Machine Learning with Applications 12 (2023) 100466

Contents lists available at ScienceDirect

Machine Learning with Applications

journal homepage: www.elsevier.com/locate/mlwa

Time to Die 2: Improved in-game death prediction in Dota 2

Charles Ringer [a], Sondess Missaoui [a], Victoria J. Hodge [a], Alan Pedrassoli Chitayat [a], Athanasios Kokkinakis [b], Sagarika Patra [b], Simon Demediuk [a], Alvaro Caceres Munoz [b], Oluseji Olarewaju [b], Marian Ursu [b], Ben Kirman [b], Jonathan Hook [b], Florian Block [b], Anders Drachen [a], James Alfred Walker [a,*]

[a] Department of Computer Science, University of York, UK
[b] School of Arts and Creative Technologies, University of York, UK

ARTICLE INFO

Keywords:
Esports
Dota 2
Deep learning
Micro-prediction
Game analytics
Recurrent neural networks

ABSTRACT

Competitive video game playing, an activity called esports, is increasingly popular to the point that there are now many professional competitions held for a variety of games. These competitions are broadcast in a professional manner similar to traditional sports broadcasting. Esports games are generally fast paced, and due to the virtual nature of these games, camera positioning can be limited. Therefore, knowing ahead of time where to position cameras, and what to focus a broadcast and associated commentary on, is a key challenge in esports reporting. This gives rise to moment-to-moment prediction within esports matches which can empower broadcasters to better observe and process esports matches. In this work we focus on this moment-to-moment prediction and in particular present techniques for predicting if a player will die within a set number of seconds for the esports title *Dota 2*. A player death is one of the most consequential events in *Dota 2*. We train our model on 'telemetry' data gathered directly from the game itself, and position this work as a novel extension of our previous work on the challenge. We use an enhanced dataset covering 9,822 *Dota 2* matches. Since the publication of our previous work, new dataset parsing techniques developed by the WEAVR project enable the model to track more features, namely player status effects, and more importantly, to operate in real time. Additionally, we explore two new enhancements to the original model: one data-based extension and one architectural. Firstly we employ learnt embeddings for categorical features, e.g. which in game character a player has selected, and secondly we explicitly model the temporal element of our telemetry data using recurrent neural networks. We find that these extensions and additional features all aid the predictive power of the model achieving an F1 score of 0.54 compared to 0.17 for our previous model (on the new data). We improve this further by experimenting with the length of the time-series in the input data and find using 15 time steps further improves the F1 score to 0.62. This compares to F1 of 0.1 for a standard RNN on the same task. Additionally a deeper analysis of the Time to Die model is carried out to assess its suitability as a broadcast aid.

1. Introduction

Esports are an increasingly popular spectacle. Thus they are an increasingly important part of the video game ecosystem, and by extension video games research (Schubert, Drachen, & Mahlmann, 2016). *Dota 2* is one of the largest esports titles[1] and sees two teams of five players attempt to battle across a map to reach the opposing teams 'ancient' and destroy it, in a genre knows as 'multi-player online battle area' (MOBA) games. *Dota 2* is a highly complex game. For example, each player selects a unique hero, from a pool of about 120, at the start of the match resulting in about $1.16e^{14}$ possible starting hero combinations. Furthermore, during gameplay players have many options for strategy, e.g. they can choose from over 200 items to purchase each with a unique effect. Additionally, timing when to focus on attacking or defending in-game objectives is very important. Each player on a team has a different role to play, similar to positions in traditional

* Corresponding author.
E-mail addresses: charles.ringer@york.ac.uk (C. Ringer), sondess.missaoui@york.ac.uk (S. Missaoui), victoria.hodge@york.ac.uk (V.J. Hodge), alan.pchitayat@york.ac.uk (A.P. Chitayat), athanasios.kokkinakis@york.ac.uk (A. Kokkinakis), moni.patra@york.ac.uk (S. Patra), simon.demediuk@york.ac.uk (S. Demediuk), alvaro.caceres@york.ac.uk (A.C. Munoz), oluseji.olarewaju@york.ac.uk (O. Olarewaju), marian.ursu@york.ac.uk (M. Ursu), ben.kirman@york.ac.uk (B. Kirman), jonathan.hook@york.ac.uk (J. Hook), florian.block@york.ac.uk (F. Block), anders.drachen@york.ac.uk (A. Drachen), james.walker@york.ac.uk (J.A. Walker).

