Shot Classification & Goal Probability Estimation Using Graph Neural Networks

Master's Thesis

(MSc.) Data Science

Pedro Satorre-Mulet[1]

Supervisors

Dr. Gabriel Anzer[2]
Prof. Dr. Tobias Scheffer[1]

[1]Institut für Informatik und Computational Science, Universität Potsdam, Germany
[2]Department of Soccer Data Analytics, RB Leipzig e.V., Germany

satorremulet@uni-potsdam.de
gabriel.anzer@redbulls.com
tobias.scheffer@uni-potsdam.de

January 27, 2024

Abstract

This thesis investigates the efficacy of Heterogeneous Graph Transformers (HGTs) in soccer analytics, focusing on shot classification and goal probability estimation. Central to this research were 3 questions: the capability of HGTs in classifying soccer shots as goals or non-goals, their proficiency in calibrating shot classification probabilities, and their comparative performance against a traditional Expected Goals (xG) model using the ROC AUC-score and Brier-score metrics. The findings, derived from implementing and comparing a HGT model, by injecting into it simpler or more complex graphs, and a (regularized logistic regression) xG model, revealed that the HGT model dealing with complex graphs significantly outperformed the others in both ROC AUC-score and Brier-score, with mean values $\overline{\text{AUC}}_{\text{HGT}_{\text{complex}}} = 0.679 \pm 1.50 \times 10^{-3}$ and $\overline{\text{Brier}}_{\text{HGT}_{\text{complex}}} = 0.155 \pm 6.41 \times 10^{-3}$. The worst-performing model was the xG model with mean values $\overline{\text{AUC}}_{\text{xG}} = 0.613 \pm 1.95 \times 10^{-6}$ and $\overline{\text{Brier}}_{\text{xG}} = 0.246 \pm 2.41 \times 10^{-4}$. This suggests that the sophisticated architecture of HGTs, proficient at capturing complex relationships within graph-structured data, offers improvements in predicting soccer shot outcomes. The study highlights the necessity for more precise event data, and suggests future exploration in enhanced HGT architectures and usage of GPUs. This thesis contributes to soccer analytics by being, to the best of my knowledge, the first to apply Graph Neural Networks (GNNs), particularly HGTs, in shot classification and goal probability calibration, and comparing the performance of these with traditional xG models. The findings support the hypothesis that utilizing HGTs can enhance shot classification and goal probability calibration in comparison with the traditional xG model, offering a novel perspective in a field that is rapidly evolving.

Keywords: Heterogeneous Graph Transformers (HGTs) · Shot Classification · Goal Probability Estimation · Expected Goals (xG)

Zusammenfassung

Diese Arbeit untersucht die Wirksamkeit von Heterogenen Graphen Transformatoren (HGTs) in der Fußball-Analytik, wobei der Schwerpunkt auf der Klassifizierung von Schüssen und der Schätzung von Torwahrscheinlichkeiten liegt. Im Mittelpunkt dieser Arbeit standen drei Fragen: die Fähigkeit von HGTs, Fußballschüsse als Tore oder Nicht-Tore zu klassifizieren, ihre Fähigkeit zur Kalibrierung von Schussklassifizierungswahrscheinlichkeiten und ihre vergleichende Leistung mit einem traditionellen Expected Goals (xG)-Modell unter Verwendung der ROC AUC-score und Brier-score Metriken. Die Ergebnisse, die sich aus der Implementierung und dem Vergleich eines HGT-Modells, in das einfachere oder komplexere Graphen eingespeist wurden, und eines xG-Modells (regularisierte logistische regression) ergaben, dass das HGT-Modell, das komplexe Graphen behandelt, die anderen Modelle sowohl im ROC AUC-score als auch im Brier-score deutlich übertraf, mit Mittelwerten $\overline{\text{AUC}}_{\text{HGT}_{\text{complex}}} = 0.679 \pm 1.50 \times 10^{-3}$ und $\overline{\text{Brier}}_{\text{HGT}_{\text{complex}}} = 0.155 \pm 6.41 \times 10^{-3}$. Das am schlechtesten abschneidende Modell war das xG-Modell mit Mittelwerten $\overline{\text{AUC}}_{\text{xG}} = 0.613 \pm 1.95 \times 10^{-6}$ und $\overline{\text{Brier}}_{\text{xG}} = 0.246 \pm 2.41 \times 10^{-4}$. Dies deutet darauf hin, dass die technisch ausgefeilte Architektur von HGTs, die komplexe Beziehungen in graphenstrukturierten Daten erfassen kann, Verbesserungen bei der Vorhersage von Fußballergebnissen bietet. Die Studie unterstreicht die Notwendigkeit präziserer Ereignisdaten und schlägt vor, in Zukunft verbesserte HGT-Architekturen und den Einsatz von GPUs zu erforschen. Diese Arbeit leistet einen Beitrag zur Fußball-Analytik, da sie meines Wissens die erste ist, die Graph Neural Networks (GNNs), insbesondere HGTs, bei der Schussklassifizierung und Torwahrscheinlichkeitskalibrierung einsetzt und die Leistung dieser mit traditionellen xG-Modellen vergleicht. Die Ergebnisse unterstützen die Hypothese, dass die Verwendung von HGTs die Schussklassifizierung und die Kalibrierung der Torwahrscheinlichkeit im Vergleich zum traditionellen xG-Modell verbessern kann, und bieten eine neue Perspektive in einem Bereich, der sich schnell entwickelt.

Schlüsselwörter: Heterogene Graphen Transformatoren (HGTs) · Schuss-Klassifizierung · Schätzung der Zielwahrscheinlichkeit · Expected Goals (xG)

Acknowledgements

I want to say thank you to Universität Potsdam for accepting me into the MSc. Data Science program as an international student. I would also like to thank Hertha BSC for granting me access to their broadcast tracking data and derived possessions data, worth 361 soccer matches from a mixture of Bundesliga (German's soccer national league) and 2. Bundesliga matches.

I want to extend my gratitude to Prof. Dr. Scheffer and the entire staff at the Institute of Computer Science at Universität Potsdam for their support and guidance. Their empathy and dedication were immensely appreciated, especially during times when it was most needed. I would also like to direct a heartfelt thank you to Gabriel for providing me with this unbelievable opportunity. His belief in my abilities, guidance throughout the duration of this thesis, and support during personal and family hardships have been invaluable.

Above all, I want to thank my family for all their unconditional and relentless support, care, affection, and love, regardless of our constant hardships. I love you with all my heart!

Declaration of originality

I confirm that this assignment is my own work, is not copied from any other person's work (published or unpublished), and has not previously been submitted for assessment either at the University of Potsdam or elsewhere.

I hereby certify that I have prepared this thesis without the help of third parties and without the use of sources and aids other than those indicated. The passages taken verbatim or in terms of content from the sources used are identified as such.

I have taken note of the "Guideline for Ensuring Good Scientific Practice for Students at the University of Potsdam (Plagiarism Guideline) - dated October 20, 2010", available online at http://uni-potsdam.de/ambek/ambek2011/1/Seite7.pdf.

I further accept that this work can be checked for plagiarism with plagiarism detection software.

Date: 27 / 01 / 2024

Name: Pedro Satorre-Mulet

Contents

Chapter 1

Introduction

This chapter introduces a brief summary of the motivation for this research in section 1.1, it continues by stating in section 1.2 the research questions tackled by this research, followed in section 1.3 by the formulation of the hypothesis of this thesis, and then section 1.4 exposes the structure that this thesis follows, quickly outlining the content of each chapter throughout the document.

1.1 Motivation

Soccer is far more than a sport watched and played by millions; it is a complex system that has intrigued coaches, analysts, and researchers alike. Soccer, also known as the "beautiful game", is a rich field of study not just for sports analysts but increasingly for data scientists; so much so, that in the last decade the number of soccer clubs and federations building up their own internal departments of soccer data science and analytics is becoming greater and greater; something that previously it was only to the reach of the most wealthy large clubs, now it is becoming democratised across clubs and federations of all sizes. With the advent of technologies that can track every move of the ball and players, soccer has become a hotbed for the application of data science to soccer data analytics. Traditionally, tabular event data such as player statistics have been the mainstay of analytics in sports, but with advances in technology, the granularity of available data for analysis has increased abundantly. Metrics that once relied on post-match statistics have now shifted to real-time tracking data, providing a plethora of information at a frequency of 25Hz, in other words, at 25 frames per second. Although traditional statistics and analytics methods offer some insights, they lack the capability to adapt and learn from the highly dynamical nature of soccer. However, the rise and exponential development of machine learning (ML) and deep learning (DL) algorithms is paving the way for more refined analyses. The recent arise of geometric deep learning (GDL) and the introduction of Graph Neural Networks (GNNs) in this field aims to bridge this gap by enabling the algorithm to consider each player and the ball as nodes in a complex graphical network, thus adding more depth to the analysis [1; 2]. Frame-wise soccer analytics, which evaluate every snapshot of a match, offer new perspectives on match dynamics and are the cornerstone of this

research.

Scoring goals is one of the most critical and thrilling aspects of soccer, since it is such a scarce yet pivotal event that can alter the outcome of the match in a split second. Its predictability is a complex phenomenon to analyze, and despite its significance and impact in this sport, our understanding of the underlying complexities that lead to a goal remains limited. While statistical models, like the Expected Goals (xG) metric, have been developed to measure the probability of a goal being scored from a specific location on the pitch using event data, while insightful, these metrics often lack the spatio granularity and do not encapsulate the intricate networked context involving all players and the ball, and thus possible relationships that might lead to a goal being scored (or not). In attempting to classify shots and predict goal probability in a frame-wise manner, given that each frame captures a unique combination of player and ball positions, and other (player and ball) physical and tactical features, this research is motivated by the ambition of leveraging the capabilities of advanced GDL techniques, like GNNs, on frame-wise soccer match tracking data, exploring the predictive power of GNNs in such multi-relational data, and allowing for a microscopic understanding of the match dynamics.

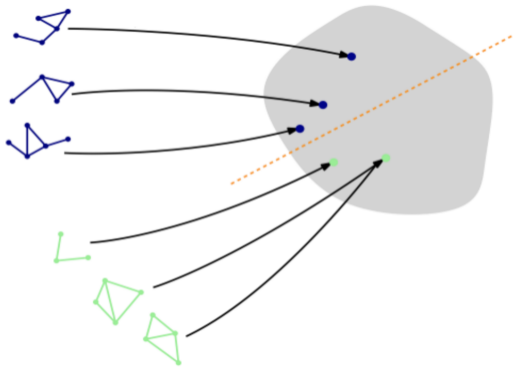


Figure 1.1: Depiction of the prediction task of binary graph classification [3].

The academic landscape for soccer analytics is relatively young, but regardless it is rich as it has been rapidly evolving in the last couple of decades. Numerous academic papers have delved into soccer analytics, and into applying advanced ML and DL techniques and models to the context of soccer. Some of the pioneering and innovative works in goal probability prediction were [4; 5; 6], which laid the foundation for and introduced the Expected Goals (xG) model as a method for predicting goal probability based on tabular event shot data. More recent advancements in soccer analytics have also begun to focus on the application of Graph Neural Networks making use of match tracking data. The work [2] uses GNNs to analyze and to try to predict successful counterattacks in soccer. This paper, together with [7] which employs Convolutional Neural Networks (CNNs) to predict goal probability but concludes by suggesting that perhaps GNNs could be a possibility of future work and improvement on this task, act as important academic precursors to this research, as they serve as inspiration for the utilization of GNNs for the task of goal probability prediction at a frame-wise level. Moreover, GNNs have been explored in various fields of research,

such as in Natural Language Processing (NLP) [8], bioinformatics [9], and traffic forecasting [10], but their application in soccer analytics is still an embryonic area of research. Thus, due to the complex spatial nature of this sport, I believe it is of crucial importance to investigate how proficient GNNs, Heterogeneous Graph Transformer Networks (HGTs) in particular, perform in trying to predict, via binary graph classification (Figure 1.1), the ultimate aim of this sport, capable of inducing goosebumps and moving millions of people's emotions — scoring a goal.

1.2 Research Questions

1. How well can an HGT classify soccer shots as non-scored shots and scored shots (i.e. goals)?
2. How well can an HGT calibrate the probabilities of these shot classifications?
3. Can an HGT outperform a simple traditional xG model at these tasks when evaluated on the ROC AUC-score and the Brier-score?

1.3 Hypotheses

The null hypothesis (H_0) is that there will be no significant difference in the performance (measured by the mean ROC AUC-score and mean Brier-score) of the "complex" HGT model, the "simple" HGT model, and the xG (regularized logistic regression) model when evaluated on the same dataset.

$$H_0 : \overline{\text{AUC}}_{\text{HGT}_{\text{complex}}} \approx \overline{\text{AUC}}_{\text{HGT}_{\text{simple}}} \approx \overline{\text{AUC}}_{\text{xG}} \quad \text{and} \quad \overline{\text{Brier}}_{\text{HGT}_{\text{complex}}} \approx \overline{\text{Brier}}_{\text{HGT}_{\text{simple}}} \approx \overline{\text{Brier}}_{\text{xG}}$$

The primary hypothesis (H_1) is that the "complex" HGT model will outperform with significant differences both the "simple" HGT model and the xG model on both evaluation metrics.

$$H_1 : \overline{\text{AUC}}_{\text{HGT}_{\text{complex}}} > \overline{\text{AUC}}_{\text{HGT}_{\text{simple}}} > \overline{\text{AUC}}_{\text{xG}} \quad \text{and} \quad \overline{\text{Brier}}_{\text{HGT}_{\text{complex}}} < \overline{\text{Brier}}_{\text{HGT}_{\text{simple}}} < \overline{\text{Brier}}_{\text{xG}}$$

The first sub-hypothesis ($H_{1\text{a}}$) is that the "complex" HGT model will demonstrate superior performance compared to the "simple" HGT model in terms of both metrics.

$$H_{1\text{a}} : \overline{\text{AUC}}_{\text{HGT}_{\text{complex}}} > \overline{\text{AUC}}_{\text{HGT}_{\text{simple}}} \quad \text{and} \quad \overline{\text{Brier}}_{\text{HGT}_{\text{complex}}} < \overline{\text{Brier}}_{\text{HGT}_{\text{simple}}}$$

The second sub-hypothesis ($H_{1\text{b}}$) is that the "simple" HGT model will perform better than the xG model in terms of both metrics.

$$H_{1\text{b}} : \overline{\text{AUC}}_{\text{HGT}_{\text{simple}}} > \overline{\text{AUC}}_{\text{xG}} \quad \text{and} \quad \overline{\text{Brier}}_{\text{HGT}_{\text{simple}}} < \overline{\text{Brier}}_{\text{xG}}$$

1.4 Content Summary

The subsequent chapters of this thesis are structured for a systematic exploration of the topic as follows: chapter 2 provides an exhaustive review of the existing literature on traditional and modern metrics and models in soccer analytics, for whoever is unfamiliar with the discussed topics, with a special focus on goal probability prediction in section 2.2, highlighting the capabilities and limitations of each of these, both to position this research within the existing academic landscape and to identify gaps that this investigation aims to fill in; section 2.3 displays an extensive technical theoretical background on related topics to this research, once again, for whoever is unfamiliar with the concepts. Chapter 3 will offer a comprehensive description of the dataset utilized, followed by an explanation of the unique challenges it poses. Moreover, the methodology adopted in this investigation for data handling and cross-validation are outlined; the specific model architectures that were tested, and the training and evaluation protocols are showcased, together with the evaluation metrics used to test them. Later, chapter 4 is dedicated to presenting the models' evaluation results, comparing the performance of the Expected Goals (xG) model, on match tracking data (instead of traditionally on event data), with 2 HGTs [11] from the *PyTorch Geometric* (PyG) library [12]. Further on, chapter 5 is dedicated to interpreting the results exposed in the previous chapter, while also discussing potential limitations and possible improvements to make; directions for future research are also suggested. Finally, chapter 6 concludes this thesis by summarizing the findings.

Chapter 2

Related Work

The objective of this chapter is to situate the reader into the context of this research in case the reader is unfamiliar with the addressed topics that set the bases of this thesis. This chapter starts by displaying in section 2.1 a glossary of key terminology and concepts that are used extensively throughout this investigation. Subsequently, section 2.2 outlines a literature review showcasing the development of soccer analytics over time — from traditional, rudimentary metrics and models to more elaborate ones, leveraging machine and deep learning techniques. In subsection 2.2.3, some of these models that make use of machine and deep learning techniques are exposed, with a special mention to those few works that exist so far in the literature which have utilized Graph Neural Networks (GNNs) in the context of soccer. This chapter ends by presenting a theoretical background on GNNs in section 2.3.

2.1 Key Terminology

In this section, I present an alphabetical list of definitions and explanations of essential technical terminology and concepts, to facilitate the comprehension of this thesis' content:

- **Activation Function:** A mathematical gate within Neural Networks (NNs) between the input feeding the current neuron and its output going to the next layer. It is a non-linear transformation that decides whether a neuron should be activated or not, essentially determining the output of that neuron given a set of inputs. Hence, an activation function introduces non-linear properties to the NN, which allows it to learn and perform more complex tasks, and helps regulating the flow of information through the NN by controlling how much of the signal from the input is allowed to pass to the next layer. This is analogous to deciding how much of the neuron's computed sum is meaningful to contribute to the NN's learning. The choice of activation function can significantly affect the performance and training efficiency of a NN.

- **Attention:** Mechanism that enables Neural Network (NN) models to weigh the relevance of different input features differently, providing a means to focus selectively on parts of

the data that are most pertinent to the task at hand. This concept is inspired by human attention, where focus is adjusted dynamically to process the most pertinent information in a given context. The mechanism works by assigning a relevance score to each part of the input data — these scores, often resulting from a trainable component of the NN model, dictate the amount of "attention" or weighting each part receives. The NN can then give preferential treatment to more informative features during the learning process, enhancing its ability to make accurate predictions or generate relevant outputs. Attention mechanisms are a crucial part of many recent state-of-the-art models.

- **Cross-Validation (CV):** Robust statistical technique used in Machine Learning (ML) to assess the generalizability of a predictive model. It involves partitioning the original dataset into sets of "training" and "testing" data multiple times in different ways to validate the model in a robust manner.

- **Data-Loaders:** A component or utility designed to efficiently load and preprocess data from a dataset into a format suitable for analysis or model training. They play a crucial role in managing data workflow in Machine Learning (ML) pipelines. Data-Loaders often include functionality to load data in batches. Batch processing is essential in ML, especially when dealing with large datasets that cannot be loaded entirely into memory at once. It helps in efficient memory management and can speed up the training process by loading data asynchronously, meaning they load the next batch of data in the background while the current batch is being processed. This parallel processing helps in optimizing the usage of computational resources and reduces idle time during model training.

- **Deep Learning (DL):** A subset of Machine Learning (ML) where multi-layered (artificial) Neural Networks (NNs) learn from vast amounts of data. DL models involve a great level of complexity and are capable of discovering intricate structures in high-dimensional data. These models are termed "deep" due to the presence of multiple "hidden" layers that enable the representation of data at various levels of abstraction. As data passes through these layers, each layer computes a transformation using weights and biases, typically followed by a non-linear "activation" function. This process allows the NN to learn complex relationships between inputs and outputs, capture hierarchical patterns, and ultimately make predictions or classifications.

- **Event** (Stream) **Data:** Detailed log-like data of soccer matches. It records data about the timestamp, player involved, location (on the pitch) of each specific on-ball, and disciplinary in-game event. Event data providers usually tag and record these on-ball events manually via the employment of television broadcast footage of soccer matches.

- (Model) **Generalization/Robustness:** The ability of a Machine Learning (ML) model to perform well on new, previously unseen data, as opposed to the data on which it was trained. It indicates the model's utility in real-world applications; it reflects how well the

ML model has learned the underlying patterns in the data, rather than simply memorizing the training samples.

- **Geometric Deep Learning (GDL):** An emerging field in Machine Learning (ML) that extends Deep Learning (DL) techniques to data with non-Euclidean structures, such as social networks, protein interactions, and 3D shapes, which are better represented as graphs, for instance. GDL aims to generalize the methods of DL to this broader class of data structures. The key challenge and focus of GDL is to develop learning algorithms that respect the intrinsic geometric properties of data. This means understanding and leveraging how data points are connected or related to each other in these complex structures.

- **Graph Neural Networks (GNNs):** A class of Neural Networks (NNs) designed to perform inference on data represented as graphs. Unlike traditional NNs that assume the data to be in regular structures like vectors (in fully connected NNs) or grids (in convolutional NNs), GNNs can handle data with an irregular structure, which is a natural form for many real-world problems. Graphs consist of "nodes" (or "vertices") and "edges", and they can represent a wide range of data types — GNNs leverage the connections and the features of these nodes and edges to learn how to make predictions. They are particularly adept at tasks such as node classification, link prediction, and graph classification. GNNs learn a representation (or embedding) for each node that captures not only its own features but also the features of its neighbors. This is typically done through a process called Message Passing, where nodes aggregate feature information from their neighbors and combine it with their own features to update their state. This process can be performed multiple times, and the resulting node embeddings can be used for downstream Machine Learning (ML) tasks.

- **Heterogeneous Graphs:** A flavour of graphs in graph and network theory where nodes and/or edges can be of different types. This contrasts with homogeneous graphs, where all nodes and edges are of the same type. Heterogeneous graphs are used to represent complex systems where different kinds of entities and relationships exist. Nodes represent entities, and there can be multiple types of entities in the graph. Edges represent relationships between entities (nodes), and there can be various types of relationships.

- **Heterogeneous (Graph) Learning:** A specialized area within graph Machine Learning (ML) that focuses on learning from heterogeneous graphs. It involves extracting knowledge and learning from the varied types of nodes and/or edges using Graph Neural Networks (GNNs). This diversity mirrors real-world data complexity, where different types of entities and interactions coexist and influence each other.

- **Heterogeneous Graph Transformer Networks (HGTs):** An advanced form of Graph Neural Networks (GNNs) specifically designed to handle heterogeneous graphs. The HGT framework extends the principles of Transformer networks, adapting them to the unique

challenges posed by heterogeneous graph data. Drawing inspiration from the Transformer architecture, HGTs utilize attention mechanisms; however, these are modified to account for the different types and attributes of nodes and edges in a heterogeneous graph. The use of attention in HGTs is crucial, it allows the model to dynamically weigh the importance of different nodes and edges based on their types and connections. This is particularly important in heterogeneous graphs where the relevance of a node or an edge can vary significantly depending on its type and context.

- **Hyper-Parameters:** The configuration settings used to structure the learning process in Machine Learning (ML) algorithms. Unlike model parameters, which are learned during training, hyper-parameters are set prior to the training process and are not derived from the data. They are external to the model and their value cannot be estimated from the data; hence, they are typically specified by the practitioner (researcher or data scientist). They have a significant impact on the performance of ML models because they dictate the behavior of the learning algorithm itself.

- **Hyper-Parameter Optimization/Tuning (HPO / HPT):** The process of finding and selecting the optimal combination of hyper-parameters that yields the best model performance, usually evaluated through Cross-Validation (CV) or other evaluation strategies; it is a crucial step in building an effective Machine Learning (ML) model, since hyper-parameters significantly influence the model's performance.

- **K-Fold Cross-Validation:** Technique used where data is divided into k "folds" (or subsets). For each iteration, one fold is retained as the "validation" data to test the model, and the remaining $k - 1$ folds are used as "training" data. The process is repeated k times, with each of the k folds used exactly once as the validation data. This method provides the benefit of maximizing both the training and validation data and allows for the evaluation of the model's performance across different subsets of data, offering a more comprehensive insight into the model's ability to generalize to unseen data. The validation results from all k iterations are then averaged to produce a single estimation of the final performance metric, which is less sensitive to the partitioning of data compared to a single naive train-test split.

- **Logistic Regression:** Statistical model and a fundamental Machine Learning (ML) algorithm used for binary classification tasks — where the objective is to predict a binary outcome (e.g.: yes/no, true/false, success/failure). Despite its name suggesting a regression algorithm, logistic regression is used for classification problems. This model predicts the probability that a given input sample belongs to a certain class of the binary target variable. The core function of logistic regression is the "logistic" function, also known as the "sigmoid" function, which is an S-shaped curve (Figure 2.2) that can take any real-valued number and map it between 0 and 1. Logistic Regression model calculates a linear combination of the input features, but instead of outputting the result directly, it passes

this result through the logistic/sigmoid function to generate the probability. The decision of which class to assign to a new input is then made by setting a threshold (decision boundary), typically at 0.5; if the predicted probability is higher than this threshold, the input is assigned to one class, otherwise to the other.

- **Machine Learning (ML):** A subset of Artificial Intelligence (AI) that empowers computer systems with the ability to learn from data and improve their performance over time without being explicitly programmed. It involves the development of algorithms that can process, analyze, and make predictions or decisions based on data. These algorithms build models based on sample "training" data, to make predictions or decisions on previously unseen "test" data. This realm intersects with statistics, computer science, and information theory, harnessing patterns and inferences from data to enable decision-making processes, often surpassing what humans or traditional computational approaches can perform.

- (Artificial) **Neural Networks (NNs):** Computational models inspired by the human brain's neural networks. They consist of interconnected units or nodes called (artificial) "neurons", which are organized in layers. A typical NN comprises an "input" layer that receives the data, one or more "hidden" layers that compute the transformations of the inputs, and an "output" layer that produces the final prediction or classification. Each connection between neurons has an associated "weight", which is adjusted during the training process. The strength of these connections signifies the importance of the input features. Neurons process the inputs by aggregating them and applying an "activation" function, which determines whether and to what extent the signal should affect the next layer.

- **Stratified Cross-Validation:** A variation of Cross-Validation (CV) where the data is split into "training" and "validation" sets in such a way that each set contains approximately the same percentage of samples of each target class (or label) as the complete set. This is particularly useful for ensuring that each fold of the CV is representative of the overall distribution of the data, especially in cases where the data is unbalanced. In an unstratified CV, there's a risk that the random partitioning could result in a training set that has a very different class distribution than the validation set, which can be problematic for evaluating the model's performance. Stratified CV mitigates this risk by preserving the class distribution within each fold, leading to more reliable and robust model evaluation, particularly for classification problems where a class imbalance is present. This helps to ensure that the performance metrics obtained from the CV process are not biased due to a disproportionate representation of classes.

- **Supervised Machine Learning:** A paradigm within the broader field of Machine Learning (ML) that focuses on the development of models that are capable of making predictions or decisions based on input "training" data that has been labeled. The "supervised" aspect refers to the presence of a guiding 'teacher' or 'supervisor' in the form of labeled data

within the target variable. In this setting, an algorithm is 'trained' on a pre-defined set of 'training examples', which include both input data and the corresponding correct outputs. The goal is for the model to learn the mapping from inputs to outputs, enabling it to predict the output for new, unseen "test" data based on the patterns it has learned during its training phase. The most common supervised learning tasks include classification, where the outputs are categories, and regression, where the outputs are numerical.

- (Broadcast) **Tracking Data:** Positional data of soccer matches. The recorded data consists of the location of the 22 players and the ball on the pitch. This data is recorded from television broadcast footage and can be extracted either using computer-vision techniques or by manual logging. The position is extracted from the primary broadcast camera through a dynamic calibration of the pitch, which adjusts to the camera's horizontal movement across the pitch. However, the generation of positional data is not uninterrupted, therefore tracking data becomes unavailable during instances when the broadcast director cuts the feed of the main broadcast camera and switches to other secondary cameras for close-ups of players, or when replays are being aired. Nonetheless, it must be noted that replays are mostly displayed when the ball is out-of-play.

- (Optical) **Tracking Data:** Positional data of soccer matches. Multiple state-of-the-art cameras are set up all around the stadium, ensuring all angles are covered, and they record frames at a very high frequency per second. The recorded data is extracted using computer-vision techniques, and consists of the location of the 22 players and the ball on the pitch. The current state-of-the-art frame rate is of 25Hz (i.e. 25 frames per second).

- **Transformers:** A type of Neural Network (NN) architecture used primarily in the field of Deep Learning (DL), particularly for tasks involving sequential data, such as Natural Language Processing (NLP) and recently also introduced in Geometric Deep Learning (GDL), implemented together with Graph Neural Networks (GNNs). The core innovation of Transformers is the self-attention mechanism. This allows the model to weigh the importance of different parts of the input data differently. Unlike Recurrent Neural Networks (RNNs) that process data sequentially, Transformers process entire sequences of data in parallel. Since Transformers do not process data sequentially, they use positional encodings to maintain the order of the sequence. These encodings provide context about the position of each token in the sequence. This parallel processing allows for significantly faster training times and the ability to handle long-range dependencies in data more effectively. The original Transformer model has an encoder-decoder architecture, where the encoder is used to process the input data and the decoder to generate the output. Transformers are highly scalable and efficient, allowing them to work with large datasets.

- **xG (Expected Goals):** The prior likelihood of the instant before the action of taking a shot resulting in a goal; a quality-score for the shot-attempt taken.

2.2 Soccer Analytics' Metrics and Models

Soccer analytics has undergone a significant evolution over the past few decades, transitioning from rudimentary statistics to sophisticated metrics and models that leverage advanced machine learning techniques that capture the intricacies of the sport. The metrics and tools employed in soccer analytics have fluctuated since data started gaining weight and attention in this sport. This section aims to provide an exhaustive review of both traditional and modern metrics and models in soccer analytics, with a particular focus on goal probability prediction at the end. The objective is to position this thesis within the existing academic landscape, identify gaps that this research could help filling and areas where this investigation could possibly contribute to.

2.2.1 Traditional Metrics

Count-Based Metrics

Traditionally, the (count) number of goals scored and assists produced by a player or a team took all the attention of coaches, soccer analysts and the media; nevertheless, concentrating only on these disregards all of the other events or actions that occur on the pitch during a soccer match. While the number of goals scored is the ultimate metric that decides the outcome of a match, they represent only a fraction of the events that occur during the playing time. For instance, only about 1% of all attacking plays and roughly 10% of all shots taken result in a goal [13]. Assists serve as a key indicator of a player's ability to create goal-scoring opportunities. Regardless, not an expressive enough metric as, for example, a player can produce a magnificent assist to another player inside the box, leaving that player alone against the goalkeeper or even with an empty goal, but the receiving player might miss the chance and not score a goal. Therefore, this metric is dependent on the finishing ability of the receiving player, making them somewhat incomplete as a standalone metric.

One of the earliest and most straightforward metrics, the total number of passes made by a player or a team during a match, provides a rudimentary measure of ball control and possession. However, this metric does not account for the quality or effectiveness of the passes. Tackles and interceptions are defensive metrics that are crucial for evaluating a player's or team's ability to regain possession. Nevertheless, they can sometimes be misleading as a high number may indicate a team that is often out of position. These metrics, although simple, provide a rough foundational understanding of a team's performance. Even though these basic metrics do provide a rough foundational understanding, they are not expressive enough, as they cannot grasp the spatio-temporal dynamic intricacies of "the beautiful game"; therefore, soccer analysts later began to dig deeper into analyzing and developing other metrics related to passes and assists, such as the risk and reward of passes, or the expectation of a pass [14; 15]; and related to tackles and interceptions, by linking these to the idea of the probability of a pass and how this probability decreases based on the physics of the ball's and players' direction, speed and players' reaction times [16; 17].

Match State-Based Metrics

Teams' possession percentage throughout a match is another commonly used metric, offering insights into which team controlled the ball and thus the game for the most extended periods of time throughout the match. This metric started gaining popularity and becoming a more commonly reported metric within live broadcasts' and media's analyses since Pep Guardiola's FC Barcelona team and the Spanish National team started implementing the "tiki-taka" style of play [18]. This style of play, which is a very possession-retention focused, rapid and aesthetic style of play, guided both of these teams to conquer three of the most prestigious trophies at a club and national level between the years 2008 and 2013, such as the UEFA Champions League, the UEFA European Cup and the FIFA World Cup [19; 20]. While indicative of a team's dominance over the ball, it does not necessarily correlate with goal-scoring opportunities, as a team could potentially have a possession percentage of over 75 or 80% throughout the match, and not be able to create more than one or two chances in the entire match, and even these chances could potentially not be of good scoring quality.

Additionally, the number of penalty-box entries and the number of entries in the final (attacking) third of the pitch are also considered valuable for assessing a team's attacking skilfulness, as they measure how often does the team enter critical and dangerous areas of the pitch during their attacking phase [21]. Nonetheless, these metrics only confine to two-thirds of the pitch and do not account for the quality of the entries themselves or of the subsequent actions of the players once inside this zone of the pitch. The problem with all of these previous metrics is that the context of the game state is not taken into consideration, and this is crucial because, for example, sometimes the aim of a possession is to not concede a goal instead of trying to score one.

2.2.2 Modern Metrics and Models

In addition to soccer, similar advanced, more modern models have been formulated for various other sports, such as American football [22], rugby [23], basketball [24], and ice hockey [25; 26]. Nonetheless, the challenges of quantifying a player's impact in soccer are heightened due to the sport's low-scoring nature and limited on-ball actions. As a result, these models have only gained traction recently in the realm of soccer analytics, spurred by the increased availability of further comprehensive data.

There are 2 main types of data sources for soccer matches that can be employed for action (not event) valuation: event stream data and (optical or broadcast) tracking data [27]. Event stream data provides a log for the timing and locations (on the pitch) of each specific on-ball in-game event like (and not limited to) tackles, shots, interceptions, and passes, and also a log for disciplinary events such as bookings (yellow or red cards), player substitutions, and more [28; 29]. On average, there tends to be approximately a few thousand events per match. A major limitation that event (stream) data has is that it does not record and include any information regarding players who were not interacting with the ball, and thus lacking the majority of the spatio-temporal contextual

information surrounding the event and the state of the match itself [28].

In the 2010s decade, it was approved by FIFA (International Association Football Federation) the deployment of wireless sensor technology for monitoring the positions of players in competitions [30; 31]. Therefore, on the other hand, optical (or broadcast) tracking data captures the positions of all 22 players and the ball at high frequency time-intervals [32; 33]. Earlier in time, tracking data had a frequency of 10Hz [34; 35], although the current state-of-the-art is set at 25Hz — i.e. 25 frames per second played of the match [36; 37; 38; 39]. This means that on average, for every match (accounting for added-time at the end of each half), approximately 150 000 frames of tracking data are generated and recorded. This degree of detailed and granular data allows to accurately measure the trajectories, speeds, velocities, and accelerations of the ball and all the players on the pitch. Even though tracking data has been a huge advancement for soccer analytics, it still has some limitations: it currently does not take into consideration or record the body orientation of players, and it also does not record the spin and curved trajectories that the ball many times take, at the moment. It is worth noting that tracking data is commonly not disseminated across different leagues or even between clubs within the same league.

Although some research has been conducted on action valuation using tracking data [14; 17; 40; 41], the bulk of the work so far in soccer analytics relies on event (stream) data, primarily due to its broader coverage across leagues and greater accessibility to clubs. However, the tide is changing, now the tendency is to make efforts to democratize tracking data, and for research and analyses to start depending heavily on tracking data [42; 43; 44] or employing a combination of event data together with tracking data, synchronizing these together [13; 45].

It is important to emphasise and understand the distinction between actions and events. The difference between actions and events lies in the fact that actions are a specific category of events that necessitate player involvement. For instance, a shooting event qualifies as an action because a player executes it, while an event marking the conclusion of the 1st half of a soccer match does not fall under the category of an action [29]. There currently exists 3 flavours of methods for valuing players' actions in soccer making use of event data [27]: count-based approaches [46; 47; 48], action-based approaches [29; 49], and expected possession value (EPV) approaches [40; 50; 51; 52; 53; 54; 55]. Count-based methods focus on the accumulation of actions without considering the contextual elements of any individual action; it evaluates players by firstly, attributing a weight to every flavour of action type, and secondly, by determining a weighted total of how frequently a player executes each of these actions throughout a match — these weights are learned by training a model that associates these frequencies with either the final result of the match or the tally of goals scored or conceded against. Given that both EPV and action-based methods concentrate on evaluating individual actions based on their specific attributes, they share a closer conceptual similarity with each other than with count-based methods [27]. In a broad sense, both action-based and EPV methods evaluate players' actions based on the subsequent Equation 2.1:

$$V(a_i) = Q(S_i) - Q(S_{i-1}) \tag{2.1}$$

where $V(a_i)$ is the total value of an action, a, at a given instant, i, of the soccer match; $Q(S_i)$ is the (quality) value of a given soccer match's state, S_i, at a given instant, and $Q(S_{i-1})$ is the quality value of the previous match's state, S_{i-1}. EPVs have also been utilized in other sports at an elite level, such as rugby [23; 56] and basketball [24; 57], where they were first introduced in 2014, but any EPV model or framework that I will make reference to from now on-wards in this section, will make reference to EPVs utilized in the context of soccer analytics. EPVs will be further described and exposed in sub-subsection 2.2.2 appearing later on in this current chapter.

