Exploitation of Hidden Context in Dynamic Movement Forecasting: A Neural Network Journey from Recurrent to Graph Neural Networks and General Purpose Transformers

Lukas Schelenz*, Shobha Rajanna*, Denis Gosalci*, Lucas Heublein*, Jonas Pirkl*, Jonathan Ott*, Felix Ott*, Christopher Mutschler*, Tobias Feigl*†
*Fraunhofer Institute for Integrated Circuits IIS, 90411 Nürnberg, Germany
†Friedrich-Alexander-Universität Erlangen-Nürnberg, 91058 Erlangen, Germany
{lukas.schelenz, shobha.rajanna, denis.gosalci, lucas.heublein, jonas.pirkl, jonathan.ott, felix.ott, christopher.mutschler, tobias.feigl}@iis.fraunhofer.de

2025 IEEE/ION Position, Location and Navigation Symposium (PLANS) | 979-8-3315-2317-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/PLANS61210.2025.11028353

***Abstract*—Forecasting within signal processing pipelines is crucial for mitigating delays, particularly in predicting the dynamic movements of objects such as NBA players. This task poses significant challenges due to the inherently interactive and unpredictable nature of sports, where abrupt changes in velocity and direction are prevalent. Traditional approaches, including (S)ARIMA(X), Kalman filters (KF), and Particle filters (PF), often struggle to model the non-linear dynamics present in such scenarios. Machine learning (ML) methods, such as long short-term memory (LSTM) networks, graph neural networks (GNNs), and Transformers, offer greater flexibility and accuracy but frequently fail to explicitly capture the interplay between temporal dependencies and contextual interactions, which are critical in chaotic sports environments.**

In this paper, we evaluate these models and assess their strengths and weaknesses. Experimental results reveal key performance trade-offs across input history length, generalizability, and the ability to incorporate contextual information. ML-based methods demonstrated substantial improvements over linear models across forecast horizons of up to 2s. Among the tested architectures, our hybrid LSTM augmented with contextual information achieved the lowest final displacement error (FDE) of 1.51m, outperforming temporal convolutional neural network (TCNN), graph attention network (GAT), and Transformers, while also requiring less data and training time compared to GAT and Transformers. Our findings indicate that no single architecture excels across all metrics, emphasizing the need for task-specific considerations in trajectory prediction for fast-paced, dynamic environments such as NBA gameplay.

***Index Terms*—Trajectory Forecasting, Time-series, Machine Learning, Deep Learning, Recurrent Network, Long Short-term Memory, Transformer, Self-attention, Graph Attention Network, Human Behaviour, Sport Analytics**

This work has been carried out within the DARCII project, funding code 50NA2401, sponsored by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) and supported by the German Aerospace Center (DLR), the Bundesnetzagentur (BNetzA), and the Federal Agency for Cartography and Geodesy (BKG). This work was also supported by the Bavarian Ministry for Economic Affairs, Infrastructure, Transport and Technology through the Center for Analytics Data Applications (ADA-Center) within the framework of "BAYERN DIGITAL II" (20-3410-2-9-8).

I. Introduction

In recent years, ML-driven forecasting has gained significant attention across various domains, including sports analytics [1], stock market predictions [2], [3], handwriting classification [4], electricity pricing analytics [5], weather forecasting [6], robotic trajectory prediction [7], [8], and healthcare. Within the realm of sports analytics, research has predominantly concentrated on predicting game outcomes [9], [10], while relatively few studies have addressed the forecasting of human motion, particularly the trajectories of athletes [11]. Accurate trajectory forecasting plays a pivotal role in enhancing real-time game analysis, offering tactical advantages [12], and mitigating injury risks [13], [14].

Athlete movements are inherently nonlinear and dynamic, influenced by factors such as spatial positioning, proximity to opponents, and abrupt changes in speed and direction [15]. Traditional statistical models, including ARIMA [3] and linear regression, as well as Bayesian methods such as KFs [16] and PFs [17], rely on linearity assumptions that render them inadequate for capturing sudden transitions and complex multi-agent interactions [11], [18]. These limitations restrict their ability to effectively handle long-term dependencies and contextual interactions in dynamic environments. To address these challenges, ML-based models have been explored for their capability to learn nonlinear patterns. Recurrent neural networks (RNNs), particularly LSTM networks [19], gated recurrent units (GRUs) [20], and Legendre memory units (LMUs) [21], are effective in capturing sequential patterns and temporal dependencies. However, they may suffer from vanishing memory and limited scalability when processing extended sequences [22]. GNNs [23], [24], [25], [26], [27], particularly GATs [28], [29], offer improved modeling of multi-agent interactions and provide a more interpretable framework for interaction analysis. Nonetheless, GNNs often lack temporal depth, reducing their effectiveness in sequential tasks. Transformers [22], [30], [31], [32], [33], originally

developed for natural language processing (NLP), present an alternative approach by employing self-attention mechanisms to capture long-range dependencies and global patterns. Despite their demonstrated success in various fields, these architectures remain underexplored for human motion forecasting in sports [34], [35]. Their adoption has been constrained by high computational costs and the absence of explicit domain-specific contextual modeling, which limits their applicability to real-time sports analytics.

Contributions. This study presents a comprehensive evaluation and comparative analysis of several ML-based forecasting models, including TCNNs, LSTMs, LMUs, GATs, and Transformers, and their hybrid configurations, for predicting the short-term movements of players in fast-paced and dynamic environments exemplified by NBA games. The investigation focuses on quantifying performance trade-offs across multiple criteria, including input history length, computational complexity, generalization capacity, and the ability to leverage contextual interactions. To address limitations in existing approaches, the proposed framework integrates explicit contextual modeling within a recurrent architecture. Unlike conventional methods that treat spatial and temporal features independently, this approach incorporates transformed distance functions within an LSTM pipeline, emphasizing the representation of shorter, more informative proximities. Additionally, team dynamics and inter-player interactions are explicitly modeled through hybrid architectures, such as RNN-GAT-RNN and fusion models that combine recurrent and attention-based mechanisms. The inclusion of Transformer-based components enables the framework to effectively capture both global dependencies and fine-grained dynamics, overcoming scalability challenges through domain-specific encodings. These innovations collectively enable the models to outperform state-of-the-art approaches in trajectory prediction, particularly in chaotic environments characterized by abrupt, non-linear player movements and complex inter-player interactions, such as those observed in NBA basketball games.

Outline. Sec. II offers a review of related literature. Sec. III describes preprocessing pipeline and the proposed methods. Sec. IV details the experimental setup, followed by the discussion of results in Sec. V. Sec. VI concludes the paper.

II. Related Work

Classical Approaches. Human motion forecasting has applications across a wide range of domains, including public safety [28], healthcare [1], and sports analytics [36], [11], [9], [10]. In the context of basketball, accurate player trajectory prediction offers substantial benefits for tactical decision-making, performance evaluation, and real-time analytics. Traditionally, model-based approaches such as KF [16] and PF [17] have been employed to address this task. These methods are effective in scenarios where motion is linear or near-linear, leveraging recent observations to generate predictions with computational efficiency. However, their performance is significantly constrained in dynamic, multi-agent environments such as basketball, where player movements are characterized by abrupt changes in velocity and direction. The reliance on simplistic motion assumptions and limited historical context further diminishes their effectiveness in capturing the complexities of real-world trajectories [11].

Spatial and Temporal Models. ML-based approaches have increasingly supplanted traditional statistical methods, offering greater flexibility in modeling complex, non-linear patterns. CNNs [37], [38], [39], for example, are adept at extracting spatial features from trajectory data but often fall short in capturing temporal dependencies and inter-player dynamics. Nikhil et al. [40] demonstrated the use of CNNs for player position prediction. However, the model's inability to represent complex interactions or long-term temporal dependencies limited its applicability to team sports. RNNs, particularly LSTMs [41], address these limitations by modeling sequential data and leveraging historical information to capture long-term dependencies. This capability has led to notable performance gains over KFs and PFs in dynamic environments [11]. Nonetheless, the reliance of LSTMs on gating mechanisms to control memory introduces computational inefficiencies. LMUs [21] seek to mitigate these challenges by employing a mathematically grounded state-space approach, enabling the effective retention of long-term dependencies while minimizing the dependence on learned gating mechanisms.