[1] https://esportsobserver.com/top10-games-2020-total-winnings/

https://doi.org/10.1016/j.mlwa.2023.100466
Received 29 November 2022; Received in revised form 4 April 2023; Accepted 5 April 2023
Available online 10 April 2023

team sports, and teams must coordinate these different roles in order to perform effectively. Gameplay is further complicated by the 'fog of war', a mechanic which means that the players in *Dota 2* have a limited view of the battle field. Team must expend resources ensuring that they maximise the amount of the map that is visible to them and choosing exactly how to execute this is important to strategy. Players can also potentially use this 'fog of war' to escape potentially risky situations by moving to a part of the map where the opposing team cannot see them.

During a *Dota 2* game, the players must closely monitor their hero's status, in particular, the likelihood of dying. Thus, a method of predicting the likelihood that a player is going to die within a game of *Dota 2* is likely to be useful for a number of applications, e.g. as an aid for broadcasters or as a coaching aid for players. However, in current esports analytics, often research focuses on using performance metrics, i.e. input features, to model likelihood of team success rather than necessarily a player's likelihood to die (Katona et al., 2019). This death prediction task is difficult and complex. For example, some heroes have abilities which allow them to heal themselves or their team-mates while heroes can purchase items in-game that allow them to heal or teleport away from danger. Hence, it is likely that we cannot simply use a few metrics, e.g. player health, to predict deaths. Likewise, for the performance of players and heroes to be accurately analysed, the historical data needs to be carefully considered and tailored to the specific task. Applying performance metrics without careful selection introduces noise and leads to biased or skewed analyses.

It is possible to curate large datasets of highly granular, high-dimensional, high-volume data from virtually every match for *Dota 2* due to the open nature of the replay system, a tool which allows players to download 'replays' of past matches and play them inside the game client. There exists, therefore, an attractive opportunity to develop telemetry based systems which are capable of predicting in-game moments, for example, when a player is likely to die. Such systems can then be operationalised by esports broadcasts to enrich content delivery, e.g. by informing commentators that a certain player is likely to die soon. This becomes more attractive when you consider how difficult commentating on *Dota 2* is. As we discuss above, the game is highly complex. As such it can be hard for commentators to assimilate the high volume of information present in the game in real time. Aids for the commentators, then, are like to be beneficial in improving the experience of watching *Dota 2* broadcasts. Furthermore, such a system is able to detect dangerous moments in a game before it develops. Analysing the output of the model to identify when a dangerous situation may be developing would allow coaches to train players to avoid situations where the danger would not have previously been observed.

This research is a follow up to our previous work on this topic, Katona et al. (2019), and provides an exploration into various methods for improving the predictive performance of the 'Time to Die' architecture. In this work, we find that utilising learnt embeddings for categorical data improves predictive power. In the original Time 2 Die paper hero IDs are 1-hot encoded. However, 1-hot encoding is computationally expensive as weights for all encoding inputs need to be updated for each training sample, unlike a learnt embedding. In particular we add embeddings for two types of features, hero IDs and items IDs. Embeddings have the advantage that they can be extended should new heroes/items be released and so models can be updated during deployment, fine-tuning only the embeddings which have been added due to the new content. Additionally, recent developments into replay parsing tools by the WEAVR project have make it possible to gather information not possible in the original paper, e.g. 'status effects'. These status effects have the potential to aid predictive power because they describe player state in a way previously not modelled, e.g. if the player is 'stunned' i.e. they cannot move or take actions, or is 'smoked', i.e. concealed under the effects of the item Smoke of Deceit. Finally, we experiment with using a recurrent architecture to model the temporal aspect of a game. *Dota 2* is a dynamic game and thus modelling the changing game state in the seconds running up to a given observation point may lead to advances in the predictive performance of the model. However, the original model used a single 'snapshot' of the game, represented as a single data vector, as input to the model. Historic change in the gamestate was modelling through as set of delta values calculated from one snapshot to the previous, e.g. how much the distance between two players has changed. In this work, we instead explore modelling this historic state change explicitly. We do this with two approaches, firstly replacing a portion of the original Time 2 Die architecture with recurrent layers and secondly replacing the shared-joint architecture of the original Time 2 Die architecture with a single set of recurrent layers.