Expected Goals (xG) and Variants

The limitations of traditional metrics together with the emergence of machine learning, data science, soccer data creators and providers has revolutionized soccer analytics, leading to the development of more refined and predictive models. From that point on, more attention was drawn to the shots. Goals are provided by shots, and since the number of goals scored in a match are scarce (and sometimes none are scored) and are usually a rare event in comparison to any other event during a soccer match, following a Poisson distribution [58], learning anything from them alone would be hard; thus, looking at shots makes more sense. There has been a little bit of a debate on the origin of the Expected Goals (xG) metric, but over time it seems that there is an overall agreement within the field experts on [6] being the father of the dubbed name of this metric, although, to the best of my knowledge, the concept and estimation method surrounding xG was introduced earlier in the literature in [59; 4; 5]. xG is a shot-based model (hence, an event-based model), and has become a cornerstone in modern soccer analytics, so much so that it has become one of the most widely used and referenced metrics not only in soccer analytics, but also by media and commentators during matches and post-match analyses across the globe. As seen in Figure 2.1 and in Equation 2.2 below, the xG model quantifies the likelihood of the action of taking shot in resulting in a goal, mainly based on various factors such as:

- The ball's 2D location (x-, y-coordinates) on the pitch,
- The ball's distance (d) to the centre of the goal being shot at (in the instant of the shot being taken),
- The angle (α) that the ball "sees" this goal (in the instant of the shot being taken) [60].

$$\text{Expected Goals (xG)} = \sigma \left[f\left(x_{\text{Ball}},\ y_{\text{Ball}},\ d_{\text{Ball - Centre of Goal}},\ \alpha_{\text{Ball - Goal}} \right) \right] \tag{2.2}$$

where σ is the logistic sigmoid (or logit) activation function, mainly employed in binary classification (also known as "logistic regression") problems. In this case, the σ function aims to dissect and classify shots into 2 distinct groups (classes or labels): shots that resulted in a goal, ending up at the back of the net, and shots that did not end up being a goal and hence missed (or blocked, or saved by the goalkeeper). It intends to achieve this by assigning a probability, the probability of scoring that shot (following Equation 2.3) [4; 5], in the range [0, 1] — with a splitting threshold probability of 0.5 — to each single shot:

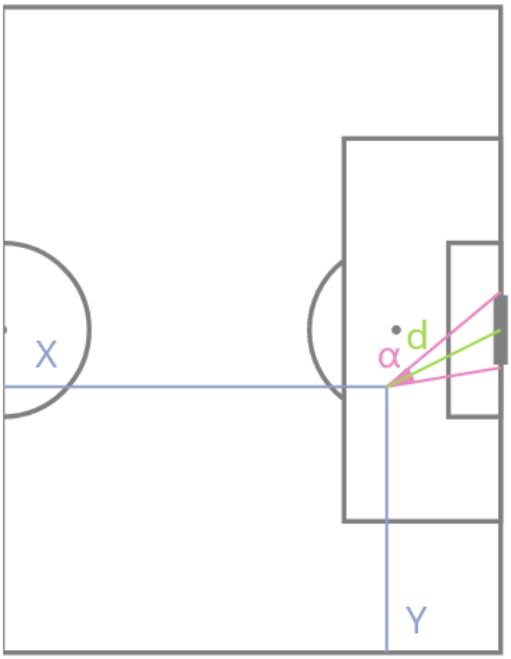


Figure 2.1: Depiction of the xG model, graphically displaying the most important features for the model [61], positioning the ball on an arbitrary location of the pitch.

$$\sigma\left(y_{\text{predicted}}\right) = \frac{e^{y_{\text{predicted}}}}{e^{y_{\text{predicted}}}+1} \equiv \frac{1}{1+e^{-y_{\text{predicted}}}} \tag{2.3}$$

where $y_{\text{predicted}}$ represents the predicted value that is output by the learning model $f(...)$ in Equation 2.2. The work in [59], to the best of my knowledge, seems to be the first which made use of the logistic regression model to estimate goal probability by analyzing shots at goal in soccer. For example, if a shot is given a probability of 0.65 by the model, the model will automatically assign (or classify) this shot to the class of shots resulting in a goal. On the other hand, if a shot is assigned a probability of 0.2 by the model, the model will then classify this shot to the class of shots not resulting in a goal, see Figure 2.2.

Some advantages of utilizing logistic regression are that it is a relatively simple and linear model, has a clear probabilistic interpretation, can be regularized to avoid overfitting, and it outputs well-calibrated predicted probabilities. Nevertheless, it also has several limitations, such as that it assumes a linear decision boundary, which might not always capture the true decision boundary, since it is a linear model it is not powerful and capable enough to capture complex (non-linear) relationships, and last but not least, it is not as flexible as other more complex models.

A high-level interpretation of the xG metric is that it can also be viewed as a quality-score for the shot attempt taken [63], whose value ranges between 0 and 1 [64]; for example, if a player produces a shot with an xG value of 0.7, this means that if the exact same shot was replicated 10 times, one would expect it to end up being a goal 7 of those 10 times. Other high-level interpretations of xG

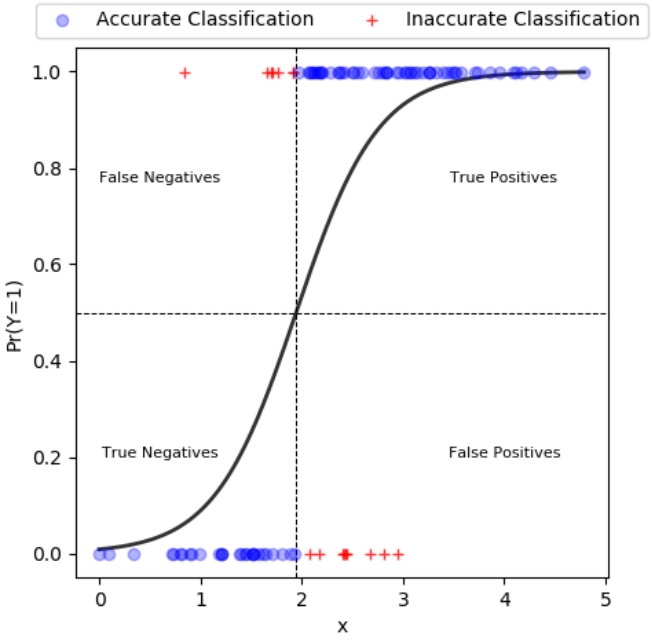


Figure 2.2: Graphical illustration of the logistic sigmoid (or logit) function [62], utilized in logistic regression (also known as "binary classification") problems, visualizing how samples are classified in the logistic regression model via the use of this function.

are, for instance, how many goals would a player (or a team) be expected to score from the shots they are making, based on historical data of lots of shots from lots of locations on the pitch, and which of those shots ended up being a goal, or more broadly speaking, it could also be viewed as the efficiency of players or teams in their finishing; therefore, xG can be useful for coaches and sporting directors in identifying players who consistently under-perform, score close to, or outperform their expected goals (xG). In consequence, the xG metric is also of great use for coaches to try and educate players, from a statistical point of view, from which positions on the pitch can they optimize their probability of scoring a goal. As we can see in Figure 2.3, the probability of scoring a goal from a location close to the "D" (name given to the semi-circle on the edge of the larger box) is not at all the same, as the probability of scoring a goal from inside the six-yard box, in front of the goal. In simple words, the closer and more centered you are from the goal, the higher the probability of scoring a goal [64].

xG has now become the most popular and well-known goal-related soccer metric among coaches, players, media, data enthusiasts, professional data analysts and scientists. As a matter of fact, xG (and slightly varying versions of its model) has been employed in multiple scientific works. In [65] it is stated that defender proximity is important to be taken into account when building the xG model. In [66] xG is used to try and estimate a soccer's match ending score. "Shot efficiency" is defined in [67] as the number of actual goals scored divided by the number of expected goals (xG), suggesting that this metric can be employed for evaluating teams' performance. xG is utilized in [68] to create a new variation of player rating method called "expected goal plus-minus" (xGPM). A

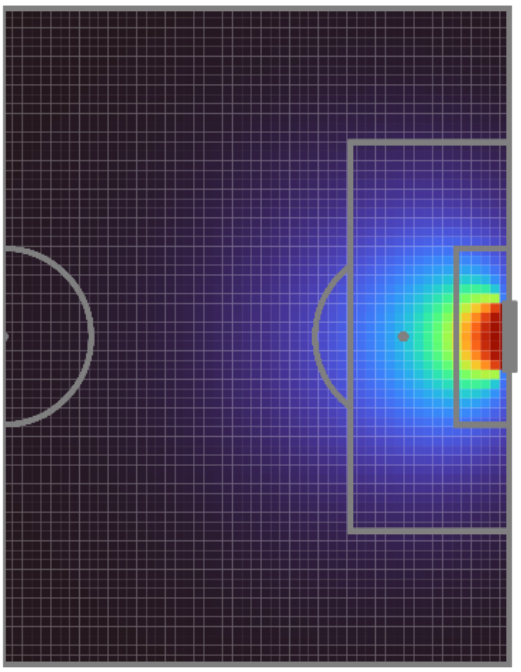


Figure 2.3: Heat-map visualization of the xG model, representing the likelihood of scoring a goal based on the ball's location, distance to the centre of the goal, and the angle that the ball "sees" this goal (at the moment of the shot being taken) [61]. A very similar, but older probability map for xG can be found in [4]

new variant of the xG model is aimed to be created in [69] using qualitative data about players (from the FIFA video game series) to enhance the prediction accuracy of shot outcomes. In [13] event data and tracking data are synchronized to develop a novel xG model, making use of an extreme gradient boosting algorithm to predict the goal probabilities from the shots, outperforming previous xG models. xG is utilized in [70] to examine the shooting tendencies and efficiency between 4 distinct levels of leagues in German soccer, and how these change as players turn older. The work by [71] utilizes explainable artificial intelligence (XAI) tools to interpret complex xG models. In [72] previously untested features (such as team form, match importance and Elo rating) are fed into the xG model to try and refine it further. Player performance is investigated in detail through xG in [73]. The work in [74] presents the "Kos Angle", an innovative feature for xG models, which takes into account the angles occupied by players within the shooting angle plane, between the ball at the instant of shooting and the goal.

xG plays too a pivotal role in several of the more sophisticated models I will display later on in this subsection. Despite its popularity, the xG model has a few limitations. At the moment, xG is not able to accurately account for the defensive pressure exerted on the player who is going to take the shot. xG is currently not able to take into account individual players' (attacking or defensive) skills; for example, it is not the same that Lionel Messi or Cristiano Ronaldo strikes the ball and takes a shot from a specific location than any other player, as we know from historical experience, Messi or Ronaldo will have a higher chance of scoring that particular shot. Nonetheless, there are continuous

efforts in research in the field of soccer analytics to try and incorporate these limitations into the xG model and improve it — even trying to introduce other potentially crucial aspects of the sport into the model as well, including but not limited to, taking into account how many players are located between the ball and the goal (including the rival's goalkeeper), how much space of the mouth of the goal do these players cover at the instant the shot is being taken, and how the rival goalkeeper's positioning at the instant of the shot can decrease the xG value of the attacking player [75].

Advanced Tactical Metrics and Models

Since the number of goals scored in a match is such a scarce event and due to all the flaws about the previously mentioned metrics, these factors have led to the development of larger "possession value-type", process-based metrics and models; as we have just seen before, xGC and xGB are a hint of this trend. These flavour of models offer a more comprehensive understanding of both individual player's and team performances, and also aim to capture the context of the match and assign a numerical value to each event within a possession chain (or sequence) in the match, describing their influence towards not conceding or scoring a goal.

One such model, and by many dubbed as the very first of this type, is the Markov Chain-based model designed in [50]. This model uses a state-space representation to simulate the continuous flow of a soccer match, by inspecting all the various ways in which a possession chain can evolve and predict various outcomes, including goal probabilities. Arbitrarily long possession chains are subtly handled via the use of absorption states (either a possession turnover or a goal being scored [27]), and each action taken within the possession chain is weighted according to the improvement (or deterioration) of the probability of scoring a goal after that action has been taken. This model offers a probabilistic framework for understanding soccer but has the downside that it assumes that each event is independent, which may not always be the case in the dynamic context of a soccer match [50].

Expected Threat (xT)

Another advanced action-valuing framework is the Expected Threat (xT) model [53]. xT, which was built upon the previously exposed Markov-Chain model, is an event-based iterative model that, is designed to quantify the potential level of threat or danger associated with ball possession in different areas of the pitch. It aims to provide a deeper understanding of build-up play and ball progression around the soccer pitch. The model works by overlaying a "value surface" on the soccer pitch, dividing it into distinct zones. Each zone has a specific value that represents the probability of a goal being scored if a team has possession in (or moves the ball to) that zone of the pitch. Each possession chain, defined by shot and move actions transition probabilities, is modelled as a discrete-time Markov process [76]. xT at iteration number n indicates the likelihood of scoring a goal within the following n actions [27; 53]. The model considers the xT value of both the starting and ending locations (or zones) on the pitch of a player's action (either shooting, passing or dribbling). Thus, xT reckons simultaneously, the immediate shooting threat and the future threat potential [53]. Hence,

following Equation 2.1, a high-level interpretation of this model is, if a player (for example) makes a pass which displaces the ball from a position on the pitch where it is improbable for their team to score, to another area on the pitch where it is more likely to score, then these players incremented the xT in favour of their team, as seen in Equation 2.4 below:

$$V_{\text{xT}}(a_i) = \text{xT}(l') - \text{xT}(l) \tag{2.4}$$

where $V_{\text{xT}}(a_i)$ is the total xT value of a successful action, a, at a given instant, i, of the soccer match; $\text{xT}(l)$ is the xT value at a given location on the pitch, l, and $\text{xT}(l')$ is the xT value when the ball is moved (taking action a) to the next location, l'. This model is helpful for tactical analysis. Although, the downside of the xT model is that it does not account for the attacking and defending structures of both teams, and therefore, it implies that the xT of the attacking team at a given arbitrary location on the pitch is the same regardless of the teams' positioning structures at the moment in time. Another model called Contextual xT was proposed to account for these limitations about players' positioning in the original xT model [76]. Other works have also extended the xT model [77] and others have used the original xT model for their research [78].

VAEP

Moreover, the Valuing Actions by Estimating Probabilities (VAEP) framework [29; 49] uses an event data language called 'SPADL' (Soccer Player Action Description Language), in which combines altogether 9 players' actions and match events, together with a CatBoost learning algorithm [79] to estimate the added (or lost) value, and thus the impact, of each player's actions contributing to the outcome of either scoring or conceding a goal before and after each action taken. This framework therefore, aims to assign a value to every single action that a player makes during a match, where this value indicates how much has this player's action increased the probability for their team of scoring a goal, or how much has this player's action increased the probability for their team of conceding a goal in the next k actions, [27], as illustrated in Equation 2.5, where k is a hyper-parameter:

$$Q(S_i) = P^k_{\text{Score}}(S_i, t) - P^k_{\text{Concede}}(S_i, t) \tag{2.5}$$

The impact of a player's action, on both scoring and conceding goals, is examined independently, as both of these effects could potentially be asymmetrical and vary depending on the context of the given match state[27; 29]. Each match state is defined as a set of α actions: the current action taken at a given instant, a_i, and $\alpha - 1$ actions taken in the past; where α is an integer hyper-parameter, representing how many actions into the past do we want the match state to consider within its context. For example, if $\alpha = 4$: $S_i = \{a_{i-3}, a_{i-2}, a_{i-1}, a_i\}$ [27]. Usually, this model is trained and tested using $\alpha = 3$ and $k = 10$ [80]. Thus, inserting Equation 2.5 into Equation 2.1, the total VAEP value of an action is calculated as the aggregate of the 'offensive' contribution and the 'defensive' contribution values, as shown underneath in Equation 2.6:

$$V_{\text{VAEP}}(a_i) = \Delta P_{\text{Score}}(a_i, t) - \Delta P_{\text{Concede}}(a_i, t) \tag{2.6}$$

An interactive tool demonstrating how $V_{\text{VAEP}}(a_i)$ is assigned to distinct actions based on different match state situations can be found in [81], and an explanation of how this interactive tool functions in [82]. This framework offers a comprehensive view of individual players' performance and impact to their team by assigning a value to all possible action types (based on the context of the match), and is capable of reasoning a given player's action's impact on any potential subsequent action taken. Nevertheless, a limitation of the VAEP model is that it only accounts for on-the-ball actions, which makes sense since it is only making use of event data, so the model does not understand everything else that is simultaneously occurring off-the-ball. Another limitation is that when the VAEP's ranking metric is employed for scouting purposes, it is complicated to make an accurate one-to-one comparison between players belonging to different leagues, since the leagues themselves have different levels of competitiveness and toughness (i.e. it might me harder or easier to stand out and perform in some leagues than others); and sometimes even between players of different clubs within the same league, as it is simpler to perform if you are in a top 5 club, surrounded by very talented players, than it is to perform if you belong to a bottom 5 club, surrounded by players with not so much skill as the previously mentioned ones. This framework can be applied for the purpose of: player evaluation — VAEP scores can be utilized to compare players across different positions and roles; for tactical analyses — understanding the effectiveness of different game strategies; and for player scouting for transfer market windows — clubs and sporting directors can make use of the VAEP's player ranking metric to identify valuable players that they could potentially sign, or to identify players who are under-performing in their teams and then consider to sell them in the next transfer window [29].

The xT model and the VAEP framework are somewhat similar, but they differ in how each model depicts the soccer's match state and the way each assesses the value of actions. For instance, VAEP adopts a comprehensive feature-based representation of the soccer match's state, that accounts for both the action and the context of the match, whereas xT utilizes a simplified match state representation focused solely on location. Also, VAEP divides the soccer match into fixed-length action sequences and considers events beyond turnovers of possession. In contrast, xT is structured around possessions, segmenting the match accordingly and estimating the probability of a goal happening within that specific possession. In consequence, VAEP is capable of extracting the context of the players' actions, and on the other hand, xT is capable of only valuing actions which are ball-progressing. VAEP more effectively encapsulates the balance between risk and reward for actions, while xT offers greater robustness. Both approaches yield further insights into player performance, in comparison to traditional metrics [27]. A more visually-aided comparison between these two models can also be found in [83].

Pitch Control

One more significant advancement in soccer analytics is the (Potential) Pitch Control model, first presented in [84], then formally introduced in the literature in [16], and later further refined in [17]. This framework is the first from the metrics and models exposed so far in this chapter's section that is purely and solely dependent on (optical or broadcast) tracking data, and that does

not require any event (stream) data for its development. Pitch Control was initially inspired from physics, utilizing the concept of electric potential fields created by electrically charged particles. The analogy transferred to the context of soccer is that, in the same way electric charges create electric potential fields in space caused by their intrinsic electric charge attribute, players may also 'create' some sort of potential control field on the soccer pitch, specifying how much space of the pitch might a player potentially be in control of, due to each player's intrinsic attributes (such as, and not limited to positioning on the pitch, velocity, acceleration, maximum sprinting speed) and thus, how much time it would take each player to reach that space (or location) on the pitch. This model builds upon and extends the ideas of utilizing Voronoi diagrams [85; 86; 87], and Delaunay triangulations [58] in sports analytics. For instance, in soccer Voronoi diagrams were employed to describe the area controlled by each player, delimiting the perimeter of this area by deciding which points in space on the pitch are closest to each player at that given moment in time, and placing a line at any equidistant location between players.

Pitch Control takes into consideration the ball's trajectory, time of flight of the ball, the time it would take a player to, separately, control and intercept the ball, accounting also for off-sides of attacking players. Equation 2.7 describes for each individual player, p, the rate of change of its potential pitch control probability, $PPCF_p$, at a given location, r, at instantaneous time, t:

$$\frac{\partial PPCF_p}{\partial T}\left(t, \vec{r}, T \mid s, \lambda_p\right) = \left[1 - \sum_P PPCF_P\left(t, \vec{r}, T \mid s, \lambda_p\right)\right] f_p\left(t, \vec{r}, T \mid s\right) \lambda_p \tag{2.7}$$

where λ_p represents the control rate of player p (i.e. the inverse of the average time it would require for a player to successfully do a controlled touch on the ball), and $f_p\left(t, \vec{r}, T \mid s\right)$ is the interception probability, which can also be interpreted as the player's arrival probability (i.e. the probability that at time t, a player p is capable of reaching a given location r on the pitch with a certain time-window T).

As one can interpret from Figure 2.4, Pitch Control represents a value surface across the whole soccer pitch which quantifies how much control a team has over different areas of the pitch at any given time, by calculating which team is most likely to (re-)gain possession of the ball at any given location on the pitch, given that the ball was hypothetically displaced to that specific location — simply put, it quantifies the degree of control or influence that each team potentially has over every location on the pitch at each instant in time. These surface values' probabilities are colour coded employing a heat-map for better interpretability, as one can pick up from looking at Figure 2.4. Hence, an area of the pitch is considered to be controlled by a team if a player from that team could reach it before any other player from the opponent team.

This framework can be exploited in multiple applications: if one creates a dynamical animated Pitch Control video of a match using its tracking data, this can be employed to be watched together with its respective match video or recording for post-match analysis. It is particularly fruitful for leveraging players' positioning for strategical tactical analysis — for pointing out and correcting players' positionings in particular plays where they created beneficial space for the rival

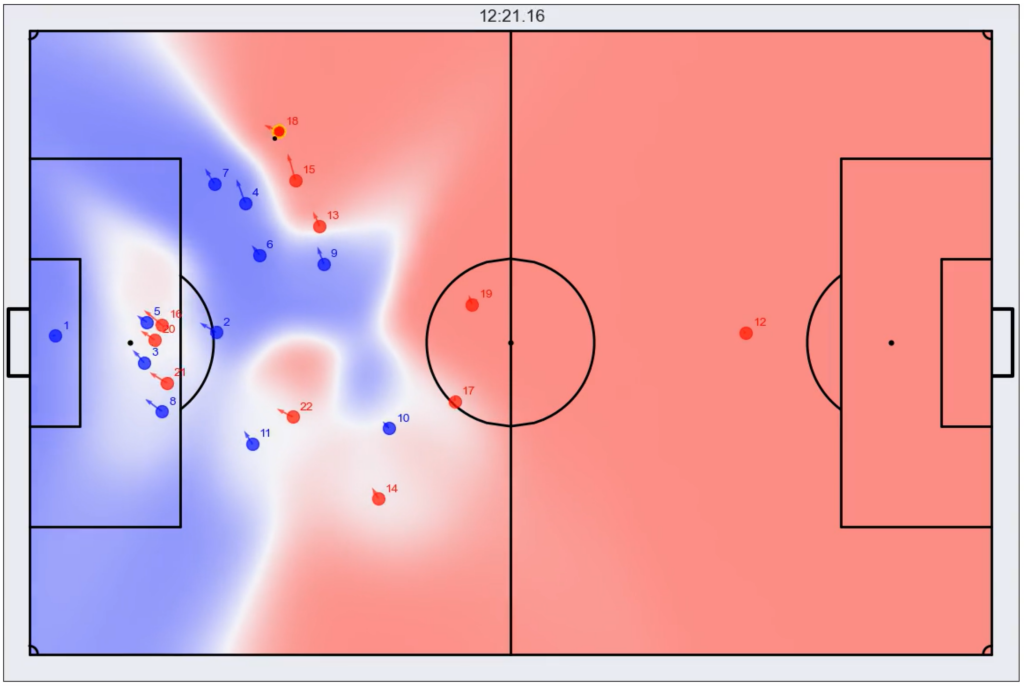


Figure 2.4: Computed Pitch Control of an arbitrary frame of an arbitrary match, using the broadcast tracking data I had access to. Blue and red regions depict the areas of the pitch controlled by the home and away teams, respectively; the white regions represent the areas of the pitch where there is more uncertainty over the control over the ball and therefore, the ball would potentially be in dispute, where it is not clear which player, belonging to which team, would arrive there first to be in control of it. The black dot (next to red player #18) is the ball, and the player surrounded by the yellow ring (in this case also red player #18) indicates the player currently in possession of the ball. For further Pitch Control-related (static and dynamic) visualizations made by myself, please refer to the Appendix B.

or allowed rival players to be on-side and learn from these mistakes, for assessing effectiveness of different team formations, identifying key moments and locations in a match (i.e. off-ball scoring opportunities), evaluating defensive (or offensive) pressure being exerted on the rival team; optimizing players' decision-making when passing the ball around and their other hypothetical passing options or the way they move across the pitch — thus, understanding the spatial (control) dynamics of the sport and assess players' performance. One limitation this framework has got is that it does not inform or report an actual numerical value, for example, of the potential passing options, to evaluate which could have been the best option to take in the end. Hence, the Pitch Control model is optimally exploited when used alongside video recordings of the soccer match and an EPV model to account for the limitation I just mentioned.

EPV (Tracking Data)

The Expected Possession Value (EPV) framework in [40; 88; 89], is the first EPV model displayed so far in this chapter's section that is dependent on (optical) tracking data — all the previously showcased EPV models relied exclusively on event (stream) data. In consequence, the use of tracking data allows this framework to offer a (almost) real-time indicator for the anticipated outcome of a possession, by integrating spatio-temporal features of all 22 players on the pitch and the ball, at all time, estimating frame-by-frame the likelihood of the outcome of that possession. When players move around the pitch (on- and off-the-ball) in a certain way, they continuously and dynamically change that likelihood.

This EPV framework aims to quantify the expected outcome of a possession at any given time. The framework, portrayed in Equation 2.8, is designed to be decoupled, breaking down the goal value into the expected values of different actions that advance the possession, allowing for a more nuanced understanding of different actions in a soccer match. These actions, A, are categorized into three main types: passes (p), shots (s), and ball-drives (d):

$$\begin{aligned} EPV(t) \;&\equiv\; \mathrm{E}\big[X \,|\, T_t\big] \\ &= \mathrm{E}\big[X \,|\, A = p\big]\,\mathrm{P}\big(A = p\big) \;+\; \mathrm{E}\big[X \,|\, A = s\big]\,\mathrm{P}\big(A = s\big) \;+\; \mathrm{E}\big[X \,|\, A = d\big]\,\mathrm{P}\big(A = d\big) \end{aligned} \tag{2.8}$$

where T_t represents all the spatio-temporal contextual information (given by the used optical tracking data) at instantaneous time t; having in mind that passes and ball drives can be attempted towards any given location on the soccer pitch (and not just directly to the players' feet), $\mathrm{E}\big[X \,|\, A = p\big]$ is the passing value surface (dependent on the potential destination location of the ball), and $\mathrm{E}\big[X \,|\, A = d\big]$ is the drive value surface (also dependent on the potential destination location of the ball); $\mathrm{E}\big[X \,|\, A = s\big]$ is an (variant of) expected goals (xG) model. In this framework, the passing component takes into consideration the probability of a successful pass and the probability of a turnover (in the case of an unsuccessful pass) — logistic regression is employed for estimating these mentioned probabilities, whereas a deep neural network is utilized for the value expectation estimation.

These decoupled components are evaluated in an independent manner. The xG (variant) model is estimated employing a logistic regression making use of spatial features; the passing value surface is estimated utilizing a custom deep convolutional neural network called SoccerMap [90] (as briefly mentioned earlier in the previous paragraph); the ball-drive value surface is also estimated making use of a custom-designed deep neural network; and overall, an action policy is constructed to join the decoupled expectations together using a deep neural network architecture, on top of Pitch Control and Pitch Influence [21] surfaces. It is ensured that all models are tuned and calibrated for accuracy for each individual decoupled component, and also for the fully joint EPV model, in a careful manner. Another advantage of this decoupled nature of the framework is, that the decoupled components can also be adjusted or expanded individually. If one believes that one team's passing

tendencies are significantly distinct from another team, one can potentially modify the passing model within the framework to better suit that team's requirements. This eliminates the need to build an entirely new model from the ground up or rely on a black-box model that only provides outcomes without insights into its behavior or rationale.

This EPV model (together with the use of tracking data) emphasizes the importance of the role of context in soccer, such as the phase of the game (build-up, progression, finalization), and other contextual factors like player positioning, motion, distances, and angles. These are incorporated into the framework to improve its interpretability and predictive ability. The EPV framework is particularly functional, thanks to the granular insights into the status of a possession that this model furnishes, for evaluating the impact of passes: being able to create more advanced versions of pass-maps, finding optimal goal-keeper actions (goal-kicks in particular), identifying player passing profiles (risk/reward trade-off); for evaluating decision-making (whether at the team level or for individual players) and for scouting (of own and rival teams' players): identifying high-value actions, space creation of value, fine-grained analysis of off-ball positioning, quantifying the effects of (attacking or defensive) pressure, analyzing rival players' marking patterns, qualitative on- and off-ball performance of players, and also ranking possessions by the final produced danger/quality.

2.2.3 Applied Machine and Deep Learning In Soccer

The practical application of ML and DL concepts and techniques in sports is not new [91; 92; 93], although it is relatively innovative in soccer. In the beautiful game, ML has been utilized in the last couple of decades to investigate numerous aspects of the sport — I have already mentioned several research works in subsection 2.2.2 that employed ML and DL, but following, I will mention multiple investigations that made use of ML for other aspects of the beautiful game that are not exclusively related to metrics.

In the work [94] some of the current challenges and future directions of the application of ML in soccer are discussed. Multilevel multivariate logistic regression is used in [95] to explore the effect of contextual variables on the attacking style of play in elite soccer. Probabilistic movement models and zones of control are investigated in [96] employing several different ML techniques, such as Kernel Density Estimations (KDEs). Pass effectiveness is researched in [97] utilizing Principal Component Analysis (PCA) and multiple linear regression. Support Vector Machine (SVM) and a leave-one-out cross-validation approach are leveraged in [98] to analyze team performance. Key performance metrics of players are analyzed in [99] by performing HPO, making use of a Grid Search with a 5-fold CV on multiple ML models, such as Random Forests and k-Nearest Neighbors (among others). Optimization techniques are employed in [100] on self-propelled particle (SPP) models to simulate and investigate player decision-making, by maximizing the weighted combination of pass probability, Pitch Control, and Pitch Impact. In-game behavior of players is explored in [101] utilizing two dimensionality reduction techniques: t-distributed Stochastic Neighbor Embedding (t-SNE) and Uniform Manifold Approximation and Projection

(UMAP), as well as previously transforming the data using standardization.

The Extreme Gradient Boosting (XGBoost) model, together with other ML techniques, is employed in [13] to develop another xG model and thus, dig further into goal-scoring probability estimation. Hierarchical clustering is utilized in [32] to group the origins of goals in the German soccer national league. XGBoost models are also used in [15; 45; 102] to examine the difficulty of passes ("expected pass" or "xPass"), to detect counterpressing, and to detect off-ball advantages, estimating the likelihood of a pass increasing the EPV value to exploit the rival team's spatial weaknesses, respectively. This same year, [103] made use too of an XGBoost model in order to define a novel metric called "unexpected pass" ("un-xPass") or also known as "creative decision rating" (CDR), a metric which valuates the level of creativity in soccer players' passing skills, and thus creating a metric for assessing passing creativity in soccer; CDR evaluates two primary aspects of a player's passes: how original they are, and their effectiveness in enhancing the team's probability of scoring a goal.

On the other hand, the practical application of DL in soccer is much more of a novelty. DL has contributed to the development of the analysis and comprehension of the intricacies of this sport within the last 5 to 10 years. In the 2016 publication [30], the authors discuss several challenges and opportunities that, at the time, were anticipated for the application of DL in soccer. Later, DL is used in [104] and in [105] via the application of (deep) Convolutional Neural Networks (CNNs), for developing a pass prediction model and to rate players' positioning on the pitch, respectively. Deep CNNs are once again employed in [40], whose CNN architecture is later dubbed as SoccerMap [90], utilized for the task of estimating EPVs. Their CNN architecture was based on [106; 107]. Then [108] utilizes both (vanilla) Recurrent Neural Networks (RNNs) and Long Short-Term Memory Networks (LSTMs, another flavour of RNNs) for trying to predict soccer match results. In the publication [109], Variational Autoencoders (VAEs) are utilized to automatize the labelling of soccer situations. Vanilla Autoencoders (AEs) are employed in [110] to encode the signature playing style of coaches, which are previously grouped together using clustering techniques.

The application of Reinforcement Learning (RL) in soccer has also just started gaining weight in the recent years. For instance, in the works of [111; 112] deep RL frameworks are implemented in order to train policy networks to try to optimize soccer players' actions in critical situations, and in general for trying to optimize players' decision-making during a match. In their latest work this year, [113] implement once again a deep RL framework, which generates the optimal ball target locations across the entire pitch area, aiming to maximize the expected outcome of that possession. Also recently this same year, [114] uses CNNs to break down each match into unique, non-intersecting parts and subsequently classify these parts into different phases, to then be capable of contextualizing the team formations used in each of the different phases of play. To cap it off, in 2021, even entities of significant prestige and influence like Google DeepMind and Liverpool FC collaborated to explore the potential of DL and Artificial Intelligence (AI) in soccer and what it can do for the sport [115].

GNNs Applied In Soccer

To the very best of my knowledge, before the start of this thesis's work there were 16 papers which applied Graph Neural Networks (GNNs) within the context of soccer. During the bureaucratic process of obtaining the official approval from my university for the topic of this thesis, which lasted a couple of weeks, another paper which applies GNNs in soccer was published; and once the work for this thesis had already started, another 2 papers were published towards the end of 2023 in which GNNs were once again applied in soccer. Thus, increasing the total amount of research works applying GNNs in the setting of soccer to 19. It is impressive that all of these 19 papers have been published in a span of 2-3 years, the first [116] being published in 2020. Even though GNNs have been around since 2005 [117; 118] and its potential computational capabilities described in 2009 [119], this demonstrates the novelty of the application of this flavour of NNs to this sport, how meteoric its popularity has become in this field of research in recent years, and the excitement that exists surrounding the potential of its applicability.

In [116] Graph Convolutional Networks (GCNs) are implemented, as well as Transformers, to detect group activity (within a soccer match) leveraging both tracking data and Computer Vision (CV) techniques on broadcast footage. GCNs are used too in [120; 121] for helping in the identification of the phases of play in a match, and in assessing how defenses influence the availability of passing options for the opposing team, respectively. Graph Recurrent Neural Networks (GRNNs) are employed in [122] to encapsulate the dynamics between players and the ball, to then forecast both, players' running trajectories and the outcomes of possession phases. In [123] GCNs and Graph Attention Networks (GATs) are leveraged to try to predict sports outcomes — even though this work is designed for American football and Counter-Strike (an e-sport) whose data and match states can have a graph-based representation, we can extrapolate their investigation to work well on soccer as well (as they explicitly mention), since soccer can too be represented in such manner. Heterogeneous Message-Passing GNNs are utilized in [124] to predict the trajectories of multiple robots playing soccer. Once more, GCNs are employed in [125] to identify and spot players' actions via unsupervised player classification. Message Passing Graph Neural Networks (MPGNNs) are used in [126; 127] to track players' trajectories using a multi-camera set-up around stadiums, and to forecast the movement and location of players that are present on the pitch but do not appear during broadcast footage, respectively. Utilizing GATs, which are GNNs that leverage the innovative and state-of-the-art Attention mechanism introduced in [128], together with Variational Recurrent Neural Networks (VRNNs) and VAEs, [1] aims to detect tactical patterns in soccer matches — all architectures were implemented from scratch employing the *PyTorch* library [129]. GCNs and GATs are used to estimate the local and target value functions of an RL model, called Deep Q-Learning Network (DQN), which is then employed to enable agents (players) to gain insights from their experiences by making use of the mentioned value functions to understand the potential worth of each contemplated action in [130].

Graph node representation learning methods, namely the Node2Vec [131] method and the GraphWave [132] algorithm, are applied in [133] to be able to build player recommendation systems for in-match player substitutions. In the work from [134] Graph Variational Recurrent Neural Networks

(GVRNNs) are implemented to model the relationship between players on the pitch and, once again, predict the long-term trajectories of players; these predicted trajectories are then contrasted against true real movements that players actually made to assess players who create off-ball scoring opportunities for the rest of teammates. In the investigation from [2], gender-specific GNNs are built, using CrystalConv layers [135] from the *Spektral* Python GNN library [**?**], for modeling the likelihood of successful counterattacks. GRNNs are once more applied in [136] to contextualize the scenario on the pitch, where the GRNN component of the larger network depicts the interactions among players through a fully connected (FC) graph, and after be capable of predicting when will the next passing action occur. A GNN dubbed as Tactical Graph Networks (TGNets) in [137], which is a streamlined, hybrid ML framework designed to be responsive to interactions among players, is utilized with the intention to achieve an optimal data representation for analyzing players trajectories with ML algorithms. In the work of [138], published in May, Graph Convolutional Recurrent Neural Networks (GCRNNs) are employed to detect latent attributes and to comprehend the spatial and temporal connections among players in the graph-based representation of the data, and later using the full framework, to be able to predict shooting actions during a match. Temporal Graph Neural Networks (TGNs), which were recently first introduced in [139], are implemented in [140] (published in September) where it investigates after a passing action has been taken in soccer, the prediction of the receiver player and the resulting outcome of that pass attempt.