Transformer Architectures. Originally developed for NLP, Transformers [22] extend the capabilities of sequence modeling by employing self-attention mechanisms. These architectures effectively capture long-range dependencies while mitigating the vanishing gradient problems typically encountered in RNNs. Recent research has demonstrated the potential of Transformers in motion prediction tasks. For instance, Giuliari et al. [42] applied Transformers to static motion data, while Mascaro et al. [43] introduced a framework integrating spatial and temporal attention for 3D skeletal motion prediction. Transformers excel at simultaneously capturing global dependencies and fine-grained local patterns, theoretically making them well-suited for complex trajectory prediction problems. However, their practical application in real-time sports contexts remains limited due to high computational demands and the absence of explicit interaction modeling, that hinder their ability to effectively capture dynamic inter-agent relationships.

Graph Neural Networks. There are many adaptations of GNNs [28], [44], [45] that enable interaction-aware modeling. By representing players as nodes and interactions, such as spatial proximity or passes, as edges, GNNs provide a powerful framework for capturing inter-player dynamics. Mo et al. [28] demonstrated the effectiveness of GNNs in vehicle trajectory forecasting, while Xu et al. [25] advanced this approach by introducing attention-based GNNs (GATs) to model adaptive interaction strengths. Despite their demonstrated utility in other domains, these methods have not been adapted for basketball, where the complexity of team strategies demands a nuanced understanding of both spatial and temporal relationships. Effective modeling this context requires capturing dynamic interactions and evolving game scenarios, which remain unexplored using existing GNN-based approaches.

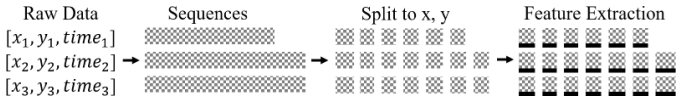


Fig. 1: Preprocessing of the NBA dataset.

Summary. Existing literature does not sufficiently address the complex interplay between temporal dependencies, inter-player interactions, and the dynamic context inherent in sports environments. Traditional statistical models are overly simplistic, relying on assumptions that limit their applicability in capturing the nuanced, non-linear motion patterns characteristic of such settings. ML-based approaches, while more flexible, frequently prioritize either temporal or spatial aspects, with limited efforts to seamlessly integrate both dimensions. The reviewed methods fail to incorporate context information, resulting in suboptimal performance for forecasting dynamic motion in sports. These limitations highlight the need for methods that effectively bridge these gaps, enabling more accurate and context-aware trajectory predictions.

III. Methodology

This Section presents the general and model-specific preprocessing pipelines (Sec. III-A). Sec. III-B introduces all methods. Sec. III-C explains method-specific postprocessing.

A. Preprocessing

General. To effectively train the models, several preprocessing steps are applied, see Fig. 1. Initially, raw data is collected and transformed into a unified structure. Each trajectory is represented as a 3D tensor of shape [num. timesteps, num. objects, num. features], where num. timesteps refers to the trajectory duration, num. objects represents the entities in the scene, e.g., players, ball, goals, and num. features includes attributes such as absolute position (pos_x, pos_y) and velocity (v_x, v_y). The NBA dataset is sampled at 25 Hz. Interpolation is applied to all trajectories to ensure a uniform time interval of 0.04 s. The dataset is then split into training, validation, and test sets, with 90% of the data allocated for training, further divided into 90% for training and 10% for validation. The test set is sourced from a single, unseen game to maintain evaluation integrity. To enhance generalization, trajectories in the training set are shuffled, while those in the test set remain unshuffled to preserve temporal continuity. Finally, the data is segmented into overlapping windows using a sliding window approach (step size = 1 to capture all windows). Each window is defined by a window size comprising a history length (input context) and a forecast length (steps to predict). For each window we convert its positions to velocities (v_x, v_y). All these position and velocity features are normalized. So, we ensure consistent magnitudes and prevent large positional values from dominating the training process.

Constant Velocity, Linear, TCNN, LSTM and LMU may directly use features (*positions* and / or *velocities*) of the general preprocessing pipeline.

GNN builds upon the general preprocessing pipeline with additional steps to transform the data into a graph representation suitable for graph-based learning. For each trajectory segment, a graph is constructed, where nodes represent entities such as players, and edges capture interactions or relationships, such as spatial proximity or passing connections. Each node is assigned a feature vector containing attributes such as *positional* coordinates (x, y), *velocity* components (v_x, v_y), and other contextual features, such as distances to opponents or team affiliation. The graph's connectivity is encoded using an edge index tensor, which is stored alongside the node features. These graph representations preserve both individual characteristics and relational dynamics, enabling GNN to learn from the temporal interactions within the scene. The dataset is then organized into batches, with each batch consisting of graphs corresponding to individual trajectory segments.

Transformer follows the general pipeline with adjustments specific to the architecture. Positional encoding is applied to the input data to explicitly capture temporal dependencies, enhancing the input tensor (*position*, *velocity*) and enabling the attention mechanism to focus on key timesteps. These features are normalized and scaled to ensure relevance for the attention mechanism. The resulting tensors, which combine normalized trajectory features and positional encodings, enable the model to effectively leverage both spatial and temporal relationships for prediction tasks. For the Transformer model specifically, a CNN-based temporal embedding is used to implicitly encode local sequential patterns before the data enters the attention layers, enhancing the representation capacity without relying on explicit positional encodings. This is inspired by Baevski et al. [46], that use convolutional layers to extract temporal features directly from raw sequences, allowing the model to learn local time-dependent structures more effectively.

B. Main Processing

In preliminary studies, we optimized each model individually by varying combinations of input features, including positions, velocities, distances, and context information.[1] Here, we report only the best-performing configurations for each model w.r.t. a fixed input sequence length of 2s.[2]

Constant Velocity naively extrapolates future positions by taking the *velocity* from the last known position and applying it uniformly across all forecast steps.

Linear may capture temporal context dependencies as our flattened sequential input (*velocities*) provides it. Its architecture includes 2 fully connected layers (128 neurons each), followed by ReLU functions and dropout for regularization.

TCNN uses 1D causal convolutions to ensure the output at time t only depends on inputs (*velocities*) at or before t [47]. We found 5 convolutional layers with increasing dilation factors (1, 2, 4, 8) capture long-range temporal dependencies

[1]Note that, if a model only predicts velocities, we reconstruct positions by concatenating them starting from the last known, i.e., initial, position.

[2]In individual tests, we found that no model benefits from an input sequence length longer than 2s, see our discussion in Sec. V.

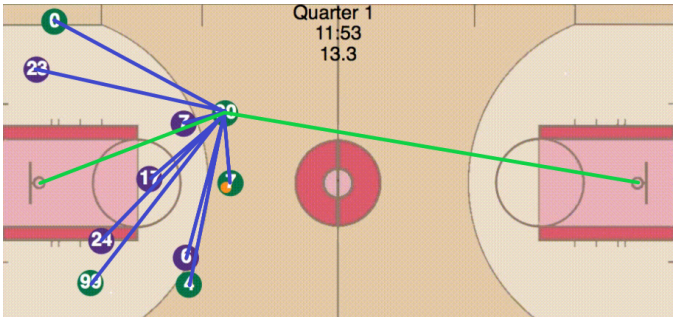


Fig. 2: The game court in the NBA dataset with context.

efficiently. Each layer uses residual connections, weight normalization, and ReLU activations. A Kernel size of 2 and 42 filters control its capacity and receptive field. TCNNs process in parallel, train faster, and maintain temporal causality.

LSTM. Memory cell-based models, such as LSTMs [19], are particularly effective at capturing complex temporal dependencies in sequential data. This architecture allows for the storage and selective processing of information across multiple timesteps, enabling the model to identify and leverage even distant dependencies within the data. In scenarios where past events significantly influence future outcomes, the ability to learn long-term dependencies is crucial. Capturing these relationships facilitates more accurate predictions and trajectory calculations. The operation of an LSTM cell relies on the interaction of several components that regulate the flow of information, including the input gate, forgetting gate, output gate, and cell state. Each of these gates plays a specific role in the process of information storage and processing.

The mathematical description of these cell operations is:

$$\begin{aligned} i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}), \\ f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}), \\ g_t &= tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}), \\ o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}), \\ c_t &= f_t \odot ct - 1 + i_t \odot g_t, \\ h_t &= o_t \odot tanh(c_t), \end{aligned} \tag{1}$$

where h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the input at time t, h_{t-1} is the hidden state of the layer at time $t-1$ or the initial hidden state at time 0, and i_t, f_t, g_t, and o_t are the input, forgotten, hidden, and output gates, with sigmoid function σ and Hadamard product $\odot$.

