilayer perceptron (MLP) without using any table keypoints.
- **Concatenation Method:** The 2D coordinates of all 14 points are concatenated into a single vector. This vector is then transformed into a location token via an MLP.
- **Dynamic Method:** Instead of direct concatenation, a small transformer encoder processes the table keypoints and condenses their information into the ball position token. This token is then used as location token $l_i$, the other tokens are discarded.

To evaluate model performance across different architectural complexities, we vary the number of transformer layers $L$, the number of attention heads $H$, and the embedding dimension $d$. Table 4 summarizes the configurations explored.

| Size | Layers $L$ | Heads $H$ | Embedding Dimension $d$ | Number of Parameters |
|---|---|---|---|---|
| Small | 8 | 4 | 32 | $0.06 \times 10^6$ |
| Base | 12 | 4 | 64 | $0.3 \times 10^6$ |
| Large | 16 | 4 | 128 | $1.6 \times 10^6$ |
| Huge | 16 | 8 | 192 | $3.2 \times 10^6$ |

Table 4. Transformer architecture variants. The number of trainable parameters is calculated for the model with connect-stage SPT architecture and concatenation token embedding module.

## B. Annotation Details

For each trajectory, we annotate the 13 table keypoints in the first frame. Since the camera remains static throughout each video, these annotations are consistently used for all subsequent frames within the trajectory. The annotated keypoints are illustrated in Figure 8. Annotating the spin direction is particularly challenging, as the spin is not directly observable in broadcast footage. To infer the spin type, we analyze the paddle's orientation at the moment of impact.

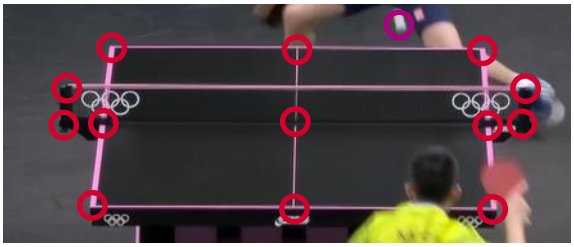


Figure 8. The 13 table keypoints (red circles) and the ball position (purple circle) are highlighted.

- **Topspin**: If the paddle is angled towards the table, the shot is labeled as topspin. This is illustrated in Figure 9a.
- **Backspin**: If the paddle is angled away from the table, the shot is labeled as backspin. This is illustrated in Figure 9b.

This annotation approach provides a practical method for inferring spin direction despite the limitations of broadcast video data.

## C. Regressing camera matrices

Although our method operates in an end-to-end manner without requiring camera calibration as input, the intrinsic and extrinsic camera matrices are necessary for computing


(a) Topspin


(b) Backspin

Figure 9. Example frames for the annotation of the spin direction. If the paddle is facing the table (a), the shot is annotated as topspin. If the paddle is facing away from the table (b), the shot is annotated as backspin.

the 2D reprojection error. For each trajectory, we manually annotate the 13 table keypoints once. Since the corresponding 3D world coordinates are known due to standardized table sizes in professional matches, the camera matrices can be estimated by minimizing the reprojection error of these 13 points. However, this regression process is inherently unstable, and even small annotation errors can lead to significant inaccuracies in the estimated camera matrices. To mitigate this issue, we employ the RANSAC algorithm [16] to robustly filter out erroneous annotations. In each RANSAC iteration, we randomly select six non-planar keypoints and compute an initial estimate of the camera matrices using the Direct Linear Transformation (DLT) algorithm [1]. This initial estimate is then refined using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization algorithm [5]. With the refined matrices, we determine the number of inliers by checking the reprojection error. A point is classified as an inlier if the reprojected 3D point is within 3 pixels of the corresponding 2D annotation. This procedure is repeated 100 times, and the setting with the highest number of inliers is selected. Using all identified inliers, we compute the final camera matrices by first applying the DLT algorithm and subsequently refining the result with the BFGS optimization. The RANSAC-based approach is essential for mitigating the impact of slight errors in the 2D annotations and obtaining accurate camera matrices. We highlight that our model, which does not require camera calibration, is very robust to minor errors in the 2D input, highlighting the benefits of implementing an end-to-end approach.