The key contributions of this paper are as follows:

- A replication of the original Time to Die paper.
- A greater understanding of the role that hyper-parameters have on the performance of the Time to Die model.
- An improved model with a new data configuration and new architecture which utilises embedding layers for certain categorical data features as well as status effect data features. We further revise the model architecture to include recurrent layers to capture temporal data effects. Many papers in the literature on esports prediction rely on data snapshots and do not capture temporal patterns which our analyses show to be important for accurate prediction.
- Experimentation with temporal models, and an exploration of the assumption that a single game time step holds the Markov property (Katona et al., 2019)
- An exploration of the impact that embeddings have on the models, and what is learnt by them.
- An investigation of whether the new model is fast enough to be used in esports broadcasts.

The rest of the paper is laid out in the following manner. Section 2 discusses past literature which has informed this work. Section 3 discusses the dataset gathered for this work, including how it differs from the dataset used in the original Time to Die work. Section 4 details the neural network architectures used. Section 5 discusses the experiments carried out and their outcomes. Section 6 details the implications of these results as well as providing additional analysis. Section 7 concludes our findings. Finally, Section 8 proposes directions for future work.

2. Related work

Despite the rising popularity of esports establishing the state-of-the-art is challenging due to commercial confidentiality resulting in many systems and findings being unavailable to the public. That said, there are emerging trends in academic esports research across a broad range of topics and disciplines including; AI, analytics, psychology, education, visualisation (Block et al., 2018), ethnography, marketing, management and business, e.g. Hamari and Sjöblom (2017), Schubert et al. (2016), Seo (2013), Xue, Pu, Hawzen, and Newman (2016), Yang, Harrison, and Roberts (2014) and Yannakakis (2012). In particular, research into in-game event, role, behaviour and win prediction are the most pertinent and as such will be discussed below.

2.1. In-game event prediction

Prior work into predictive models that are designed to uncover information about events during the game is the research most closely related to this article. Naturally the most important prior work is the original Time to Die paper (Katona et al., 2019) because this work is a direct follow up to that. That work, much like this, used a neural network to predict which players are going to die within 5 seconds during *Dota 2* games. Other closely related work includes Marshall, Mavromoustakos-Blom, and Spronck (2022) who use telemetry data

from *Counter-Strike: Global offensive* (CS:GO) trained into a long short-term memory (LSTM) recurrent neural model to predict in-game deaths within a three-second window with F1-score of 0.38. They identified a number of important data features for accurate prediction including a player's death count, health, enemies in range and equipment value as important.

Closely related, Cleghern, Lahiri, Özaltin, and Roberts (2017) predicted hero health in *Dota 2* using a combination of techniques: an auto-regressive moving average model (ARMA) to predict small changes in health with a peak accuracy of 77.2% and a statistical (logistic and linear regression) estimation model (see Cleghern et al., 2017) which predicts large changes in health 10 seconds ahead. However, this latter model has poor prediction accuracy, although when it does correctly predict a health change, the accuracy is around 80% for the direction and magnitude of the health change. Cleghern et al. (2017) only use health data, unlike the present work which uses a large number of features. Furthermore, their dataset comprises just 542 matches, with no ability differentiation so it is not clear if the results would work for professional and/or amateur matches. Lopez-Gordo, Kohlmorgen, Morillas, and Pelayo (2020) predicts player performance at the single-event level, namely shot accuracy (hit or miss) in a first-person shooter (FPS) video game achieving 74% prediction accuracy using a two-layer feed-forward ANN. At the team level, Schubert et al. (2016) described a graph-based method for detecting spatio-temporally bounded team encounters (team fights) in *Dota 2*, and, early on in esports research, noted the potential for predictive analytics in esports. Ke et al. (2022) developed a framework for defining and extracting teamfight definitions in *Dota 2*. They evaluated whether team fights (key but relatively rare events) held a signal useful for match outcome prediction. Testing different RNNs, they reached over 50% accuracy using a bidirectional LSTM, and just 5 minutes of match data. Accuracy increased with further data input illustrating the importance of having rich training data.