The cherry on top of the cake: in October, Google DeepMind in collaboration with Liverpool FC, published their first joint paper [141], after the promising paper [115] mentioned earlier in this subsection 2.2.3. In [141] they developed a GNN-based AI assistant tool, dubbed "*TacticAI*", exploiting geometric DL to inspect corner-kicks, and predominantly makes use of GATv2 [142] Group Convolutional layers. *TacticAI* inherently generates graphical player representations that adhere to various symmetries of the soccer pitch. Utilizing these graph representations, *TacticAI* is capable of predicting different outcomes of a corner kick. Additionally, *TacticAI* functions as a retrieval tool, enabling the extraction of similar corner-kick scenarios through player model similarities, and also as a generative recommendation system. It proposes modifications to player positions and speeds to optimize or reduce the estimated likelihood of a shot; therefore, *TacticAI* integrates predictive, retrieval, and generative functionalities, aiding coaches in efficiently evaluating and experimenting with different player arrangements for each corner-kick strategy, and choosing those with the greatest anticipated success rate.

Despite the gigantic advancements in soccer analytics and soccer data science in the last 10 to 15 years, there are still existing exciting gaps to be filled, and this thesis aims to address one of them. One of the most, if not the most, crucial aspect of this sport is scoring goals — at the end of the day, or the match I should say, 1 single goal is what makes the difference in this sport; it is what could make the difference, not only between winning, drawing or losing a match, it could also make the difference (at a club level) between relegating one division down, staying in the same league, promoting to one division up, qualifying for or being disqualified from international competitions, such as the UEFA competitions (Champions League, Europa League, Conference League) or Libertadores, winning a title (cup)... and these usually go hand-in-hand with more

personal consequences: from a personnel point of view, any of the previously mentioned scenarios could be the difference between positive or negative salary and job-status modifications for coaches, players, and club staff; and from a fanatical point of view, once again, any of the previously mentioned scenarios could be the difference between anger, disappointment, sadness or even crying, and happiness or euphoria (or again, maybe even crying). Incredible how the simple fact of scoring a goal or not can change so many things in such drastic ways, this is the reason why I believe that exploring and investigating methods to try to optimize the probability of scoring a goal is so important. In the research carried out in [7] both GoogLeNet and a 3-layered CNN are used to predict goal-scoring opportunities — in their conclusion and discussion of potential future work and improvements, they state that GNNs could improve upon their work.

I would like to take it a step further using as inspiration multiple works that I have mentioned throughout this whole section 2.2 (soccer analytics' metrics and models), but in particular this last subsection 2.2.3 (ML and DL applied in soccer) and sub-subsection 2.2.3 (GNNs in soccer). I believe that one can observe that there is a gap in the literature landscape in using GNNs, even more specifically Heterogeneous Graph Transformer Networks (HGTs), to classify shots and thus predict goal probability, as far as I know, in a direct manner. Therefore, this thesis aspires to leverage GNNs to model the complex relationships between all 22 players, the ball, and many other of their physical and spatial features to classify shots into scored and non-scored shots, and hence, predict the probability of scoring a goal. The intention is to use a relatively simple xG model as a baseline model to have as a benchmark, and then make use of 2 advanced GNNs, Heterogeneous Graph Transformer Networks (HGTs) [11], from the *PyTorch Geometric* (PyG) library [12].

2.3 Theoretical Background

This section intends to sequentially present the historical developments and technical theoretical background of the relevant topics in Geometric Deep Learning (GDL) that are relevant for the work of this thesis, briefly stopping at the most relevant milestones and authors through the years.

2.3.1 Graph Theory

This subsection will provide a foundational understanding of Graph Theory, which is essential for comprehending the subsequent topics related to Graph Neural Networks (GNNs) and Transformer Networks. Graph Theory is a branch of mathematics and computer science that involves studying the properties and applications of graphs, which are mathematical structures used to model pairwise relations between objects or entities. A graph is made up of nodes (also called vertices or points) connected by edges (also called links or lines). The study of graphs is crucial in various fields, including computer science, physics, sociology, and biology, as it provides a fundamental way to represent network structures and graph-structured objects.

A graph G is often denoted as $G(N, E)$, where N represents the set of nodes and E the set of edges. Graphs can be categorized into several types:

1. Undirected Graphs: Edges have no orientation. They are simply connections between nodes without any direction. The edge (i, j) is identical to the edge (j, i). Thus, indicating that the relationship that edges represent between nodes is bidirectional.
2. Directed Graphs (Digraphs): Edges have orientation/direction. An edge from node i to node j is not the same as an edge from node j to node i. These are typically represented by arrowheads showing the direction from the source node to the target node. Thus, indicating that the relationship that edges represent between nodes is unidirectional.
3. Weighted Graphs: Edges are assigned a certain weight (a numerical value) or cost, reflecting the strength or capacity of the connection between pairs of nodes. This feature is useful in scenarios where the path cost is a consideration, like routing and network optimization problems.
4. Labeled Graphs: These graphs have labels attached to the edges or nodes (or both). Labels can represent various attributes relevant to the nodes or edges.
5. Homogeneous Graphs: The nodes and edges of these graphs are assumed to be of the same type, respectively. No distinction is made between different nodes or between different edges.
6. Heterogeneous Graphs: The nodes and/or edges of this flavour of graphs are capable of having different types, respectively. Distinctions are made between different nodes and/or between different edges.

Graph Theory's foundational concepts include paths, cycles, and connectivity, which explore the ways in which nodes are linked. The degree of a node in a graph is the number of edges connected to it. In the case of directed graphs, this is further divided into in-degree and out-degree, representing incoming and outgoing edges, respectively. Graphs can be classified into several types based on their characteristics. For instance, a complete (or fully connected) graph is one where every pair of nodes is connected by an edge. A bipartite graph divides nodes into 2 disjoint sets where edges only connect nodes from different sets. Trees are connected graphs without cycles (acyclic graphs).

In computational contexts, graphs are typically represented using adjacency matrices or adjacency lists. An adjacency matrix, see Equation 2.9 and Figure 2.5, is a 2D array where the element at row i and column j indicates the presence or absence (and perhaps the weight) of an edge between nodes i and j.

$$A_{ij} = \begin{cases} 1 & \text{If there is an edge between nodes } i \text{ and } j, \\ 0 & \text{Otherwise.} \end{cases} \tag{2.9}$$

If an edge does have some weight, it will have a non-zero value > 1 in the adjacency matrix. Adjacency lists, on the other hand, represent a graph by listing all nodes connected to each node,

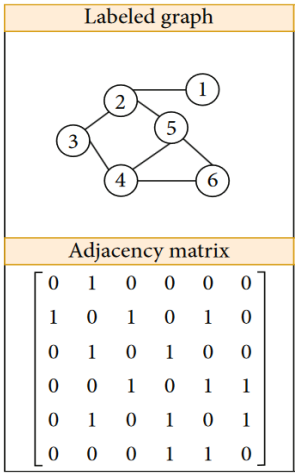


Figure 2.5: **Top:** Simple visualization of a graph with labeled nodes. **Bottom:** Its respective adjacency matrix, representing the graph's connectivity [143].

offering a more space-efficient representation, especially for sparse graphs. Optionally, one can also include matrices to represent the features of the nodes and of the edges.

Graph Theory has broad applications across various domains, it provides the basis for analyzing the structure of networks, enabling an understanding of their properties and behaviors. This analysis plays a significant role in algorithm and data structure design, especially for problems involving network connectivity, flow, and optimization. The effectiveness of graph theory in these areas lies in its ability to abstract and model complex systems as a set of relationships, facilitating the application of various analytical techniques and algorithms to solve real-world problems.

This foundational overview of graph theory sets the stage for understanding more complex structures and algorithms, particularly in the realm of graph-based Machine Learning (ML) models like GNNs and Graph Transformer Networks (GTNs).

2.3.2 Graph Neural Networks

This subsection aims to provide some important theoretical background knowledge on Graph Neural Networks (GNNs). GNNs represent a significant advancement in the field of Deep Learning (DL) and Artificial Intelligence (AI). They are Neural Networks (NNs), first developed by [117; 118], specifically designed to operate on graph structures (or data with irregular structures), enabling the effective processing of data in non-Euclidean domains. Introduced almost 2 decades ago, GNNs have recently acquired and grown in popularity and scope.

Traditional ML methods typically preprocess graph-structured data into simpler representations like vectors. This process, however, may lead to the loss of critical topological information and dependencies inherent in the original graph structure [119]. Traditional DL architectures, such

as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), excel in Euclidean spaces (grid-like data) like images and sequences but struggle with graph-structured data [117; 118]. Hence, GNNs emerged as a solution to directly process graph-structured data without the need for preprocessing [119]; GNNs address this limitation by leveraging the properties of Graph Theory, making them suitable for a wide range of applications including social network analysis, recommendation systems, and bioinformatics [144; 145]. In social networks, they help in recommendation systems and community detection. In e-commerce, they are used for fraud detection and personalized recommendations. In the field of drug discovery and bioinformatics, GNNs play a vital role in predicting molecular properties and interactions. Their ability to handle complex relational data makes them an invaluable tool in numerous fields [144].

Graphs can represent complex relationships and interdependencies within data. In a GNN, graphs are treated as sets of nodes and edges, where nodes and edges have feature representations that encapsulate their properties, and the GNN learns to propagate and transform the information in these features across the graph, in a process denominated Message Passing [146]. On a high-level interpretation, this process effectively captures both the node's features and the structural information of the graph. GNNs apply neural network transformations to these features, ensuring that the output accounts for both, the features of individual nodes and their respective positions, in the overall graph structure [118; 147]. Message Passing is a core principle in GNNs. Nodes in the graph send and receive information (or messages), in the form of feature vectors, to and from their neighboring nodes through the graph's edges. Edges may not or may as well have feature vectors of their own, to be aggregated with the nodes' feature vectors. These messages are aggregated and transformed to update the node features. The Message Passing mechanism enables GNNs to incorporate local neighborhood information, making the learned representations rich and context-aware [146].

Consider a graph $G = (N, E)$ where N is the set of nodes and E is the set of edges. Each node i has an associated feature vector (node representation) h_i. The Message Passing process typically follows 2 phases during the forward pass of the network: the message-passing and the readout phases [146]; these steps are described below for each layer l in the GNN:

1. A message m_{ji}^l computed by the differentiable message function M^l, see Equation 2.10, is the message or information received by node i through the edge of the graph from node j in layer l.

$$m_{ji}^l = M^l\big(h_i^l, h_j^l, e_{ij}\big) \tag{2.10}$$

   where h_i^l and h_j^l are the feature vectors of nodes i and j at layer l, and e_{ij} is the feature vector of the edge between them.

2. Then, all the messages received by node i from its neighbors N(i) at layer l are aggregated together in the aggregation function A^l in Equation 2.11; the aggregation function A^l may be, but not limited to, a min, max, sum or mean operation.

$$a_i^l = A^l\Big(\{m_{ji}^l \mid \forall\, j \in \mathrm{N}(i)\}\Big) \tag{2.11}$$

3. Finally, node i's feature vector in the current layer h_i^l is updated in the next layer into h_i^{l+1}, via the differentiable updating function U^l in Equation 2.12, which combines h_i^l with the aggregated messages from the current layer a_i^l . Often, the updating function U^l is a NN or a non-linear activation function, for example, some version of ReLU, the softmax, or the logistic sigmoid function — which to use depends on whether the current layer l is a hidden layer or a final readout output layer [146].

$$h_i^{l+1} = U^l(h_i^l,\, a_i^l) \equiv U^l\Big(h_i^l,\, A^l\Big(\{m_{ji}^l \mid \forall\, j \in \mathrm{N}(i)\}\Big)\Big) \tag{2.12}$$

Message Passing has an iterative nature, each layer of a GNN can be seen as one round of Message Passing. The more layers are stacked, the further can the information be propagated through the graph, depending on the connectivity of the graph. Therefore, since the depth of a GNN (number of GNN layers) determines how far information can travel, a node's representation in deeper layers encapsulates information from a larger neighborhood. The elegance of Message Passing in GNNs lies in how it effectively captures the interplay between node features and graph topology. This mechanism allows GNNs to learn rich representations of nodes and edges, making them powerful. The specific design of the message, aggregation, and update functions can vary, leading to different GNN architectures, each with its own strengths and applications. The effectiveness of this approach is evident in applications like molecular property prediction, where the structure of the molecule (represented as a graph) plays a crucial role in determining its properties [146; 148].

As one can observe illustrated in Figure 2.6, there are 3 main prediction tasks that one can perform with GNNs — node classification, graph classification, and edge (or link) prediction:

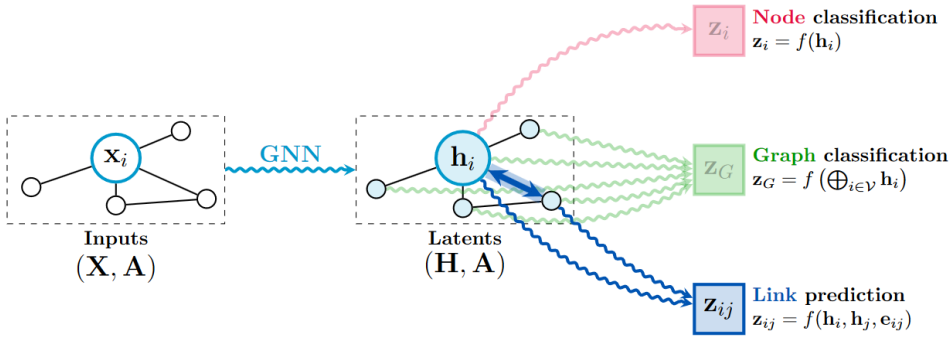


Figure 2.6: GNN prediction tasks [149].

In node classification, the goal is to predict the label or attribute of a node based on the attributes of the other nodes and the structure of the graph. GNNs aggregate information from a node's neighbors (and potentially their neighbors' neighbors, and so on) during Message Passing, allowing each node to effectively gather and learn from the collective information present in its local neighborhood.

Common applications include identifying the role of a person in a social network, categorizing scientific papers in citation networks, or predicting protein functions in biological networks. Graph classification involves predicting a label or property for an entire graph, as opposed to individual nodes. In this task, GNNs aggregate information from all nodes (and possibly edges) in a graph during Message Passing to make a prediction about the graph as a whole. This often involves some form of readout or (global graph) pooling layer that combines node (perhaps also edge) representations into one single graph representation. Common applications include, for instance, predicting the properties of molecules (where each molecule is a graph) and fraud detection (where transactions might form graphs representing patterns of behavior). In edge prediction, the task is to predict whether an edge should exist between two nodes. This can be a prediction of missing links in an existing network or the likelihood of future link formation. GNNs learn representations of nodes that capture not just their own attributes but also the structure of their neighborhood. These representations can then be used to infer the likelihood of a link between any pair of nodes in the graph. Examples include predicting friendships or connections in social networks, inferring protein-protein interactions in biological networks, and recommendation systems (such as suggesting new products to buy or films to watch).

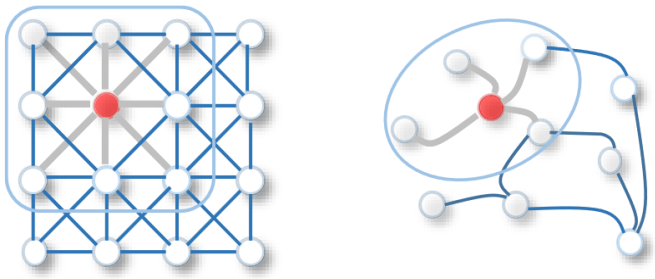


Figure 2.7: Comparison of 2D convolution on **(a)** grid-like data, with convolution on **(b)** graph-structured data [145].

The concept of GNNs has evolved significantly since its inception. Early models were focused on recurrent architectures, where a fixed neural network model was applied recursively over nodes until a stable state was achieved. Modern GNNs have moved towards feed-forward architectures, incorporating lessons from the success of CNNs and attention mechanisms in other domains. This evolution has led to the development of various types of GNNs, such as Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs) [146; 144]. One of the most prominent types of GNNs is the GCN. The concept of convolution in GCNs is generalized from grid data (like images) to graphs, observe Figure 2.7. They define a convolution operation on the graph based on the graph's adjacency matrix. This approach allows GCNs to efficiently learn from the graph's topology, making them particularly effective for tasks like node classification and link prediction; view Figure 2.8 for a simple depiction of a GCN. The simplicity and efficiency of GCNs have contributed to their widespread adoption in various domains [148; 144].

On the other hand, GATs introduce the attention mechanism (later described in subsection 2.3.3), widely used in sequence models like Transformers (also discussed later on in subsection 2.3.3), into

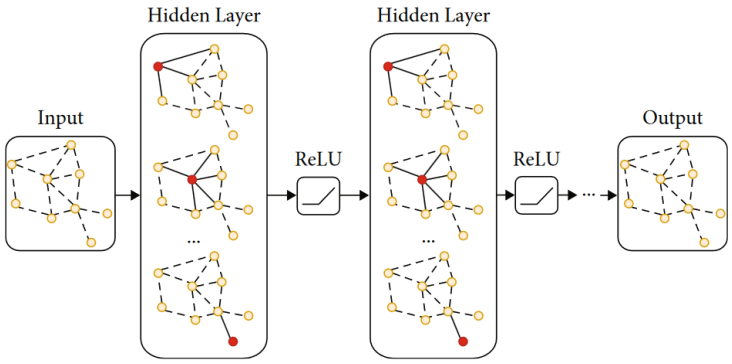


Figure 2.8: Illustration of the concept of a simple GCN model [143].

GNNs. In GATs, the weights of the edges are not predefined but are learned through attention instead, view Figure 2.9 for an illustration of this. This means, on a high-level interpretation, that the influence of one node over another is dynamically determined based on their features. This approach allows GATs to focus on more important neighbors during feature aggregation, enhancing the model's ability to learn complex patterns in graph data. GATs have shown remarkable performance in node classification and graph classification tasks, demonstrating the power of attention mechanisms in graph-based learning [150; 142].

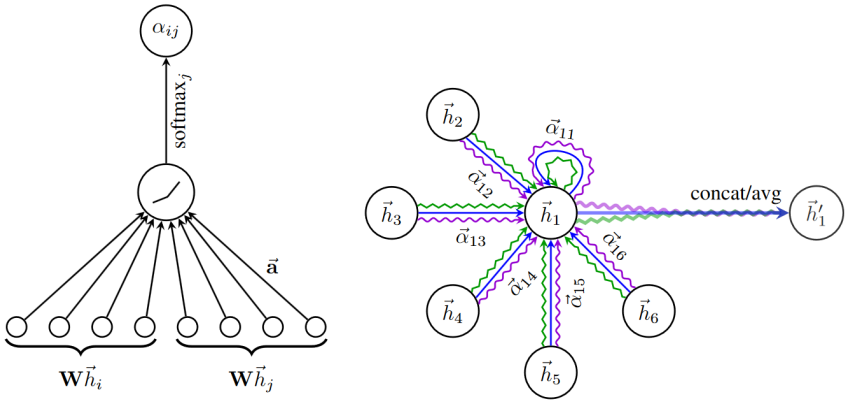


Figure 2.9: **Left:** The non-normalized attention coefficients $e_{i,j} = a(\mathbf{W}\vec{h_i}, \mathbf{W}\vec{h_j}) \equiv \vec{\mathbf{a}}^T[\mathbf{W}\vec{h_i}||\mathbf{W}\vec{h_j}]$ utilized by GATs. $\mathbf{W}$ are learnable weight matrices, $\vec{h}$ are the node features of nodes i and j, parameterized by a weight vector $\vec{\mathbf{a}}$. Then a LeakyReLU activation is applied, finishing off by obtaining the normalized attention coefficients $\alpha_{i,j}$ by normalizing $e_{i,j}$ across all choices of nodes j using the softmax function. **Right:** An illustration of multi-head attention (with $K = 3$ heads) by node $i = 1$ on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain the resulting node embedding $\vec{h}'_1$ for node $i = 1$ [150].

This overview of GNNs covers the fundamental concepts, challenges, and applications, drawing from a range of sources and research papers. The insights provided here are essential for understanding the impact and potential of GNNs in various domains, including the context of this thesis.

2.3.3 Transformers

This subsection intends to expose knowledge about the state-of-the-art architecture of Transformer Networks and their Attention mechanism, which were first developed and introduced in the literature by [128]. A transduction model in the context of ML and data science refers to a type of model that is designed to transform input data into a corresponding output through a learned mapping. The key characteristic of a transduction model is that it operates on a specific set of inputs and outputs, learning the relationships directly from the data — they don't just learn general rules or patterns that can be applied to any new data (like induction models), instead, they are more focused on the specific examples they are trained on. Traditional sequence transduction models like RNNs and CNNs, despite their efficacy, are limited by their sequential computation, hindering parallelization and efficiency. The Transformer model avoids this by relying solely on attention mechanisms, dispensing with recurrence and convolutions entirely. This architecture dramatically enhances parallel processing capabilities, expediting training and improving performance on tasks like machine translation, for instance.

The Transformer, see Figure 2.10, adopts an encoder-decoder structure to its architecture. Both the encoder and decoder are composed of stacks of N identical layers (in [128] 6 are used), each consisting of 2 and 3 sub-layers, respectively: 1 (in the case of the decoder, 2) Multi-Head Self-Attention mechanism, and a position (or point)-wise fully connected feed-forward network. Residual Connections [151] and layer normalization [152] are employed around each sub-layer: LayerNorm(Sub-layer(x) + x), where the function utilized by the sub-layer itself is represented by Sub-layer(x). To enable the functioning of the Residual Connections, every sub-layer within the model, including the embedding layers, generates outputs with a fixed dimensionality d_{model} (in [128] d_{model} = 512). In the decoder component's stack, the first Multi-Head Self-Attention sub-layer is modified (in comparison to the Self-Attention sub-layer in the encoder's stack) with masking to avoid positions from attending to subsequent positions. This masking, along with the offset of the output embeddings by one position, guarantees that the predictions at position i are based solely on the known outputs at positions preceding i. This whole design allows the model to process data in parallel and capture complex dependencies, enhancing both efficiency and effectiveness.

It has been mentioned that Transformers use Multi-Head Self-Attention sub-layers, but, what is Attention? An Attention mechanism (or function) is essentially a process that maps a query and a collection of key-value pairs to an output, with the query, keys, values, and output all being represented by vectors (or matrices). The output is derived as a weighted sum aggregation of the values, where the weights for each value are determined through a compatibility function between the query and its respective key [128]. A high-level interpretation of Attention in the context of Transformers is that it is a mechanism to dynamically focus on certain parts of the input sequence when processing or generating each element of the output sequence. It allows the model to consider the entire context of a sequence when making predictions, instead of relying solely on immediate neighbors or a fixed-size window around the current element; thus, enhancing its ability to capture context and relationships in data.

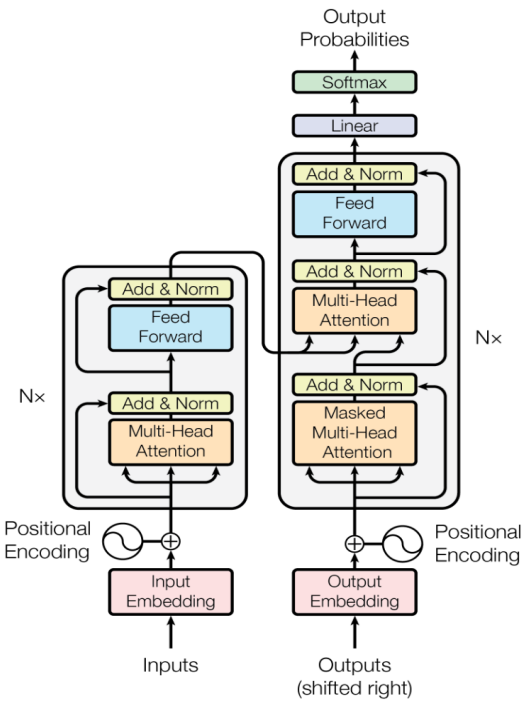


Figure 2.10: Model architecture of the Transformer [128].

For a better understanding of what the queries, keys, and values may represent since they are abstract concepts, let's provide a simple example in the context of machine translation: suppose one is willing to translate the English sentence "the defender slide-tackled the striker" to Spanish. In the encoder, the input is the English sentence mentioned, and the queries, keys, and values are all derived from this input; each query determines how each word in the English sentence relates to every other word in the sentence — for the word "tackled" in "the defender slide-tackled the striker" a query determines its relationship to "defender", "striker", "slide", etc.; the keys are used to score the relevance of each word in the English sentence to the query — the key for "striker" helps to establish its relevance to the query for "defender"; the values correspond to the actual words in the English sentence, and are weighted based on the Attention scores produced (explained shortly in detail in the next paragraph) by the queries and the keys — these weighted values then form a context-enriched representation of each word in the English sentence.

In the decoder, the goal is to generate the Spanish translation, and the roles of the queries, keys, and values adapt to this objective; queries are derived from the part of the Spanish sentence generated so far, and each query represents the current state of the output sequence, guiding what the model will generate next — if the Spanish sentence has "el defensa" so far, the query assists in deciding what word should come next; in the self-Attention sub-layer, keys and values are also derived from the Spanish sentence generated so far, similar to how it works for the encoder; whereas in the cross-Attention sub-layer, keys and values originate from the encoder's output, allowing the decoder to reference the original English sentence — for generating the next word after "el defensa", the decoder uses keys and values from the encoder's output to reference relevant

parts of “the defender slide-tackled the striker”, ensuring accurate and contextually appropriate translation. In summary, in the encoder, queries, keys, and values, derived from the English sentence, work together to create a context-rich representation of each word. In the decoder, queries (from the Spanish sentence being formed), and keys and values (from the encoder’s output of the English sentence) interact to generate a contextually accurate Spanish translation, word by word.

Transformers employ a simple yet powerful Attention function called the Scaled Dot-Product Attention; one can see its basic structure on the left side of Figure 2.11. This Attention mechanism, as seen in Equation 2.13, calculates Attention scores based on the dot product of the query Q with all keys K; the queries and keys have the same dimension, denoted as d_k. Then, the result is scaled by the square root of the keys’ dimensionality d_k. This scaling helps stabilize the gradients, especially for larger dimensions. The result of this is later normalized by making use of the non-linear softmax function (Equation 2.14) to obtain the weights on the values V. In the case of the first Multi-Head Self-Attention sub-layer in the decoder, masking is employed right before applying the softmax function. Finally, the output produced by the softmax function is multiplied by the actual values V; these values V have dimension d_v. Therefore, the output Attention matrix is computed in [128] as:

$$\text{Attention}\big(Q, K, V\big) \;=\; \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.13}$$

where

$$\text{softmax}(x_i) \;=\; \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.14}$$

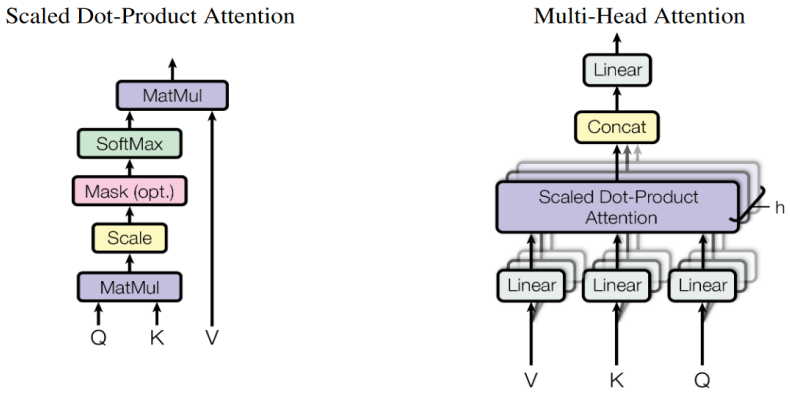


Figure 2.11: **Left:** The (scaled dot-product) Attention mechanism. **Right:** An illustration of multi-head Attention [128].

A key innovation in the Transformer is the use of Multi-Head (Self-)Attention, displayed on the right side of Figure 2.11. Instead of performing Attention once, the Transformer does it multiple h times in parallel, with each ’head‘ attending to different parts of the input sequence. The queries, keys, and values are linearly projected h times (in [128] h = 6 is used) with different learned linear projections to d_k, d_k and d_v dimensions, respectively. The (scaled dot-product) Attention function is performed

in parallel on each of these projected versions, producing d_v-dimensional output values. These outputs are then concatenated and projected again, as shown in Equation 2.15, resulting in the final Multi-Head Attention values [128]. This allows the model to attend to information from different representation sub-spaces at different positions, and hence, capture various aspects of the sequence, like different syntactic or semantic features, enhancing its ability to capture various aspects of the input data.

$$\begin{aligned} \text{Multi-Head Attention}\big(Q, K, V\big) &= \text{Concat}\big(\text{head}_1, ..., \text{head}_\text{h}\big)W^O \\ \text{where} \quad \text{head}_\text{i} &= \text{Attention}\big(QW_i^Q, KW_i^K, VW_i^V\big) \end{aligned} \tag{2.15}$$

where the parameter matrices $W_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^O \in \mathbb{R}^{d_\text{model} \times d_k}$ are the projections. Multi-Head Attention is employed in 3 different parts of the Transformer's architecture: Multi-Head Self-Attention occurs within both the encoder and the decoder components' stacks; and then, Multi-Head Attention takes place between the encoder and the decoder structures, where the input queries for the decoder's second Multi-Head Self-Attention sub-layer originate from the previous decoder's first masked Multi-Head Self-Attention sub-layer, and the remaining input "memorized" keys and values for the decoder's second Multi-Head Self-Attention sub-layer arrive from the output of the encoder [128].

The encoder and decoder in the model each contain a fully connected feed-forward network that operates independently on each position, involving two linear transformations with a ReLU activation in between. The linear transformations vary in parameters across layers and can be viewed as 2 convolutions with a kernel size of 1. The input and output vectors' dimensions are set at d_model, and the dimension of the inner layer is d_{ff} (equal to 2048 in [128]). The output of the decoder is then fed into a learned linear transformation and softmax function to convert it into the predicted next-token probabilities.

Since Transformers lack recurrence in their architecture, they use positional encodings (Equation 2.16) by adding these to the input and output embeddings, which are the inputs for the encoder and decoder stacks respectively, to incorporate information about the position of the tokens in the sequence. These positional encodings have the same dimension d_model as the input and output embeddings. These encodings ensure that the model can recognize and utilize the order of the input. The positional encodings use sine and cosine functions of different frequencies, allowing the model to learn to attend by relative positions [128]:

$$\begin{aligned} \text{PE}_{(pos,\, 2i)} &= \sin\left(\frac{pos}{10000^{2i/d_\text{model}}}\right) \\ \text{PE}_{(pos,\, 2i+1)} &= \cos\left(\frac{pos}{10000^{2i/d_\text{model}}}\right) \end{aligned} \tag{2.16}$$

where i is the dimension and pos is the position. This choice of sinusoidal functions enables the model to interpret the sequence order effectively.

The Transformer model markedly reduces training time compared to RNNs and CNNs. This efficiency stems from its parallelizable nature, enabling it to process entire sequences at once, unlike RNNs that process sequentially. The introduction of Transformers marked a significant advancement in DL and NLP. The Transformer's architecture is highly versatile and its influence extends beyond specific tasks, inspiring a wave of research into Attention-based models, and setting the stage for future innovations in AI and ML, such as Graph Transformer Networks (GTNs).

2.3.4 Graph Transformer Networks

It has been proven in the literature that GNNs are a good approach to leverage the novel Attention mechanism [153]. Since Transformers have just been explained in the previous subsection 2.3.3, it just seems appropriate for the context of this thesis to expose how these have been adapted to also be used in the setting of graph learning and incorporated into GNNs, in a sophisticated architecture called Graph Transformer Networks (GTNs) [154].

I will not go into much technical detail regarding this particular architecture, or explain explicitly how the Attention mechanism is utilized in this network, and will focus rather on the higher-level interpretation even though it is related to the context of this thesis. The reason for this is that I rather focus on the technical aspects of another GTN, namely the Heterogeneous Graph Transformer Networks (HGTs) discussed in the following subsection 2.3.5, since these are the ones that I have utilized in the practical work of this thesis. For anyone interested in further technical aspects of the GTNs discussed in this subsection, please refer to [154]. Nonetheless, since this is still an important network within the literature landscape of the context of this thesis I do not want to fully omit it, and so I will provide a relatively brief overview of GTNs.

GTNs represent an innovative approach in the realm of GNNs. These neural networks are designed to address the limitations of traditional GNNs, which typically operate on fixed and homogeneous graphs. GTNs excel in handling heterogeneous graphs, comprising various types of nodes and edges, by generating new graph structures (i.e. adjacency matrices) for effective node representation learning. This is achieved in an end-to-end manner, enabling GTNs to identify useful connections between unconnected nodes and form meta-paths, which are essentially multi-hop connections. GTNs can be viewed as a graph analogue of Spatial Transformer Networks [155] which explicitly learn spatial transformations of input images or features [154]. A key feature of GTNs is their ability to learn and generate new graph structures (or adjacency matrices) based on data and tasks, without the need for domain-specific knowledge. This process involves learning the soft selection of edge types and the composition of relations to create useful meta-paths; view Figure 2.12 for a conceptual visualization of this. The GTN framework adapts to different datasets by generating appropriate

graph structures, which significantly enhances the effectiveness of node representation compared to traditional methods that rely on predefined meta-paths [154].

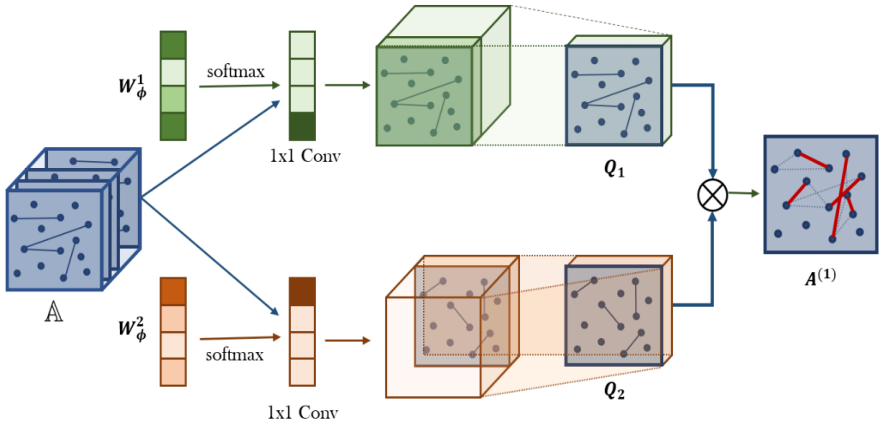


Figure 2.12: Graph Transformer (GTN) layer [154].

At the core of GTNs lies the Graph Transformer layer (Figure 2.12), which is instrumental in learning the soft selection of edge types. This GTN layer generates new graph structures by dynamically identifying beneficial meta-paths, enabling the network to adaptively transform the input graph based on the task at hand. This layer's functionality can be encapsulated in the following Equation 2.17, representing the adjacency matrix A^l of the meta-path graph at the current layer l:

$$A^l = \text{softmax}\left(W^l\right) A^{l-1} \tag{2.17}$$

where W^l is the trainable weight matrix of the current layer l, the softmax function is applied to W^l to normalize the weights, and A^{l-1} is the adjacency matrix from the previous layer $l-1$. This is important for learning effective node representations, therefore improving the model's interpretability [154].

In node classification tasks, GTNs have demonstrated phenomenal performance. By leveraging their ability to learn new graph structures and then applying (graph) convolution on these graphs using GCNs, see Figure 2.13, GTNs have achieved remarkable accuracy in classifying nodes in heterogeneous graphs, outperforming traditional methods that depend on manually defined meta-paths [154].