## D. Augmentation Details

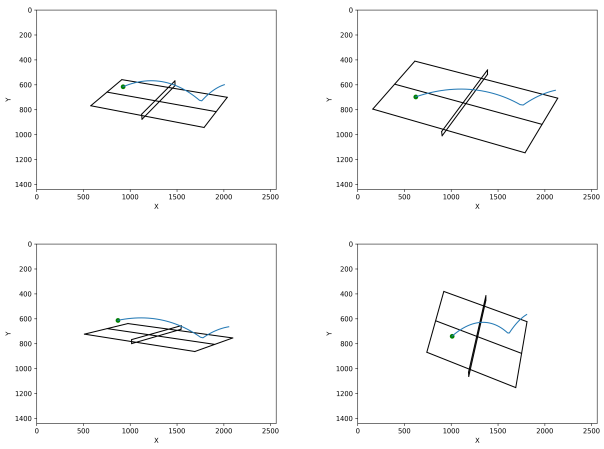


Figure 10. A trajectory sampled from 4 different camera perspectives. During training, we transform the trajectory with such randomly sampled camera parameters to introduce different viewpoints.

During training, we **randomly sample plausible camera parameters** to simulate diverse camera perspectives, increasing the diversity of the model inputs and ensuring that the model generalizes well across different viewpoints. Examples of trajectories visualized from various sampled camera parameters are shown in Figure 10.

In the synthetic dataset, we do not only store the ball's 3D position at each time step but also record intermediate positions. For these intermediate positions, we employ a "virtual" framerate of 500 Hz, which is ten times higher than the actual framerate. This allows us to accurately simulate **motion blur** in a physically consistent manner. Rather than using the exact ball position at a given timestamp, we randomly select a point within a temporal window around each frame. For the experiments in this paper, we define this window such that the selected point has a timestamp within a maximum deviation of $0.4 * \frac{1}{50\,\text{Hz}}$. Motion blur is applied not only to the 2D ball positions but the 3D ground truth positions are shifted accordingly.

The **sudden end** augmentation is designed to simulate scenarios in which the opposing player interferes, leading to an abrupt termination of the trajectory. For each trajectory, we remove a randomly selected number of coordinates at the end, however, we ensure that the trajectory always includes the bounce on the table. This augmentation is applied with a probability of $50\%$ during training, allowing the model to learn from complete trajectories while also adapting to

cases where the ball's motion is unexpectedly interrupted.

The **Gaussian blur** augmentation is implemented by introducing random noise to the 2D ball position and the 2D table keypoints. The noise is sampled from a Gaussian distribution with a standard deviation of 2 pixels in both the $x$- and $y$-directions, effectively simulating annotation inaccuracies.

# E. Further Experiments

This section presents additional experiments that provide deeper insights into the model's behavior. All experiments are based on the best model, which is described in Section 6.4. It uses the concatenation token embedding method, the connect-stage SPT architecture, and all data augmentations.

## E.1. Spin Prediction Coordinate System

| Method | Synthetic | | Real | | | |
|---|---|---|---|---|---|---|
| | $\Delta\vec{\omega}$ | $\Delta\vec{r}_{\text{world}}$ | *acc* | $F_1$ | ROC-AUC | $\Delta\vec{r}_{\text{img}}$ |
| world | 48.7 Hz | 5.5 cm | 92.0 % | 0.917 | 0.990 | **0.19 %** |
| ball | 48.3 Hz | 5.4 cm | **94.0 %** | **0.938** | **1.000** | 0.22 % |

Table 5. Comparison of different spin prediction coordinate systems. The best results on the real data are highlighted in bold.

In Section 3.2, we discussed that while the trajectory is analyzed in the world coordinate system, the spin is evaluated in the ball coordinate system. According to Equation 2, the predicted spin can be transformed between coordinate systems. Thus, there are two approaches for predicting the spin:

- The network is trained to predict the spin in the world coordinate system, and the predicted trajectory is used in Equation 2 to transform the spin into the ball coordinate system.
- The network is trained to directly predict the spin in the ball coordinate system, eliminating the need for any transformation.

Since the first approach relies on the predicted trajectory for coordinate transformation, it may introduce additional errors. Conversely, using the same coordinate system for both trajectory and spin could simplify training, as the network does not need to learn the transformation.