As an assistive tool for esports commentators, authors have developed tools to predict in-game video highlights (showing events of interest). Many incorporate recurrent architecture layers to capture temporal patterns. Zhang, Liu, Wang, and Sun (2020) use deep neural networks to analyse the correlations between the esports game's spatio-temporal features and the highlights in the corresponding gamecast. Similarly, Ringer, Walker, and Nicolaou (2019) used spatio-temporal features to predict in-game events as well the streamer's emotional state in livestreams of the MOBA game League of Legends. Kang and Lee (2020) couple a multi-layer perceptron win–loss classifier with a deep learning network to analyse the video highlights and identify points when the rate of change of the win–loss prediction is highest. Luo, Guzdial, and Riedl (2019) use Convolutional Neural Networks to predict in-game events from *Dota 2* gameplay video frames. They predict events including item usage, fights, game end with 95% accuracy though this would still cause three mistakes per second when parsing livestream data at 60 FPS. Wang et al. (2020) use a time-enabled transformer recurrent neural network to analyse behaviour sequences in the lead-up to important in-game events as described in the game log.

2.2. Role and behaviour prediction

Analysing and predicting player roles and player behaviour allows players and spectators to understand games more. Eggert, Herrlich, Smeddinck, and Malaka (2015) used supervised logistic regression to classify players into pre-determined roles using performance metrics. The authors crowd-sourced their labelled data through asking users to label player roles in game replays. They found that player performance can affect the quality of logistic regression classification — if players perform poorly at a role then classification is less accurate. In contrast, Demediuk, York, Drachen, Walker, and Block (2019) use non-performance metrics which are less reliant on player quality. They use unsupervised ensemble clustering to group players into roles using data on player movement, resource and ability prioritisation. Drachen et al. (2014) study team behaviour, and analyse how spatio-temporal behaviour of teams in *Dota 2* varies across different skill tiers (player abilities) using statistical (ANOVA) analyses and clustering of time-series data. They found the spread of teams and the positions of players varies with skill level. The results of Drachen et al. (2014) and Eggert et al. (2015) indicate that the players' skill levels affect players' roles and behaviour and algorithms' abilities to classify them which has implications for death prediction.

2.3. Win prediction

The majority of academic work applying predictive models to esports has focused on predicting the outcome of esports matches and a number of algorithms have been explored. The interested reader may see Hodge et al. (2019) for an overview of *Dota 2* win prediction. Many authors use logistic regression: Agarwala and Pearce (2014), Kinkade, Jolla, and Lim (2015), Makarov, Savostyanov, Litvyakov, and Ignatov (2018), Maymin (2021), Pobiedina, Neidhardt, Calatrava Moreno, and Werthner (2013), Schubert et al. (2016), Song, Zhang, and Ma (2015), Wang (2016) and Yang, Qin, and Lei (2017), other statistical techniques include Gaussian processes (Bailey, 2020) and naïve Bayes algorithm (Wang & Shang, 2017). Machine learning algorithms include decision trees (Rioult, Métivier, Helleu, Scelles, & Durand, 2014; Yang et al., 2014), random forests (Ani, Harikumar, Devan, & Deepa, 2019; Hodge et al., 2019, 2017; Johansson & Wikström, 2015), support vector machines (Anshori, Mar'i, Alauddin, & Abdurrahman Bachtiar, 2018), and k-nearest neighbour (Conley & Perry, 2013). These approaches use data snapshots capturing only one, or occasionally, multiple time slices. More recent work aims to capture temporal patterns using neural network algorithms such as recurrent neural networks (RNNs) (Silva, Pappa, & Chaimowicz, 2018; Wang et al., 2020; White & Romano, 2020; Yu, Zhang, Chen, & Xie, 2018), long short-term memory NNs (Akhmedov & Phan, 2021) (including bi-directional LSTMs Kim & Lee, 2020) and two-stage spatial temporal networks (Yang et al., 2022). Wang et al. (2020) use a time-enabled transformer recurrent neural network to encode behaviour sequences, capture important events and predict the winner. It can also be used for highlight detection as discussed in Section 2.1.

Akhmedov and Phan (2021) and Hodge et al. (2019) have integrated their win predictors with a live game data feed to provide near real-time win prediction.

2.4. Overview

Authors identify the importance of using a rich set of data features for prediction. We expand our range of features in this paper compared to the previous Time to Die paper (Katona et al., 2019) to capture more knowledge. Our new approach adds recurrent deep learning to capture temporal relationships in the data and its features, feature embeddings to efficiently capture more features and more feature values coupled with a rich set of in-game data features to predict player deaths. Similarly to Hodge et al. (2019), we note that other papers in the literature often use small training data sets and do not use professional game-play data even though authors have found that players' abilities effect prediction accuracy. We use a large data set of 9822 professional *Dota 2* matches. Additionally, authors posit that data which does not rely on an individual player's performance is important for prediction. We do not use performance-related data. The two most closely related papers Katona et al. (2019) and Marshall et al. (2022) achieve a precision of 0.38 and F1 score of 0.38 respectively. Through our new data configuration, model architecture enhancements and optimised time-series length, we achieve an F1 score of 0.62 and precision of 0.52. We show in Section 5.3.1 that this new model is able to inference fast enough to be used in esports broadcasts.