The forward propagation of the graph convolutions performed by the GCNs [148] in GTNs are computed, following Equation 2.18, as:

$$H^{l+1} = \sigma\left(A^l H^l W^l\right) \tag{2.18}$$

where H^{l+1} is the node feature representation at the next layer $l+1$, σ is a non-linear activation function (such as ReLU, for example), H^l is the node feature representation at the current layer l,

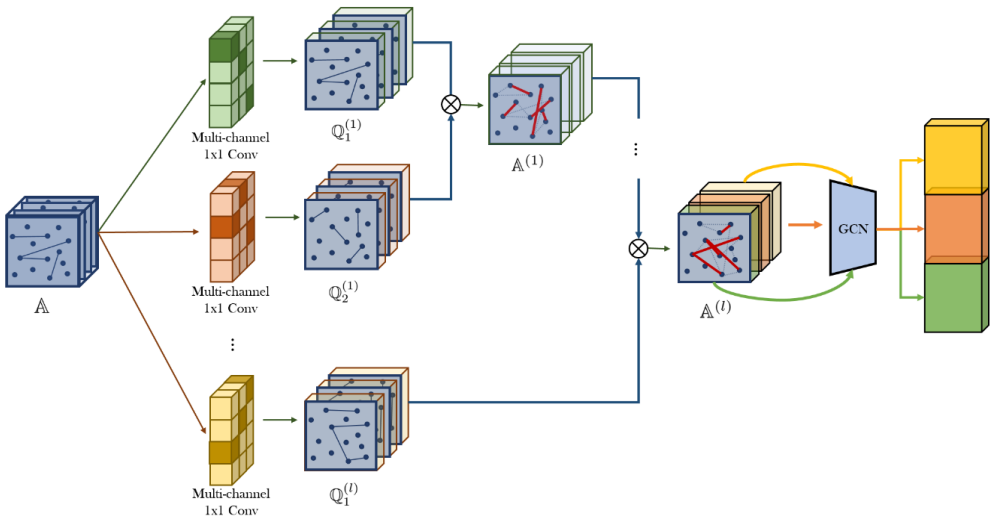


Figure 2.13: Graph Transformer Networks' (GTNs) architecture [154].

and W^l is the trainable weight matrix of the current layer l [154].

GTNs offer significant advantages over earlier GNNs, especially in their handling of heterogeneous graphs. Unlike the first GNNs that were first developed, which often required manual preprocessing to handle different node and edge types, GTNs automatically learn these structures, thus reducing the need for domain-specific knowledge and manual effort. This makes GTNs more versatile and effective in a wider range of applications [154].

GTNs represented a significant step forward in the field of GNNs. By addressing the limitations of traditional GNNs in handling heterogeneous graphs, GTNs opened up new avenues for graph-based learning and analysis. Their ability to learn and generate graph structures (adjacency matrices) autonomously makes them a powerful tool for a variety of applications, setting a new standard for performance and versatility in GNNs. Thus, since heterogeneous graphs have been mentioned quite a bit in this subsection, it is time to talk in-depth about them in the next subsection 2.3.5, dive deep into heterogeneous (graph) learning, and the specific heterogeneous GNN that I applied for this investigation — which is the heart of this thesis from the data science and GDL point of view (as I already exposed in section 2.2 the main related work from the soccer's domain point of view).

2.3.5 Heterogeneous Graph Transformer Networks

Heterogeneous Graph Learning and Neural Networks

Heterogeneous graph learning, a domain at the intersection of Graph Theory and ML of growing importance in the data science and AI fields, delves into the study, manipulation, and learning of the patterns of heterogeneous graphs, i.e. graphs containing multiple types of nodes and edges. These graphs, which are a closer reflection of many of the complex real-world scenarios and networks,

compared to homogeneous graphs, are instrumental in capturing multi-type and multi-modal interactions among different entities [156].

Heterogeneous graphs — from now on referred to as HGs (and when referring to homogeneous graphs, I will just explicitly write the whole name out) — are characterized by their ability to encapsulate different types of entities (represented by nodes) and different types of relations (represented by edges), making them omnipresent in real-world scenarios. They are extraordinary in representing highly complex networks, for instance, traffic dynamics and recommendation systems. Their capability to handle diverse flavours of nodes and edges enables them to model intricate connections in these networks, a capability that is absent in homogeneous graphs. This versatility has spurred significant growth in HG-related research, particularly in data mining, ML, and GDL [157].

Representing data in HGs involves unique challenges. In a HG, each node type and edge type can have different attributes (features) and semantics. Heterogeneous Graph Neural Networks (HGNNs) extend the concept of GNNs from homogeneous graphs to HGs [158; 156]. Therefore, when designing embedding methods, especially HGNNs, it is crucial to address the heterogeneity in attributes to effectively fuse information. This includes considering the structure, which is usually semantic-dependent, and dealing with different feature spaces for different node types [158]. Another characteristic that HGNNs have over traditional GNN models is that HGNNs allow for feature matrices of different dimensions for different node types and different edge types, which is common in HGs and which traditional GNNs cannot deal with. HGNNs address this by allowing different message and update functions for different types of nodes and edges. This adaptability is key to processing the diverse information present in HGs [12]. The aggregation process in HGNNs is crucial for combining information from different types of neighbors. It involves using NN architectures like RNNs [158] and/or Attention mechanisms [158; 159; 160] to aggregate content embeddings from different neighboring groups and balance their impacts based on the node and/or edge types.

While unsupervised HGNNs focus on capturing the generalizable aspects of node embeddings, semi-supervised HGNNs aim for task-specific embeddings, often employing attention mechanisms to emphasize the most relevant structural and attribute information [157]. The Heterogeneous Graph Attention Network (HAN) [159] exemplifies this approach. It uses a hierarchical Attention mechanism to capture both node-level and semantic-level importance, effectively integrating the rich and diverse information present in heterogeneous graphs [157].

These paragraphs have offered an overview of heterogeneous graph learning. Recent advancements in GDL and GNNs have led to the development of frameworks such as *PyTorch Geometric* (PyG) [12] (this being the framework used for this thesis), which provides specialized HGNNs for HGs, but also for homogeneous graphs, based on the literature landscape. PyG is capable of adapting its Message Passing base formulation to accommodate the diversity in node and edge types in HGs. Finally, I arrive at the very last sub-subsection of this section 2.3 on technical data science and GDL

theoretical background. In this last portion, I will talk about the type of HGNNs that I employed for the practical part of this thesis, these being Heterogeneous Graph Transformer Networks (HGTs). I chose HGTs because I believed that these were the most appropriate GNNs that PyG offered and which suited best for the soccer domain context of this investigation, since the ball and the players are modeled as distinct types of nodes, both the ball and players have a different number of features, the graphs constituting of the ball and all players are fully connected, edges connecting players with each other having also a different number of features compared to the edges connecting the ball to each player.

Heterogeneous Graph Transformer Networks (HGTs)

This last sub-subsection, of this section 2.3 on technical data science and GDL theoretical background, exposes some important knowledge on Heterogeneous Graph Transformer Networks (HGTs), which are the GNN architecture that I test in this thesis' investigation.

HGTs are an advanced architecture in GNNs, view Figure 2.14, specifically designed to handle HGs. Unlike traditional GNNs that assume homogeneity in nodes and edges, HGTs address the complexity where different types of nodes and edges exist. The need for such a model arises because real-world data frequently consists of diverse elements interacting in various ways, making homogeneity an unrealistic assumption. In HGs, nodes and edges are associated with different types, and these types play a critical role in the graph's structure and semantics, making them rich in semantic information. However, this heterogeneity poses significant challenges in modeling and learning. Traditional GNNs often fall short in capturing the intricate interactions between different types of nodes and edges. This limitation leads to a loss of valuable semantic information and inaccurate representations of the graph structure [11]. HGs in HGTs are defined as $G = (N, E, A_N, R_E)$, where N is the set of nodes of multiple types, E also represents the set of edges of multiple types, A_N is the set of node types in the graph, and R_E represents too the set of edge types in the graph. The type of each node $\tau(n_i)$ and the type of each edge $\phi(e_{ij})$ is defined by the following mapping functions $\tau(n_i) : N \to A_N$ and $\phi(e_{ij}) : E \to R_E$, respectively.

HGT's unique structure allows it to integrate data from various high-order neighboring nodes via Message Passing between layers. This process effectively creates "soft" meta-paths. Therefore, even when HGT uses only immediate one-hop edges as its input, without inserting any crafted meta-paths, its Heterogeneous Mutual Attention mechanism has the capability to autonomously identify and derive meta-paths that are crucial for a range of downstream tasks in an implicit manner [11; 156].

In HGTs, a critical concept is the use of "meta-relations". A meta-relation in a HG is a triplet $\langle \tau(n_{\text{source}}), \phi(e_{\text{source-target}}), \tau(n_{\text{target}}) \rangle$ that includes the type of the source node $\tau(n_{\text{source}})$, the type of the neighboring target node $\tau(n_{\text{target}})$, and the type of the edge $\phi(e)$ connecting them together. This concept is pivotal in HGTs, as it helps in capturing the heterogeneous nature of the graph more effectively, by assigning different Attention weight matrices to different meta-relations to

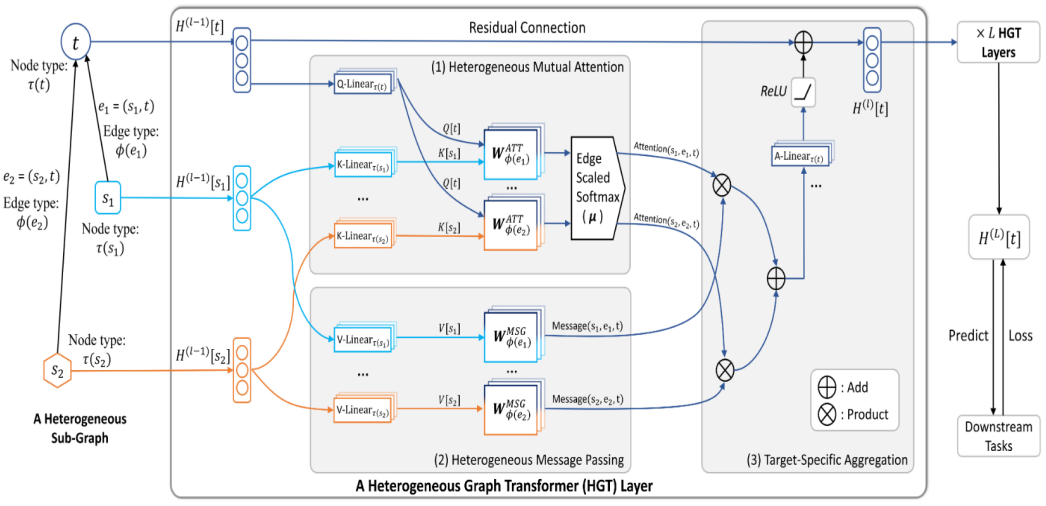


Figure 2.14: HGTs' architecture; node-types are color-coded. Has 3 main components: **(1)** Heterogeneous Mutual Attention mechanism, allowing for meta-relation triplets; **(2)** Heterogeneous Message Passing from the source node(s); **(3)** Target-Specific (Heterogeneous Message) Aggregation [111].

implicitly learn meta-paths, empowering the model to take type information into consideration. By considering these meta-relations, HGTs can model the complex and diverse interactions within the graph, which is not possible with conventional GNNs (designed for homogeneous graphs) [11; 144].

A summarized overview of the workflow of the architecture of HGTs (take a look at Figure 2.14 to follow along) is the next: provided a sampled heterogeneous sub-graph, HGT identifies every connected node pair — in these pairs, a source node n_{source} is linked to a target node n_{target} through an edge e (or $e_{\text{source-target}}$); The primary aim of HGTs is to aggregate information from n_{source} to develop a contextualized representation for the target node n_{target}. This procedure is split into three distinct parts: the "Heterogeneous Mutual Attention" mechanism, "Heterogeneous Message Passing", and "Target-Specific Aggregation". The output from the l-th layer of HGT is represented as H^l and serves as the input for the subsequent $l+1$-th layer. By stacking L of such layers, the representations for all nodes across the graph, and hence of the entire graph, denoted as H^L, are obtained. These comprehensive node representations are then utilized either for training or applied to various downstream tasks [111].

The architecture of HGTs (Figure 2.14) incorporates the Heterogeneous Mutual Attention mechanism — **(1)** in Figure 2.14 — to tackle the limitation that Attention mechanisms in other Attention-based GNNs have regarding their assumption of all node-types having the same feature distributions; in Heterogeneous Mutual Attention, each node-type is allowed to have its own feature distribution. Taking inspiration from the original Transformer [128], this new Attention mechanism calculates the Attention coefficients based on the meta-relations (these triplets being analogous to the Q, K, V triplets from the original Transformers), allowing the model to weigh the importance of each node's message differently depending on the type of relationship (edge) it has with other nodes, by mapping

the target node n_{target} into a 'Query' vector $Q(n_{\text{target}})$, mapping the source node n_{source} into a 'Key' vector $K(n_{\text{source}})$. This approach ensures that the model captures the unique characteristics of each node- and edge-type in the graph [111]. The fundamental distinction lies in the approach towards projections: the standard Transformer applies a uniform set of projections, for instance across all words in the context of machine translation, whereas HGTs assign a unique set of projection weights to each meta-relation. In order to maximize the sharing of parameters but simultaneously preserving the unique attributes of distinct (meta-)relations, HGTs design their weight matrices for interaction operators in a particular way. They divide them into three projections: one for the source node, another for the edge, and a third for the target node. More specifically, HGTs compute the Attention with h heads for each edge defined as $e = (n_{\text{source}}, n_{\text{target}})$. As shown in Equation 2.19, the Attention vector for each node pair is calculated from this novel Attention mechanism as:

$$\begin{aligned} &\text{Attention}_{\text{HGT}}(n_{\text{source}}, e_t, n_{\text{target}}) = \\ &= \text{softmax}_{\forall n_{\text{source}} \in N(n_{\text{target}})} \left(\Big|\Big|_{i=1}^{h} \text{Attention-head}_i(n_{\text{source}}, e_t, n_{\text{target}}) \right) \end{aligned} \tag{2.19}$$

where $||$ represents the concatenation operator (like in [128]), h is the number of Attention-heads, and every Attention-head has a vector dimension of $\frac{d}{h}$ [111]. Within the context of HGTs, Attention-heads are, as observed in Equation 2.20, the dot product between the transformed Key vector K and the Query vector Q, effectively measuring the relevance or 'attention' the target node n_{target} should pay to the source node n_{source} under the context of edge e with edge-type t. The softmax function is applied to normalize the Attention scores across all source nodes linked to the target node n_{target}, ensuring that they sum up to 1. This normalization is non-trivial for the aggregation step in the Transformer architecture. The Attention-heads are computed as:

$$\begin{aligned} &\text{Attention-head}_i(n_{\text{source}}, e_t, n_{\text{target}}) = \\ &= \left[K_i(n_{\text{source}}) \, W_{\phi(e)}^{\text{Attention}}(e_t^T) \, Q_i(n_{\text{target}})^T \right] \cdot \frac{\mu_{\left\langle \tau(n_{\text{s}}), \, \phi(e_{\text{source-target}}), \, \tau(n_{\text{t}}) \right\rangle}}{\sqrt{d}} \end{aligned} \tag{2.20}$$

where $K_i(n_{\text{source}})$ represents the Key vector for the i-th Attention-head, it is a projection of the source node n_{source} into a space where its type $\tau(n_{\text{source}})$ is considered, and the projection is specific to the type of the source node (see Equation 2.21); $W_{\phi(e)}^{\text{Attention}}(e_t^T) \in \mathbb{R}^{\frac{d}{h} \times \frac{d}{h}}$ is a learnable Attention weight matrix that is dependent on the edge-type $\phi(e)$. It allows the model to capture different semantic relations for different edge-types. This matrix transforms the Key vector to align it with the Query vector in the context of the edge-type; Similar to $K_i(n_{\text{source}})$, $Q_i(n_{\text{target}})$ represents the Query vector for the i-th Attention-head. It projects the target node n_{target} into a space considering its node type (see Equation 2.21); and finally, $\mu \in \mathbb{R}^{|A_N| \times |R_E| \times |A_N|}$ is a prior tensor, called Edge-Scaled Softmax (examine **(1)** in Figure 2.14), to represent the general importance of every meta-relation triplet. This is based on the understanding that not all edges have the same level of influence on the target nodes. Therefore, μ works as a flexible scaling factor within the Attention mechanism, adjust-

ing to reflect the varying contributions of different edges to the target nodes. [11]. The respective Query and Key vectors of the i-th Attention-head are projected as follows:

$$\begin{aligned} K_i(n_{\text{source}}) &= \text{K-Linear}^i_{\tau(n_{\text{source}})}\Big(H^{l-1}[n_{\text{source}}]\Big) \\ Q_i(n_{\text{target}}) &= \text{Q-Linear}^i_{\tau(n_{\text{target}})}\Big(H^{l-1}[n_{\text{target}}]\Big) \end{aligned} \tag{2.21}$$

To finish off the Heterogeneous Mutual Attention mechanism, for each target node n_{target}, the method involves collecting all the Attention vectors from its neighboring nodes $N(n_{\text{target}})$. Following this, a softmax function is applied to these vectors, as previously shown in Equation 2.19. This process ensures that the sum of all Attention values from the neighbors of n_{target}, denoted by $\sum_{\forall n_{\text{source}} \in N(n_{\text{target}})} \text{Attention}_{\text{HGT}}(n_{\text{source}}, e_t, n_{\text{target}})$, equals a ones-vector $\vec{1}$ of dimensions $h \times 1$, thereby normalizing the Attention scores [11].

The Heterogeneous Message Passing is another important component in HGTs. It occurs in tandem with computing the Heterogeneous Mutual Attention, inspect **(2)** in Figure 2.14, where there is a transfer of information from source nodes to target nodes. This process, similar to the new Attention mechanism, aims to include the meta-relations associated with edges in the Message Passing phase. The purpose here is to mitigate the disparities in the distributions of nodes and edges that belong to different types. For the edge e of type t of a pair of nodes $e_t(n_{\text{source}}, n_{\text{target}})$, the multi-head Message is calculated in this parallel process, following Equation 2.22 as:

$$\text{Message}_{\text{HGT}}(n_{\text{source}}, e_t, n_{\text{target}}) = \Big|\Big|_{i=1}^{h} \text{Message-head}_i(n_{\text{source}}, e_t, n_{\text{target}}) \tag{2.22}$$

$$\text{where} \quad \text{Message-head}_i(n_{\text{source}}, e_t, n_{\text{target}}) = \text{M-Linear}^i_{\tau(n_{\text{source}})}\Big(H^{l-1}[n_{\text{source}}]\Big)\, W^{\text{Message}}_{\phi(e)}$$

where $W^{\text{Message}}_{\phi(e)} \in \mathbb{R}^{\frac{d}{h} \times \frac{d}{h}}$ is a learnable Message weight matrix that is dependent on the edge-type $\phi(e)$. It allows the model to capture different semantic relations for different edge-types. To compute the i-th Message-head, denoted as $\text{Message-head}_i(n_{\text{source}}, e_t, n_{\text{target}})$, the process begins by transforming the source node n_{source} of type $\tau(n_{\text{source}})$ into the i-th Message vector. This transformation is achieved through a linear projection, $\text{M-Linear}^i_{\tau(n_{\text{source}})}$, which maps from $\mathbb{R}^d \rightarrow \mathbb{R}^{\frac{d}{h}}$. Following this, the Message weight matrix $W^{\text{Message}}_{\phi(e)}$, is employed to incorporate the dependency of the edge-type. The concluding step involves concatenating all h Message-heads to form the $\text{Message}_{\text{HGT}}(n_{\text{source}}, e_t, n_{\text{target}})$ for each pair of nodes [11].

On a higher-level interpretation, this new heterogeneous version of Message Passing allows the model to aggregate information from neighboring nodes during forward propagation but does so in a way that respects the graph's heterogeneity. Each node receives messages from its neighbors, and these messages are transformed based on the type of the node and the relationship (edge-type) it

shares with its neighbors [11].

After computing both the Heterogeneous Multi-head Attention and Heterogeneous Multi-head Message, the next step, called Target-Specific Aggregation, is to aggregate these from the source nodes for the target node n_{target} (Equation 2.23). It is important to remember that the softmax step, as outlined in Equation 2.19, normalizes the Attention vectors for each target node n_{target} so that their sum equals one. This normalization allows for the use of the Attention vector as a weighting factor. By employing this weighting factor, we can average the Messages originating from the source nodes [11]. This process results in an updated vector, represented as $\tilde{H}^l[n_{\text{target}}]$:

$$\tilde{H}^l[n_{\text{target}}] = \oplus_{\forall n_{\text{source}} \in N(n_{\text{target}})} \Big[\text{Attention}_{\text{HGT}}(n_{\text{source}}, e_t, n_{\text{target}}) \cdot \\ \cdot \text{Message}_{\text{HGT}}(n_{\text{source}}, e_t, n_{\text{target}}) \Big] \tag{2.23}$$

where $\oplus$ is the point-wise adding operation, for all the neighboring source nodes adjacent (connected) to the target node. This process aggregates data for the target node n_{target} from all its adjacent neighbors (the source nodes), each having unique feature distributions. The ultimate phase, Equation 2.24, involves mapping the target node n_{target}'s vector back with its type-specific distribution, which is indexed by its respective node-type $\tau(n_{\text{target}})$. To achieve this, the node embedding updated vector $\tilde{H}^l[n_{\text{target}}]$ is subjected to a linear projection, labeled A-Linear$_{\tau(n_{\text{target}})}$. This is then followed by the application of a non-linear activation function σ, and complemented by a Residual Connection, as outlined in [151]:

$$H^l[n_{\text{target}}] = \sigma\Big(\text{A-Linear}^i_{\tau(n_{\text{target}})} \tilde{H}^l[n_{\text{target}}]\Big) + H^{l-1}[n_{\text{target}}] \tag{2.24}$$

In this manner, one obtains the output of the l-th layer of the HGT, denoted as $H^l[n_{\text{target}}]$, for the target node n_{target}. Real-world graphs often exhibit a "small-world" property, meaning that by stacking a few HGT layers L (where L is a relatively small number), each node can effectively connect with a significant portion of the other graph's nodes, which may vary in types, edges and edge-types connecting them. Consequently, HGT is capable of producing highly contextualized representations H^L for every node. This representation can be integrated into various models to execute downstream tasks in HGNNs, such as node classification, graph classification and edge prediction [11].

As demonstrated throughout this sub-subsection, the overall architecture of the HGT model heavily relies on the meta-relation — $\langle \tau(n_{\text{source}}), \phi(e_{\text{source-target}}), \tau(n_{\text{target}}) \rangle$ — to separately parameterize each of the learnable weight matrices. This approach balances model capacity and computational efficiency. Unlike the standard Transformer in [128], which uses uniform operators, HGTs tell apart the operators for diverse node- and edge-types, making it more skillful at handling the distribution variances found in HGs. Compared to existing models that use distinct matrices for each entire meta-relation, HGT's triplet parameterization more effectively utilizes the HG schema for parameter sharing. This parameter sharing allows (meta-)relations with fewer occurrences to quickly adapt and generalize due to shared parameters. Simultaneously, the operators for different

relationships (edge-types) can still retain their unique properties by utilizing a considerably smaller set of parameters [111].

HGTs are designed to handle web-scale graph data efficiently. This capability is facilitated by a specialized Mini-Batch graph sampling algorithm dubbed as "HGSampling". This innovative approach enables efficient and scalable training of HGTs on large-scale graphs by ensuring a balanced representation of different node- and edge-types in the sampled sub-graphs [111]. The development of HGTs opened up new avenues in graph learning, especially in handling complex and large-scale HGs. Future research may explore extending HGTs with more advanced Attention mechanisms, for instance. Incorporating temporal dynamics to GNNs is already under exploration [161]. HGTs represent a significant advancement in the field of GNNs. Their ability to efficiently model the complex and varied interactions in HGs makes them a powerful tool for a wide range of applications and contexts, including sports data science and analytics. This thesis, for instance, aims to apply HGTs in soccer – a sport well-known for its high complexity. Here, HGTs will be applied for a binary graph classification task (see Figure 1.1), specifically classifying soccer shots, as a scored shot or a non-scored shot, based on the positions and other attributes of the ball and all 22 players at the instant of the shot-attempt. This approach inherently includes predicting the likelihood of shots resulting in a goal. Transitioning from this theoretical foundation, the subsequent chapter 3 delves into the methodology employed, detailing how these advanced graph learning techniques are practically applied in soccer analytics.

Chapter 3

Methodology

In this chapter, I delve into the intricate methodology that underpins the research presented in this thesis. My exploration commences with section 3.1, where I describe the data utilized and its significance in the context of soccer analytics. Following this, in section 3.2 I address the data cleaning (subsection 3.2.1), feature engineering (subsection 3.2.2), data augmentation (subsection 3.2.3), and data transformation (subsection 3.2.4) methods, highlighting the steps taken to prepare and refine the data for effective analysis. I then explain in section 3.3 the Cross-Validation (CV) strategy carried out, detailing the robust approach adopted to ensure the validity and generalizability of the models. Subsequently, in section 3.4, I very briefly state the architecture of the two pivotal models of this thesis: the traditional 'Expected Goals' (xG) model and the 'Heterogeneous Graph Transformer' (HGT) model. This drives us to the models' training-validation procedure in section 3.5, where I outline the policies employed in training and validating the models to obtain the best configuration of hyper-parameters for the models, followed by the metrics explored to assess the performance of these models during the models' evaluation protocol in section 3.6. Each section is designed to provide a comprehensive understanding of the methodologies employed, ensuring a thorough and transparent insight into the mechanics of this research.

3.1 The Data

The data used for this research is a combination of spatio-temporal contextual information in the form of 'tracking data' and possessions-based 'event data', for 361 soccer matches, which was generously provided by the German soccer club Hertha BSC. The 'event data' that is commonly used by soccer clubs is a log in which each on-ball event (passes, tackles, shots, interceptions, etc.), disciplinary event (bookings, player substitutions, etc.), the timestamp and location of each of these events is recorded. The limitation of event data is that it does not include information about the players who were not interacting with the ball at each log entry. Thus, if one intends to also take into account and capture more information related to the rest of the players on the pitch who are not interacting directly with the ball, another form of data, called 'tracking data', is required.

3.1.1 Possessions-based Data

The possessions-based data I was provided with was pre-derived by Hertha BSC from the tracking data that will be described in the following subsection 3.1.2, and it explicitly describes a chronological log of the possessions occurring within a soccer match, for all 361 matches, and thus shows which player is in possession of the ball and for how long (or how many frames). The dimensions of this set of data is of (674 786, 35). Even though there are 35 columns present, only several are crucial for the context of this thesis, since columns such as '**nextcornerfor_timeto**', meaning how many frames are left for the next corner in favour of the home team to occur, or, '**is_setpiece**', a binary variable stating whether the current possession was a set piece or not, are not relevant, and so they do not have to be taken into account. The initial 12 relevant columns taken into account for the context of this research are shown in the Table 3.1. I say 'initial' 12 relevant columns, because these were the relevant ones when I was provided with the data, but later I performed some feature engineering (see subsection 3.2.2), and so a few more relevant columns were created.

3.1.2 Tracking Data

In general, tracking data fills in the gap for the limitations of event data, by providing the positions of all 22 players and the ball on the pitch at any given instance of the game. The tracking data I was provided with originates from live TV broadcast footage at a frequency of 25 Hz (i.e. 25 frames per second, or what is the same, 25 rows of data represents 1 second of the soccer match); originally, this data is provided to Hertha BSC by Skill-Corner at a frequency of 10 Hz, but Hertha BSC pre-upscaled this frequency to 25 Hz before handing it over to me, to match the frequency of the state-of-the-art optical tracking data (this being instead tracking data collected by installed high-tech cameras in the stadiums, together with the application of computer vision techniques). This tracking data for the 361 matches belongs to a mixture of matches played during the German Bundesliga and 2. Bundesliga in the season 2021/2022.

When loading the tracking data of every single soccer match on a Pandas DataFrame (1 for every match), they all contained the same 107 columns, but on average each match consisted of 150 000 rows (frames) of data. Therefore, on average, the dimensions of each set of tracking data are (150 000, 107). Since the data is so voluminous, all of these 361 matches were stored in 361 different '.parquet' files, which made it very tedious to explore and handle the tracking data. As mentioned in the previous subsection 3.1.1, even though there are 107 columns present, only a portion of these are crucial for the context of this thesis. Due to the nature of live TV broadcast footage, it can happen that at some points during soccer matches, while the ball is in play, the cameraman points and focuses the camera on fans on the stands, making a zoom-in on a player running on the pitch or on the referee, or even behind the scenes of the live-stream it is decided that a specific play should be repeated for the fans watching from home on the TV. This has the inconvenience that these moments could coincide with the ball being in play, and thus for the tracking data to not be able to be recorded accurately; for these situations, some variables called 'Visable' (for the ball) and 'VisableX' (for player 'X') are present in the tracking data. Having in mind that this research intends to analyze shot-attempts, and following the logic that in the vast majority of times that a shot-attempt occurs, it occurs in an important relevant area of the pitch where it is important that

Column Name	Description
frames	Match's frame (number) in which the current possession started
players	Current player (ID) in possession of the ball
teams	Team's ID (team "1.0" or "2.0" — team "1.0" being the home team in the match), indicating which team is currently in possession of the ball
duration	Duration (in number of frames) of the possession of the ball of the current player in possession
is_start	Boolean indication on whether it is the start or the end of the possession of the ball of the current player
MatchId	Match's ID
nextxgfor	Float value (in [0, 1]) indicating the sum of the xG values produced by the possessions (players' actions) leading up to a shot for the team currently in possession, if a shot-attempt is made within the next 10s; Otherwise "0.0"
nextxgagainst	Float value (in [0, 1]) indicating the sum of the xG values produced by the possessions (players' actions) leading up to a shot against the team currently in possession, if a shot-attempt is made within the next 10s; Otherwise "0.0"
nextgoalfor	Match's frame (number) in which a goal will occur for the team currently in possession
nextgoalagainst	Match's frame (number) in which a goal will occur against the team currently in possession
nextgoalfor_timeto	Number of frames remaining for the next goal for the current team in possession
nextgoalagainst_timeto	Number of frames remaining for the next goal against the current team in possession

Table 3.1: Table of the relevant columns in the possessions-based data, at the time of receiving the data.

the live TV stream shows the fans at home what is occurring, for simplicity purposes, I made the assumption that all shots must have been broadcasted. Thus, I did not take into account any of these mentioned variables.

The 'initial' 48 relevant columns taken into account for the context of this research are shown in the Table 3.2. One additional note to have in mind is that for each tracking data DataFrame for each match, the frame number of the match is represented by the index of the DataFrame.

Due to possible privacy conflicts, Hertha BSC (the data provider) anonymized the data by encoding the names of players and teams. For instance, the player jersey numbers 1 to 22 are used so they do not align with the ground truth jersey number of the players. Since I have just mentioned the

Column Name	Description
X, **Y**, **Z**	The ball's 3D (x-, y-, and z-) coordinates (in [cm] units)
X1 - **X11**	The x-coordinate of the home team's players; "1" always represents the home team's goalkeeper (in [cm] units)
X12 - **X22**	The x-coordinate of the away team's players; "12" always represents the away team's goalkeeper (in [cm] units)
Y1 - **Y11**	The y-coordinate of the home team's players (in [cm] units)
Y12 - **Y22**	The y-coordinate of the away team's players (in [cm] units)
BallPossesion	Team currently in possession of the ball (team "1.0" or "2.0" — team "1.0" being the home team in the match)
Section	Current half of the match being played in ("1.0" for the 1st Half or "2.0" for the 2nd Half)

Table 3.2: Table of the 'initial' relevant columns in the broadcast tracking data, at the time of receiving the data.

positional coordinates of both the ball and the players in Table 3.2, it seems appropriate to also mention the coordinate system on the soccer pitch. In Figure 3.1, I provide a simple visualization of the players on the pitch. The origin (0, 0) of the coordinate system is located at the very center of the pitch where kick-off takes place (the center spot). The very far right goal line is situated at (+) 10 500 cm from the halfway line, and the top touchline is located at (+) 6 800 cm from the center spot; logically, the same coordinates are used in the opposite direction, in negative coordinates, for symmetry. For visualization purposes, a margin of 300 cm was allowed to surround the playable part of the pitch.

To make sense of both sets of data together, I synchronized both the possessions and tracking data together, to ensure they are aligned with each other, and to incorporate via feature engineering multiple crucial features for the context of this research. The procedure to achieve this is described within the following section 3.2, in subsection 3.2.2. In the mentioned subsection, I will also outline how the following information was extracted, but for now, I will simply display in Figure 3.2 the 'initial' distribution of the total number of shots and those shots that were scored (goals): 8053 shots were made, and 933 goals were scored.

As it can be appreciated, there is a very high-class imbalance of approximately 9:1; approximately for every 10 shots, 9 are not scored and 1 is scored (11.6% of the shots are scored). This poses a great challenge for the research at hand since this is something that must be taken into heavy consideration, in order to achieve a solid and unbiased classifier.

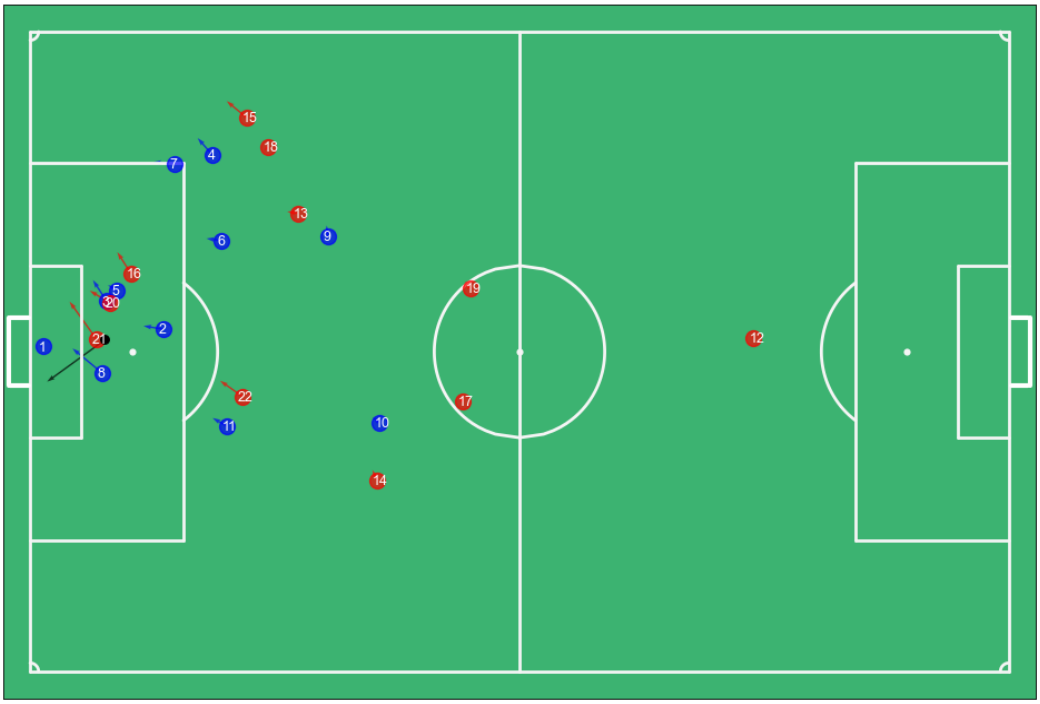


Figure 3.1: Players and the ball on the pitch of an arbitrary frame of an arbitrary match in the tracking data. Arrows represent their velocity vector. Blue players are the home team's players, the red players are the away team's players, and the black dot represents the ball.

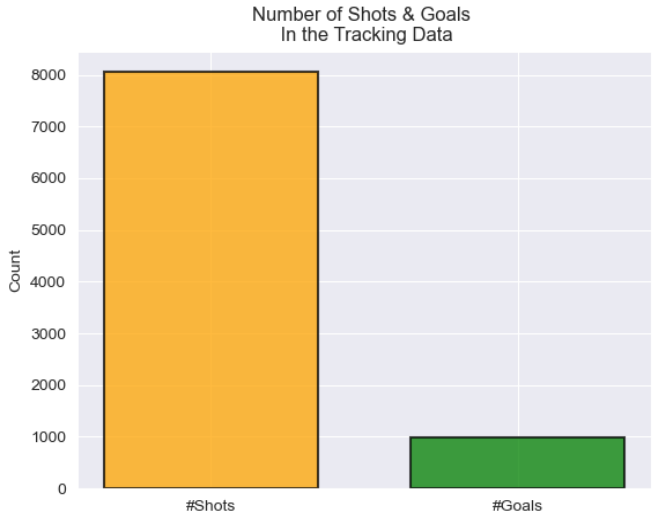


Figure 3.2: Distribution of total number of shots and the scored shots (goals) in the tracking data (for all 361 matches).

3.2 Data Handling

This section on the methods employed to handle the data is pivotal in shaping the foundation upon which the analytical models are built. It begins with 'Data Cleaning' (subsection 3.2.1), an important process where we refine the datasets by removing irrelevant columns and inspecting for missing or 'NaN' values, ensuring the reliability and accuracy of the data. Next, in 'Feature Engineering' (subsection 3.2.2), I delve into how some crucial relevant features are engineered and extracted from the already available data, enhancing the datasets' potential to yield meaningful insights. This is followed by 'Data Augmentation' (subsection 3.2.3), where I strategically expand the final sets of data to address issues of having little data (for the context of this thesis) and enhance model robustness. Finally, 'Data Transformation' (subsection 3.2.4), encompasses the techniques utilized to convert the data into a format that is optimized for the models used, ensuring it is in the ideal state for effective analysis and modeling. Each of these steps is meticulously executed to guarantee the integrity and quality of the data, which is central for the success of the subsequent modeling phases.