We implemented a two-layer LSTM model, where the successive arrangement of two LSTM layers enhances the model's ability to capture multi-layered dependencies. The first layer extracts basic patterns from the input data and passes them to the subsequent layer, which models deeper and more complex dependencies. This layered structure enables the model to capture both short-term and long-term dependencies, thereby improving the accuracy of trajectory predictions. Model performance is further enhanced by providing supplementary information about the environment, see Fig. 2.

The model utilizes data based solely on the *velocities* of the target object, teammates, and opponents. However, to provide a more comprehensive understanding of the environment, additional context information regarding the *distances* to neighboring objects is incorporated. The environment comprises both dynamic elements, such as players, and static fixed points. In the case of the NBA dataset, these fixed points include the two basketball hoops, which serve as landmarks. A challenge in incorporating distances is that they can be arbitrarily large between objects. To address this, it is necessary to weight relevant distances more heavily to better represent valuable information. For this purpose, we apply a transformation function to scale the distances to a suitable value range:

$$f(x) = sgn(x) \cdot 2 \cdot e^{-\frac{1}{2}\cdot|x|}. \tag{2}$$

The transformation is applied to the distances to scale them into an appropriate value range. Its purpose is to optimize the influence of the distances on model predictions and enhance the informative value of the environmental properties. By prioritizing shorter distances, this transformation ensures that spatial relationships are more meaningful for the model. The resulting data is then organized into input-output pairs suitable for training the LSTM unit.

CNN-LSTM. To efficiently prepare the context information for the LSTM-based model, a CNN is incorporated into the architecture (CNN-LSTM), as depicted in Fig. 3. The convolutional layer is employed to extract relevant features from the input data, particularly from complex spatial contexts essential for trajectory prediction. By utilizing convolutional layers, the model can identify local dependencies and patterns in the data before these are processed by the temporal-sequential LSTM model. Here, M represents the number of dimensions, N denotes the number of objects (11 players, including 10 players and one ball), T is the length of the input sequence, and $T^{'}$ is the length of the output sequence. v corresponds to

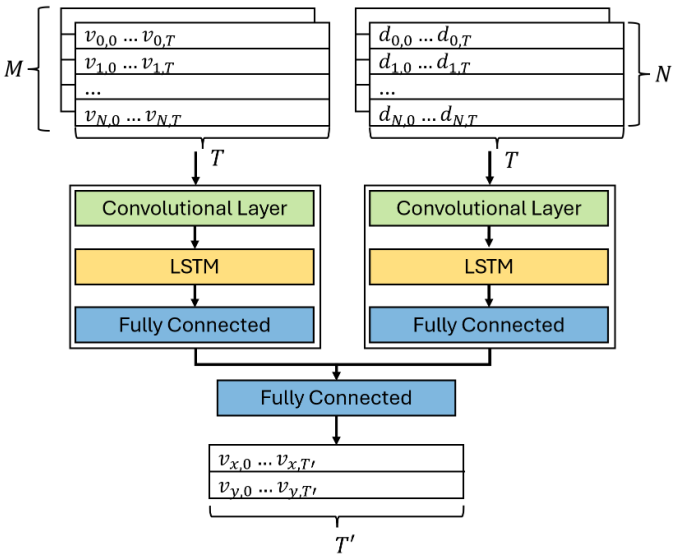


Fig. 3: Pipeline of the CNN-LSTM model.

the velocities of the objects, and d represents the distances to neighboring objects 2. The convolutional layer has a kernel size of (2×5), operating along the time axis, processing two input channels, and producing eight output channels. A LeakyReLU activation function, with a negative gradient factor of 0.1, is applied between the convolutional layer and the subsequent LSTM. The LSTM consists of two layers (hidden size = 128), followed by a linear layer with 128 inputs and 100 outputs for encoding purposes. To integrate the context information, an additional linear layer is added, providing 400 inputs and 100 outputs for predicting the x- and y-coordinates over a two-second period. Note that, the described block represents a module list comprising four distinct information fields. The first field contains the *velocity* information, the second field holds the transformed *distances* to the nearest object, while the third and fourth fields represent the *distances* to the two basketball hoops.

LMU. Although recurrent gating architectures, such as vanilla RNN, LSTM, and GRU, have proven empirically successful in managing hidden states, they do not always provide a direct mathematical foundation for how memory is formed and sustained over time. In contrast, LMUs adopt a more principled state-space approach, grounded in continuous-time filtering. Rather than relying primarily on learned gating mechanisms, LMUs utilize Legendre polynomials to effectively retain and transform past information, offering a theoretically sound framework to address the vanishing gradient problem and capture long-term dependencies.

The LMU [21] relies on approximating a continuous-time delay $F(s) = e^{-\theta s}$ using an ordinary differential equation (ODE) with window length θ:

$$\theta \dot{m}(t) = Am(t) + B\,u(t), \tag{3}$$

where $m \in \mathbb{R}^d$ is the memory state, and $u \in \mathbb{R}$ is the input. The matrices $A \in \mathbb{R}^{d\times d}$ and $B \in \mathbb{R}^d$ encode Legendre polynomial coefficients. These matrices are calculated as

$$A = [a_{ij}]\,,\ a_{ij} = \begin{cases} -1 & \text{if } i < j, \\ (-1)^{i-j+1} & \text{if } i \geq j, \end{cases} \tag{4}$$

$$B = [b_i]\,,\ b_i = (2i+1)(-1)^i, \quad i,j \in [0, d-1]. \tag{5}$$

The Equation (3) is discretized to obtain

$$m_t = \bar{A} m_{t-1} + \bar{B} u_t, \tag{6}$$

where $\bar{A}$ and $\bar{B}$ are the discretized versions of A and B, computed using methods such as Euler's method or zero-order hold. In a standard LMU layer, the memory state m_t is updated using the discretized ODE, and a hidden state h_t is computed by transforming both the input x_t and the memory state by

$$u_t = e_x^T x_t + e_h^T h_{t-1} + e_m^T m_{t-1}, \tag{7}$$

$$m_t = \bar{A} m_{t-1} + \bar{B} u_t, \tag{8}$$

$$h_t = f\big(W_x x_t + W_h h_{t-1} + W_m m_t\big), \tag{9}$$

where e_x, e_h, and e_m are encoding vectors and W_x, W_h, and W_m are learnable weight matrices.

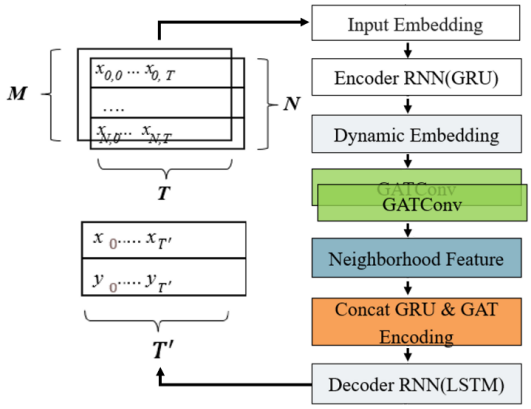


Fig. 4: Pipeline of the GNN model.

Our model utilizes two LMU layers. The first LMU layer transforms the input, which consists of 22 dimensions (11 objects, each with two features, i.e., *positions* and *velocities*), into a hidden size of d. A dropout layer is applied after the first LMU layer, followed by a second LMU layer with the same dimensionality. The final fully connected layer projects the output back to two dimensions, predicting the x- and y-coordinates over the forecast horizon (T'). The hyperparameter d represents the dimensionality of the memory state m and the size of the state-space matrices A and B. In this model, d is set to 256, providing sufficient capacity to capture long-term dependencies while ensuring computational efficiency. The parameter θ defines the window length for the continuous-time delay approximation, controlling how far into the past the memory state m retains information. For this model, θ is set to 25, balancing the ability to capture temporal dependencies with model complexity. This configuration ensures that the LMU effectively represents sequential patterns in the input trajectory while maintaining efficiency.