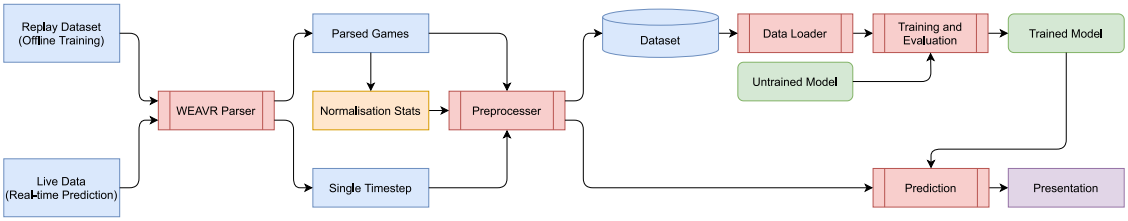


Fig. 1. Data flow through the program for both the offline training of the model and the live single timestep data used for real-time live prediction.

3. Data

Data were gathered from a set of *9822* matches from 'replay' files retrieved through the OpenDota platform via its API support.[2] All matches were collected from patch *7.27*[3] from all *Dota 2* regions and played in a period between June and December 2020. We split the data as 65% for training the models, 5% for sweeping the hyper-parameters to identify the best settings for each model (this process is described in Section 4.4), 10% for validating each model and 20% was kept back as an unseen test set for our evaluations in Section 5.

This study used professional and premium matches only, as the model and methodology described in this paper is more directly applicable to the esports domain, thus to professional and premium matches that attract broadcast interest. As observed in the literature (Chitayat et al., 2020; Semenov, Romov, Korolev, Yashkov, & Neklyudov, 2016), characteristics of play style and decision making can change drastically depending on player skill level. For this reason a complete dataset of high skilled tournament matches would more directly represent the target performance for the model.

3.1. Processing and normalisation

The data processing pipeline used in this study for both the offline training and real-time live prediction is illustrated in Fig. 1.

The replay files allow audiences and players to re-watch their matches. The game data are stored as a time-series of game snapshots in the file, each snapshot is timestamped and represents 0.033 seconds in game time. We refer to one snapshot as one time step. Replay files vary in size between *50–150* Mb. For this reason, replay files contain extremely high-fidelity information, however it is compressed in such way that utilising these files in their raw would not be feasible. Thus, we used the Clarity Parser library[4] to extract the data used in this study (refer to Section 4). The Clarity Parser is a open source Java library specifically designed to read and extract data from *Dota 2* and *Counter-Strike: Global Offensive* replay files.

The replay files were then processed by the WEAVR infrastructure which enhances the variety and amount of relevant data that can be retrieved. Additional features such as the standardisation of Combat Log Events and game state data allow for a more complete overview and representation of the game in its entirety. Furthermore, the WEAVR framework is designed to work in a live setting, and thus the techniques discussed in this work move from being theoretically useful to potentially deployable. The data were in the form of CSV files which are easily accessible and contain the information necessary. Additionally, each row represents a different time step and the rows are ordered temporally, thus making prediction on time series data straightforward.

Once the replay files have been parsed into CSV files two more processing steps are carried out. Firstly, the files are processed to add extra features which are calculable from the dataset but not explicitly contained in it. Distance and movement features are added in the form of distance to other heroes and structures such as towers as well as delta features describing how these values have changed from timestamp to timestamp. We also convert the representation of certain categorical features, necessary to implement the original Time to Die model. In this case, the hero ID is converted to a one-hot encoding and the items that a certain hero has are converted to a set of binary features which describe if the hero has a certain item, from a small subset of items. The embedding model proposed in this work does not require either processing step so is faster. Next a set of binary historic visibility features is added. These describe if a hero has been visible to the opposing team in the last 10 seconds, with one feature used for each second. This is required because heroes often go in and out of the enemies field of vision and a hero may go out of vision but their location known and thus they are still in danger. Finally we calculate the training labels for the model as similar if a hero dies in the next five seconds or not.