3.2.1 Data Cleaning

As it was shortly mentioned in the previous section 3.1, many of the present columns in both of the sets of data (possession and tracking) were irrelevant to the context of this thesis, and so they were directly dropped, leaving us with the columns that were mentioned in Tables 3.1 and 3.2. Then, an inspection for the presence of missing and 'NaN' values was carried out for the remaining relevant columns. Luckily, for both possession and tracking data, all relevant columns had no missing or 'NaN' values, except for the '**nextgoalfor**', '**nextgoalagainst**', '**nextgoalfor_timeto**', and '**nextgoalagainst_timeto**' columns in the possessions data, which contained 'NaN' values. But this was not a problem, because these columns (as well as the '**nextxgfor**' and the '**nextxgagainst**' columns) would not be kept for the final set of usable data for the models, but instead were used to engineer some important features (further described in the next subsection 3.2.2) which would then be used for the final set of usable features. Therefore, overall, no imputation methods were required to be performed. It was realized that these 'NaN' values had a particular pattern in the possessions data.

For simplicity, let's say a match score was 1-0 for the home team. Since these columns describe when will the next goal occur (for or against the team currently in possession) and how much time (in number of frames) is left for that to occur within the match, 'NaN' values were placed in these columns for the rest of the match whenever this event was not going to occur anymore. For instance, in this simple example, for the home team, 'NaN' values would be placed in these columns for all the possessions of that team after their 1st and only goal was scored, and for the away team, all their possessions would contain a 'NaN' value in these columns for this match because they never scored a goal. This motivates to move on to the next subsection, to describe how these mentioned features, and others, were used to create and extract useful features which most were then used in

the final set of features to input into the models.

3.2.2 Feature Engineering

In this subsection, I will first explain the features engineered for the possessions data, and then for the tracking data. In between both, I will outline how both the possession and tracking data were synchronized to obtain tracking data containing a binary indicator for all frames (rows of data), representing the instant that a shot-attempt and the instant that a goal occurred within all 361 matches.

Feature Engineering On Possessions Data

Since the main events of interest in this research are the shots occurring in a soccer match, and mostly whether they resulted in a goal or not, the first features that were aimed to be extracted from the possessions data were the shots and the goals occurring in the matches, using the mentioned '**nextxgfor**', '**nextxgagainst**', '**nextgoalfor**', '**nextgoalagainst**', '**nextgoalfor_timeto**', and '**nextgoalagainst_timeto**' columns. Since the data I was provided with was possessions-based "event" data, structured in frame windows of possessions (as one can deduce and understand from the data description in Table 3.1), without a column that stated the exact frame number in which a (or the next) shot would occur — unlike traditional event data — or a column from which this information could be extractable, it was not possible to extract the exact moment (or frame) in which a shot occurred. Instead, the best that it was able to be done, was to extract a fixed frame window of 250 frames (i.e. 10s) in which a shot would be occurring, using the '**frames**', '**teams**', '**nextxgfor**', '**nextxgagainst**', '**nextgoalfor**', '**nextgoalagainst**', '**nextgoalfor_timeto**', and '**nextgoalagainst_timeto**' columns. This inaccurate extraction of when exactly do shots occur is possibly the largest and main source of noise and inaccuracy (in the data) in this research, even though some meticulous data checks were performed to try and attenuate this.

To obtain the shots information and extract when goals occurred 2 new binary columns, '**Will_Be_a_Shot**' and '**Will_Be_a_Goal**', were generated (Figure C.1). These features are vital for the predictive task at hand in this thesis, since '**Will_Be_a_Shot**' will be employed to filter and extract the approximate frames (and hence, rows of data) in the tracking data (when they are later synchronized) in which a shot occurred; and '**Will_Be_a_Goal**' will indicate from those filtered shot frames (rows), which shots were scored and which were not, becoming the target variable for the models.

- Creating '**Will_Be_a_Shot**': This binary column indicates whether a shot-attempt will occur within the next 10s (250 frames). This was determined using the '**nextxgfor**' and '**nextxgagainst**' columns, if either of these columns has a non-zero positive value, it suggests that a shot occurred within the next 10s (250 frames), thus setting '**Will_Be_a_Shot**' to 1.0.

- Creating '**Will_Be_a_Goal**': Similar to '**Will_Be_a_Shot**', this binary target variable indicates whether a goal will be scored within the next 10s (250 frames). This was determined based on the time-to-next-goal columns ('**nextgoalfor_timeto**' and '**nextgoalagainst_timeto**'), with a goal being flagged with a 1.0 if it is expected to occur within the set 250-frame window.

Then I handled unrecorded goals (which are also shots); meticulous data checks were carried out for 8 different spotted scenarios where goals were clearly scored but were not properly recorded in the dataset due to possessions data logging issues. This was accomplished through a series of conditional checks across consecutive rows (representing sequential possessions):

- Scenarios 1 & 3 (Figure C.2): These scenarios check for instances where consecutive possessions belong to the same team being in possession (either the home or the away team), and there was an increment in the team's goal count ('**nextgoalfor**'). This implied a goal had been scored but not recorded, necessitating an update in both '**Will_Be_a_Shot**' and '**Will_Be_a_Goal**' to indicate a shot and a goal occurrence.
- Scenarios 2 & 4 (Figure C.2): Similar to before, but focusing on the opposition team's goals ('**nextgoalagainst**'). This check ensures that goals scored against the possession team were captured, even if they were not logged properly in the data.
- Scenario 5 (Figure C.3): This checks when the possession switches from the home team (1) to the away team (2). It identifies the case where a goal was scored for team 1 (in possession) but it was reflected in the data only after the possession had switched to team 2. The conditions ensure that the frame in which the goal was expected ('**nextgoalfor**') was different before and after the possession change and that the frame count aligned with the frame-timing of the goal. If these conditions were met, it was inferred that a goal was scored but not properly logged, and the values in '**Will_Be_a_Shot**' and '**Will_Be_a_Goal**' were updated accordingly.
- Scenario 6 (Figure C.3): Similar to scenario 5, but focuses on a goal scored against team 1 (the team initially in possession). It checks for a scenario where the goal against team 1 ('**nextgoalagainst**') becomes a goal for team 2 after the possession change. The conditions confirm the timing and frame alignment for the goal occurrence. If met, it indicated an unrecorded goal against the team initially in possession, updating the '**Will_Be_a_Shot**' and '**Will_Be_a_Goal**'.
- Scenario 7 (Figure C.3): This scenario is the counterpart to scenario 5, where the possession switches from team 2 to team 1. It checks for a goal scored for team 2 but was only reflected in the data when team 1 gained possession. The conditions assessed the frame numbers and timing of the goal. Meeting these conditions implied a goal occurrence during team 2's possession that was not properly logged.
- Scenario 8 (Figure C.3): Counterpart to scenario 6, focusing on team 2 initially in possession and the next possession entry in the data switching to team 1. Checks for unlogged goals against team 2, which became apparent in the data only after the possession switch. Conditions verified the alignment of goal timing and frame counts. If true, it indicated a goal was scored against team 2 that was missed in the possessions' log.

These checks ensure that (probably not all, but) many unrecorded goals were captured in the target variable '**Will_Be_a_Goal**'. The code of the conditional checks for these 8 different scenarios can be found in the Appendix C. For simplicity, it was assumed that whenever a goal occurred, a shot-attempt also must have occurred.

Feature Engineering On Tracking Data

In this sub-subsection the feature engineering performed on the tracking data is outlined, where the main objectives were to synchronize (or link) the possessions data to the tracking data, and exclusively from the tracking data to extract: the ball's and the players' velocities and speeds; the ball's distance and angle to the target goal (the goal to which all shots were enforced to be shot at); the players' distance and angle to the target goal; the distance between the ball and every player, and the distance between each pair of players on the pitch; a binary indicator for each player, flagging whether their team was currently in possession of the ball or not; and another binary indicator for each pair of players on the pitch, flagging whether these 2 players in the pair belong to the same team or not — all of these features were derived from the positional (tracking) data.

The continuing 4 paragraphs describe the way in which the possessions and the tracking data were linked and synchronized to identify when shots and goals occur in each of the 361 sets of tracking data. First, due to the nature of the '**nextxgfor**' and the '**nextxgagainst**' columns (review Table 3.1 if needed), and the fact that their positive non-zero values are recorded with precision to the 5^{th} decimal place, for simplicity purposes it was assumed that every positive non-zero values in these columns were unique for each shot, and thus could be used as an indication of when a unique shot-attempt occurs (within the next 10s or 250 frames); since the probability of different sequences of possessions leading up to a shot-attempt resulting with the exact same value for their sum of xG, to the 5^{th} decimal place, is relatively low. Having in mind these considerations, I first extracted the list of unique sum of xG values recorded, for each of the 361 matches, within the '**nextxgfor**' and '**nextxgagainst**' columns in the possessions data. Then, the tracking data for a specific match was loaded into a Pandas DataFrame, and a new column called '**(Sum_of)_xG**' was initialized. Subsequently, in the possessions data, all the possessions belonging to that particular match (using the '**MatchId**' column) were extracted, and following from those, only the possessions leading up to a shot were identified (i.e. '**nextxgfor**' or '**nextxgagainst**' > 0). Later, using the '**is_start**', '**frames**', '**duration**', '**nextxgfor**' and '**nextxgagainst**' columns from the possessions data of all those possessions leading up to a shot and the loaded tracking data for that particular match, a series of *if* conditions were designed to assign the unique '**nextxgfor**' and '**nextxgagainst**' values into the new '**(Sum_of)_xG**' column in the tracking data for the appropriate frames (i.e. rows of data) leading up to a shot in that match. This was repeated another 360 times to integrate this column and logic in the tracking data of all the matches.

Once this was achieved, the second step was to integrate the binary '**Will_Be_a_Shot**' column from the possessions data into the tracking data. This was very simple, as it followed the same logic as when created for the possessions data, it only takes to initialize a new '**Will_Be_a_Shot**' column

in the tracking data of each match and assign a value of 1.0 in the new column to all those frames (rows) where the '**(Sum_of)_xG**' column has a positive non-zero value.

To finish off, I identified the frames in the tracking data of each match where goals occurred, using the '**nextgoalfor**' and '**nextgoalagainst**' columns in the possessions data. This information was integrated into the match tracking datasets by adding a new binary column '**Will_Be_a_Goal**'. This column flags with a value of 1.0 all those rows where a goal will occur within the specific 250 frame-window (10s) of the possession and for the 125 frames (5s) after the goal is scored. Not only the exact frame of goal occurrence (given by '**nextgoalfor**' and '**nextgoalagainst**') was considered, but this flagging included the lead-up and players' celebration of the goal, incorporating a window of 10s before and 5s after the goal. This frame-window was also incorporated for the goals because the tracking data is relatively noisy; for example when this frame-window was not included, many times the '**Will_Be_a_Goal**' would have a value of 1.0 several frames after the last 1.0 value in the '**Will_Be_a_Shot**' column, and we want these to coincide when a goal occurs, so using the frame-window ensures these align when appropriate.

The subsequent paragraphs, until the end of this subsection, state the features that were engineered exclusively from the tracking data. The match tracking dataset of each of the 361 matches was taken and enriched by calculating the velocity components and speeds for both players (in 2D) and the ball (in 3D). The velocity components were calculated, for instance, for the x-component as $v_x = \frac{dx}{dt}$, where dx computes the positional difference (in the x-coordinate) in consecutive frames, and dt computes the time difference between consecutive frames. A Savitzky-Golay smoother, with a configurable smoothing window and polynomial order that fits local polynomials to the data for reducing the noise, was employed providing a smoothed estimate of the velocity, and making the data more representative of actual player and ball movements. A window size of 150 frames and a polynomial order of 1 were used — these values were chosen, based on the values that produced the best dynamical (video) visualizations of players (and the ball) moving around the pitch (where the velocity vectors were shown), via trial and error; the link to these visualizations can be found in Appendix B.1. The function accounts for the discontinuity in tracking data between the two halves of a soccer match. It separately processes the data for each half, ensuring the smoothing operation does not erroneously span the halftime interval. Once all the velocity components were calculated, the speed was computed as $|v| = \sqrt{v_x^2 + v_y^2}$. This function sets a realistic threshold for a maximum speed of 1200 cm/s to filter-out potential data errors and outliers, ensuring the integrity of the velocity calculations. The names given to all of these new 70 columns in the tracking data are the following: '**V_xBall**', '**V_yBall**', '**V_zBall**', and '**SpeedBall**' for the ball; '**V_x1**' to '**V_x11**' for the home team players' velocity in the x-component, '**V_x12**' to '**V_x22**' for the away team players' velocity in the x-component, '**V_y1**' to '**V_y11**' for the home team players' velocity in the y-component, '**V_y12**' to '**V_y22**' for the away team players' velocity in the y-component, and the same concept applies for the speed of each player with column names '**Speed1**' to '**Speed22**'.

Once the velocities and speeds were calculated for each player and the ball, I decided to filter and extract all those frames (rows of data), from every set of tracking data of every one of the 361

matches, in which a shot was taken; for every series of 1.0 values in the '**Will_Be_a_Shot**' column (indicating that a shot occurred at some point within those frames with a value of 1.0) in the tracking data, I decided to use the very last frame in which a 1.0 value occurs before it changes to a 0.0 value again. This is not trivial because, as I mentioned earlier in sub-subsection 3.2.2, this was probably the main source of noise and inaccuracy in the data that I ended up using as input for the models, since this selected frame many times did not coincide with and belong to the exact moment a shot was actually taken by a player — many times it was several frames earlier or after a shot was taken, but I was not able to come up with a better way to extract the exact instant of the shot for every shot in all the matches with the data I had. Now that we have this data structure, we can interpret each row of data as a frozen frame or snapshot of the instant a shot-attempt happened during a match.

Once all of these steps were taken, I could then extract the count of how many shots and goals occurred in the tracking data throughout all the matches (8 053 shots and 933 goals), producing Figure 3.2, which was shown in the previous section 3.1 (subsection 3.1.2). This demonstrates a very large class imbalance, as 7 120 shots were not scored and only 933 shots were scored (goals). This class imbalance is a very delicate and crucial aspect of the data that must be handled and tackled appropriately, as it will be shown in subsequent parts of this thesis, such as in subsection 3.2.3, section 3.3, section 3.5, and in section 3.6.

After extracting the frames of the instants in which only shots occurred, the shooting direction of these shots was adjusted in such a way that the shots were enforced to occur on, or aim at one same 'target' goal (I chose it to be the one on the right side of the pitch) based on which team was in possession of the ball at that instant, as this is the common setting in xG models, to avoid the directionality of shots depending on whether the shot was attempted by the home or the away team and in which half of the match it occurred in (1^{st} or 2^{nd}, determined by the '**Section**' column). To achieve this, it was first enforced for the home team to always attack from left to right and for the away team to attack from right to left, throughout the entire match, regardless of the half of the match in which it was being played. Then, if the shot frame (or row of data) indicated that the home team was in possession of the ball at the instant of the shot, no modifications were made; but if the shot frame indicated that the away team was in possession of the ball at the instant of the shot-attempt, then all the x- and y-coordinates, and all the x- and y-velocity components of the ball and all the players were mirrored. Now that all the shots are oriented towards the same target goal, we no longer identify the teams as the 'home' or the 'away' teams, but instead, as the 'attacking' or the 'defending' teams.

At this point, since I only had 8 053 rows of data (frames) representing all the shot-attempts present in the entire set of tracking data for all 361 matches, which is not good enough to build a robust model, I decided to perform data augmentation (which I will explain in further detail in the next subsection 3.2.3) to inflate the number of samples based on geometrical logic; also because it was taken into account that the following planned features to be engineered had a dependency on geometric information, so it would be easier to engineer these new features on the already augmented data, than to modify all the new engineered features during data augmentation based on

the geometrical logics applied.

Later on, the 2D Euclidean distance between the ball and the center of the target goal was calculated; following Equation 3.1, the angle that the ball "sees" the mouth of the target goal [60] was also computed (in radian units) as:

$$\theta_{\text{Ball/Player - Target Goal}} = \arctan\left(\frac{\text{Target Goal Width} \times x_{\text{Ball/Player}}}{x^2_{\text{Ball/Player}} + y^2_{\text{Ball/Player}} - \left(\frac{\text{Target Goal Width}}{2}\right)^2}\right) \tag{3.1}$$

where $x_{\text{Ball/Player}}$ and $y_{\text{Ball/Player}}$ represent the x- and y-coordinates, respectively, of the ball or the player (depending on which of the 2 one is calculating the angle for). After, the same 2 previous features were calculated for each of the players on the pitch. Moreover, the 3D Euclidean distance between the ball and each player was calculated (3D in this case, because the ball also has a z-coordinate representing its height above the ground), as well as the 2D Euclidean distance between all pairs of players on the pitch. To finish off with the feature engineering on the tracking data, 2 binary indicators were generated. The first, creates a binary column for every single player on the pitch, flagging whether that player's team was in possession of the ball at that shot frame or not. The second, creates a binary column for each pair of players on the pitch, indicating whether these 2 players belong to the same team or not.

3.2.3 Data Augmentation

As mentioned above, half way through subsection 3.2.2, since I only had 8 053 shots from the original data I was provided with, and this is not considered a good enough quantity of data to build a proper robust model, I decided to augment this number of samples using some geometrical considerations, and on top of these new samples, I also created some synthetic data by taking into account some aspects of the nature of this sport.

Before the data augmentation process was carried out, I decided to perform a 70:30 stratified train-test split, where a random seed was set to ensure reproducibility. I split the data before performing data augmentation only on the training dataset because I wanted to avoid information leakage from the newly augmented and newly created synthetic data into the hold-out test dataset. I wanted this split to be stratified to ensure that the class distribution was preserved in both the training and the test datasets. Once this split was done, the training dataset had 5 637 shots (with 4 984 non-scored shots, and 653 goals), and the hold-out test dataset had 2 416 shots (with 2 136 non-scored shots, and 280 goals).

Now that the data is appropriately split, I augmented (only) the training dataset, by first applying a geometric transformation to the shots, and then in second place, using the original and the geometrically transformed shots, I generated synthetic shots by slightly modifying the values in certain columns of the tracking data. The geometric transformation consists of creating a copy of the original set of shots (5 637 shots, 653 goals) and simply mirroring the y-coordinates and the y-velocity

components of the ball and all players, to simulate shots from symmetrical positions. Further on, synthetic shots data for 38 726 more shots (where 4 486 are goals) were generated, while preserving the class label distribution from the original shots data. These synthetic shots were generated by sampling the specified number of shots to generate, and later varying the x- and y-coordinates of the ball within ranges of 1m or 2m, depending on the location of the ball of the sampled shot, using the Normal probability distribution function; for the x-coordinate, if the location of the ball of the sampled shot was closer than 20m away from the target goal, the synthetic x-coordinate was generated using the Normal distribution function with a standard deviation of 1m (100cm), and if the ball was 20m away from the target goal or further away, the synthetic x-coordinate was generated using the Normal distribution function with a standard deviation of 2m (200cm); for the generated synthetic y-coordinate, the Normal distribution function was always used, with a standard deviation of 1m (100cm). It was also ensured that the closest attacking player to the ball was found, and his coordinates were modified to relocate him right next to the new synthetic position of the ball. A random seed was set to ensure the reproducibility of the generation of the synthetic shots. A quick note is that I arbitrarily chose to generate 38 726 samples of synthetic shots, to obtain a final round number of 50 000 shots for the total augmented training dataset — which is quite a decent amount of data for an xG model.

Figure 3.3: Distribution of the binary target variable '**Will_Be_a_Goal**' in the augmented training dataset: 50 000 shots, 44 208 non-scored shots, and 5 792 scored shots (goals).

Afterwards, the original shots data, the geometrically transformed shots, and the synthetically generated shots data were concatenated together and randomly shuffled. Once all the remaining planned features were engineered, on both the augmented training and the hold-out test datasets, as outlined previously in sub-subsection 3.2.2, the final set of tracking data of the shot frames has dimensions (50 000, 675) for the augmented training dataset and (2 416, 675) for the hold-out test dataset. Each has the target variable '**Will_Be_a_Goal**' distributed as shown in Figures 3.3 and 3.4, respectively. As a taster for the reader, on Figures B.1, B.2, B.3, and B.4 in Appendix B, I provide visualizations

of the locations on the pitch for the 1st 100 shots and the 1st 100 goals, from both the augmented training and hold-out test datasets, with a color-coded indication on whether the shot resulted in a goal or not.

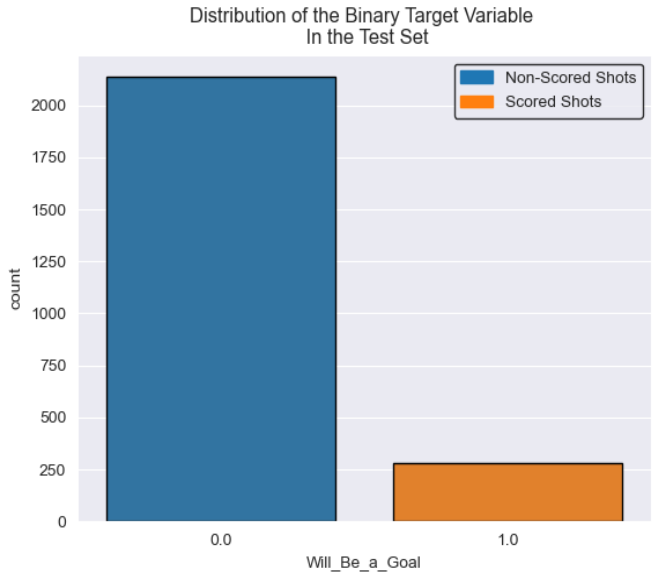


Figure 3.4: Distribution of the binary target variable '**Will_Be_a_Goal**' in the augmented training dataset: 2 416 shots, 2 136 non-scored shots, and 280 scored shots (goals).

3.2.4 Data Transformation

This subsection delves into how the data was transformed for each of the 3 models. In first place, I describe how the final set of tracking data of the shot frames is filtered and transformed, ready to be used for the traditional and simple xG model. Further, I outline how the final set of tracking data of the shot frames is transformed into a graph representation, ready to be used for both HGT models, one using simpler input graphs than the other.

xG Model

For the traditional and simple xG model, the relevant columns to use as input for this model were selected following some of the most common features used in the literature; these features being the x- and y-coordinates of the ball, the 2D Euclidean distance between the ball and the center of the mouth of the target goal, and the angle at which the ball "sees" the mouth of the target goal — and of course the '**Will_Be_a_Goal**' column as the target variable. Later, a maximum absolute (MaxAbs) scaling was performed on the features, to scale those features containing positive and negative values to a range of [-1, 1], those features containing only negative values (< 0) to a range of [-1, 0], and features containing only positive values (> 0) to a range of [0, 1].

HGT Models

For both of the HGT models, the data transformation process was the same with just a minor difference: the "simple" HGT model will have simpler graphs as input, with nodes containing fewer features than the "complex" HGT model, which will have input graphs containing more node features (see Figures C.4 and C.5 in Appendix C). From now on, I will refer to these 2 as the simple or the complex HGT models — although, I want to stress that this terminology employed to refer to them as "simple" or "complex" does not mean, under any circumstance, that the implemented architecture of these models is simpler or more complex than the other; the architecture of these, as described in section 3.4, is identical for both. As previously for the traditional, simple xG model, I applied a MaxAbs scaling to all numerical features in the final set of tracking data of the shot frames.

The next step that was taken was to convert this final set of tracking data of the shot frames into their respective graph representation, by transforming each shot frame (row of data) into a PyG [12] undirected heterogeneous graph data ('HeteroData') object. Obtaining undirected graphs was achieved by first creating a directed graph, and then using a 'ToUndirected' transform from PyG. They are heterogeneous graphs because the ball node and all other players' nodes were treated as different types of nodes: the 'Ball' node-type and the 'Player' node-type. Here is where the main difference lies between the 2 HGT models. The simple HGT model receives simple graphs as input, where the ball node has 4 node-features: the x- and y-coordinates of the ball, the 2D Euclidean distance between the ball and the center of the mouth of the target goal, and the angle at which the ball "sees" the mouth of the target goal — the exact same ones as for the traditional, simple xG model; and the node of each of the 22 players has 3 node-features: the x- and y-coordinates of the player, and the binary indicator flagging whether that player's team is currently in possession of the ball or not. The complex HGT model receives input graphs that were a bit more complex, where the ball node has 9 node-features: the x-, y-, and z-coordinates of the ball, the x-, y-, and z-velocity components of the ball, the speed of the ball, the 2D Euclidean distance between the ball and the center of the mouth of the target goal, and the angle at which the ball "sees" the mouth of the target goal; and the node of each of the 22 players has 8 node-features: the x- and y-coordinates of the player, the x- and y-velocity components of the player, the speed of the player, the 2D Euclidean distance between the player and the center of the mouth of the target goal, the angle at which the player "sees" the mouth of the target goal, and the binary indicator flagging whether that player's team is currently in possession of the ball or not.

These undirected heterogeneous graphs are fully connected, where in each graph, the edges connecting the ball node with the players' nodes were distinguished from those edges connecting a pair of player nodes. For the 22 edges connecting the ball node to the players' nodes, each of these edges contains 1 edge-feature: the 2D Euclidean distance between the ball and the player. For the 231 edges connecting each distinct pair of players on the pitch, each of these edges contains 2 edge-features: the 2D Euclidean distance between these 2 players, and the binary indicator flagging whether these 2 players belong to the same team or not.

3.3 Cross-Validation Strategy

The issue of highly imbalanced data, particularly in binary classification tasks, poses a challenge for achieving reliable and robust predictions. This thesis aims to address this problem by employing a specialized cross-validation (CV) technique (Figure 3.5) to minimize the variance and uncertainty of the validation metrics' estimates.

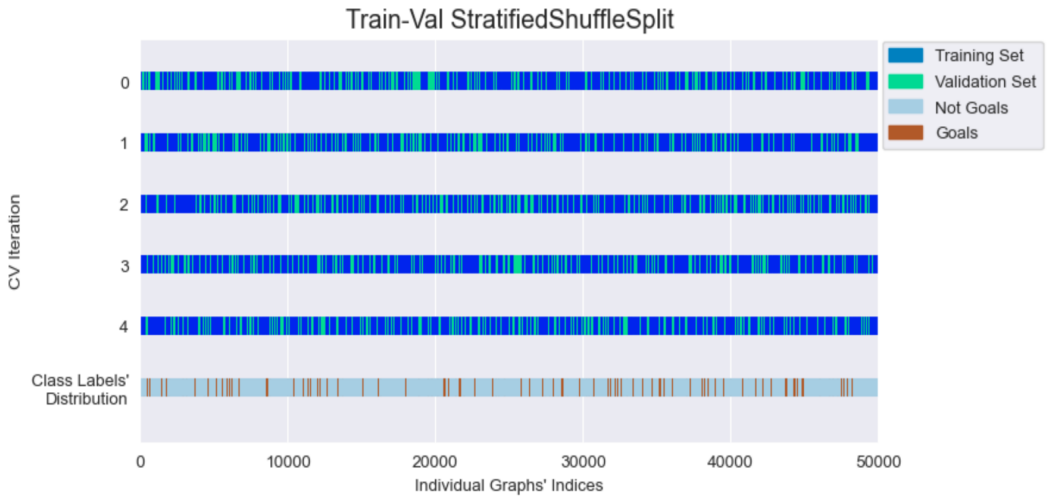


Figure 3.5: The 5-fold stratified shuffled training-validation split that was utilized. The first 5 rows depict how the (augmented) training data is split into training and validation sub-datasets in a random and stratified manner. The last row shows the class labels' distribution across the entire (augmented) training dataset.

As illustrated in Figure 3.5, a 5-fold stratified shuffled training-validation split protocol was performed during the hyper-parameter tuning and model selection part for all 3 tested models. Stratification ensures that each fold maintains the same proportion of class labels as the original (augmented) training dataset. This is vital in imbalanced datasets because it prevents scenarios where a fold might end up with very few or no samples of the minority class. If the class distribution in each fold does not represent the overall distribution, the model's evaluation might not be accurate or realistic. Shuffling the training and validation indices in each fold ensures that the data distribution within each fold was random. This randomness helps in mitigating the risk of any biases or patterns that might be present due to the order of the data, leading to a more robust evaluation. By averaging the results over multiple folds, the variance is reduced in the mean validation metric estimate. This is particularly important in imbalanced datasets where the model's performance can vary significantly depending on the specific samples it is trained on or validated against. This approach provides assistance in assessing how well do the models generalize to unseen data. Since the models were evaluated on different subsets of the data, it provided a more realistic indicator of how the model will perform on data it has not encountered before.

3.4 Model Architectures

This current section states very briefly and concisely the 2 model architectures utilized in this research. The choice of hyper-parameters to be tested and chosen for each model will be tabulated in the next section 3.5 about the models' training procedure.

The traditional and simple Expected Goals (xG) model employs a simple regularized logistic regression, from the SciKit-Learn library [162], on tabular data. An illustration of how predictions were classified by the logistic regression model is shown on Figure 2.2 in section 2.2, sub-subsection 2.2.2.

On the other hand, both HGT models (the "simple" and the "complex") make use of the exact same architecture (observe Figure 3.6), as it was previously reported in the introductory paragraph of sub-subsection 3.2.4 from the subsection 3.2.4 on 'Data Transformation'. Both HGT models were designed to handle binary classification tasks on complex graph-structured data. The models are a subclass of *PyTorch*'s 'torch.nn.Module', ensuring seamless integration with *PyTorch* [129]. The code for the HGT models' architecture can be found in Figures C.6 and C.7 in the Appendix C.

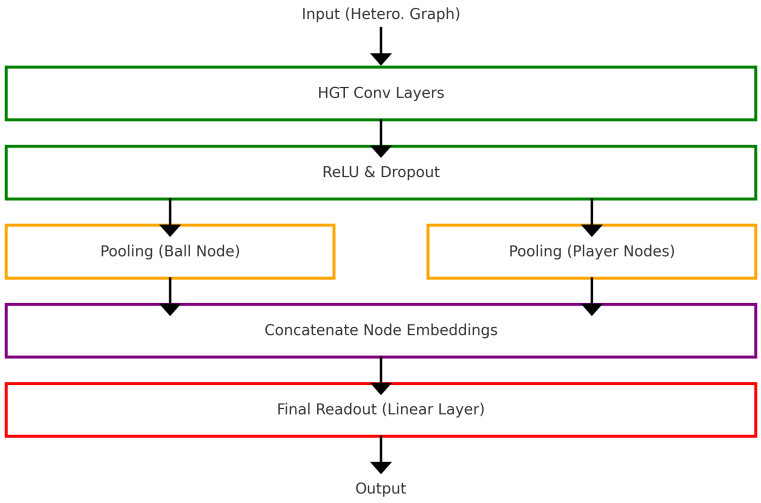


Figure 3.6: Architecture for both the "simple" and the "complex" HGT models. **Note:** "simple" and "complex" refer to the complexity of the input heterogeneous graphs, rather than the HGT models' architecture itself.

The architecture for these 2 HGT models (Figure 3.6) commences with the initialization of hyper-parameters: the number of HGT convolutional layers, the number of hidden neural network units, the number of attention heads in the HGT layer, the message aggregation strategy, the dropout rate, and the global graph pooling strategy. The core of the architecture comprises a sequence of HGT convolutional layers, utilizing the 'HGTConv' class from PyG. These layers enable the incorporation of heterogeneous graph convolution operations. The input and output channels of these layers are meticulously controlled, with the initial layer's input channels being dynamically set based on the condition of the layer index. The attention mechanism within each

HGT convolutional layer is governed by the specified number of attention heads, coupled with a selected message aggregation strategy, enhancing the model's capacity to capture complex inter-node relationships in heterogeneous graphs. To address the risk of overfitting and improve the model's generalization capability, a dropout mechanism was integrated post-activation (using the ReLU function in each convolutional layer) for regularization purposes. The architecture further employs a global graph pooling strategy, parameterized to allow mean, sum, or max global pooling operations, a common technique and step for graph-level classification tasks. This strategy is pivotal in aggregating node-features across the graph, resulting in 2 separate pooling operations for the distinct node-types: the 'Ball' and 'Player' nodes. These pooled features were subsequently concatenated, forming a comprehensive representation of both node-types. Finally, the architecture culminates in a linear transformation layer, designed to consolidate the combined node-features into a singular output value. This single output unit is particularly tailored for binary graph classification tasks. The sophistication of these HGT models lies in their ability to assimilate intricate node- and edge-features from heterogeneous graphs, efficiently transforming these into predictive insights.

3.5 Models' Training-Validation Procedure

This section outlines in detail the models' training protocols. Even though they are very similar to each other, the training protocol varies slightly in some details between the traditional xG model and the HGT models; either due to the models' nature, such as the models' hyper-parameter spaces, or due to computational cost, like the number of hyper-parameter combinations tested, since I run all the computations locally on CPU — for instance, the logistic regression for the xG model is far less computationally intensive in comparison to the HGT models. If one desires to follow along the following subsections together with the code, one can find the link to access the code in Appendix A.

3.5.1 xG Model

For the traditional xG model, a random seed (with a value of 7) was set for reproducibility purposes. The (augmented) training dataset was then split into training and validation sub-datasets following the cross-validation (CV) technique depicted in Figure 3.5 in section 3.3.

Then, the logistic regression model from SciKit-Learn was instantiated. Notably, the model was enforced to employ a 'balanced' class weighting to address the inherent high class imbalance in the dataset. This weighting strategy aims to enhance the model's sensitivity towards the minority class (scored-shots, i.e. goals). This 'balanced' class weighting was eventually enforced because when it was used as a hyper-parameter, where the choice was this 'balanced' class weight or assigning no class weights to the class labels, the resulting best set of hyper-parameters combination (producing the best mean validation metric) stated to not use any class weights, but this was misleading because when the confusion matrix was checked for this initial set of best hyper-parameters combination, it was observed that this model was not capable of capturing and classifying any samples as true positives (TP) or false positives (FP) whatsoever. For this reason, it was decided to enforce this

'balanced' class weighting, since this did eventually result in the model being capable of capturing samples as TPs and FPs too. These class weights were 0.566 and 4.32 (rounded to 3 significant figures), for the majority (non-scored shots) and the minority (scored shots, i.e. goals) classes, respectively.

Moreover, an exhaustive hyper-parameter tuning approach was taken using a grid-search. This grid-search was performed on the set of hyper-parameter options (the hyper-parameter space) tabulated in Table 3.3.

Hyper-Parameter	Options
Inverse of Regularization Strength (C)	[0.001, 0.002, 0.004, 0.005, 0.006, 0.008, 0.01, 0.02, 0.04, 0.05, 0.06, 0.08, 0.1, 0.2, 0.4, 0.5, 0.6, 0.8, 1, 2, 4, 5, 6, 8, 10, 20, 40, 50, 60, 80, 100]
Norm of the Penalty (penalty)	["l1", "l2"]
Optimization Algorithm (solver)	["sag", "saga"]
Tolerance For Stopping Criteria (tol)	[0.0001, 0.001]
Max. Number of Iterations Taken For the Solvers To Converge (max_iter)	[100, 500, 1000]

Table 3.3: Table of the logistic regression (traditional xG) model's hyper-parameter space. **Note:** "saga" supports both the "l1" and "l2" penalties — but "sag" only supports "l2".

All the 558 different possible combinations of hyper-parameters in Table 3.3 were tested to identify the optimal configuration that maximizes the model's performance, validated using the area under the Receiver Operating Characteristic (ROC) curve (the ROC AUC) metric. The use of the stratified shuffled split for CV ensures that each fold was a good representative of the whole, maintaining the percentage of samples for each class. The combination of grid search with stratified shuffled K-fold CV not only rigorously validates the model across the data but also optimizes the hyper-parameters to enhance the model's generalizability and predictive power. This training-validation protocol took approximately 10 minutes to be fully executed.

After training and validating the model, the optimal configuration of hyper-parameters resulted to be: $\{$**C** = 0.001, **penalty** = "l2", **solver** = "sag", **tol** = 0.001, **max_iter** = 100$\}$, maximizing the logistic regression's performance with a mean validation ROC AUC-score of 0.618 ± 0.003, where 0.003 is the standard deviation (std) across the 5 folds. Subsequently, using this optimal configuration of parameters, the logistic regression was (re-)trained on the entire (augmented) training dataset (i.e. using both the training and the validation sub-datasets) and evaluated on the

hold-out test dataset. This other protocol is further described in section 3.6, subsection 3.6.1.

3.5.2 HGT Models

The current subsection outlines the comprehensive framework for training, validating, and optimizing the HGT models. Once again, a random seed (with the same value of 7, as for the xG model) was set for reproducibility purposes.