GNN. To capture temporal dependencies, we enhance our GNN by integrating Graph Attention (GATs) [28], [29]. This integration enables the rich temporal representations generated by a RNN-based model to be processed dynamically in a graph-structured format, allowing GATs to emphasize the most relevant temporal and spatial interactions. GATs, a class of GNNs, utilize attention mechanisms to learn from graph-structured data. By dynamically assigning varying importance weights to neighboring nodes, GATs enable the model to focus on the most pertinent information for each node in the graph. The attention mechanism in GATs is computed using the following attention and aggregation functions: For the node-level attention, a given node i, the attention coefficients $\alpha_{i,j}$ between node i, and its neighbor j are calculated as

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(a^\top [Wh_i \oplus Wh_j]\right)\right)}{\sum_{k\in\mathcal{N}(i)} \exp\left(\text{LeakyReLU}\left(a^\top [Wh_i \oplus Wh_k]\right)\right)}, \} \tag{10}$$

where h_i and h_j are the feature vectors of nodes i and j, W is a learnable weight matrix, a is the attention weight vector, $\oplus$ denotes concatenation, and $\mathcal{N}(i)$ is the set of neighbors of

node i. Weighted aggregation: we compute the updated feature for node i as a weighted sum of its neighbors' features:

$$h_i' = \sigma\Big(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W h_j \Big), \tag{11}$$

where σ is an activation function (e.g., ReLU). For the multi-head attention, to enhance the model's capacity, GATs often employ multiple attention heads by computing

$$h_i' = \oplus_{k=1}^{K} \sigma\Big(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(k)} W^{(k)} h_j \Big), \tag{12}$$

where K is the number of attention heads.

GATs have demonstrated exceptional performance in some graph-based applications [48], providing a powerful mechanism to capture the intricate interactions between entities in dynamic environments. The flexibility of the attention mechanism allows GATs to learn more nuanced relationships compared to traditional GNNs. Our novel RNN-GAT-RNN-based encoder-decoder framework effectively addresses the challenges of predicting player movements in team sports. It combines RNNs and GATs to model both temporal dynamics and inter-player interactions, ensuring accurate predictions.

Fig. 4 shows the overall architecture of the fusion model. The framework is organized into two primary stages: the encoder and the decoder. The encoder comprises two key components: (1) The history encoder, which utilizes a GRU-based RNN [3] to capture sequential dependencies within the historical trajectories of players. Each player's trajectory, represented by x- and y-coordinates over time, is processed through the GRU to generate a latent representation that encodes both individual movement patterns and temporal dynamics. This provides the model with a comprehensive understanding of past movements, including position changes, velocity variations, and directional shifts. (2) The interaction encoder constructs an interaction graph where players are represented as nodes, and edges define relationships or interactions, such as distances or passes. A GAT processes this graph by calculating attention coefficients that determine the significance of neighboring players' movements for each target player. This enhances the representation of inter-player dynamics, producing interaction features that reflect each player's role and influence within the team. The decoder stage consists of a decoder that employs an LSTM-based RNN[3] to predict future trajectories based on the encoded features from the previous stage. Inputs to the decoder include dynamic features from the history encoder and interaction features from the GAT. These inputs are concatenated into a single vector, providing a comprehensive representation of the target player's dynamics and interactions. The decoder then iteratively predicts the target player's next position at each timestep, ultimately generating the entire trajectory. The processing begins with the data representation

$$\text{data} = \{H_t, E_t, y_t\}, \tag{13}$$

[3]In a preliminary study, we optimized the architecture by exploring various RNN encoder-decoder combinations, including vanilla RNN, GRU, LMU, and LSTM. In this paper, we report only the best-performing configuration.

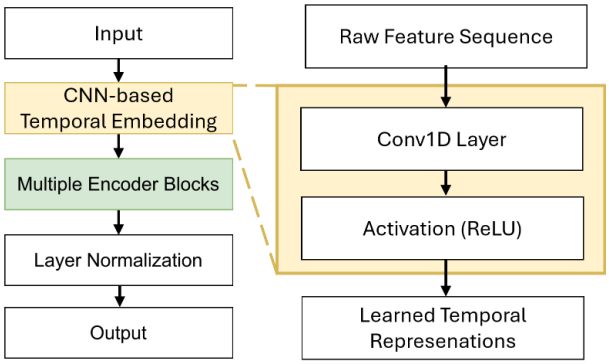


Fig. 5: Architecture (left) with positional encoding (right) of our Transformer model; information flows from top to bottom.

where H_t represents the historical tracks of all players, E_t is the edge set defining the graph structure, and y_t is the ground truth trajectory of the target player. The interaction graph at time t is processed as

$$G_t = \text{GNN}_{\text{inter}}(R_t, E_t), \tag{14}$$

where R_t represents the sequential features *position* and *velocity* of all players at time t, and E_t defines the graph structure at that timestep. To predict future trajectories we combine dynamics r_{t_0} and interaction features g_{t_0} as

$$f_{t_0} = \text{RNN}_{\text{fut}}([g_{t_0}, r_{t_0}]), \tag{15}$$

f_{t_0} represents the predicted trajectory of a target player. Fusion and residual layers further improve predictive accuracy by integrating feature fusion and residual connections. Temporal and relational features are concatenated, merging historical and interaction data, while residual connections reintroduce the original input features at later stages. This architecture allows the model to capture complex patterns effectively and mitigates vanishing gradients.

Transformer Model. RNNs and their variants have demonstrated efficacy in modeling sequential data, owing to their ability to capture temporal dependencies. However, their inherent sequential processing limits computational efficiency and impedes the modeling of long-range dependencies in extended sequences. Although GNNs, especially GATs, excel in modeling spatial relationships by dynamically attending to relevant graph nodes, they seem less effective in capturing temporal information over prolonged time periods.

Fig. 5 shows the architecture of the transformer model. The input sequence is processed through an encoder-only transformer, which mitigates the error accumulation commonly encountered in decoder-based transformer models. This encoder simultaneously processes the entire sequence, utilizing positional embeddings to represent the temporal order of the input data. Specifically, a CNN-based temporal embedding module is used to extract local sequential patterns from the input (*velocity* vectors and *positions*). A 1D convolutional layer with a fixed kernel size transforms the raw features into a richer representation space, captures implicit temporal

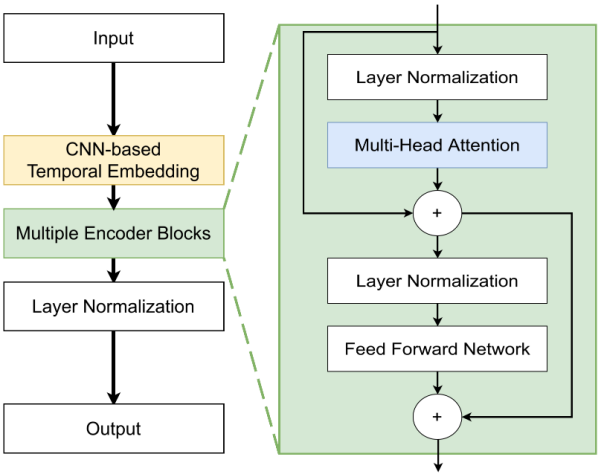


Fig. 6: Visualization of the data flow of an encoder block.

dependencies and preserves the order. This method increases the feature capacity available to the attention mechanism and stabilizes the learning process. Velocities effectively capture motion trends and relative dynamics. However, we found that absolute positions provide essential spatial context that enables the model to better understand formations, spacing, and positional roles. We think, this input strategy is effective for transformer architectures due to their global receptive field. Instead, most recurrent models such as vanilla RNN, GRU, and LSTM perform best on relative features, e.g., velocities.

Each encoder block, see Fig. 6, comprises three main components: a multi-head attention (MHA) mechanism, a feed-forward network (FFN), and layer normalization. The MHA mechanism enables the model to simultaneously attend to different parts of the input sequence, effectively capturing both short-range and long-range dependencies. This is followed by a position-wise FFN with a hidden size of 1,024, which processes the output at each position independently. Residual connections are applied after both the MHA and FFN layers to stabilize the training process and enhance gradient flow. Layer normalization is employed to ensure stability and normalize the data as it propagates through the network.

The MHA mechanism utilizes eight attention heads (num.heads $= 8$) and is computed by

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \tag{16}$$

where Q, K, and V are the query, key, and value matrices, respectively. The dimension of each key vector is $d_k = 4$. These matrices are derived from the input sequence X through linear transformations

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V, \tag{17}$$

with W_Q, W_K, and W_V as learnable weight matrices. We pass the output of the MHA to the feed-forward network

$$\text{FFN}(x) = \sigma(xW_1 + b_1)W_2 + b_2, \tag{18}$$

where W_1, W_2, b_1, and b_2 are learnable parameters, and σ is an activation function, typically ReLU.

The transformer model consists of six encoder blocks (num. encoder blocks $= 6$). Each block has an input embedding size of $d_{\text{model}} = 256$, which matches the dimensionality of the input embeddings and positional encodings. The input sequence length is determined by the trajectory data, with each sequence processed entirely by the encoder.