Table 5 compares both approaches. Training the network directly in the ball coordinate system results in slightly better performance across all spin-related metrics. However, trajectory prediction benefits from predicting the spin in the world coordinate system. Overall, the differences are minor. Choosing the coordinate system depends on whether trajectory accuracy or spin accuracy is more critical for the specific application. This allows for flexibility in the model design, enabling it to be tailored to the specific requirements of the task at hand.

## E.2. Positional Encoding

| Method | Synthetic | | Real | | | |
|---|---|---|---|---|---|---|
| | $\Delta\vec{\omega}\downarrow$ | $\Delta\vec{r}_{\text{world}}\downarrow$ | *acc* ↑ | $F_1$ ↑ | ROC-AUC ↑ | $\Delta\vec{r}_{\text{img}}\downarrow$ |
| rotary | 48.7 Hz | 5.5 cm | **92.0 %** | **0.917** | **0.990** | **0.19 %** |
| added | 52.6 Hz | 5.8 cm | 90.0 % | 0.897 | 0.987 | 0.26 % |

Table 6. Comparison of different positional encodings. The best results on the real data are highlighted in bold.

The standard approach for incorporating positional information in transformers is by adding a fixed sinusoidal positional encoding to the token embeddings. However, our model utilizes a rotary positional encoding, which is commonly used in language models. Table 6 compares both methods. As the rotary positional encoding achieves better performance across all metrics, we conclude that it is more suitable for our task.

## E.3. Loss Target

| Method | | Synthetic | | Real | | | |
|---|---|---|---|---|---|---|---|
| $\mathcal{L}_{\text{trajectory}}$ | $\mathcal{L}_{\text{spin}}$ | $\Delta\vec{\omega}\downarrow$ | $\Delta\vec{r}_{\text{world}}\downarrow$ | *acc* ↑ | $F_1$ ↑ | ROC-AUC ↑ | $\Delta\vec{r}_{\text{img}}\downarrow$ |
| ✓ | ✓ | 48.7 Hz | 5.5 cm | **92.0 %** | **0.917** | **0.990** | **0.19 %** |
| × | ✓ | 60.6 Hz | - | 78.0 % | 0.769 | 0.890 | - |
| ✓ | × | - | 5.5 cm | - | - | - | 0.22 % |

Table 7. Comparison of joint prediction with individual models for each task. ✓ indicates which loss function is used during training, while × indicates the absence of the specific loss. The best results on the real data are highlighted in bold.

Our model is designed to jointly predict both trajectory and spin. For training both task, we simply sum the two loss functions in Equation 7. In this section, we examine whether joint prediction is beneficial or if the two tasks should be handled by separate models.

Table 7 compares joint prediction with separate models. The results clearly show that joint prediction outperforms the separate models across all metrics. This suggests that the network extracts useful information from one task that enhances the other. Thus, joint prediction is an effective approach.

## E.4. Model Size

| Method | Synthetic | | Real | | | |
|---|---|---|---|---|---|---|
| | $\Delta\vec{\omega}\downarrow$ | $\Delta\vec{r}_{\text{world}}\downarrow$ | *acc* ↑ | $F_1$ ↑ | ROC-AUC ↑ | $\Delta\vec{r}_{\text{img}}\downarrow$ |
| small | 64.9 Hz | 10.7 cm | **92.0 %** | **0.917** | 0.956 | 0.29 % |
| base | 51.0 Hz | 6.2 cm | 90.0 % | 0.895 | **0.998** | 0.25 % |
| large | 48.7 Hz | 5.5 cm | **92.0 %** | **0.917** | 0.990 | 0.19 % |
| huge | 48.8 Hz | 5.1 cm | 86.0 % | 0.850 | 0.971 | **0.17 %** |

Table 8. Comparison of different model sizes. The best results on the real data are highlighted in bold.

Table 8 compares different model sizes, which are defined in Table 4. All models demonstrate good performance in spin prediction. However, increasing the model size improves trajectory prediction accuracy. Additionally, the spin prediction performance of the largest model is slightly worse than that of the other models, possibly due to overfitting. Therefore, we identify the large model as the best compromise between spin prediction and trajectory prediction.

## F. Reproducibility and Open Resources

To facilitate reproducibility and further research, we provide the following resources:

- Synthetic trajectories used for training.
- Annotations for the real-world dataset.
- Trained model weights.
- Source code for both training and inference.

All resources are publicly available at
https://kiedani.github.io/CVPRW2025/.