First, the stage was set for hyper-parameter optimization (HPO). A diverse range of potential hyper-parameters was defined (see Table 3.4), encompassing aspects like the minority (scored shots) class label's weights, batch size, architecture specifics of the HGT model (like the number of layers, hidden units, attention heads, etc.), and training parameters (dropout rate, global pooling strategy, learning rate, weight decay, and so on). This extensive hyper-parameter space is indicative of the nuanced approach required to fine-tune GNN models, where each parameter can significantly impact model performance. A subset of these, in particular, 10 combinations were randomly selected, emphasizing a balance between exploration of the hyper-parameter space and computational feasibility, since computational power was sadly a big limitation in this thesis work, as all computations were run locally on a CPU.

Hyper-Parameter	Options
Positive Class Weights (For Scored-Shots, i.e. Goals)	['balanced' weight (= 4.32), None]
Batch Size	[512, 1024, 2048]
Number of HGT Convolutional Layers	[1, 2, 3]
Number of Hidden Layers' Units	[16, 32, 64, 128]
Number of Attention Heads	[1, 2, 4]
Message Aggregation Strategy	["sum", "mean", "max"]
Dropout Rate	[0.0, 0.25, 0.5]
Global Graph Pooling Strategy	["sum", "mean", "max"]
Learning Rate	[0.00001, 0.0001, 0.001]
Weight Decay	[0.0001, 0.001]

Table 3.4: Table of the HGT models' hyper-parameter space.

For each of these 10 combinations of hyper-parameters: the (augmented) training graphs data was randomly split in a stratified manner into training and validation sub-datasets in each fold (following the CV strategy depicted in Figure 3.5); for each fold, these training and validation graphs were respectively injected into distinct PyG data-loaders for memory-efficiency, to merge several graphs (determined by the batch size) to a mini-batch (following Figure 3.7) — the data-loader dealing with the training graphs re-shuffles the graphs at every epoch, whereas the one dealing with the validation graphs it does not re-shuffle the graphs at every epoch. For each fold, the HGT model was instantiated with a subset of the hyper-parameters and then it was trained and evaluated for 50 epochs. During each epoch, the HGT model performed the forward propagation through Message

Passing, the training loss was then computed using the binary cross-entropy (BCE) with logits loss function from *PyTorch* [129] (with the option of using enhanced weights for the minority class — the goals), gradient clipping was also employed — a technique crucial for stabilizing the training process, especially within the first few epochs, and after, it was optimized (calculating the gradients of the loss with respect to the model parameters and updating these) during backpropagation using the Adam optimizer; in each epoch, the training and validation loss, and the training and validation ROC AUC-score were calculated and saved to be able to follow the learning process of the HGT model throughout the epochs; the learning curves of this training-validation protocol, for each HGT model on their respective best fold, can be viewed in Appendix D). An early-stopping strategy with a patience of 15 epochs, by keeping track of the best validation ROC AUC-score, was utilized in each fold to prevent overfitting, and for computational- and memory-efficiency.

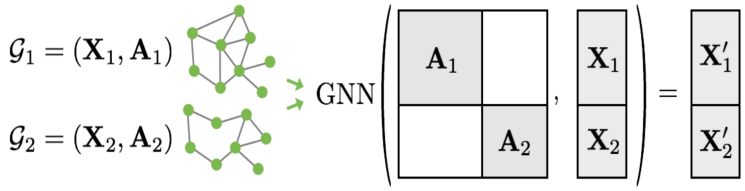


Figure 3.7: Graph-level mini-batching. Adjacency matrices are stacked diagonally (creating a giant graph that holds multiple isolated subgraphs), and the node and target features are concatenated in the node dimension [3].

The use of the ROC AUC-score as a performance indicator, especially in a binary classification task with high class imbalance, highlights the focus on both discrimination and ranking capabilities of the model. This choice is particularly pertinent in scenarios, such as the current one, where the predictive power on the minority class (goals) is as critical as the overall accuracy. Furthermore, for each hyper-parameter combination, the best validation ROC AUC-score within every fold was logged, and then the mean of these was calculated. This training-validation protocol took approximately 45 hours to be fully executed for the simple HGT model, and approximately 50 hours for the complex HGT model.

After training and validating the model on all the 10 randomly selected combinations of hyper-parameters, the optimal configuration of hyper-parameters for the simple HGT model resulted to be: {**Positive_Class_Weight** = 4.31629834, **Batch_Size** = 512, **Num_HGTConv_Layers** = 2, **Hidden_NN_Node_Units** = 128, **Num_Attention_Heads** = 4, **Message_Aggregation_Strategy** = "mean", **Dropout_Rate** = 0.25, **Global_Graph_Pooling_Strategy** = "sum", **Learning_Rate** = 0.0001, **Weight_Decay** = 0.0001}, maximizing the performance with a mean validation ROC AUC-score of 0.673 ± 0.004, where 0.004 is the std across the 5 folds; and the optimal configuration of hyper-parameters for the complex HGT model resulted to be the same as for the simple HGT model, but maximizing the performance with a mean validation ROC AUC-score of 0.718 ± 0.003 instead, where 0.003 is once again the std across the 5 folds. Subsequently, using this optimal configuration of parameters, both HGT models were (re-)trained on the entire (augmented) training dataset and evaluated on the hold-out test dataset. This other protocol is further described in the next section.

3.6 Evaluation Procedure and Metrics

In this section, I delve into the final training and testing (evaluation) procedure for the 3 models (almost identical for each) and describe the key metrics utilized to assess and evaluate the performance of these predictive classifying models. Each metric provides a unique lens through which the model's effectiveness, calibration, and reliability in predicting soccer shot outcomes are evaluated, catering to the subtleties of the shots' (and therefore of the data's) imbalanced nature and the critical importance of precise classification and probabilistic forecasting.

3.6.1 Final Train-Test Protocol

This subsection outlines the final training and testing protocol employed on the models after selecting the optimal set of hyper-parameters for that model, and thus using these on the models to make a final evaluation.

xG Model

For the xG model, the number of train-test iterations was set to equal the number of folds (= 5) as for the training-validation CV strategy exposed in section 3.3, to start with. At each train-test iteration: the regularized logistic regression was instantiated with the best-selected combination of hyper-parameters while a different random seed was set at each train-test iteration (to allow reproducibility, but ensuring the model received the data in different ways at each iteration); then, the model was fitted on the entire (augmented) training dataset, and evaluated on the hold-out test dataset, recording the test Brier-score, the test ROC AUC-score, and displaying both the Confusion Matrix and the ROC curve on the test dataset of that train-test iteration. The mean results from this final protocol are presented in chapter 4 and their interpretation and discussion in chapter 5. This final training-testing protocol took approximately 5 seconds to be fully executed.

HGT Models

Very similarly, for each of the HGT models (simple and complex), the number of train-test iterations was set to equal the number of folds (= 5) as for the training-validation CV strategy exposed in section 3.3, to start with. At each train-test iteration: the entire (augmented) training graphs data was injected into a PyG data-loader, using the best-selected batch size, where the training graphs were re-shuffled at every epoch; the HGT model was instantiated with the best-selected combination of model hyper-parameters while a different random seed was set at each train-test iteration (to allow reproducibility, but ensuring the model received the data in different ways at each iteration); following, the HGT model was trained, identically as for the previous training-validation protocol, for 50 epochs (the same number of epochs as for the training-validation protocol) using the best-selected training configuration hyper-parameters. At every epoch, the training loss and the training ROC AUC-score were computed and recorded (as with the training-validation protocol, the learning curves of each iteration for this final training-testing protocol can be found in Appendix

D); the best training ROC AUC-score within each train-test iteration was kept track of throughout the 50 epochs so that the learned weights of the model at the epoch that produced the best training ROC AUC-score within that train-test iteration were saved.

Once the HGT model was trained for 50 epochs (and still within the same train-test iteration), the hold-out test graphs data was injected into another PyG data-loader, using the best-selected batch size, where the test graphs were not re-shuffled at every train-test iteration; the best learned weights of the HGT model for that train-test iteration were then loaded, and after, the HGT model made predictions and was evaluated on the hold-out test data, computing the test Brier-score and the test ROC AUC-score for that iteration — these predicted test class labels, class labels' probabilities, and the test metrics' scores were logged for each train-test iteration. Both the Confusion Matrix and the ROC curve were also displayed for the evaluation of the test dataset for each of the train-test iterations. The mean results from this final protocol are presented in chapter 4 and their interpretation and discussion in chapter 5. This final training-testing protocol took approximately 20 hours to be fully executed for each of the simple and complex HGT models.

3.6.2 Evaluation Metrics

This subsection describes the evaluation metrics used for the final training and testing protocol outlined in the previous subsection 3.6.1. These are the ROC AUC-score and the Brier-score. To understand the ROC AUC-score, I will first give a brief description of the Confusion Matrix and of the Recall (or Sensitivity) metric.

Recall (Sensitivity)

Recall, also known as True Positive Rate (TPR) or Sensitivity, is the ratio of TP from all the data that actually and truly is positive, view Figure 3.8 and Equation 3.2:

	Predicted Positive	Predicted Negative	
Actual Positive	TP *True Positive*	FN *False Negative*	Sensitivity $\frac{TP}{(TP+FN)}$
Actual Negative	FP *False Positive*	TN *True Negative*	Specificity $\frac{TN}{(TN+FP)}$
	Precision $\frac{TP}{(TP+FP)}$	Negative Predictive Value $\frac{TN}{(TN+FN)}$	Accuracy $\frac{TP+TN}{(TP+TN+FP+FN)}$

Figure 3.8: Confusion Matrix Metrics [163].

$$\text{TPR} \equiv \text{Recall} \equiv \text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.2}$$

where TP represents the true positives (i.e. the positive class that was correctly predicted, or classified, as a positive class), and FN are the false negatives (i.e. the positive class that was incorrectly classified as a negative class). Recall can be interpreted as a measure of how well the model identifies all positive class instances, i.e. the model's ability to identify all actual goals (scored shots). It is particularly important in the context of this thesis, as it focuses on the model's ability to detect the positive class (the minority class). High Recall indicates that the model is effective in identifying the positive class (scored shots).

Area Under the ROC Curve

First, we must understand what a Receiver Operating Characteristic (ROC) curve is to then make sense of the interpretation and value that brings the Area Under this Curve (known as AUC, AUC-ROC, or ROC AUC-score). The ROC curve is a graphical representation that illustrates the performance of a binary classifier at all possible classifying threshold settings, with True Positive Rate (TPR), also known as Recall or Sensitivity, plotted on the y-axis, and with False Positive Rate (FPR) plotted on the x-axis. The formulas defining TPR and FPR are displayed in Equations 3.2 and 3.3, respectively:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{3.3}$$

where FP represents the false positives (i.e. the negative class that was incorrectly classified as a positive class), and TN are the true negatives (i.e. the negative class that was correctly classified as a negative class). Figure 3.9 shows what a moderate, or typical, ROC curve appears to be like, what the ROC curve of an ideally perfect model, and what a randomly performing model (a diagonal line) would look like:

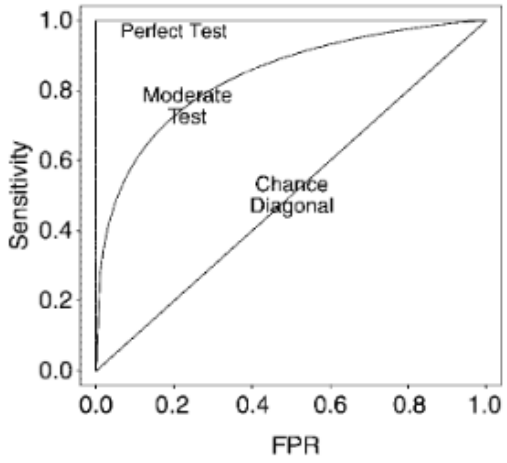


Figure 3.9: Graphical illustration of the ROC curve [164].

The ROC AUC-score is a vital and intuitive metric in evaluating binary classification models, particularly in cases with significant class imbalance. As its name very well describes already, the ROC AUC-score represents a numerical measure that quantifies the entire area underneath

the model's empirical ROC curve — it provides a single numerical value that summarizes the model's ability to discriminate between positive and negative classes (in this thesis' case, between non-scored shots and scored shots) across all possible classification probability thresholds. The ROC AUC metric also assesses the model's ranking ability — how well it can rank a random positive instance (goal, or scored shot) higher than a random negative instance (non-scored shot). These properties are crucial for building reliable models, especially for imbalanced datasets, such as shot-based goal prediction settings. ROC AUC-score provides a balanced metric that is not skewed by the uneven distribution of the classes (in this case, goals and non-scored shots). As seen in Figure 3.9, a model with an ROC AUC $= 1$ represents an ideally perfect model, a model with an ROC AUC $= 0.5$ (the diagonal line across the plot) suggests a model that performs no better than random guessing, and below this model quality threshold of ROC AUC $= 0.5$ it indicates a model performing worse than random guessing. Typically, one should obtain an ROC curve for their model that has a respective ROC AUC-score that lies between 0.5 and 1 — the closer to 1, the better.

The ROC AUC-score is a threshold-independent metric, which means that, unlike accuracy or precision, ROC AUC-score evaluates the model across all possible probability thresholds, giving a holistic measure of performance. Therefore it is also useful for comparing different models to each other, as it summarizes the model's ability to differentiate classes regardless of the threshold. In datasets with a large class imbalance (as it is the case of soccer shots' outcome), traditional metrics can be misleading, as they might be skewed towards the majority class. It is crucial for models to distinguish between the minority and majority classes effectively, thus the ROC AUC-score provides a more balanced perspective, since it evaluates the model at all possible thresholds, as mentioned. In real-world applications, where making a distinction between classes is determinant (like soccer shots), assessing the ROC AUC-score on unseen test data can inform about the reliability of the model in operational settings.

Brier-Score

When we make classification predictions, we are inherently predicting probabilities of that instance or sample to be from a specific class, thus we also need to investigate how accurate, or calibrated, these predicted probabilities are. The Brier-score is a metric utilized for evaluating the accuracy of probabilistic predictions in binary classification tasks. It is especially useful in contexts where predicting the probability of an event (such as a goal, or scored shot) is as important as the event's occurrence. The Brier-score is a mean squared error metric for probabilistic predictions. It measures the accuracy of the predicted probability (of each shot ending up being a goal) against the actual outcome (whether the shot actually resulted in a goal or not). Equation 3.4 displays its form for binary classification settings:

$$\text{Brier-Score} = \frac{1}{N}\sum_{i=1}^{N}\left(p_i - o_i\right)^2 \tag{3.4}$$

$$\text{Brier-Score}_{\text{Shots}} = \frac{1}{N_{\text{Total Shots}}} \sum_{i_{\text{Shot}}=1}^{N} \left(p_{i_{\text{Shot}}} - o_{i_{\text{Shot}}} \right)^2 \tag{3.5}$$

where N is the number of predictions, p_i represents the predicted probability of the occurrence of the class, and o_i is the actual true outcome of that class label (1 if that class event occurred, 0 if it did not). An adaptation of the interpretation of the Brier-score within the context of shot-based goal probability prediction [60] is shown in Equation 3.5. The Brier-score evaluates how close the predicted probabilities are to the actual class outcomes. The lower the score the better accuracy, with a Brier-Score $= 0$ being perfect probabilistic accuracy, and thus representing the ideal perfect model.

This metric captures two aspects of model performance: calibration (how close predicted probabilities are to actual class labels) and refinement (how well the model separates the classes). Since the Brier-score is sensitive to probability estimates, unlike some metrics that only consider classification labels, this metric evaluates the quality of the predicted probabilities, which is critical in imbalanced datasets. This makes it robust to class imbalances, as it remains a reliable measure of performance even when classes are imbalanced, as it evaluates the squared difference between predicted probabilities and actual outcomes of the classes. Hence, the Brier-score is an invaluable metric for the context of this thesis, as it provides a comprehensive measure of how well the models predict the likelihood of a shot resulting in a goal. This metric is particularly important due to the naturally imbalanced nature of goal occurrences, ensuring that predictions are evaluated not only on how accurate but also on how calibrated and sharp they are in terms of probabilistic estimates.

Chapter 4

Results

In this chapter, I present the outcomes of the analytical processes, laying out the results derived from the implementation of the 3 models. The findings are a culmination of the designed methodologies detailed in previous chapters. I systematically showcase the performance of these models, evaluating their effectiveness in the context of shot classification and the calibration of goal probabilities in soccer. These results not only offer insights into the specific domain of soccer analytics but also contribute to the broader field of GDL.

Following the trail of thought from the end of subsection 3.6.1 in the previous chapter, I will now expose the models' performance on the mean ROC AUC-score across the 5 train-test iterations, after training and testing the models on their respective best-selected combination of hyper-parameters. Figures 4.1, 4.2, and 4.3 showcase the ROC curves obtained from the predictions made at every train-test iteration on the hold-out test dataset.

The xG model produced a mean test ROC AUC-score of $\overline{\text{AUC}}_{\text{xG}} = 0.613 \pm 1.95 \times 10^{-6}$, see Figure 4.1, where 1.95×10^{-6} represents the standard error of the mean (SEM). This mean test ROC AUC value for the xG model has a margin of error of $\pm 5.41 \times 10^{-6}$; this extremely tight margin of error indicates highly consistent performance (i.e. extremely low variance) across the 5 train-test iterations, which signifies that the mean value is precise. The xG model's mean test Brier-score is $\overline{\text{Brier}}_{\text{xG}} = 0.246 \pm 2.41 \times 10^{-4}$, with a margin of error of 6.69×10^{-4}, which indicates the average squared difference between the predicted probabilities and the actual outcomes is relatively high, reflecting less calibration in predictions.

As observed in Figure 4.2, the simple HGT model achieved a mean test ROC AUC-score of $\overline{\text{AUC}}_{\text{HGT}_{\text{simple}}} = 0.665 \pm 3.58 \times 10^{-3}$, which is significantly higher than the xG model, as confirmed by Table 4.1. This mean test ROC AUC value for the simple HGT model has a margin of error of $\pm 9.93 \times 10^{-3}$, indicating a bit more variance than the xG model but still a relatively precise mean estimate. The simple HGT model significantly reduces the test Brier-score to a mean value of $\overline{\text{Brier}}_{\text{HGT}_{\text{simple}}} = 0.162 \pm 1.72 \times 10^{-3}$, as confirmed by Table 4.1, with a margin of error of 4.77×10^{-3}, suggesting that the simple HGT model is better calibrated than the xG model in its

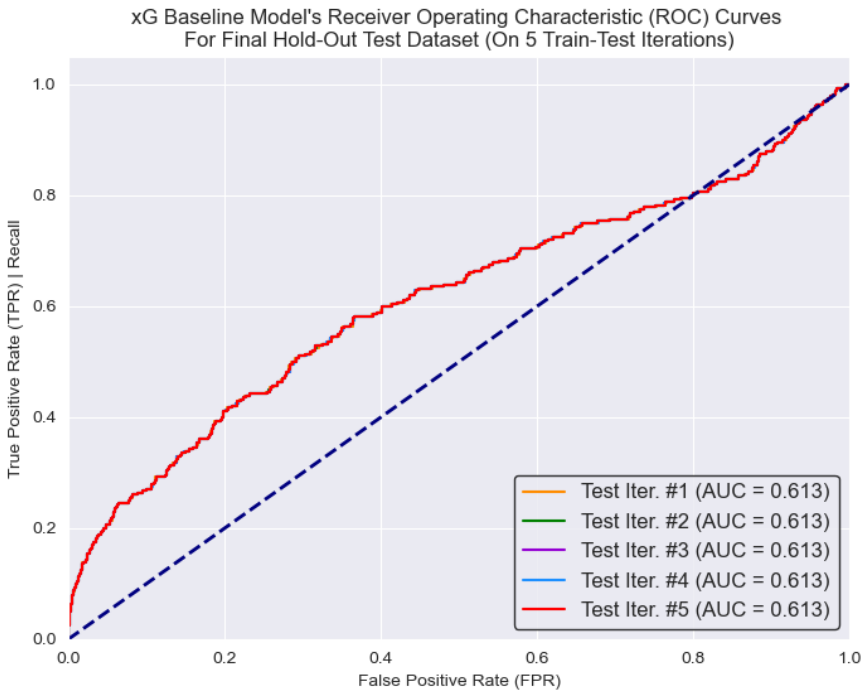


Figure 4.1: ROC curves for the xG model, with their respective ROC AUC-score for each of the 5 train-test iterations on the hold-out test dataset.

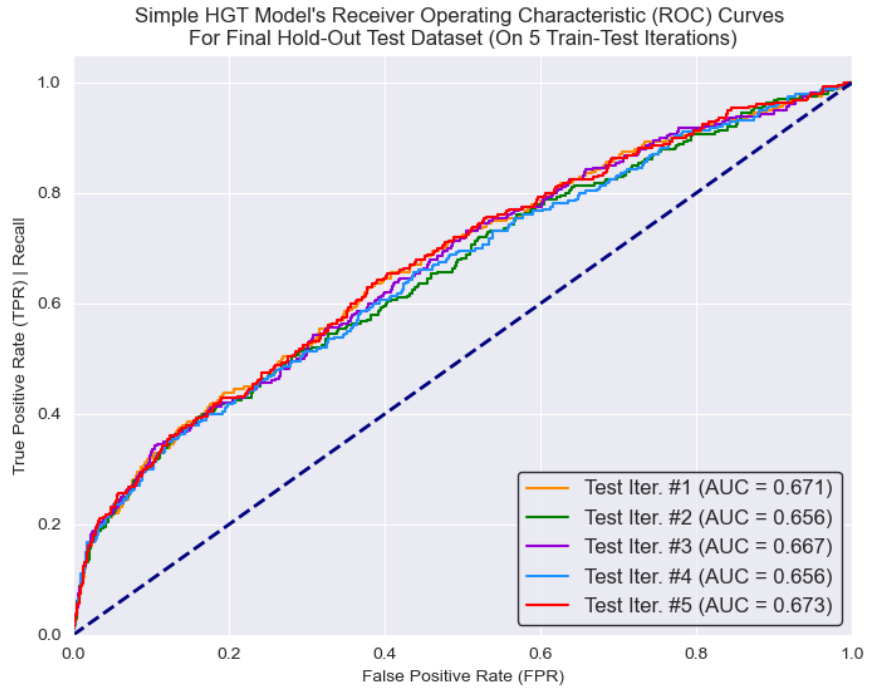


Figure 4.2: ROC curves for the simple HGT model, with their respective ROC AUC-score for each of the 5 train-test iterations on the hold-out test dataset.

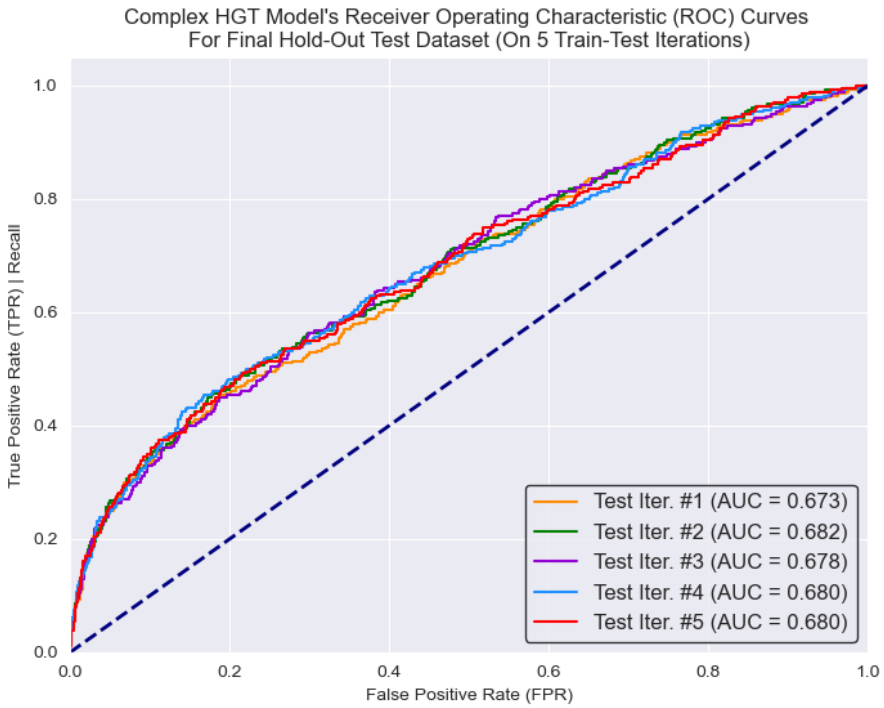


Figure 4.3: ROC curves for the complex HGT model, with their respective ROC AUC-score for each of the 5 train-test iterations on the hold-out test dataset.

probability estimates.

It is appreciated in Figure 4.3, that the complex HGT model further improved the mean test ROC AUC-score, obtaining $\overline{\text{AUC}}_{\text{HGT}_{\text{complex}}} = 0.679 \pm 1.50 \times 10^{-3}$. This mean test ROC AUC value for the complex HGT model has a margin of error of $\pm 4.15 \times 10^{-3}$ still suggesting a relatively precise mean estimate; both HGT models also have a tight margin of error, still indicating consistent performance across the 5 train-test iterations. The consistent ROC AUC-scores across the test iterations for each model suggest stable performance when evaluated on the hold-out test dataset. The complex HGT model achieves the lowest mean test Brier-score of $\overline{\text{Brier}}_{\text{HGT}_{\text{complex}}} = 0.155 \pm 6.41 \times 10^{-3}$, indicating the best probabilistic calibration among the 3 models. However, with a slightly larger margin of error of 1.78×10^{-2}, it suggests that while it generally performs better than the other 2 models, there exists more variability in its performance on the test Brier-score across the different train-test iterations.

As perceived in Figures 4.1, 4.2, and 4.3, the ROC curves of all 3 models display a consistent pattern of improvement from the xG baseline to the simple HGT and then to the complex HGT model.

To statistically compare the results of these 3 models to each other, I carried out an independent 2-sample t-test on each pair of models for each evaluation metric, with a threshold significance level

(α) of 0.05 ($\equiv 5\%$). The results of these statistical tests are tabulated in the Tables 4.1, 4.2, and 4.3.

Test ROC AUC-Score		**Test Brier-Score**	
t-Statistic	14.3	**t-Statistic**	48.4
p-Value	0.00	**p-Value**	0.00
Power of the t-Test	1.00	**Power of the t-Test**	1.00

Table 4.1: t-Tests' results: xG model vs. simple HGT model.

It is appreciated in both Tables 4.1 and 4.2, that the independent two-sample t-tests for both metrics provided clear evidence that both HGT models significantly outperformed the traditional xG model on both evaluation metrics since the p-values for both metrics are lower than the significance level α when both HGT models are separately and directly compared to the xG model.

Test ROC AUC-Score		**Test Brier-Score**	
t-Statistic	43.6	**t-Statistic**	14.3
p-Value	0.00	**p-Value**	0.00
Power of the t-Test	1.00	**Power of the t-Test**	1.00

Table 4.2: t-Tests' results: xG model vs. complex HGT model.

Table 4.3 shows that the complex HGT model's improvement on the simple HGT model with respect to the mean test ROC AUC-score is statistically significant with a t-statistic of 3.61 and a p-value of 0.014 (which is below the threshold of 0.05). The power of this test is 0.883. However, the complex HGT model did not show a statistically significant improvement over the simple HGT model with respect to the mean test Brier-score, based on the p-value of 0.319 (being larger than the threshold of 0.05). The power of this test is 0.167.

Test ROC AUC-Score		**Test Brier-Score**	
t-Statistic	3.61	**t-Statistic**	1.12
p-Value	0.014	**p-Value**	0.319
Power of the t-Test	0.883	**Power of the t-Test**	0.167

Table 4.3: t-Tests' results: simple HGT model vs. complex HGT model.

The complex HGT model outperforms both the simple HGT and xG models in terms of the ROC AUC-Score. All models are significantly different from each other in terms of the ROC AUC-Score with the complex HGT model showing the best mean performance. In terms of the Brier-score, the complex HGT model also has the lowest mean score, indicating the best performance in probabilistic predictions, but the difference between the simple and complex HGT models is not statistically significant.

Chapter 5

Discussion

The subsequent discussion, analysis, and interpretation in this chapter of the findings presented in the previous chapter 4 aim to draw meaningful conclusions, identify trends, and suggest potential areas for future research. Initially, section 5.1 delves into the nuances of the results. This exploration helps in explaining the practical significance and the theoretical contributions of our research. Subsequently, section 5.3 candidly addresses the constraints and challenges encountered during this study, acknowledging the aspects where my methodology or data might have influenced the results. Finally, this chapter culminates with section 5.4 by proposing potential improvements for refining the approach carried out in this research and suggesting avenues for future work. This last section aims to inspire ongoing inquiry and development in the fields of soccer analytics and GDL techniques, building upon the groundwork laid by this thesis.

5.1 Interpretation of the Results

Based on the findings from the previous chapter 4, the mean test ROC AUC-score suggests that the model has a relatively low discriminative ability to distinguish between the positive class (goals) and the negative class (non-goals). The increase in ROC AUC-score for the simple HGT model reflects a better discriminative ability over the baseline xG model. The best-performing model on the test ROC AUC-score was the complex HGT model, suggesting that possibly, the additional complexity within the heterogeneous graphs used as input to the HGT architecture contributed positively to the model's performance on its discriminative ability to distinguish between non-scored shots and scored shots (goals). All models show curves above the diagonal line of no-discrimination, which indicates that they have predictive power. Each step up in model complexity seems to provide a boost in ROC AUC performance, as shown by the progressive lift of the ROC curves, from model to model, towards the top-left corner of the plots in Figures 4.1, 4.2, and 4.3, which is indicative of better Recall (also known as Sensitivity or TPR) and Specificity (equivalent to 1 - FPR). The improvement from the xG to the HGT models could be attributed to the sophisticated architecture of HGTs, which likely enables them to capture complex relationships within the data that are pertinent to shot classification tasks.

A similar pattern follows for the models' performance in calibrating the predicted probability estimates produced by them. The xG model produced a mean test Brier-score larger than the simple HGT model, which in return also produced a larger Brier-score than the complex HGT model. The results show a clear trend where the HGT models outperform the baseline xG model, not only in predictive performance (ROC AUC-score), but also in probabilistic calibration (Brier-score). The complex HGT model demonstrates the best overall performance, balancing discriminative ability and probabilistic calibration, albeit with some variability as indicated by its margin of error for the Brier-score. It is worth noting that despite the improvements, even the complex HGT model's ROC AUC-score indicates there is room for further refinement, as scores closer to 1.0 would indicate a stronger predictive capability; it would also be interesting to see how these further refinements could allow (or not) to achieve a statistically significant difference in Brier-score between the HGT models themselves.

Overall, these findings support the hypothesis of this thesis that incorporating GNNs, specifically HGT models, can enhance shot classification (goal or no goal) and goal probability predictions in soccer analytics, in particular, my primary hypothesis (H_1) was that HGTs would perform better than the traditional xG model on these tasks, as formulated in chapter 1, section 1.3.

In Tables 4.1 and 4.2, the significant t-statistics and the p-values of 0.00 for both the mean test ROC AUC-Score and the mean test Brier-score strongly suggest that the differences in the means are not due to random chance. The power of these tests being 1.00 reinforces the reliability of these results, suggesting certainty on correctly rejecting the null hypothesis H_0, aligning with the primary hypothesis H_1, and strongly supporting the second sub-hypothesis H_{1b} (view section 1.3 in chapter 1).

Table 4.3 indicates that the comparison between the simple and complex HGT models yielded mixed results; as briefly stated in the findings in the previous chapter 4, Table 4.3 suggests that the complex HGT model's improvement on the simple HGT model in terms of the test ROC AUC-score is statistically significant. The fact that the power of this test is 0.883, indicates that there is a high probability of correctly rejecting the null hypothesis H_0 for this evaluation metric between these 2 models. However, the complex HGT model did not show a statistically significant improvement over the simple HGT model in terms of the test Brier-score. The power of this test being 0.167, indicates that there is a low probability of correctly rejecting the null hypothesis H_0 for this evaluation metric between these 2 models, thus, suggesting that the observed difference in this metric could be due to random chance. Therefore, these t-tests' results partially support the overall primary hypotheses in H_1 since, as mentioned before, the second sub-hypothesis H_{1b} is strongly supported, and the mean test ROC AUC-score between both HGT models supports the first sub-hypothesis H_{1a}, but the mean test Brier-score between these HGT models does not support H_{1a}. This means that further research is required to determine whether the difference in this latter metric between the simple and the complex HGT models is indeed due to random chance or if it could be statistically significant. These results signal that while the complex HGT model has a marginally better ability to discriminate between scored shots (goals) and non-scored shots than

the simple HGT model, it currently does not significantly improve the calibration of the predicted probabilities.

Having in mind the reported difference in performance between the xG model and the HGT models, one must also take into account the training time of the models and weigh whether it is worth training the HGT models for almost over 70 hours (due to their computational complexity) or, on the other hand, simply training the traditional xG model for only around 10 minutes. On the other hand, perhaps by using more precise shots data from state-of-the-art event data provided by external specialized 3rd parties, rather than estimating the shot frames from possessions-based data, the difference in performance might be larger than it currently seems, and may feel worth employing the HGT models.

5.2 Interpretability of the Best-Performing Model

Another important aspect is to visualize and understand the interpretability of the HGT model. To do so, the test class labels' predicted probabilities were analyzed from the best-performing (the complex HGT) and the worst-performing (xG) models. The shot frame cases where there was the largest discrepancy between these 2 models, where the best-performing model is favored, were extracted first; these largest discrepancy cases occurred when the true class label of the shot frame was a scored shot (a goal), and the HGT model also predicted a goal, with its predicted probability as large and as far as possible from the xG model's predicted probability. Figure 5.1 shows one of the most extreme and visually better examples, it displays a match situation where the predicted goal probability given by the complex HGT model was of 0.911 and the xG model assigned it 0.534.

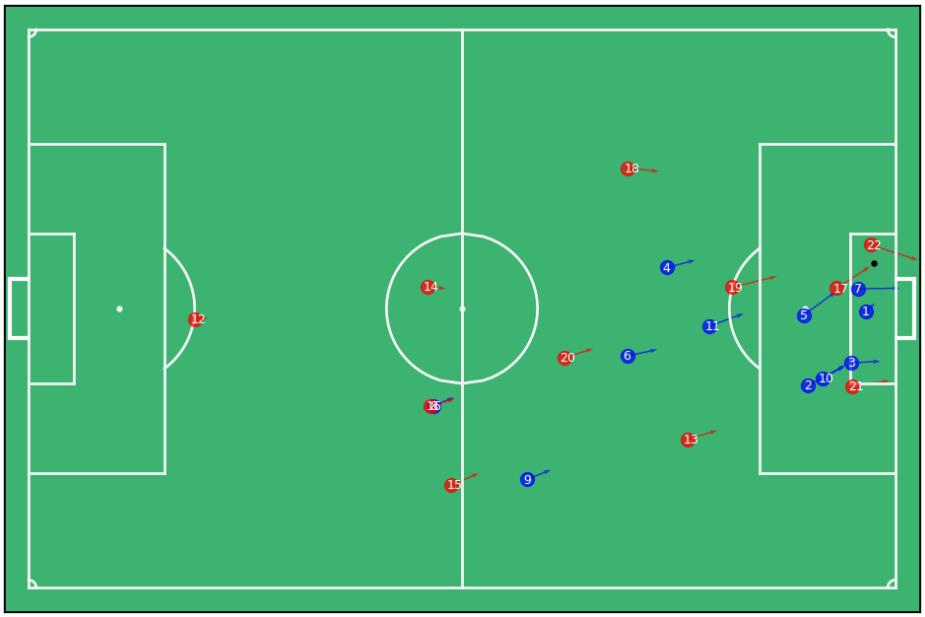


Figure 5.1: Frame showing the away team's player #22 shooting the ball towards the target goal, where the home team's goalkeeper #1 is out of position, leaving an empty goal behind.

This large difference of almost 38% in the predicted probabilities is probably due to the intricate and sophisticated architecture of the HGT convolutional layers, capable of making sense of all the spatial and physical information injected into the HGT model through the input heterogeneous graphs, where all the coordinates, velocities and distances of all the players and the ball are encoded, whereas the traditional xG model just takes into account a very small quantity of features related only to the ball. In this particular example illustrated in Figure 5.1, we can see how the attacking (red) player #22 shoots the ball towards an empty goal, and the defending goalkeeper is out of position leaving the goal empty behind him/her. In this situation, it makes reasonable sense that the HGT model's probability of scoring is so high and so far apart from the xG model since it is capable of capturing this semantic information of the sport as geometrical and physical information of all the players on the pitch are taken into account. On the other hand, since the xG model used in this thesis only considers the x- and y-coordinates of the ball, its distance to the center of the target goal, and the angle that "sees" the target goal, it is less sure that the shot could end up resulting in a goal since it has no information about other players on the pitch, and cannot capture the information that the HGT model does. It is not the same to make this same shot with the goalkeeper on the goal line, covering the first post, or with a defender covering some space between the ball and the empty goal, compared to the currently displayed situation.