To train the model effectively, the AdamW optimizer is utilized, incorporating weight decay to mitigate overfitting. The learning rate follows a warm-up schedule:

$$\text{lr}(t) = \frac{d_{\text{model}}^{-0.5} \cdot t}{\text{warmup steps}^{1.5}}, \tag{19}$$

where $d_{\text{model}} = 256$ and t is the current step. The number of warmup steps can be selected according to the specific training regimen. By utilizing these components, including an eight-head MHA mechanism, an FFN with a hidden size of 1,024, six encoder blocks, and a robust training strategy, the transformer model is highly effective in capturing complex player-ball interactions and long-range temporal dependencies.

C. Postprocessing

For models that output relative directed velocities v instead of absolute positions p, we reconstruct the position sequence $p_{i,n}$ from the predicted velocities v. This reconstruction relies on a fixed time step of $\Delta t = 0.04s$ between consecutive positions p_{i-1} and p_i. Each forecasting process begins from a known initial position p_{i-1}, which serves as the starting point for iteratively computing future positions: $p_i = p_{i-1} + v_i \Delta t$.

IV. Experiments

Sec. IV-A provides a detailed description of the dataset, while Sec. IV-B outlines the experimental setup. Error metrics are summarized in Sec. IV-C. The benchmark experiments were conducted on a high-performance system equipped with Tesla V100-SXM2 GPUs, and the models were implemented using *PyTorch* (version 2.4.1).

A. Dataset

We utilize the publicly available NBA dataset [49], which includes data from the 2015-2016 season, comprising 508 hours of playtime recorded at 25fps. To address potential recording errors, interpolation was applied for data smoothing. The dataset is partitioned into three subsets: 70% for training, 20% for validation, and 10% for testing. This partitioning ensures that no games are shared across subsets, enabling the model to be evaluated on completely unseen data for an unbiased performance assessment.

B. Experimental Setup

Experiment 1: Optimal Input Length. The first experiment seeks to determine the optimal input length for the models under evaluation. Specifically, it examines how varying the number of past timesteps used as input influences predictive accuracy by assessing changes in prediction error for different

input sequence lengths. The dataset comprises 128,538 trajectories, each containing 50 predicted timesteps, with various input lengths tested to identify the ideal historical context for accurate forecasting. Multiple input sequence lengths are evaluated to determine the point at which increasing the input length improves accuracy up to a certain threshold, beyond which further inclusion of timesteps either yields only marginal improvements or potentially degrades performance due to overfitting or increased uncertainty. In a preliminary study, we evaluated the effects of various input horizons (up to 10s), i.e., **0 to 2** in steps of 0.05s, and **4**, 6, 8, and 10s, for each model individually. Note that for computationally inefficient models such as GNN and Transformer, we only evaluated the bolded horizons, as no performance improvement was observed beyond 2s. This strategy offers insights into the minimal historical context needed for accurate predictions while minimizing computational overhead. The results from this experiment inform the selection of input length for subsequent evaluations of other models.

Experiment 2: Generalization Within a Unique Team. The second experiment assesses the models' ability to generalize to unseen games while remaining within the same team context. Again, we employ the same dataset with 128,538 trajectories, each containing 2s input- and forecast data.[4] The models are trained on a subset of games from a single team, with the test set comprising games from the same team that were excluded from the training process. This setup evaluates how well the models can predict player trajectories in new game scenarios, having learned the team's specific playing style and dynamics during training. By fixing the input length to 50 timesteps, the experiment establishes consistent conditions for assessing the models' ability to generalize to new yet familiar contexts, thereby emphasizing their robustness in capturing team-specific movement patterns.

Experiment 3: Cross-Team Generalization. This experiment evaluates the models' ability to generalize across different teams by training on data from one team and testing on entirely unseen games from another team. The dataset employs 48,274 trajectories, each with a fixed input length of 50 timesteps and an output length of 50 timesteps, i.e., 2s.[4] This experiment exposes the models to unfamiliar team dynamics, simulating real-world scenarios where the system must adapt to diverse strategic and competitive environments. The results propose applicability of the learned motion patterns and the models' robustness in handling varied team behaviors.

C. Error Metrics

Performance is evaluated using both spatial and angular metrics. Specifically, the final displacement error (FDE) and the average displacement error (ADE) are measured in meters, while the final angular error (FAE) and the average angular error (AAE) are measured in degrees. These metrics collectively assess how closely the predicted trajectory approximates the ground truth in terms of both position and orientation. The FDE and ADE metrics are defined as follows:

$$\text{FDE} = \frac{1}{N}\sum_{i=1}^{N} \|y_{i,T} - \widehat{y}_{i,T}\|, \tag{20}$$

$$\text{ADE} = \frac{1}{N \cdot T}\sum_{i=1}^{N}\sum_{t=1}^{T} \|y_{i,t} - \widehat{y}_{i,t}\|, \tag{21}$$

where $\widehat{y}_{i,t}$ denotes the predicted position at timestep t for sample i, $y_{i,t}$ is the corresponding ground truth, T is the forecasting horizon, and N is the total number of samples. Here, ADE quantifies the average positional mismatch at every timestep, whereas FDE focuses on the final position at $t = T$.

For angular errors, orientation is derived from velocity vectors of the ground truth trajectory $\vec{v}_y$ and the predicted trajectory $\vec{v}_{\widehat{y}}$. First, the cosine of the angle between these vectors is computed by

$$\bar{\theta} = \arccos\Big(\frac{\vec{v}_y \cdot \vec{v}_{\widehat{y}}}{\|\vec{v}_y\| \cdot \|\vec{v}_{\widehat{y}}\|}\Big). \tag{22}$$

A signed angle θ is then obtained to distinguish leftward and rightward orientations:

$$\theta = \begin{cases} \bar{\theta} & \text{if} \quad \vec{v}_y \times \vec{v}_{\widehat{y}} > 0, \\ -\bar{\theta} & \text{else.} \end{cases} \tag{23}$$

To measure orientation accuracy over time, we define AAE as

$$\text{AAE} = \frac{1}{N \cdot T}\sum_{i=1}^{N}\sum_{t=1}^{T} \big|\theta_{i,t}\big|, \tag{24}$$

and the FAE is defined as

$$\text{FAE} = \frac{1}{N}\sum_{i=1}^{N} \big|\theta_{i,T}\big|, \tag{25}$$

where $\theta_{i,t}$ is the signed angle between the ground truth and predicted velocity vectors for sample i at time t. Accordingly, AAE captures the average angular discrepancy at each timestep, and FAE captures the final angular mismatch at $t = T$. These spatial and angular metrics furnish a comprehensive assessment of trajectory prediction performance.

V. Results

We evaluated all models based on the impact of input history length on the forecasting horizon (Sec. V-B), generalization capability, forecasting uncertainty, computational complexity, and the ability to leverage implicit contextual information derived from team dynamics (Sec. V-C - V-D). For all experiments, we evaluated the following models: two classic baselines, a Constant Velocity model and a Linear model, as well as our TCNN, LSTM, CNN-LSTM, LMU, GNN, and Transformer architectures.

A. Optimal Input Features

For a fair comparison, we optimized each model individually w.r.t. input features (*positions*, *velocities*, *distances*). We found that Constant Velocity, Linear, TCNN, LSTM, and CNN-LSTM perform best on *velocities*[1]. Instead, LMU, GNN, and Transformer exploited also *positions*.

[4]We found that no model benefits from input lengths greater than 2s, and the prediction error increases significantly beyond a 2s forecast horizon.

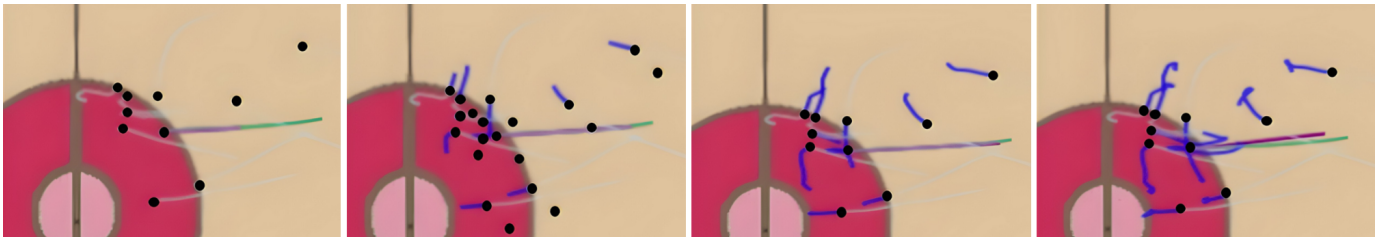


Fig. 7: Comparison of trajectory forecasting for different input lengths (from left to right: 0.04s, 1s, 2s, and 4s). Best results are when the purple curve covers the green curve, e.g., at 2s input length.