The final data processing step before training was to normalise the dataset. To achieve this the minimum and maximum value for all features that required normalisation, i.e. scalar values with a maximum values > 1 or minimum value < 0, was calculated. No features had ranges below 1. The features were re-scaled using the MinMaxEncoder in Eq. (1) where x is the feature to be normalised, and $\min_{in}$ and $\max_{in}$ are the smallest and largest observed value for that feature. Note, $\min_{in}$ and $\max_{in}$ are only calculated on the training dataset to avoid the possibility of leaking feature size information from the test set to the training set. This step is very important as the size of features has a direct impact on the importance in training the network, especially at the start of the training process. Without normalisation features which naturally have very large values, e.g. the amount of gold that a hero has, may overshadow features with a smaller range such as features with a binary range.

$$x' = \frac{x - \min_{in}}{(\max_{in} - \min_{in})} \tag{1}$$

Once processed and normalised each game file is stored as a Hierarchical Data Format (HDF) file, which is used directly for training.

4. Methods

4.1. The time to die model

This work builds upon Katona et al. (2019) and as such we begin with the architecture proposed in that work. The 'Time to Die' (TtD) model is a deep neural network which is structured as a two-stage system. The first portion of the network acts as a feature extractor for each hero (illustrated as green and orange blocks in Fig. 2). A set of fully connected layers are used for each of the ten heroes, which are each represented as a vector of features, and the resultant latent vectors are then concatenated to form the input for the second half of the network. The layers in this first part share weights between heroes, allowing the model to learn these weights more quickly, without positional bias, and in a manner which allows all heroes to be encoded in the same way. The second half of the network (illustrated as red blocks in Fig. 2) takes this latent feature vector and passes it through several fully connected layers before outputting 10 values between

[2] https://docs.opendota.com/

[3] A patch is an update to the game. It is released to fix bugs and add new features. When analysing data, it is imperative to collect data from a single release (patch) to ensure consistency and compatibility of data.

[4] https://github.com/skadistats/clarity

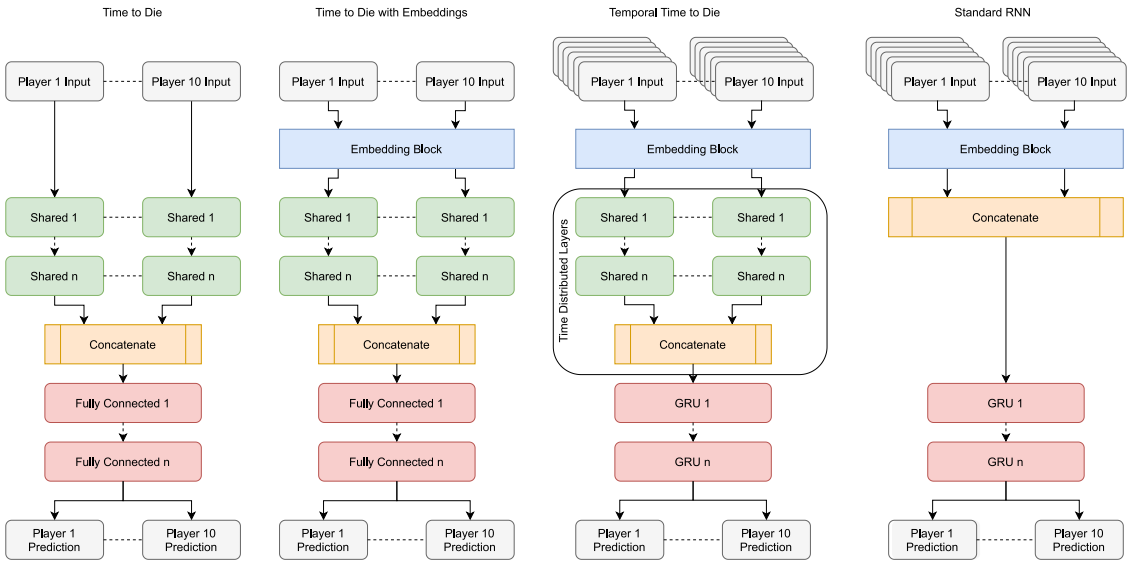


Fig. 2. Architectures for each model presented in the paper. First, the standard Time to Die model used in TtD_A and TtD_B. Second, incorporating embeddings into the model used in TtD_+ and TtD_2 (for an explanation of the embedding block see Fig. 3). Third, the Temporal Time to Die model TtD_{temp}. Fourth, a temporal model which just uses RNN layers called $Standard\,RNN$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