5.3 Limitations

Despite being capable of extracting an estimated frame for the instant that the shot-attempts occurred in, using the possessions-based data that I was provided with, I believe that if state-of-the-art event data was utilized instead to extract the exact frame in the tracking data in which the ball is hit by the attacking player performing the shot, and using the same methodologies employed in this investigation, the results of this thesis could have potentially been of greater quality.

Despite their capabilities, GNNs face several challenges. One major challenge is scalability, as the complexity of GNNs can grow significantly with the size of the graph. Another issue is over-smoothing, where repeated application of GNN layers leads to node representations becoming indistinguishable. Moreover, most GNNs assume that the graph structure is given and accurate, which might not always be the case in real-world scenarios [144; 165], as it is the case in this research.

Having no access to GPUs and running everything locally on a CPU probably caused major limitations in this research, restricting many of the computational aspects to be explored and tested in this research, such as the number of hyper-parameter combinations to be explored in the HGT models, the number of folds and epochs in the training-validation and training-testing protocols, and so on.

The training-validation and training-testing protocols used in this thesis probably also limited the minimization of the estimate's variance of the mean test evaluation metrics' scores produced by the models. Perhaps if a more sophisticated CV approach was taken, a Nested Stratified Shuffled

K-Fold CV for example, the variance of the estimates could have been lower.

5.4 Possible Improvements and Future Work

- Utilizing state-of-the-art event data, to be able to pinpoint the exact instant and frame of the shots performed by the attacking players with much greater accuracy than from possessions-based data; if access to state-of-the-art event data is not possible, then possibly coming up with a smarter or more sophisticated way of extracting the frame a shot-attempt occurs from this possessions data could be investigated.
- When creating the graph representation of the data using PyG, the 'Player' node-type could be further divided into 'Attacking Player' and 'Defensive Player' node-types for each shot frame.
- Employing a better and more robust CV strategy, such as a Nested Stratified Shuffled K-Fold CV.
- Using a greater number of CV folds (10-15, instead of 5).
- Exploring more sophisticated HGT architectures, including more components within the current HGT architecture used.
- Using GPUs and parallel computation to accelerate the computational processes and attenuate the memory limitations.
- 10 hyper-parameter combinations were randomly selected from the potential hyper-parameter space for the HGT models due to computational limitations and time-efficiency reasons, but ideally, a much greater number of configurations should be tested.
- 50 epochs were chosen due to computational limitations and time-efficiency reasons, but ideally, a much greater number of epochs should also be explored.
- Other optimizers could be tested too.
- A different number of epochs for the early-stopping strategy could be explored.

The idea behind the investigation carried out in this research using HGTs, or even other GNNs, could potentially be extended to deal with dynamical possessions instead of focusing on static shot frames, creating an EPV framework where GNNs are capable of extracting the spatio-temporal (dynamic) intricacies of the sport to estimate at each frame the probability of scoring a goal for the team currently in possession, or of conceding a goal in case of a turnover at that frame.

The hope is that one day it will be possible to employ this model (or a variant of it) not only by analysts for match analyses between gameweeks or for player scouting, but also to exploit it live during actual matches, allowing coaches and their staff to operate it to analyze the match as it takes place and make real-time decisions, or to further enhance xG visualizations during live broadcasts.

Chapter 6

Conclusion

This thesis set out to explore the application of Heterogeneous Graph Transformer Networks (HGTs) in soccer analytics, focusing on shot classification and goal probability estimations. This research was driven by the objective to evaluate and discover whether HGTs could outperform a traditional Expected Goals (xG) model in these tasks, as delineated by the research questions and hypotheses.

The findings from the implemented models shed light into potential significant advancements in the domain of soccer analytics. The complex HGT model, in particular, demonstrated a superior performance over both the simple HGT and the traditional xG models in terms of the ROC AUC-score and the Brier-score. This suggests that the sophisticated architecture of HGTs, capable of capturing complex relationships within intricate graph-structured data, provides a notable improvement in predicting shot outcomes in soccer. The results not only statistically validate through t-tests the primary hypothesis H_1 that HGTs would outperform the traditional xG model but also suggest avenues for further refinement and exploration in the field of geometric deep learning applied in soccer analytics.

However, these advancements are not without limitations existing. The reliance on possessions-based data, rather than state-of-the-art event data, possibly constrained the precision of the shot frame extraction. Furthermore, the challenges inherent to Graph Neural Networks (GNNs), such as scalability and over-smoothing, and the limitation of conducting computations on a CPU, rather than a GPU, may have restricted the full potential of this investigation.

Future research directions should focus on utilizing more precise event data, refining the graph representations, and exploring more robust cross-validation strategies and sophisticated HGT architectures. Additionally, the extension of this research to dynamic possessions, creating an Expected Possession Value (EPV) framework with GNNs, presents an exciting frontier for soccer analytics.

To the best of my knowledge, this thesis is a pioneering effort to make an opening in the literature landscape of soccer analytics by applying GNNs, specifically HGTs, to soccer shot classification and goal probability calibration. It opens the door to a multitude of possibilities where these advanced techniques could be employed not just for post-match analysis and player scouting but potentially in real-time during matches, improving decision-making processes and enhancing live broadcast visualizations.

In conclusion, while this research confirms the potential of HGTs in enhancing soccer analytics, it also underscores the need for continued exploration and innovation in this rapidly evolving field. The journey from data to decision-making in sports analytics is complex and multifaceted, and this thesis represents a step forward in that journey, offering a glimpse into a future where data science and sports intersect more profoundly.

Appendix A

Thesis Code

The code employed for this thesis can be found in the following GitHub repository.

Appendix B

Static and Dynamic Pitch Visualizations

B.1 Velocities In Tracking Data

For static and dynamical visualizations of the velocities of the players and the ball via the use of tracking data, please refer to the link to the GitHub repository for this thesis, displayed in Appendix A.

B.2 Some Shots From the Final Augmented Training and Hold-Out Test Datasets

Figures B.1, B.2, B.3, and B.4, provide visualizations of the locations on the pitch for the 1st 100 shots and the 1st 100 goals, from both the augmented training and hold-out test datasets, with a color-coded indication on whether the shot resulted in a goal or not.

B.3 Pitch Control

For static and dynamical Pitch Control visualizations, please refer to the link to the GitHub repository for this thesis, displayed in Appendix A.

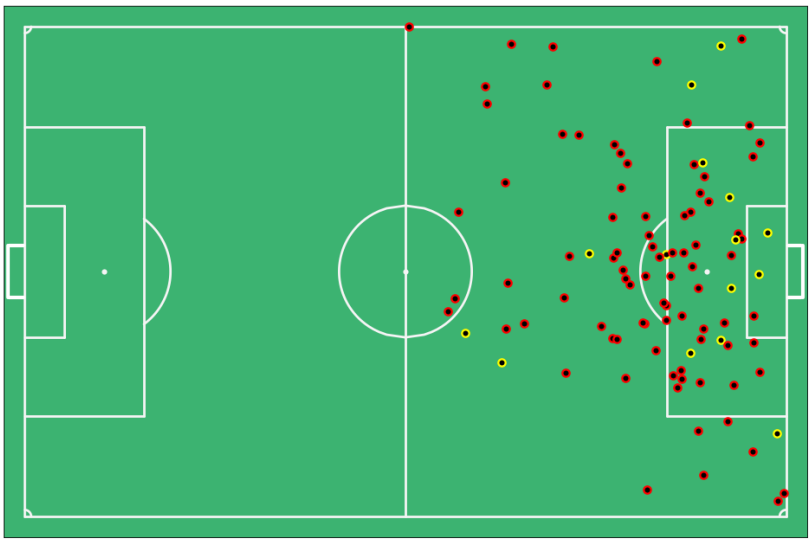


Figure B.1: Locations on the pitch of the 1st 100 shots in the augmented training dataset. Balls surrounded by a red ring represent shots that were not scored; balls surrounded by a golden ring represent shots resulting in a goal.

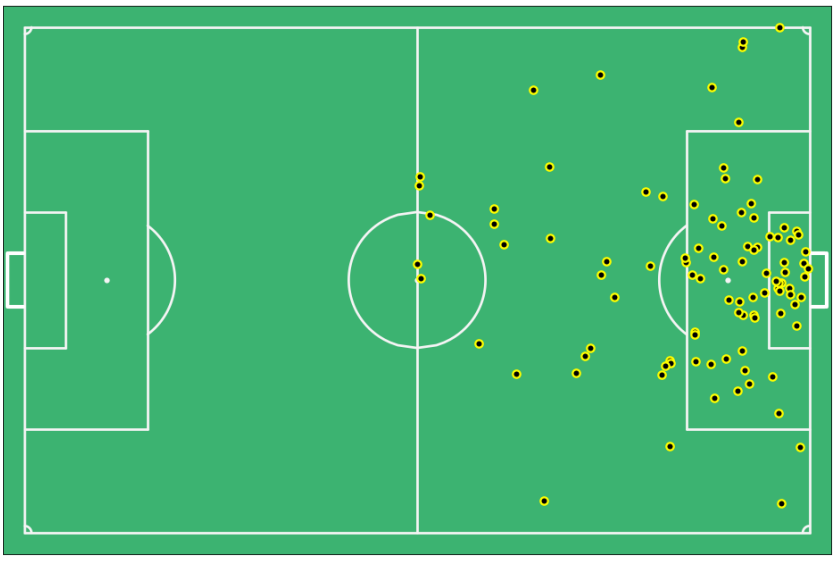


Figure B.2: Locations on the pitch of the 1st 100 goals in the augmented training dataset.

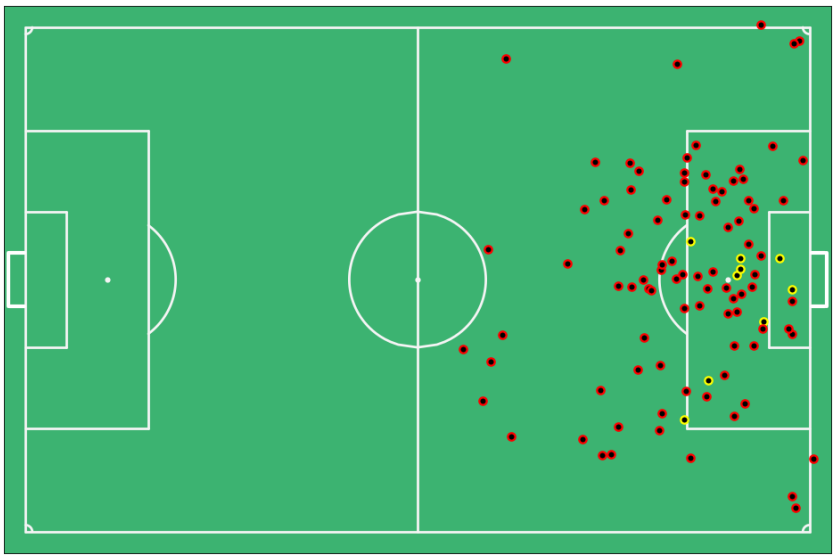


Figure B.3: Locations on the pitch of the 1st 100 shots in the hold-out test dataset. Balls surrounded by a red ring represent shots that were not scored; balls surrounded by a golden ring represent shots resulting in a goal.

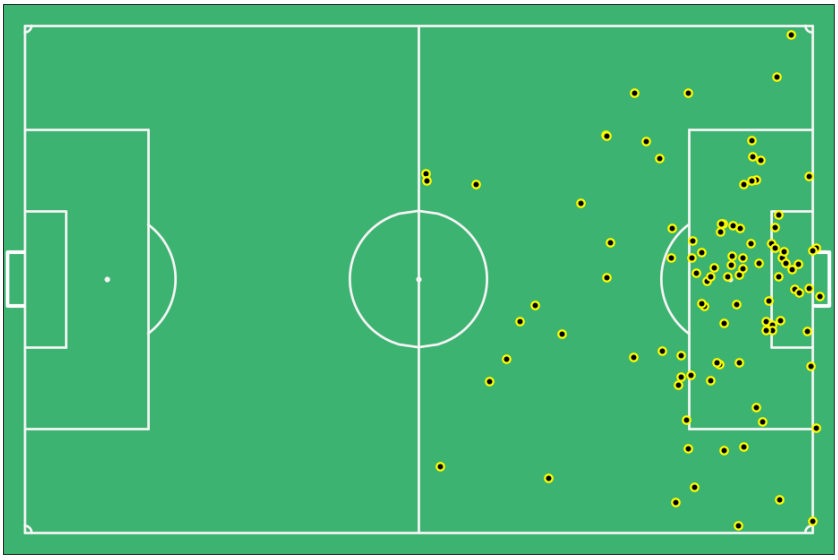


Figure B.4: Locations on the pitch of the 1st 100 goals in the hold-out test dataset.

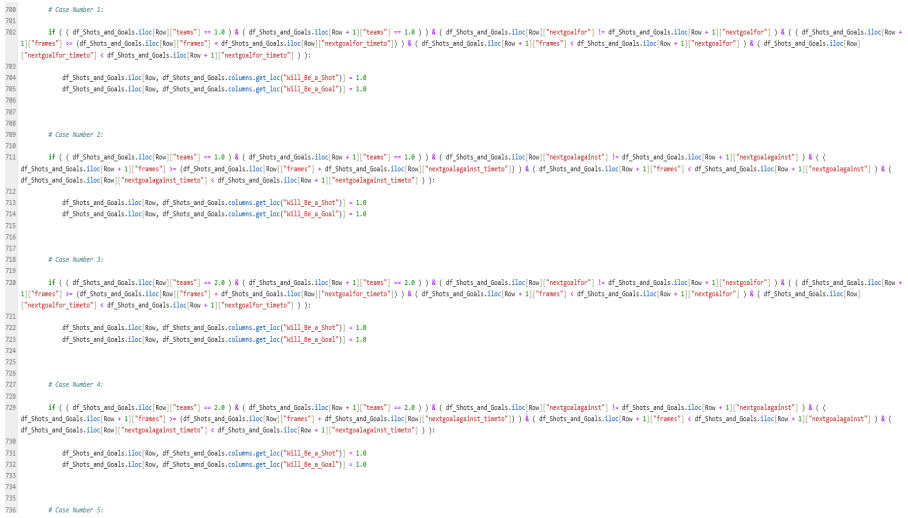
Appendix C

Code Screenshots

Note: Unfortunately, due to the format of the margins of the pages of this thesis, the code within the screenshots is not very readable without zooming in. To read the code within the screenshots, please use a zoom of 225-250%.

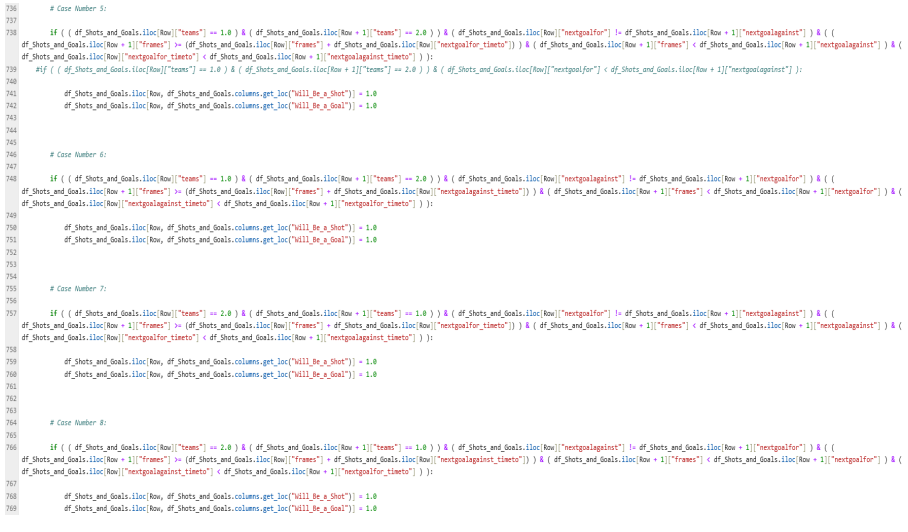
```
def Adding_Shots_and_Goals_To_Possessions_and_Targets_Dataset( df ):
    """
    Function That Adds 2 New Columns To the Possessions & Targets' Dataset: `Will_Be_a_Shot` & `Will_Be_a_Goal`, Representing Whether a Shot or a Goal Occurred/Will Occur Within 10s of the Possession.
        Optionally, It Also Drops/Removes All the Next-Goal-Related Columns; This Is Done To Avoid Having Any Missing (`NaN`) Values Present In the Dataset.

    Input: df = DataFrame We Want To Inspect
    Input: remove_next_goal_related_columns = Boolean Value Stating Whether All Next-Goal-Related Columns Should Be Removed Or Not

    Output: df_Shots_and_Goals = Possessions & Targets DataFrame With Shots & Goals Indications Included
    """

    df_Shots_and_Goals = df.copy()


    #  Create (Target) Variable `Will_Be_a_Shot` Extracting Whether There Will Be a Shot/Chance To Score Within the Next 10s (= 250 Frames), This Info./Threshold Being Intrinsically Encapsulated Within the `nextxgfor/against` Columns

    df_Shots_and_Goals["Will_Be_a_Shot"] = 0.0

    df_Shots_and_Goals.loc[ ( df_Shots_and_Goals["nextxgfor"] > 0 ) | ( df_Shots_and_Goals["nextxgagainst"] > 0 ), "Will_Be_a_Shot" ] = 1.0




    # Create (Target) Variable `Will_Be_a_Goal` Extracting Whether There Will Be a Goal Within the Next 10s (= 250 Frames)

    Frames_Window_10s = 250


    df_Shots_and_Goals["Will_Be_a_Goal"] = 0.0

    df_Shots_and_Goals.loc[ ( df_Shots_and_Goals["nextgoalfor_timeto"] <= Frames_Window_10s ) | ( df_Shots_and_Goals["nextgoalagainst_timeto"] <= Frames_Window_10s ), "Will_Be_a_Goal" ] = 1.0




    # Check For "Unrecorded" Goals That Did Very Clearly Occur, But the Possessions 10s Before the Goal Leading Up TO It Has Not Been Recorded/Logged-In Into the Dataset

    for Row in range(len(df_Shots_and_Goals.index)):

        if Row == ( len(df_Shots_and_Goals.index) - 1):

            continue


        # Case Number 1:
```

Figure C.1: Initial base conditions to create the '**Will_Be_a_Shot**' and '**Will_Be_a_Goal**' columns in the possessions data to spot unrecorded goals due to logging mistakes.



```
        # Case Number 1:

        if ( ( df_Shots_and_Goals.iloc[Row]["teams"] == 1.0 ) & ( df_Shots_and_Goals.iloc[Row + 1]["teams"] == 1.0 ) ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalfor"] != df_Shots_and_Goals.iloc[Row + 1]["nextgoalfor"] ) & ( ( df_Shots_and_Goals.iloc[Row + 1]["frames"] >= (df_Shots_and_Goals.iloc[Row]["frames"] + df_Shots_and_Goals.iloc[Row]["nextgoalfor_timeto"]) ) & ( df_Shots_and_Goals.iloc[Row + 1]["frames"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalfor"] ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalfor_timeto"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalfor_timeto"] ) ):

            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Shot")] = 1.0
            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Goal")] = 1.0



        # Case Number 2:

        if ( ( df_Shots_and_Goals.iloc[Row]["teams"] == 1.0 ) & ( df_Shots_and_Goals.iloc[Row + 1]["teams"] == 1.0 ) ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalagainst"] != df_Shots_and_Goals.iloc[Row + 1]["nextgoalagainst"] ) & ( ( df_Shots_and_Goals.iloc[Row + 1]["frames"] >= (df_Shots_and_Goals.iloc[Row]["frames"] + df_Shots_and_Goals.iloc[Row]["nextgoalagainst_timeto"]) ) & ( df_Shots_and_Goals.iloc[Row + 1]["frames"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalagainst"] ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalagainst_timeto"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalagainst_timeto"] ) ):

            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Shot")] = 1.0
            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Goal")] = 1.0



        # Case Number 3:

        if ( ( df_Shots_and_Goals.iloc[Row]["teams"] == 2.0 ) & ( df_Shots_and_Goals.iloc[Row + 1]["teams"] == 2.0 ) ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalfor"] != df_Shots_and_Goals.iloc[Row + 1]["nextgoalfor"] ) & ( ( df_Shots_and_Goals.iloc[Row + 1]["frames"] >= (df_Shots_and_Goals.iloc[Row]["frames"] + df_Shots_and_Goals.iloc[Row]["nextgoalfor_timeto"]) ) & ( df_Shots_and_Goals.iloc[Row + 1]["frames"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalfor"] ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalfor_timeto"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalfor_timeto"] ) ):

            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Shot")] = 1.0
            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Goal")] = 1.0



        # Case Number 4:

        if ( ( df_Shots_and_Goals.iloc[Row]["teams"] == 2.0 ) & ( df_Shots_and_Goals.iloc[Row + 1]["teams"] == 2.0 ) ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalagainst"] != df_Shots_and_Goals.iloc[Row + 1]["nextgoalagainst"] ) & ( ( df_Shots_and_Goals.iloc[Row + 1]["frames"] >= (df_Shots_and_Goals.iloc[Row]["frames"] + df_Shots_and_Goals.iloc[Row]["nextgoalagainst_timeto"]) ) & ( df_Shots_and_Goals.iloc[Row + 1]["frames"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalagainst"] ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalagainst_timeto"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalagainst_timeto"] ) ):

            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Shot")] = 1.0
            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Goal")] = 1.0



        # Case Number 5:
```

Figure C.2: Conditional checks for scenarios 1 to 4 in the possessions data to spot unrecorded goals due to logging mistakes.



```
        # Case Number 5:

        if ( ( df_Shots_and_Goals.iloc[Row]["teams"] == 1.0 ) & ( df_Shots_and_Goals.iloc[Row + 1]["teams"] == 2.0 ) ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalfor"] != df_Shots_and_Goals.iloc[Row + 1]["nextgoalagainst"] ) & ( ( df_Shots_and_Goals.iloc[Row + 1]["frames"] >= (df_Shots_and_Goals.iloc[Row]["frames"] + df_Shots_and_Goals.iloc[Row]["nextgoalfor_timeto"]) ) & ( df_Shots_and_Goals.iloc[Row + 1]["frames"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalagainst"] ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalfor_timeto"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalagainst_timeto"] ) ):
            #if ( ( df_Shots_and_Goals.iloc[Row]["teams"] == 1.0 ) & ( df_Shots_and_Goals.iloc[Row + 1]["teams"] == 2.0 ) ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalfor"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalagainst"] ):

            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Shot")] = 1.0
            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Goal")] = 1.0



        # Case Number 6:

        if ( ( df_Shots_and_Goals.iloc[Row]["teams"] == 1.0 ) & ( df_Shots_and_Goals.iloc[Row + 1]["teams"] == 2.0 ) ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalagainst"] != df_Shots_and_Goals.iloc[Row + 1]["nextgoalfor"] ) & ( ( df_Shots_and_Goals.iloc[Row + 1]["frames"] >= (df_Shots_and_Goals.iloc[Row]["frames"] + df_Shots_and_Goals.iloc[Row]["nextgoalagainst_timeto"]) ) & ( df_Shots_and_Goals.iloc[Row + 1]["frames"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalfor"] ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalagainst_timeto"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalfor_timeto"] ) ):

            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Shot")] = 1.0
            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Goal")] = 1.0



        # Case Number 7:

        if ( ( df_Shots_and_Goals.iloc[Row]["teams"] == 2.0 ) & ( df_Shots_and_Goals.iloc[Row + 1]["teams"] == 1.0 ) ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalfor"] != df_Shots_and_Goals.iloc[Row + 1]["nextgoalagainst"] ) & ( ( df_Shots_and_Goals.iloc[Row + 1]["frames"] >= (df_Shots_and_Goals.iloc[Row]["frames"] + df_Shots_and_Goals.iloc[Row]["nextgoalfor_timeto"]) ) & ( df_Shots_and_Goals.iloc[Row + 1]["frames"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalagainst"] ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalfor_timeto"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalagainst_timeto"] ) ):

            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Shot")] = 1.0
            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Goal")] = 1.0



        # Case Number 8:

        if ( ( df_Shots_and_Goals.iloc[Row]["teams"] == 2.0 ) & ( df_Shots_and_Goals.iloc[Row + 1]["teams"] == 1.0 ) ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalagainst"] != df_Shots_and_Goals.iloc[Row + 1]["nextgoalfor"] ) & ( ( df_Shots_and_Goals.iloc[Row + 1]["frames"] >= (df_Shots_and_Goals.iloc[Row]["frames"] + df_Shots_and_Goals.iloc[Row]["nextgoalagainst_timeto"]) ) & ( df_Shots_and_Goals.iloc[Row + 1]["frames"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalfor"] ) & ( df_Shots_and_Goals.iloc[Row]["nextgoalagainst_timeto"] < df_Shots_and_Goals.iloc[Row + 1]["nextgoalfor_timeto"] ) ):

            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Shot")] = 1.0
            df_Shots_and_Goals.iloc[Row, df_Shots_and_Goals.columns.get_loc("Will_Be_a_Goal")] = 1.0
```

Figure C.3: Conditional checks for scenarios 5 to 8 in the possessions data to spot unrecorded goals due to logging mistakes.

```
HeteroData(
  y=[1],
  Ball={
    x=[1, 4],
    pos=[1, 2],
  },
  Player={
    x=[22, 3],
    pos=[22, 2],
  },
  (Ball, Interacts, Player)={
    edge_index=[2, 22],
    edge_attr=[22],
  },
  (Player, Connects, Player)={
    edge_index=[2, 231],
    edge_attr=[231, 2],
  },
  (Player, rev_Interacts, Ball)={
    edge_index=[2, 22],
    edge_attr=[22],
  },
  (Player, rev_Connects, Player)={
    edge_index=[2, 231],
    edge_attr=[231, 2],
  }
)
```

Figure C.4: Graph structure of the PyG undirected heterogeneous “simple” graph data (‘HeteroData’) objects used as input for the “simple” HGT model.

```
HeteroData(
  y=[1],
  Ball={
    x=[1, 9],
    pos=[1, 3],
  },
  Player={
    x=[22, 8],
    pos=[22, 2],
  },
  (Ball, Interacts, Player)={
    edge_index=[2, 22],
    edge_attr=[22],
  },
  (Player, Connects, Player)={
    edge_index=[2, 231],
    edge_attr=[231, 2],
  },
  (Player, rev_Interacts, Ball)={
    edge_index=[2, 22],
    edge_attr=[22],
  },
  (Player, rev_Connects, Player)={
    edge_index=[2, 231],
    edge_attr=[231, 2],
  }
)
```

Figure C.5: Graph structure of the PyG undirected heterogeneous “complex” graph data (‘HeteroData’) objects used as input for the “complex” HGT model.

```python
class HGT_Model_Simple(torch.nn.Module):

    def __init__(self, num_HGTConv_layers, hidden_NN_node_units, num_attention_heads, message_aggregation_strategy, dropout_rate, global_graph_pooling_strategy):

        super(HGT_Model_Simple, self).__init__()

        torch.manual_seed(7)


        self.Output_Hidden_NN_Node_Units = hidden_NN_node_units

        self.Dropout_Rate = dropout_rate   # Store Dropout-Rate As a Class Attribute
        self.Global_Graph_Pooling_Strategy = global_graph_pooling_strategy   # Store Global Graph Pooling Strategy As a Class Attribute


        self.HGT_Convs_list = torch.nn.ModuleList()

        for Layer_i in range(num_HGTConv_layers):

            Input_Hidden_NN_Node_Units = hidden_NN_node_units if Layer_i > 0 else -1

            HGT_Convolutional_layer = HGTConv( in_channels = Input_Hidden_NN_Node_Units, out_channels = self.Output_Hidden_NN_Node_Units, metadata = First_Training_Graph.metadata(),
                                               heads = num_attention_heads, group = message_aggregation_strategy )

            self.HGT_Convs_list.append(HGT_Convolutional_layer)



        # Adjusted (i.e. `2 *`) For Combined Features From Nodes & Edges

        self.Final_Readout_Linear_Layer = Linear( in_channels = 2 * self.Output_Hidden_NN_Node_Units, out_channels = 1 )   # out_channels = 1  <==>  Because We Are Doing Binary Classification



    def forward(self, x_dict, edge_index_dict, ball_batch_index, player_batch_index):
```

Figure C.6: HGT models' architecture (1).

```python
    def forward(self, x_dict, edge_index_dict, ball_batch_index, player_batch_index):


        # HGT Convolutional Layers

        for HGT_Convolution in self.HGT_Convs_list:

            x_dict = HGT_Convolution( x_dict, edge_index_dict )

            x_dict = { key : F.relu(x) for key, x in x_dict.items() }

            x_dict = { key : F.dropout( x, p = self.Dropout_Rate, training = self.training ) for key, x in x_dict.items() }


        # Global Graph Pooling Strategy

        Pooling_Function = global_mean_pool if self.Global_Graph_Pooling_Strategy == "mean" else \
                           global_add_pool if self.Global_Graph_Pooling_Strategy == "sum" else \
                           global_max_pool


        # Pooling For Both Node Types

        Ball_Node_Features_pooling = Pooling_Function( x_dict["Ball"], ball_batch_index )

        Player_Node_Features_pooling = Pooling_Function( x_dict["Player"], player_batch_index )


        # Combining Node- & Edge-Features

        Combined_Node_Features_Embeddings = torch.cat( [Ball_Node_Features_pooling, Player_Node_Features_pooling], dim = 1 )


        Output = self.Final_Readout_Linear_Layer(Combined_Node_Features_Embeddings)


        return Output
```

Figure C.7: HGT models' architecture (2).

Appendix D

Supplementary Plots and Figures

In this appendix, one can observe that for both the training-validation and the training-testing protocols, both HGT models ("simple" and "complex") are still underfitting, which gives room to believe and hope that the models could keep learning and improving further if some of the limitations present in this research were to be overcome.

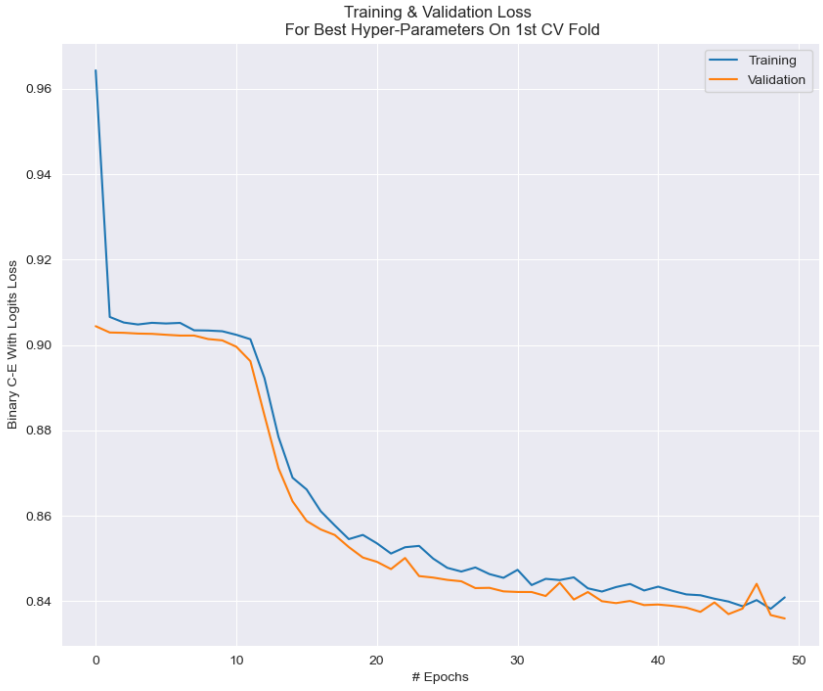


Figure D.1: "Simple" HGT model's training and validation learning curves of the binary cross-entropy (BCE) with logits loss for the best hyper-parameters on the 1st fold (the best fold).

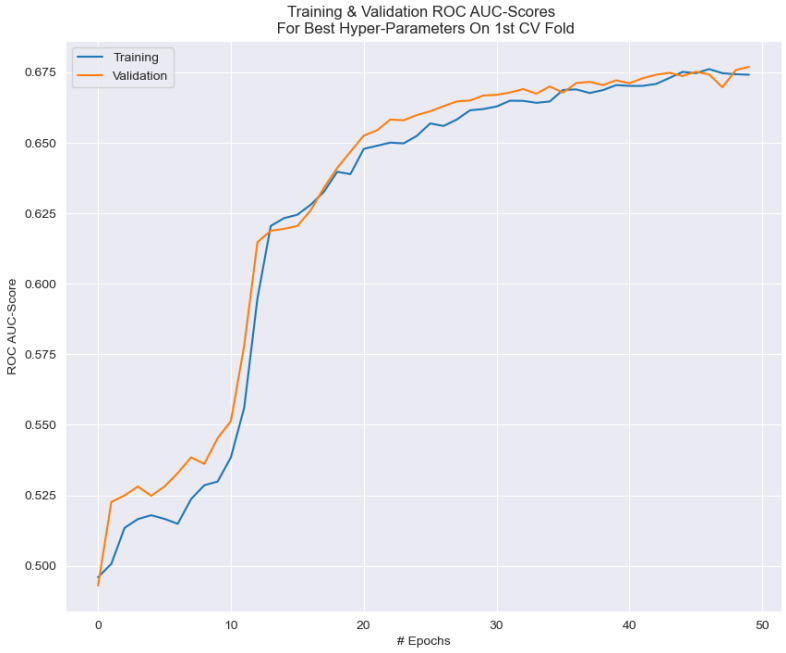


Figure D.2: "Simple" HGT model's training and validation learning curves of the ROC AUC-score for the best hyper-parameters on the 1st fold (the best fold).

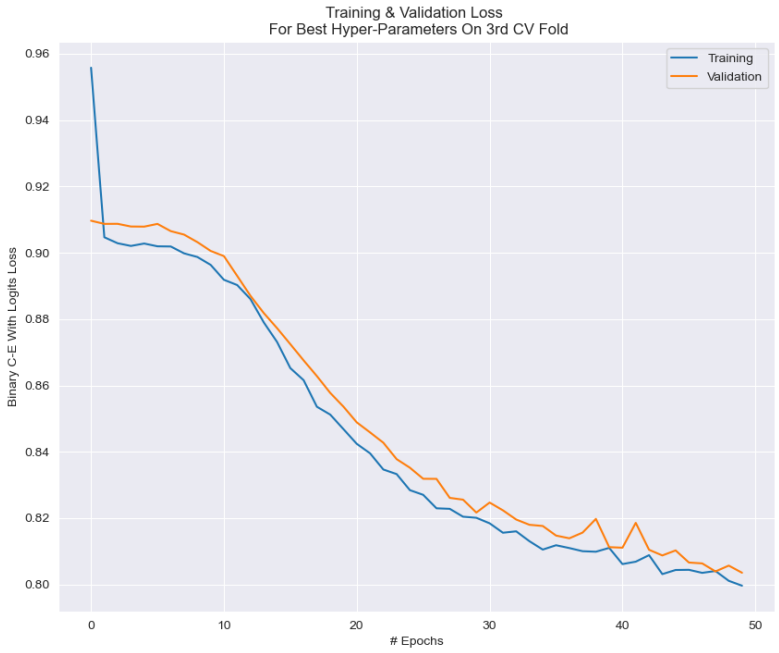


Figure D.3: "Complex" HGT model's training and validation learning curves of the binary cross-entropy (BCE) with logits loss for the best hyper-parameters on the 3rd fold (the best fold).

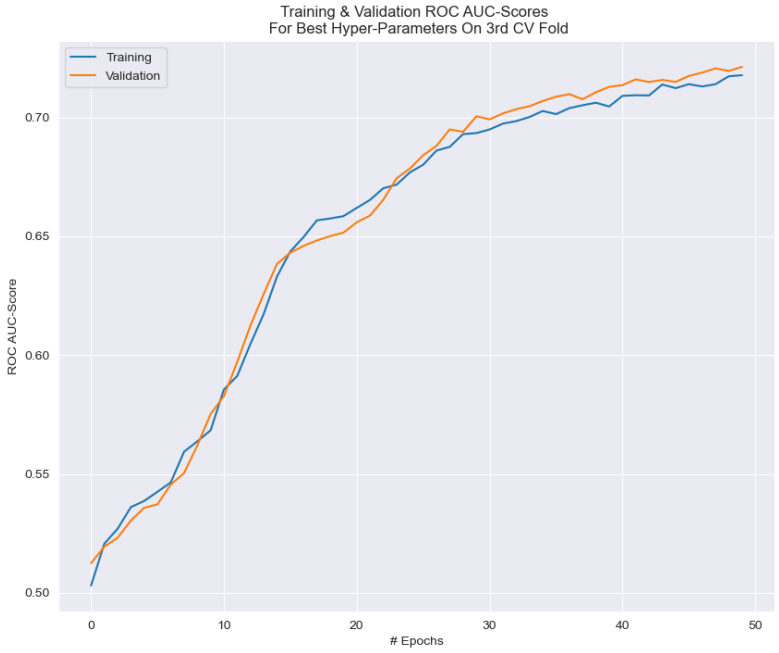


Figure D.4: "Complex" HGT model's training and validation learning curves of the ROC AUC-score for the best hyper-parameters on the 3rd fold (the best fold).