B. Experiment 1: Optimal Input Length

This experiment investigates the effect of varying the input sequence length on the accuracy of trajectory prediction. Fig. 8 presents the results, where the horizontal axis represents the forecasting horizon (in seconds) and the left vertical axis denotes the distance error (in meters). Each colored line corresponds to a different input sequence length. The data reveals that shorter input windows, e.g., less than 2s, result in significantly higher error rates, as the model lacks sufficient historical context to make accurate predictions (dark blue colored lines in the top right that represent an input length of 0.15s). As the input length increases, the error decreases 1.62m until it levels off at a plateau around 3.5s (light blue lines in the top-left sub-graph in Fig. 8), indicating that a broader temporal context improves prediction accuracy. Beyond 2s, however, the models no longer benefit significantly from additional input data; the error curve begins to level off, suggesting that further increases in the input window yield diminishing returns in terms of predictive accuracy. For simplicity, Fig. 8 only shows the results of the LSTM model. We observed similar results across all other model architectures: increasing the input length up to 2s consistently reduces error, while additional expansion results in minimal improvements. Thus, we suggest an optimal input length of around 2s for most models, providing a balance between accuracy and computational efficiency, in basketball. Interestingly, at forecast horizons of 2s and beyond, an increase in input length offers little benefit, see the top-left sub-graph, e.g., error variations remain minimal between 1.61 and 1.63m at input lengths between 1.0 and 3.0s. We

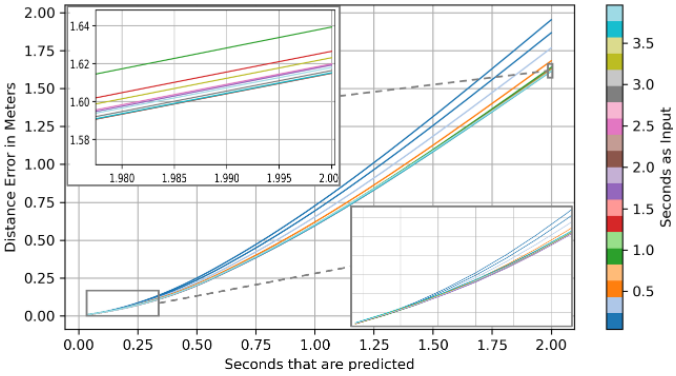


Fig. 8: Evaluation error (in *m*) of different input lengths.

believe this is due to the highly dynamic nature of basketball: players frequently change direction, making motion patterns beyond 2s less predictive. In contrast, preliminary tests on soccer and pedestrian data suggest that significantly longer input horizons are beneficial in those domains, as movements are generally more stable and less prone to sudden backward or lateral changes.

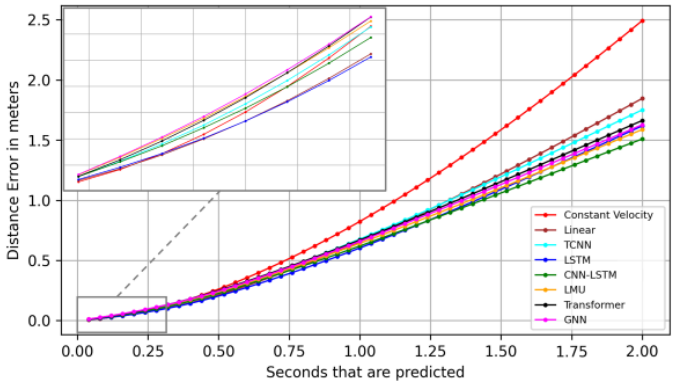
Fig. 7 illustrates the same scenario with varying input lengths (0.04s, 1s, 2s, and 4s). The green line represents the ground truth, while the purple line indicates the predicted trajectory. The black dots denote the positions of individual players, the blue path corresponds to the input trajectory, and the gray lines depict the projected positions of the players over the next two seconds. We observed that the error of the estimated trajectory increases when the input length is significantly smaller than 2s.

C. Experiment 2: Generalization Within a Unique Team

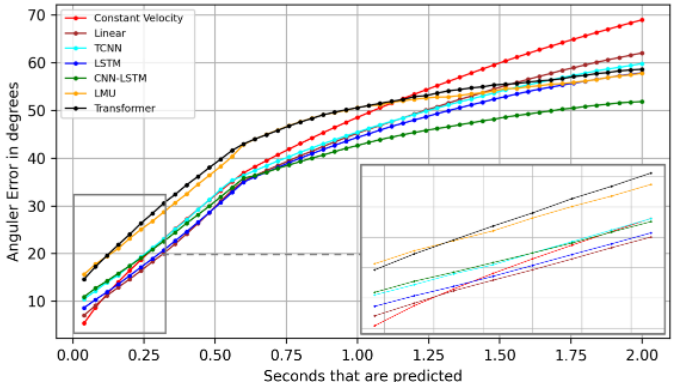
This experiment evaluates the performance of all methods w.r.t. trajectory forecasting on unknown input data, i.e., train and test on a specific team (Lakers). Fig. 9 illustrates the distance error for each model architecture.[5]

For short forecast horizons below 0.5s, all models yield significantly lower errors than for long ones. The results show that even the Constant Velocity and Linear neural network models perform well when motion dynamics and trajectory changes are minimal, i.e., below 0.5s forecast horizons, see Tbl.II and the top-left sub-graph in Fig.9a. Specifically, the Linear model achieved an FDE of 0.20m and an FAE of 28.59°. The sub-graph indicates that up to a 0.12s forecast horizon, both Constant Velocity and Linear models (brown and red curves) outperform all others. However, between 0.12s and 0.30s, the Constant Velocity model fails to capture non-linear motion, and the Linear and LSTM models begin to outperform the rest. Beyond 0.4s, all models outperform the Constant Velocity model. And beyond 1.3s, all models outperform the Linear model. We believe this is as the Constant Velocity and Linear models benefit the least from additional context information,

[5]Note that the curves in Fig. 9b exhibit a distinct elbow. We believe this is likely attributable to characteristics inherent to the dataset or to specific patterns in NBA playstyle. When applying our methods to similar datasets from other domains, such as soccer or pedestrian trajectories, this effect did not occur. Notably, the Constant Velocity baseline also displays this pattern, further suggesting a domain-specific cause.



(a) Trajectory forecasting (in m).



(b) Orientation forecasting (in °).[5]

Fig. 9: Evaluation of all models. An optimized constant velocity KF works best below 0.12,s, after which the dynamic motion becomes nonlinear and ML models are necessary.

TABLE I: Errors at a forecast horizon of 2.0s.

Model	ADE [m]	FDE [m]	AAE [°]	FAE [°]
Constant Velocity	0.99	2.49	45.42	68.98
Linear	0.76	1.85	41.76	62.05
TCNN	0.76	1.75	42.23	59.83
LSTM	0.68	1.62	40.61	57.87
CNN-LSTM	**0.67**	**1.51**	**38.92**	**51.86**
LMU	0.70	1.59	45.38	57.80
Transformer	0.74	1.66	46.12	58.64
GNN	0.71	1.62	45.18	54.20

TABLE II: Errors at a forecast horizon of 0.48s.

Model	ADE [m]	FDE [m]	AAE [°]	FAE [°]
Constant Velocity	0.10	0.24	19.31	31.22
Linear	**0.09**	0.20	**17.56**	**28.59**
TCNN	0.10	0.23	20.47	31.28
LSTM	**0.09**	**0.19**	18.25	28.60
CNN-LSTM	0.10	0.21	20.20	29.94
LMU	0.11	0.23	25.97	36.39
Transformer	0.11	0.24	26.96	38.02
GNN	0.11	0.23	22.10	32.85

as they are unable to capture temporal dependencies due to the lack of recurrent or attention mechanisms.

Out of all methods, our CNN-LSTM benefits the most from supplementary context information. Its combination of CNN and LSTM outperforms pure LSTMs with increasing forecast horizon starting at 1.28s, by up to 11cm at 2s forecast horizon, see Tbl.I. Its distance error (green curve) decreases by 40% from 2.5m (Constant Velocity) to 1.5m, see Fig. 9a. And its orientation error reduces by 25% from 68.98 ° to 51.86 °, see Fig. 9b. Instead, the LMU model achieves a slightly higher FDE of 1.59m. Thus, we think that Legendre polynomials may be worth investigating as part of a hybrid CNN-LMU architecture. Our GNN is almost on-par with LMU and slightly outperforms the Transformer model. We think that this is as GNN exploits both GAT and graph-context. So, it returns an FDE of 1.62m. Interestingly, the Transformer model performs slightly worse (FDE = 1.66m) than the recurrent (LSTM, LMU, CNN-LSTM) and GNN models. We believe that our Transformer model suffers from having too little data for effective training.