[0.0, 1.0] which represent the 'probability' that each hero dies in the next five seconds. Conceptually, the first portion of the network is tasked with understanding the state of a hero agnostic of the other players, and the second part of the network utilises these representations to model the way in which players may interact with each other. Fig. 2 contains a visual representation of this model, and we refer the reader to the original paper for more details. While the basis of the models presented in this paper are based on the Time to Die architecture several modifications and engineering techniques, discussed in detail below, are employed to improve performance.

4.2. Hero and item embeddings

In Katona et al. (2019) hero IDs, a unique ID assigned to each playable character in the game, were encoded using a one-hot vector. In general one-hot encoding suffers performance issues because they are weight and inference inefficient. The weight matrix for a fully connected one hot encoding has size $o * n + b$ where o, is the size of the encoding, n is the number of weights in the next layer, and b is the number of bias units used, traditionally $b = n$. Additionally all of these weight must be calculated for each pass through the layers, both during training and inference.

The original paper also used only a subset of 17 items, rather than modelling all possible items in the game. These items were hand-selected under the assumption they are the most likely to impact the survival chances of a player. However, at the time of writing, there are over 200 items in *Dota 2* and thus the original feature set used less than 10% of possible items. Therefore, it is very likely that modelling all of the items in the game will aid performance. However, modelling all of the items that a player owns using one-hot encoding is likely to inflate the number of weights needed to an unmanageable number, especially when considering that each hero can hold up to 18 items at the same time.

Handling of both item IDs and hero IDs can be improved by using embeddings, a technique developed initially for natural language processing applications but now utilised often for categorical features. Embeddings replace the one-hot encoding with a weight matrix of size $c * e$ where c is the number of categories, determined by the dataset, and e is the size of the embedding, a tunable hyper-parameter. This weight matrix functions as a lookup table. Rather than encoding a categorical variable like hero ID with a one-hot encoding instead the raw categorical value, in this case represented as an integer, is used as an input and then used to index into the weight matrix and select the row of length e which corresponds to that categorical value. The weight matrix between the embedding and the next layer is calculated as $e * n$, rather than the $o * n$ matrix necessary for one-hot encoding. Additionally, only the embedding relating to the input category is used during the forward pass. This has the effect of, in general, reducing the number of parameters. For example, consider hero IDs, a one-hot encoding of $o = 150$ and a first layer of $n = 1024$, each with a bias unit, results in a weight matrix between these layers of 154,624 weights. However, replacing the one hot encoding with an embeddings of $c = 150$ and $e = 24$, a value which performed well during hyper-parameter search, results in an embedding with 3600 weights and a weight matrix to the first layer of 25,600 weights, for 29,200 total weights, although not all are used at each forward pass. This dramatically reduces both the total number of parameters as well as the number of calculations required for each forward pass. Note that there are some hero ID values and many item ID values which are not in use, for example, the highest item ID values is approaching 1500 but there are only 200 items. This is due to heroes and items within the game changing as the game evolved and old IDs no longer being used. Because embeddings only use the selected weights during each pass there is no performance penalty to using oversized embeddings matrices where some elements are not used.

Another benefit of embeddings is that adding extra embeddings after a model has been trained without retraining the entire network is possible by adding extra rows to the embedding weight matrix, which is not the case with a one hot encoding. This is extremely beneficial for our model because *Dota 2* is constantly evolving, including adding new heroes. In theory, this allows us to add a new hero when one is released, and update its embedding while retaining the performance of the rest of the model, rather than needing to retrain the model. Finally embeddings naturally find an embedding space. We speculate that this embedding space can be queried via post-hoc analysis to uncover insights into heroes and items, at least pertaining to survivability, that are previously unknown.

In the models which use embeddings, we use one for hero IDs and one for item IDs. The hero ID embedding takes a single hero idea feature, whereas the item embeddings take 18 item IDs. Each hero can possess up to 18 items so we need to allow for this in the embeddings. Once the embedding has been calculated for both the hero ID and set

Table 1
The different models use different feature sets. This table breaks the features into groups and Table 3 lists which groups of features the different models use.