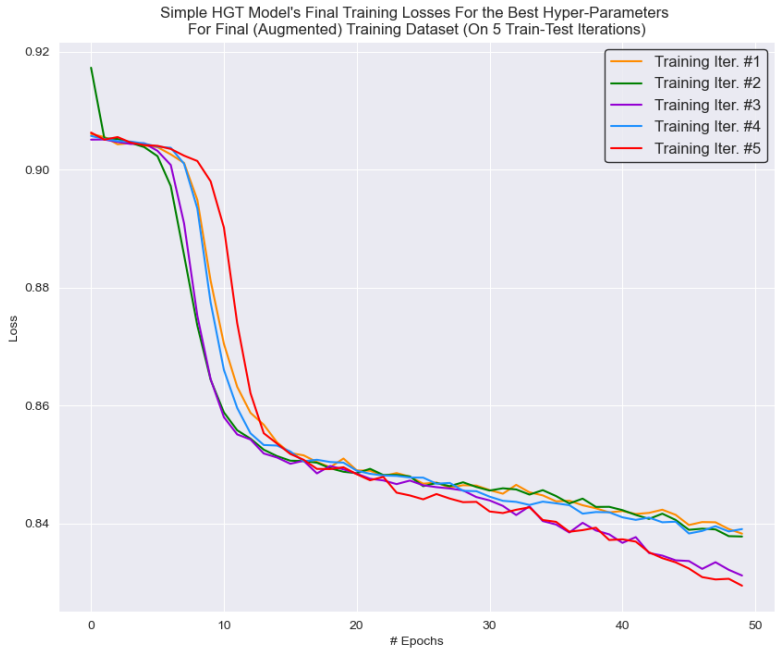


Figure D.5: "Simple" HGT model's training learning curves of the binary cross-entropy (BCE) with logits loss for the best hyper-parameters at each iteration during the final training-testing protocol.

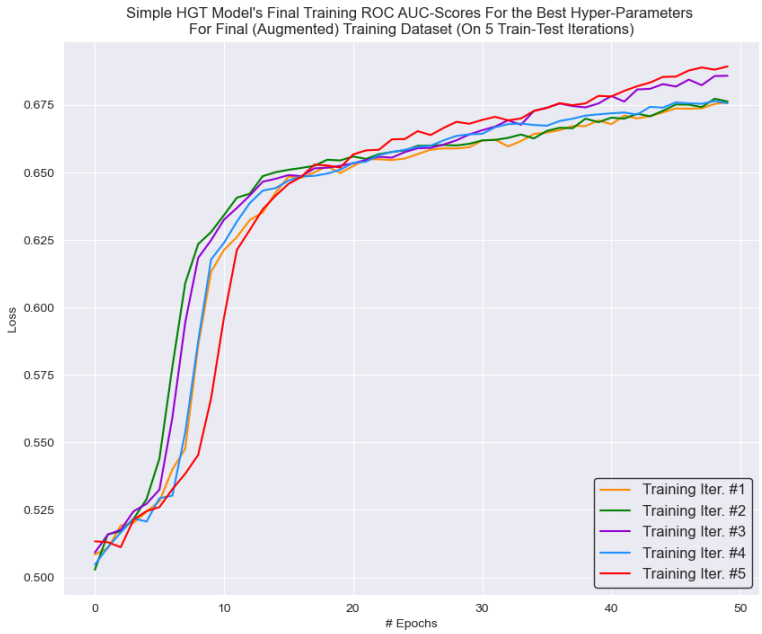


Figure D.6: "Simple" HGT model's training learning curves of the ROC AUC-score for the best hyper-parameters at each iteration during the final training-testing protocol.

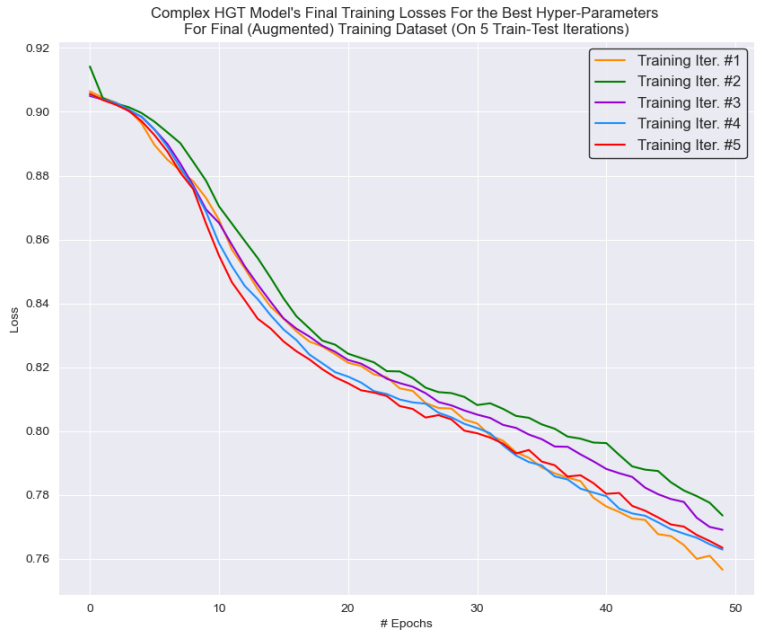


Figure D.7: "Complex" HGT model's training learning curves of the binary cross-entropy (BCE) with logits loss for the best hyper-parameters at each iteration during the final training-testing protocol.

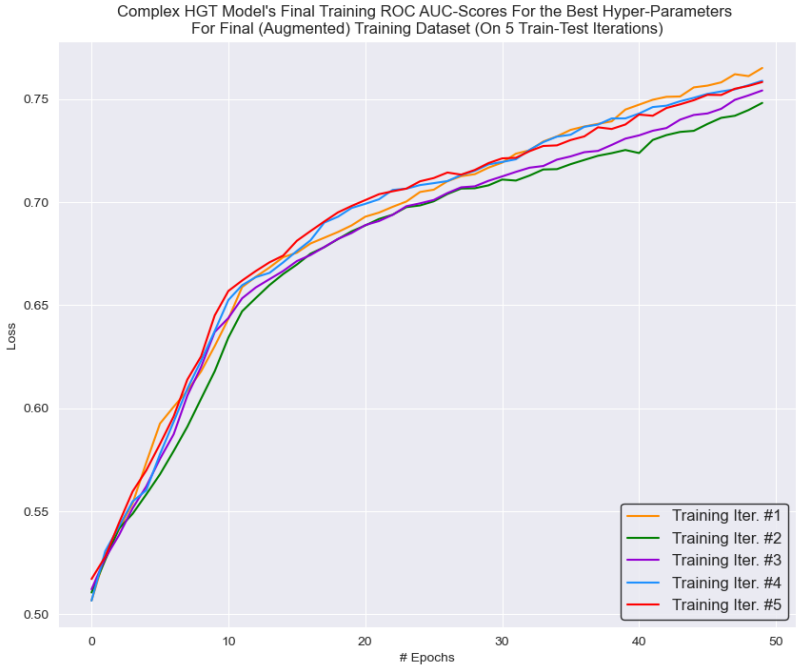


Figure D.8: “Complex” HGT model’s training learning curves of the ROC AUC-score for the best hyper-parameters at each iteration during the final training-testing protocol.

Bibliography

[1] Anzer G, Bauer P, Brefeld U, Faßmeyer D. Detection of tactical patterns using semi-supervised graph neural networks. In: MIT Sloan Sports Analytics Conference. vol. 16; 2022. p. 1-3.

[2] Sahasrabudhe A, Bekkers J. A Graph Neural Network deep-dive into successful counterattacks. In: Proceedings of the 17th MIT Sloan Sports Analytics Conference; 2023. .

[3] Geometric P. Colab Notebooks and Video Tutorials - Graph Classification with Graph Neural Networks;. Available from: https://colab.research.google.com/drive/1I8a0DfQ3fI7Njc62__mVXUlcAleUclnb?usp=sharing#scrollTo=N-FO5xL3mw98.

[4] Pollard R, Ensum J, Taylor S. Estimating the probability of a shot resulting in a goal: The effects of distance, angle and space. Int J Soccer Sci. 2004 01;2.

[5] Ensum J, Pollard R, Taylor S. Applications of logistic regression to shots at goal in association football: Calculation of shot probabilities, quantification of factors and player/team. Journal of Sports Sciences. 2004;22(6):500-20.

[6] Green S. Assessing the performance of premier leauge goalscorers. OptaPro Blog. 2012. Available from: https://www.statsperform.com/resource/assessing-the-performance-of-\premier-league-goalscorers.

[7] Wagenaar M, Okafor E, Frencken W, Wiering M. Using deep convolutional neural networks to predict goal-scoring opportunities in soccer. In: 6th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2017); 2017. .

[8] Wu L, Chen Y, Shen K, Guo X, Gao H, Li S, et al. Graph neural networks for natural language processing: A survey. Foundations and Trends® in Machine Learning. 2023;16(2):119-328.

[9] Zhang XM, Liang L, Liu L, Tang MJ. Graph neural networks and their current applications in bioinformatics. Frontiers in genetics. 2021;12:690049.

[10] Zheng C, Fan X, Wang C, Qi J. Gman: A graph multi-attention network for traffic prediction. In: Proceedings of the AAAI conference on artificial intelligence. vol. 34; 2020. p. 1234-41.

[11] Hu Z, Dong Y, Wang K, Sun Y. Heterogeneous graph transformer. In: Proceedings of the web conference 2020; 2020. p. 2704-10.

[12] Fey M, Lenssen JE. Fast graph representation learning with PyTorch Geometric. arXiv preprint arXiv:190302428. 2019.

[13] Anzer G, Bauer P. A Goal Scoring Probability Model for Shots Based on Synchronized Positional and Event Data in Football (Soccer). Frontiers in Sports and Active Living. 2021;3:624475.

[14] Power P, Ruiz H, Wei X, Lucey P. Not all passes are created equal: Objectively measuring the risk and reward of passes in soccer from tracking data. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining; 2017. p. 1605-13.

[15] Anzer G, Bauer P. Expected passes: Determining the difficulty of a pass in football (soccer) using spatio-temporal data. Data mining and knowledge discovery. 2022;36(1):295-317.

[16] Spearman W, Basye A, Dick G, Hotovy R, Pop P. Physics-based modeling of pass probabilities in soccer. In: Proceeding of the 11th MIT Sloan Sports Analytics Conference; 2017. .

[17] Spearman W. Beyond expected goals. In: Proceedings of the 12th MIT sloan sports analytics conference; 2018. p. 1-17.

[18] Lopez Jorda T. ¿Qué fue del tiqui-taca?; August 02 2020. Available from: https://www.lavanguardia.com/deportes/futbol/20200802/482627281411/tiqui-taca-tiki-taka-johan-\cruyff-pep-guardiola-barcelona\-analisis-luis-milla-pichi-\alonso-alex-delmas.html.

[19] Lazzeri H, Leveridge S. Bayern Munich of 2020 vs Barcelona of 2009: Which sextuple side is best?; February 11 2021. Available from: https://www.marca.com/en/football/barcelona/2021/02/11/6025a426e2704e69af8b462c.html.

[20] Ronay B. Spain's champions abandoned tiki-taka as a sinking ship; June 19 2014. Available from: https://www.theguardian.com/football/blog/2014/jun/19/spains-devotion-to-tika-taka-\became-old-and-stale-in-brazil.

[21] Fernandez J, Bornn L. Wide Open Spaces: A statistical technique for measuring space creation in professional soccer. In: Sloan sports analytics conference. vol. 2018; 2018. .

[22] Romer D. Do firms maximize? Evidence from professional football. Journal of Political Economy. 2006;114(2):340-65.

[23] Kempton T, Kennedy N, Coutts AJ. The expected value of possession in professional rugby league match-play. Journal of sports sciences. 2016;34(7):645-50.

[24] Cervone D, D'Amour A, Bornn L, Goldsberry K. Pointwise: Predicting points and valuing decisions in real time with nba optical tracking data. In: Proceedings of the 8th MIT Sloan Sports Analytics Conference, Boston, MA, USA. vol. 28; 2014. p. 3.

[25] Schulte O, Zhao Z, Routley K. What is the Value of an Action in Ice Hockey? Learning a Q-function for the NHL. In: Proceedings of the 2nd Workshop on Machine Learning and Data Mining for Sports Analytics; 2015. .

[26] Liu G, Schulte O. Deep reinforcement learning in ice hockey for context-aware player evaluation. arXiv preprint arXiv:180511088. 2018.

[27] Van Roy M, Robberechts P, Decroos T, Davis J. Valuing on-the-ball actions in soccer: a critical comparison of xT and VAEP. In: Proceedings of the AAAI-20 Workshop on Artifical Intelligence in Team Sports. AI in Team Sports Organising Committee; 2020. .

[28] Anzer G. Large Scale Analysis of Offensive Performance in Football-Using Synchronized Positional and Event Data to Quantify Offensive Actions, Tactics, and Strategies. Universität Tübingen; 2022.

[29] Decroos T, Bransen L, Van Haaren J, Davis J. Actions speak louder than goals: Valuing player actions in soccer. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining; 2019. p. 1851-61.

[30] Rein R, Memmert D. Big data and tactical analysis in elite soccer: future challenges and opportunities for sports science. SpringerPlus. 2016;5(1):1-13.

[31] Di Salvo V, Modonutti M. Integration of different technology systems for the development of football training. J Sports Sci Med. 2009;11:3.

[32] Anzer G, Bauer P, Brefeld U. The origins of goals in the German Bundesliga. Journal of Sports Sciences. 2021;39(22):2525-44.

[33] Andrienko G, Andrienko N, Anzer G, Bauer P, Budziak G, Fuchs G, et al. Constructing spaces and times for tactical analysis in football. IEEE Transactions on Visualization and Computer Graphics. 2019;27(4):2280-97.

[34] Bialkowski A, Lucey P, Carr P, Yue Y, Sridharan S, Matthews I. Identifying team style in soccer using formations learned from spatiotemporal tracking data. In: 2014 IEEE international conference on data mining workshop. IEEE; 2014. p. 9-14.

[35] Bialkowski A, Lucey P, Carr P, Yue Y, Sridharan S, Matthews I. Large-scale analysis of soccer matches using spatiotemporal tracking data. In: 2014 IEEE international conference on data mining. IEEE; 2014. p. 725-30.

[36] Kovalchik SA. Player Tracking Data in Sports. Annual Review of Statistics and Its Application. 2023;10:677-97.

[37] Szymski D, Weber H, Anzer G, Alt V, Meyer T, Gärtner BC, et al. Contact times in professional football before and during the SARS-CoV-2 pandemic: Tracking data from the German Bundesliga. European Journal of Sport Science. 2023;23(3):460-7.

[38] Link D, Anzer G. How the COVID-19 pandemic has changed the game of soccer. International journal of sports medicine. 2022;43(01):83-93.

[39] Stein M, Janetzko H, Seebacher D, Jäger A, Nagel M, Hölsch J, et al. How to make sense of team sport data: From acquisition to data modeling and research aspects. Data. 2017;2(1):2.

[40] Fernández J, Bornn L, Cervone D. Decomposing the immeasurable sport: A deep learning expected possession value framework for soccer. In: 13th MIT Sloan Sports Analytics Conference; 2019. .

[41] Link D, Lang S, Seidenschwarz P. Real Time Quantification of Dangerousity in Soccer Using Spatiotemporal Tracking Data. In: International Association of Computer Science in Sport (IACSS) Conference; 2016. p. 12.

[42] Kempe M, Goes FR, Lemmink KA. Smart data scouting in professional soccer: Evaluating passing performance based on position tracking data. In: 2018 IEEE 14th International Conference on e-Science (e-Science). IEEE; 2018. p. 409-10.

[43] Goes F, Kempe M, Van Norel J, Lemmink K. Modelling team performance in soccer using tactical features derived from position tracking data. IMA Journal of Management Mathematics. 2021;32(4):519-33.

[44] Forcher L, Altmann S, Forcher L, Jekauc D, Kempe M. The use of player tracking data to analyze defensive play in professional soccer-A scoping review. International Journal of Sports Science & Coaching. 2022;17(6):1567-92.

[45] Bauer P, Anzer G. Data-driven detection of counterpressing in professional football: A supervised machine learning task based on synchronized positional and event data with expert-based feature extraction. Data Mining and Knowledge Discovery. 2021;35(5):2009-49.

[46] McHale I, Scarf P. Modelling soccer matches using bivariate discrete distributions with general dependence structure. Statistica Neerlandica. 2007;61(4):432-45.

[47] McHale IG, Scarf PA, Folker DE. On the development of a soccer player performance rating system for the English Premier League. Interfaces. 2012;42(4):339-51.

[48] Pappalardo L, Cintia P, Rossi A, Massucco E, Ferragina P, Pedreschi D, et al. A public data

set of spatio-temporal match events in soccer competitions. Scientific data. 2019;6(1):236.

[49] Decroos T, Bransen L, Van Haaren J, Davis J. VAEP: an objective approach to valuing on-the-ball actions in soccer. In: IJCAI; 2020. p. 4696-700.

[50] Rudd S. A framework for tactical analysis and individual offensive production assessment in soccer using markov chains. In: New England symposium on statistics in sports; 2011. .

[51] Mackay N. Predicting goal probabilities for possessions in football. Vrije Universiteit Amsterdam. 2017.

[52] Decroos T, Van Haaren J, Dzyuba V, Davis J, Kaytoue M, Zimmermann A. STARSS: a spatio-temporal action rating system for soccer. In: Machine Learning and Data Mining for Sports Analytics ECML/PKDD 2017 workshop. vol. 1971. Springer; 2017. p. 11-20.

[53] Singh K. Introducing Expected Threat (xT); December 24 2018. Available from: https://karun.in/blog/expected-threat.html.

[54] Yam D. Attacking contributions: Markov models for football. StatsBomb–2019 [Electronic resource]–Access mode: https://statsbomb com/articles/soccer/attacking-contributions-markov-models-for-football. 2019.

[55] Mackay N. Introducing a Possession Value framework; 2019. Available from: https://www.statsperform.com/resource/introducing-a-\possession-value-framework/.

[56] Sawczuk T, Palczewska A, Jones B. Development of an expected possession value model to analyse team attacking performances in rugby league. Plos one. 2021;16(11):e0259536.

[57] Cervone D, D'Amour A, Bornn L, Goldsberry K. A multiresolution stochastic process model for predicting basketball possession outcomes. Journal of the American Statistical Association. 2016;111(514):585-99.

[58] Sumpter D. Soccermatics: Mathematical Adventures in the Beautiful Game Pro-Edition. Bloomsbury Publishing; 2017.

[59] Pollard R, Reep C. Measuring the effectiveness of playing strategies at soccer. Journal of the Royal Statistical Society Series D: The Statistician. 1997;46(4):541-50.

[60] Robberechts P, Davis J. How data availability affects the ability to learn good xG models. In: Machine Learning and Data Mining for Sports Analytics: 7th International Workshop, MLSA 2020, Co-located with ECML/PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings 7. Springer; 2020. p. 17-27.

[61] Davis J, Robberechts P. Illustrating The Interplay Between Features and Models In xG; May 14 2020. Available from: https://dtai.cs.kuleuven.be/sports/blog/illustrating-the-interplay-\between-features-and-models-\in-xg.

[62] Adams S. An Introduction to Logistic Regression in Python with statsmodels and scikit-learn; June 17 2020. Available from: https://levelup.gitconnected.com/an-introduction-to-logistic-\regression-in-python-with-\statsmodels-and-scikit-learn-\1a1fb5ce1c13.

[63] Sumpter D. Introducing expected goals; 2022. Available from: https://soccermatics.readthedocs.io/en/latest/lesson2/introducingExpectedGoals.html.

[64] Lu Z, Yang J, Sumpter D. Statistical and Visualization Methods on Evaluating Players in Football. 2020.

[65] Lucey P, Bialkowski A, Monfort M, Carr P, Matthews I. quality vs quantity: Improved shot prediction in soccer using strategic features from spatiotemporal data. 2015.

[66] Eggels H, Van Elk R, Pechenizkiy M. Explaining Soccer Match Outcomes with Goal Scoring Opportunities Predictive Analytics. In: Mlsa@pkdd/ecml; 2016. .

[67] Rathke A. An examination of expected goals and shot efficiency in soccer. Journal of Human Sport and Exercise. 2017;12(2):514-29.

[68] Kharrat T, McHale IG, Peña JL. Plus–minus player ratings for soccer. European Journal of Operational Research. 2020;283(2):726-36.

[69] Madrero Pardo P. Creating a model for expected goals in football using qualitative player information. Universitat Politècnica de Catalunya; 2020.

[70] Raudonius L, Seidl T. Shot Analysis in Different Levels of German Football Using Expected Goals. In: International Workshop on Machine Learning and Data Mining for Sports Analytics. Springer; 2022. p. 14-26.

[71] Cavus M, Biecek P. Explainable expected goal models for performance analysis in football analytics. In: 2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA). IEEE; 2022. p. 1-9.

[72] Mead J, O'Hare A, McMenemy P. Expected goals in football: Improving model performance and demonstrating value. Plos one. 2023;18(4):e0282295.

[73] Javed D, Jhanjhi N, Khan NA. Football Analytics for Goal Prediction to Assess Player Performance. In: Innovation and Technology in Sports: Proceedings of the International Conference on Innovation and Technology in Sports,(ICITS) 2022, Malaysia. Springer; 2023. p. 245-57.

[74] Karim H, Marwane L. The Kos Angle, an optimizing parameter for football expected goals

(xG) models. International Journal of Computer Science in Sport. 2023;22(2):49-61.

[75] Vatvani D. Upgrading Expected Goals; May 16 2022. Available from: https://statsbomb.com/articles/soccer/upgrading-expected-goals/.

[76] Everett G, Beal R, Matthews T, Norman T, Ramchurn G. Contextual Expected Threat using Spatial Event Data. 2022.

[77] Van Roy M, Robberechts P, Davis J. Optimally Disrupting Opponent Build-ups. In: StatsBomb Conference; 2021. .

[78] Van Roy M, Robberechts P, Yang WC, De Raedt L, Davis J. Leaving goals on the pitch: Evaluating decision making in soccer. arXiv preprint arXiv:210403252. 2021.

[79] Prokhorenkova L, Gusev G, Vorobev A, Dorogush AV, Gulin A. CatBoost: unbiased boosting with categorical features. Advances in neural information processing systems. 2018;31.

[80] Decroos ea Tom. VAEP: What is VAEP?; 2019. Available from: https://dtai.cs.kuleuven.be/sports/vaep.

[81] Decroos ea Tom. VAEP: Exploring VAEP; 2019. Available from: https://dtai.cs.kuleuven.be/sports/vaep.

[82] Robberechts P. Exploring How VAEP Values Actions; April 27 2020. Available from: https://dtai.cs.kuleuven.be/sports/blog/exploring-how-vaep-values-\actions.

[83] Decroos ea Tom. Valuing On-the-Ball Actions in Soccer: A Critical Comparison of xT and VAEP; December 18 2019. Available from: https://dtai.cs.kuleuven.be/sports/blog/valuing-on-the-ball-actions-in\-soccer-a-critical-comparison\-of-xt-and-vaep/.

[84] Spearman W. Quantifying Pitch Control. In: 2016 OptaPro Analytics Forum; 2016. .

[85] Kim S. Voronoi analysis of a soccer game. Nonlinear Analysis: Modelling and Control. 2004;9(3):233-40.

[86] Masheswaran R, Chang YH, Su J, Kwok S, Levy T, Wexler A, et al. The three dimensions of rebounding. MIT SSAC. 2014.

[87] Fonseca S, Milho J, Travassos B, Araújo D. Spatial dynamics of team sports exposed by Voronoi diagrams. Human movement science. 2012;31(6):1652-9.

[88] Fernández J, Bornn L, Cervone D. A framework for the fine-grained evaluation of the instantaneous expected value of soccer possessions. Machine Learning. 2021;110(6):1389-427.

[89] Fernández de la Rosa J. A framework for the analytical and visual interpretation of complex spatiotemporal dynamics in soccer. 2022.

[90] Fernández J, Bornn L. Soccermap: A deep learning architecture for visually-interpretable analysis in soccer. In: Machine Learning and Knowledge Discovery in Databases. Applied Data Science and Demo Track: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part V. Springer; 2021. p. 491-506.

[91] Owens N, Harris C, Stennett C. Hawk-eye tennis system. In: 2003 international conference on visual information engineering VIE 2003. IET; 2003. p. 182-5.

[92] Singh Bal B, Dureja G. Hawk Eye: A Logical Innovative Technology Use in Sports for Effective Decision Making. Sport Science Review. 2012;21.

[93] Naik BT, Hashmi MF, Bokde ND. A comprehensive review of computer vision in sports: Open issues, future trends and research directions. Applied Sciences. 2022;12(9):4429.

[94] Herold M, Goes F, Nopp S, Bauer P, Thompson C, Meyer T. Machine learning in men's professional football: Current applications and future directions for improving attacking play. International Journal of Sports Science & Coaching. 2019;14(6):798-817.

[95] Gonzalez-Rodenas J, Aranda R, Aranda-Malaves R. The effect of contextual variables on the attacking style of play in professional soccer. 2020.

[96] Brefeld U, Lasek J, Mair S. Probabilistic movement models and zones of control. Machine Learning. 2019;108(1):127-47.

[97] Goes FR, Kempe M, Meerhoff LA, Lemmink KA. Not every pass can be an assist: a data-driven model to measure pass effectiveness in professional soccer matches. Big data. 2019;7(1):57-70.

[98] Kusmakar S, Shelyag S, Zhu Y, Dwyer D, Gastin P, Angelova M. Machine learning enabled team performance analysis in the dynamical environment of soccer. IEEE access. 2020;8:90266-79.

[99] Almulla J, Alam T. Machine learning models reveal key performance metrics of football players to win matches in qatar stars league. IEEE Access. 2020;8:213695-705.

[100] Alguacil FP, Fernandez J, Arce P, Sumpter D. Seeing in to the future: using self-propelled particle models to aid player decision-making in soccer. In: Proceedings of the MIT sloan sports analytics conference; 2020. p. 1-23.

[101] García-Aliaga A, Marquina M, Coterón J, Rodríguez-González A, Luengo-Sánchez S. In-game behaviour analysis of football players using machine learning techniques based on player statistics. International Journal of Sports Science & Coaching. 2021;16(1):148-57.

[102] Llana S, Madrero P, Fernández J, Barcelona F. The right place at the right time: Advanced off-ball metrics for exploiting an opponent's spatial weaknesses in soccer. In: Proceedings of the 14th MIT Sloan Sports Analytics Conference; 2020. .

[103] Robberechts P, Van Roy M, Davis J. un-xPass: Measuring Soccer Player's Creativity. In: Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining; 2023. p. 4768-77.

[104] Hubáček O, Šourek G, Železnỳ F. Deep learning from spatial relations for soccer pass prediction. In: Machine Learning and Data Mining for Sports Analytics: 5th International Workshop, MLSA 2018, Co-located with ECML/PKDD 2018, Dublin, Ireland, September 10, 2018, Proceedings 5. Springer; 2019. p. 159-66.

[105] Dick U, Brefeld U. Learning to rate player positioning in soccer. Big data. 2019;7(1):71-82.

[106] Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2015. p. 3431-40.

[107] Pathak D, Krahenbuhl P, Darrell T. Constrained convolutional neural networks for weakly supervised segmentation. In: Proceedings of the IEEE international conference on computer vision; 2015. p. 1796-804.

[108] Tiwari E, Sardar P, Jain S. Football match result prediction using neural networks and deep learning. In: 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO). IEEE; 2020. p. 229-31.

[109] Fassmeyer D, Anzer G, Bauer P, Brefeld U. Toward automatically labeling situations in soccer. Frontiers in Sports and Active Living. 2021;3:725431.

[110] Cintia P, Pappalardo L. Coach2vec: autoencoding the playing style of soccer coaches. arXiv preprint arXiv:210615444. 2021.

[111] Rahimian P, Oroojlooy A, Toka L. Towards optimized actions in critical situations of soccer games with deep reinforcement learning. In: 2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA). IEEE; 2021. p. 1-12.

[112] Rahimian P, Van Haaren J, Abzhanova T, Toka L. Beyond action valuation: A deep reinforcement learning framework for optimizing player decisions in soccer. In: 16th Annual MIT Sloan Sports Analytics Conference. Boston, MA, USA: MIT; 2022. p. 25.

[113] Rahimian P, Van Haaren J, Toka L. Towards maximizing expected possession outcome in soccer. International Journal of Sports Science & Coaching. 2023:17479541231154494.

[114] Bauer P, Anzer G, Shaw L. Putting team formations in association football into context. Journal of Sports Analytics. 2023;9(1):39-59.

[115] Tuyls K, Omidshafiei S, Muller P, Wang Z, Connor J, Hennes D, et al. Game Plan: What AI can do for Football, and What Football can do for AI. Journal of Artificial Intelligence Research. 2021;71:41-88.

[116] Sanford R, Gorji S, Hafemann LG, Pourbabaee B, Javan M. Group activity detection from trajectory and video data in soccer. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops; 2020. p. 898-9.

[117] Gori M, Monfardini G, Scarselli F. A new model for learning in graph domains. In: Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.. vol. 2. IEEE; 2005. p. 729-34.

[118] Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G. The graph neural network model. IEEE transactions on neural networks. 2008;20(1):61-80.

[119] Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G. Computational capabilities of graph neural networks. IEEE Transactions on Neural Networks. 2008;20(1):81-102.

[120] Campos JC. Determining the phases of play using Graph Neural Network Embeddings.

[121] Stöckl M, Seidl T, Marley D, Power P. Making offensive play predictable-using a graph convolutional network to understand defensive performance in soccer. In: Proceedings of the 15th MIT sloan sports analytics conference. vol. 2022; 2021. .

[122] Dick U, Tavakol M, Brefeld U. Rating player actions in soccer. Frontiers in Sports and Active Living. 2021;3:682986.

[123] Xenopoulos P, Silva C. Graph neural networks to predict sports outcomes. In: 2021 IEEE International Conference on Big Data (Big Data). IEEE; 2021. p. 1757-63.

[124] Du H, Chen Z, Wang Y, Huang Z, Wang Y, Xiong R. Multi-agent trajectory prediction based on graph neural network. In: 2021 IEEE International Conference on Real-time Computing and Robotics (RCAR). IEEE; 2021. p. 632-7.

[125] Cartas A, Ballester C, Haro G. A graph-based method for soccer action spotting using unsupervised player classification. In: Proceedings of the 5th International ACM Workshop on Multimedia Content Analysis in Sports; 2022. p. 93-102.

[126] Komorowski J, Kurzejamski G. Graph-Based Multi-Camera Soccer Player Tracker. In: 2022 International Joint Conference on Neural Networks (IJCNN). IEEE; 2022. p. 1-8.

[127] Omidshafiei S, Hennes D, Garnelo M, Wang Z, Recasens A, Tarassov E, et al. Multiagent offscreen behavior prediction in football. Scientific reports. 2022;12(1):8638.

[128] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. Advances in neural information processing systems. 2017;30.

[129] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems. 2019;32.

[130] Liu J, Niu Y, Shi Y, Zhu J. Graph neural network based agent in Google Research Football. In: 2nd International Conference on Artificial Intelligence, Automation, and High-Performance Computing (AIAHPC 2022). vol. 12348. SPIE; 2022. p. 888-97.

[131] Grover A, Leskovec J. node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining; 2016. p. 855-64.

[132] Donnat C, Zitnik M, Hallac D, Leskovec J. Learning structural node embeddings via diffusion wavelets. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining; 2018. p. 1320-9.

[133] Yılmaz Öİ, Öğüdücü ŞG. Learning football player features using graph embeddings for player recommendation system. In: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing; 2022. p. 577-84.

[134] Teranishi M, Tsutsui K, Takeda K, Fujii K. Evaluation of creating scoring opportunities for teammates in soccer via trajectory prediction. In: International Workshop on Machine Learning and Data Mining for Sports Analytics. Springer; 2022. p. 53-73.

[135] Xie T, Grossman JC. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. Physical review letters. 2018;120(14):145301.

[136] Dick U, Brefeld U. Action rate models for predicting actions in soccer. AStA Advances in Statistical Analysis. 2023;107(1-2):29-49.

[137] Raabe D, Nabben R, Memmert D. Graph representations for the analysis of multi-agent spatiotemporal sports data. Applied Intelligence. 2023;53(4):3783-803.

[138] Goka R, Moroto Y, Maeda K, Ogawa T, Haseyama M. Prediction of Shooting Events in Soccer Videos Using Complete Bipartite Graphs and Players' Spatial-Temporal Relations. Sensors. 2023;23(9):4506.

[139] Rossi E, Chamberlain B, Frasca F, Eynard D, Monti F, Bronstein M. Temporal graph networks for deep learning on dynamic graphs. arXiv preprint arXiv:200610637. 2020.

[140] Rahimian P, Kim H, Schmid M, Toka L. Pass Receiver and Outcome Prediction in Soccer Using Temporal Graph Networks. In: 10th Workshop on Machine Learning and Data Mining for Sports Analytics, Turin, Italy; 2023. .

[141] Wang Z, Veličković P, Hennes D, Tomašev N, Prince L, Kaisers M, et al. TacticAI: an AI assistant for football tactics. arXiv preprint arXiv:231010553. 2023.

[142] Brody S, Alon U, Yahav E. How attentive are graph attention networks? arXiv preprint arXiv:210514491. 2021.

[143] Liang F, Qian C, Yu W, Griffith D, Golmie N. Survey of graph neural networks and applications. Wireless Communications and Mobile Computing. 2022;2022.

[144] Zhou J, Cui G, Hu S, Zhang Z, Yang C, Liu Z, et al. Graph neural networks: A review of methods and applications. AI open. 2020;1:57-81.

[145] Wu Z, Pan S, Chen F, Long G, Zhang C, Philip SY. A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems. 2020;32(1):4-24.

[146] Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE. Neural message passing for quantum chemistry. In: International conference on machine learning. PMLR; 2017. p. 1263-72.

[147] Battaglia PW, Hamrick JB, Bapst V, Sanchez-Gonzalez A, Zambaldi V, Malinowski M, et al. Relational inductive biases, deep learning, and graph networks. arXiv 2018. arXiv preprint arXiv:180601261. 2018.

[148] Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:160902907. 2016.

[149] Veličković P. Everything is connected: Graph neural networks. Current Opinion in Structural Biology. 2023;79:102538.

[150] Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y. Graph attention networks. arXiv preprint arXiv:171010903. 2017.

[151] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2016. p. 770-8.

[152] Ba JL, Kiros JR, Hinton GE. Layer normalization. arXiv preprint arXiv:160706450. 2016.

[153] Zhang B, Xiao J, Jiao J, Wei Y, Zhao Y. Affinity attention graph neural network for weakly supervised semantic segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2021;44(11):8082-96.

[154] Yun S, Jeong M, Kim R, Kang J, Kim HJ. Graph transformer networks. Advances in neural information processing systems. 2019;32.

[155] Jaderberg M, Simonyan K, Zisserman A, et al. Spatial transformer networks. Advances in neural information processing systems. 2015;28.

[156] Zhao J, Wang X, Shi C, Hu B, Song G, Ye Y. Heterogeneous graph structure learning for graph neural networks. In: Proceedings of the AAAI conference on artificial intelligence. vol. 35; 2021. p. 4697-705.

[157] Wang X, Bo D, Shi C, Fan S, Ye Y, Philip SY. A survey on heterogeneous graph embedding: methods, techniques, applications and sources. IEEE Transactions on Big Data. 2022;9(2):415-36.

[158] Zhang C, Song D, Huang C, Swami A, Chawla NV. Heterogeneous graph neural network. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining; 2019. p. 793-803.

[159] Wang X, Ji H, Shi C, Wang B, Ye Y, Cui P, et al. Heterogeneous graph attention network. In: The world wide web conference; 2019. p. 2022-32.

[160] Yang T, Hu L, Shi C, Ji H, Li X, Nie L. HGAT: Heterogeneous graph attention networks for semi-supervised short text classification. ACM Transactions on Information Systems (TOIS). 2021;39(3):1-29.

[161] Sahili ZA, Awad M. Spatio-Temporal Graph Neural Networks: A Survey. arXiv preprint arXiv:230110569. 2023.

[162] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011;12:2825-30.

[163] W D. Understanding the Confusion Matrix; Jul 29 2020. Available from: https://medium.com/@danyal.wainstein1/understanding-the-confusion-\matrix-b9bc45ba2679.

[164] Obuchowski NA. Receiver operating characteristic curves and their use in radiology. Radiology. 2003;229(1):3-8.

[165] Henaff M, Bruna J, LeCun Y. Deep convolutional networks on graph-structured data. arXiv preprint arXiv:150605163. 2015.