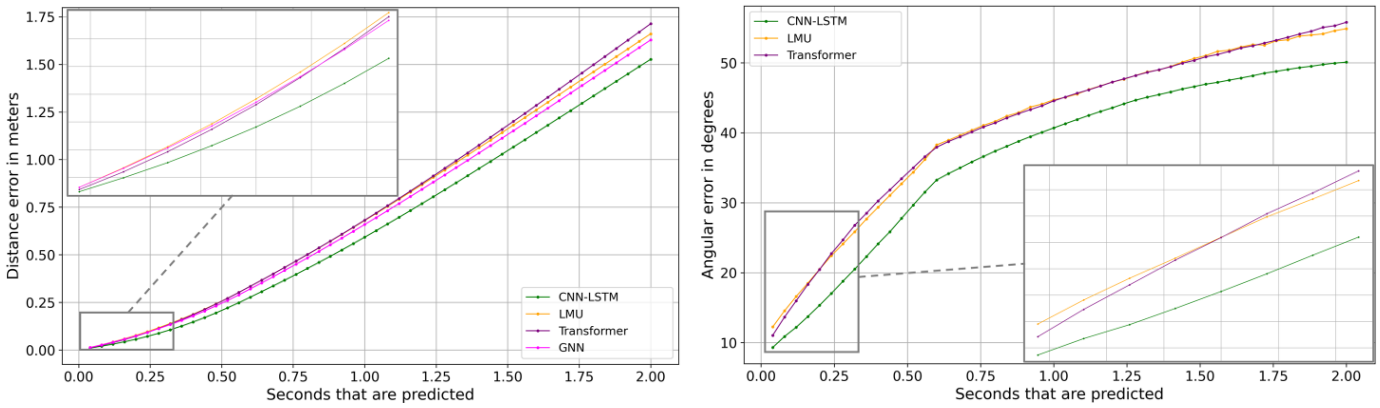
In essence, our findings highlight the importance of selecting models based on the forecast horizon. For very short horizons (up to 0.12s), classical approaches such as a Constant Velocity Kalman Filter may be sufficient. However, for forecast horizons beyond 0.12s, we recommend using recurrent, graph-based, or attention-based models. For horizons exceeding 0.76s, at least an LSTM model should be employed, and beyond 1.3s, a CNN-LSTM architecture is advisable.

D. Experiment 3: Cross-Team Generalization

In this experiment, we only benchmark the best-performing models (CNN-LSTM, LMU, Transformer, and GNN) to predict trajectories for new games featuring unknown players. Initially, the models were trained on data of a specific team. Although new trajectories were provided, the players within each team were always known. This setup enabled the model to learn patterns and correlations specific to a given team without being exposed to unknown player attributes.

Fig. 10 demonstrates the performance of our models trained on Lakers on an unknown match between Cleveland and Houston. The players in this match are also new to the model. The results show that the models are robust to unknown data, with errors nearly identical to those observed in earlier tests, compare Tbl. III. This suggests that the models generalize even when confronted with unfamiliar games and players. Tbl. III presents a comparison between the models of Experiment 2, that were trained and tested on a specific team (Lakers) to the results of Experiment 3.

The minimal differences in error (CNN-LSTM: Δ FDE=0.02m) between the two experiments indicate that the models do not overfit to specific players or teams. Instead, they demonstrate the ability to make robust, generalized predictions even on unknown (left out) data. For readability, this time we only plot methods that exploit context. Notably, our CNN-LSTM model (green curve) outperforms the more complex Transformer and GNN models across all forecast horizons. This suggests that a carefully engineered hybrid architecture



(a) Trajectory forecasting (in m). (b) Orientation forecasting (in $^\circ$).

Fig. 10: Evaluation for unseen games.

TABLE III: Difference in error between Experiment 2 and 3.

Model	**ΔADE [m]**	**ΔFDE [m]**	**ΔAAE [°]**	**ΔFAE [°]**
CNN-LSTM	**0.01**	**0.02**	**0.87**	**0.68**
LMU	0.04	0.07	4.02	2.92
Transformer	0.02	0.05	4.70	2.84
GNN	0.04	0.03	2.47	2.78

may outperform attention-based models such as GNNs (with GAT) and Transformers, particularly in data-scarce settings and dynamic sports.

VI. Summary

This paper evaluated various classic and ML-based forecasting models, including Constant Velocity, Linear neural network, LSTM, LMU, TCNN, GNN (with GAT), and Transformers, in the context of NBA player position forecasting.

Our benchmarks for determining the optimal input sequence length indicate that a duration of approximately 2s is ideal for most architectures. Shorter input windows lack sufficient historical context for accurate predictions, while longer windows yield diminishing returns in predictive accuracy.

To evaluate the effects of unseen trajectories on our models, we benchmark them in a single-team context, i.e., a random split. For short forecast horizons ($<$ 0.5s) all models perform well, with minimal error differences. Constant Velocity and Linear models perform best up to 0.12s, thanks to the minimal trajectory changes in this range. At medium forecast horizons (0.12–0.4s) Constant Velocity begins to falter due to its inability to capture non-linear motion. Linear and LSTM models start to outperform others. And at long forecast horizons ($>$ 0.4s) all models surpass the Constant Velocity baseline. Beyond 1.3s, even the Linear model is outperformed by all others, highlighting its limitations in modeling complex temporal dynamics. Especially our CNN-LSTM Model shows the strongest performance across long horizons, benefiting the most from additional context. It reduces FDE by 40% (from 2.5mto 1.5m), outperforming pure LSTM models beyond 1.28s. Our LMU model achieves slightly higher FDE (1.59m) than CNN-LSTM, indicating strong but slightly inferior performance. This suggests potential in combining CNN and LMU using Legendre polynomials. The GNN model performs comparably to LMU (FDE = 1.62m) and outperforms the Transformer, likely due to its ability to leverage both attention and graph-context information. And our Transformer model slightly underperforms (FDE = 1.66m). We believe this is likely due to insufficient training data, which limits the model's ability to fully leverage attention mechanisms.

We also assessed the generalizability of our temporal context models (LMU, CNN-LSTM, GNN, Transformer) by testing their performance on unknown (left out) games and players from different teams. The CNN-LSTM model offers robust performance (lowest FDE of 1.53m), with marginal increases in error (FDE by 0.02m). We observed similar trends for the LMU (0.07m), Transformer (0.05m), and GNN (0.03m) models. So, our models adapt to unfamiliar team dynamics.

Our findings highlight the importance of integrating context information for accurate and generalizable player position forecasting. Moreover, our models help mitigate delays inherent in modern signal processing pipelines, offering practical applications in real-time sports analytics and other domains that demand precise motion forecasting.

References

[1] U. Brefeld and A. Zimmermann, "Guest editorial: Special issue on sports analytics," in *Data Mining and Knowledge Discovery Jo.*, vol. 31, no. 7, 2017, pp. 1577–1579.

[2] J. Zou, Q. Zhao, Y. Jiao, H. Cao, Y. Liu, Q. Yan, E. Abbasnejad, L. Liu, and J. Q. Shi, "Stock Market Prediction via Deep Learning Techniques: A Survey," 2023.

[3] S. Siami-Namini and A. S. Namin, "Forecasting Economics and Financial Time Series: ARIMA vs. LSTM," in *arXiv:1803.06386 [cs.CV]*, 2018.

[4] F. Ott, D. Rügamer, L. Heublein, B. Bischl, and C. Mutschler, "Joint Classification and Trajectory Regression of Online Handwriting using a Multi-Task Learning Approach," in *Proc. IEEE Conf. Winter Conf. on Applications of Computer Vision (WACV)*, Waikoloa, HI, 2022.

[5] W. Zhang, F. Cheema, and D. Srinivasan, "Forecasting of Electricity Prices Using Deep Learning Networks," in *Asia Pacific Power and Energy Engineering Conf. (APPEEC)*, 2018, pp. 451–456.

[6] G. Jain and S. Singh, "A Review on Weather Forecasting Techniques," in *IJARCCE*, vol. 5, 2016, pp. 177–180.

[7] F. Ott, T. Feigl, C. Löffler, and C. Mutschler, "ViPR: Visual-Odometry-aided Pose Regression for 6DoF Camera Localization," in *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Seattle, WA, 2020, pp. 187–198.

[8] F. Ott, L. Heublein, D. Rügamer, B. Bischl, and C. Mutschler, "Fusing Structure from Motion and Simulation-Augmented Pose Regression from Optical Flow for Challenging Indoor Environments," in *Jo. Visual Communication and Image Representation (JVCIR)*, vol. 104256, 2024.

[9] E. Morgulev, O. H. Azar, and R. Lidor, "Sports Analytics and the Big-data Era," in *Intl. Jo. Data Science and Analytics*, vol. 5, no. 1, 2018, pp. 213–222.

[10] M. W. Lam, "One-match-ahead Forecasting in Two-team Sports with Stacked Bayesian Regressions," in *Jo. Artificial Intelligence and Soft Computing Research*, vol. 8, no. 3, 2018, pp. 159–171.

[11] T. Feigl, "Daten-getriebene Methoden zur Bestimmung von Position und Orientierung in Funk- und Trägheits-basierter Koppelnavigation," Ph.D. dissertation, FAU Erlangen-Nürnberg, 2021.

[12] Z. Wei, X. Zhu, B. Dai, and D. Lin, "Rethinking Trajectory Prediction via Team Game," in *arXiv:2210.08793 [cs.CV]*, 2022, pp. 1–13.

[13] L. Zhang, Q. She, and P. Guo, "Stochastic Trajectory Prediction with Social Graph Network," in *arXiv:1907.10233 [cs.CV]*, 2019, pp. 1–12.

[14] W.-J. Chen, M.-J. Jhou, T.-S. Lee, and C.-J. Lu, "Hybrid Basketball Game Outcome Prediction Model by Integrating Data Mining Methods for the National Basketball Association," in *Entropy Jo.*, vol. 23, no. 477, 2021, pp. 1–17.

[15] D. Brüggemann, S. Bracke, H. Gottschalk, M. Rottmann, K. Maag, R. Chan, and M. Schubert, "Ansätze zur Verbesserung KI-basierter Systeme für das autonome Fahren," in *Qualitätsmanagement in den 20er Jahren - Trends und Perspektiven*, B. Leyendecker, Ed., 2021, pp. 100–119.

[16] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," University of North Carolina at Chapel Hill, USA, Tech. Rep., 1995.

[17] P. Djuric, J. Kotecha, J. Zhang, Y. Huang, T. Ghirmai, M. Bugallo, and J. Miguez, "Particle Filtering," in *IEEE Signal Processing Magazine*, vol. 20, no. 5, 2003, pp. 19–38.

[18] D. K. Stephanos, G. Husari, B. T. Bennett, and E. Stephanos, "Machine Learning Predictive Analytics for Player Movement Prediction in NBA: Applications, Opportunities, and Challenges," in *Proc. Intl. Conf. ACM Southeast (ACM SE)*, 2021, pp. 2–8.

[19] R. C. Staudemeyer and E. R. Morris, "Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks," in *arXiv:1909.09586 [cs.CV]*, 2019, pp. 1–32.

[20] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," in *arXiv:1412.3555 [cs.NE]*, 2014, pp. 1–9.

[21] A. Voelker, I. Kajić, and C. Eliasmith, "Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks," in *Proc. of Conf. Advances in Neural Information Processing Systems (NIPS)*, 2019, pp. 15 544–15 553.

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *arXiv:1706.03762 [cs.CV]*, 8 2017.

[23] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A Comprehensive Survey on Graph Neural Networks," in *IEEE Trans. on Neural Networks and Learning Systems*, vol. 32, no. 1. IEEE, 2020.

[24] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang, "Connecting the dots: Multivariate time series forecasting with graph neural networks," in *Proc. Intl. conf. on knowledge discovery and data mining (KDD)*, 2020, pp. 753–763.

[25] Y. Xu, L. Wang, Y. Wang, and Y. Fu, "Adaptive Trajectory Prediction via Transferable GNN," in *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 6520–6531.

[26] Scarselli, Franco and Gori, Marco and Tsoi, Ah Chung and Hagenbuchner, Markus and Monfardini, Gabriele, "The graph neural network model," in *IEEE Trans. on Neural Networks*, vol. 20, no. 1, 2009, pp. 61–80.

[27] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," in *Expert Systems with Applications*, vol. 207, 2022, p. 117921.

[28] X. Mo, Y. Xing, and C. Lv, "Graph and Recurrent Neural Network-based Vehicle Trajectory Prediction for Highway Driving," in *Proc. Intl. Conf. IEEE Intelligent Transportation Systems Conf. (ITSC)*, 2021, pp. 1934–1939.

[29] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio *et al.*, "Graph attention networks," in *stat*, vol. 1050, no. 20, 2017.

[30] E. Aksan, M. Kaufmann, P. Cao, and O. Hilliges, "A spatio-temporal transformer for 3d human motion prediction," in *3DV*, 2021, pp. 565–574.

[31] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *arXiv:1810.04805 [cs.CV]*, 2019.

[32] N. Carion, F. Massa, G. Synnaeve, R. Cadene, and C. Schmid, "End-to-end object detection with transformers," in *European Conf. on Computer Vision (ECCV)*, 2020.

[33] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, and B. Schiele, "An image is worth 16x16 words: Transformers for image recognition at scale," in *arXiv:2010.11929 [cs.CV]*, 2020.

[34] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human Trajectory Prediction in Crowded Spaces," in *Proc. Intl. Conf. IEEE Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 961–971.

[35] A. Mohamed, K. Qian, M. Elhoseiny, and C. Claudel, "Social-stgcnn: A Social Spatio-temporal Graph Convolutional Neural Network for Human Trajectory Prediction," in *Proc. Intl. Conf. IEEE Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 14 424–14 432.

[36] X. Li, X. Ying, and M. C. Chuah, "Grip++: Enhanced Graph-based Interaction-aware Trajectory Prediction for Autonomous Driving," in *arXiv:1907.07792 [cs.CV]*, 2019, pp. 1–16.

[37] I. Goodfellow, Y. Bengio, and A. Courville, *Convolutional Networks*. MIT Press, 2016.

[38] L. Alzubaidi, J. Zhang, A. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. Fadhel, M. Al-Amidie, and L. Farhan, "Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions," in *Jo. of Big Data*, vol. 8, no. 2, 2021, pp. 1–74.

[39] Aksan, Emre and Kaufmann, Manuel and Hilliges, Otmar, "Structured prediction helps 3d human motion modelling," in *Proc. IEEE Conf. Winter Conf. on Applications of Computer Vision (WACV)*, 2019, pp. 7144–7153.

[40] N. Nikhil and B. T. Morris, "Convolutional Neural Network for Trajectory Prediction," in *Proc. European Conf. on Computer Vision (ECCV)*, 2019, pp. 186–196.

[41] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," in *Neural Computation Jo.*, vol. 9, no. 8, 1997, pp. 1735–1780.

[42] F. Giuliari, I. Hasan, M. Cristani, and F. Galasso, "Transformer Networks for Trajectory Forecasting," in *arXiv:2003.08111 [cs.CV]*, 2020, pp. 1–14.

[43] E. V. Mascaro, S. Ma, H. Ahn, and D. Lee, "Robust Human Motion Forecasting using Transformer-based Model," in *Proc. Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2022, pp. 10 674–10 680.

[44] X. Mo, Y. Xing, and C. Lv, "Interaction-aware Trajectory Prediction of Connected Vehicles Using CNN-LSTM Networks," in *Proc. Intl. Conf. IEEE Industrial Electronics Society (IECON)*, 2020, pp. 5057–5062.

[45] Z. Almahmoud, P. D. Yoo, O. Alhussein, I. Farhat, and E. Damiani, "A Holistic and Proactive Approach to Forecasting Cyber Threats," in *Nature Scientific Reports Jo.*, vol. 13, no. 1, 2023, pp. 8049–8061.

[46] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations," in *CoRR*, vol. abs/2006.11477, 2020.

[47] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in *Proc. Intl. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 156–165.

[48] H. Cheng, M. Liu, L. Chen, H. Broszio, M. Sester, and M. Y. Yang, "Gatraj: A graph-and attention-based multi-agent trajectory prediction model," *ISPRS JPRS*, vol. 205, pp. 163–175, 2023.

[49] K. Linou, D. Linou, and M. de Boer. (2016) NBA-Player-Movements Dataset. [Online]. Available: https://github.com/linouk23/NBA-Player-Movements