Original Time to Die features	Embeddings	Status effect
TeamFightParticipation, CreepsStacked	heroID	wardsDestroyed, isBroken, direBotTowersDestroyed
Ability Level, Ability Cast Range	itemID	isDisarmed, isHexed, direMidTowersDestroyed,
TowerKills, RoshanKills		isMagicImmune, isMuted
Ability Mana Cost Ability Cooldown		isSilenced, isSmoked, direBotRangeRaxDestroyed
Ability Activated, Ability ToggleState		isStunned, ultimateRead, direTopMeleeRaxDestroyed
Item Cooldown, Stuns		supportGoldSpent, heroDamage, direBotMeleeRaxDestroyed
TauntCooldown, BKBChargesUsed		buybackCost, buybackCooldown, direTopRangeRaxDestroyed
AbilityPoints, PrimaryAttribute		radiantTopTowersDestroyed, radiantMidTowersDestroyed
		radiantBotTowersDestroyed, radiantTopRangeRaxDestroyed
		radiantTopMeleeRaxDestroyed, radiantMidRangeRaxDestroyed
		radiantMidMeleeRaxDestroyed, radiantBotRangeRaxDestroyed
		radiantBotMeleeRaxDestroyed, direTopTowersDestroyed,
		direMidRangeRaxDestroyed, direMidMeleeRaxDestroyed.

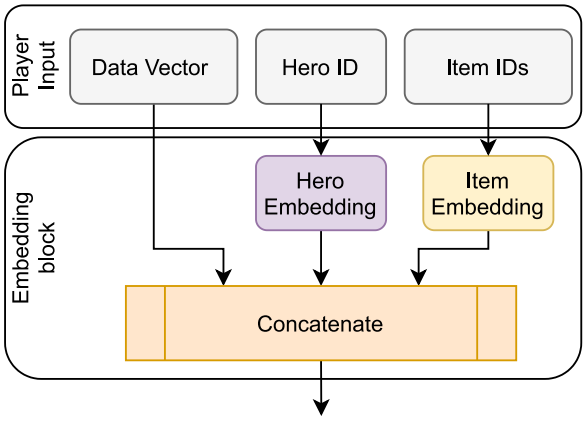


Fig. 3. Embedding block for Time to Die model.

of items IDs they are concatenated along side the other features for a heroes to form the input vector for the shared portion of the network. Fig. 3 demonstrates this process.

4.3. Feature set

We know from Katona et al. (2019) that models that are provided with more features about the game are more likely to perform well. As a result we initially take a maximalist approach to feature selection, and provide the models with as many features as possible. We consider our input data to be a set of 10 feature vectors, each of which relates to a single hero. Each of these feature vectors is constructed from features which describe the current state of the hero at a given time stamp. Table 1 shows the features the model can be trained with, split into categories of broadly similar features: features from the original Time to Die model, embeddings features and status effect features.

While we pose this work as a continuation of Katona et al. (2019) it should be noted that not all features from the original paper are represented like for like in this paper. This is due to the nature of our new parsing process. However, having consulted with an expert player we believe that these features have minimal effect on the predictive capability of the model, because they generally track historic stats, e.g. how many wards have been placed, rather than the current state of the hero. Our 'full' dataset is listed in Table 2 with the features grouped by which part of the game they describe. Our 'full' dataset contains a set of features not in the original paper. These largely pertain to status effect, e.g. if the player is stunned, i.e. they are currently unable to move or act. Intuitively, these status effects should be very useful when determining if a player is likely to die or not because they describe player state in a way not previously modelled. Our feature set, alongside our embeddings, also allows us to model ownership of every item in the game, rather than the small number of items modelled in the original Time to Die model. The features in the original paper and not present in ours, as well as the features not in the original paper are detailed in Table 1.

This work initially experiments with three different feature sets, firstly one which mirrors the original Time to Die model (Katona et al., 2019) as closely as possible. This feature set is used for both the 'TtD_A' and 'TtD_B' models. Secondly, a feature set that uses broadly the same features as the original paper but formats certain features so that they can be used with an embedding layer, namely hero IDs and item IDs, used for 'TtD_+'. This formatting for the embedding layer additionally allows models trained with 'TtD_+' to model all items rather than the small selection in the original paper. Finally, a feature set which combines the modifications utilised in 'TtD_+' alongside extra features such as status effect, used for our 'TtD_2' model. Table 3 provides a comparison of these four models. In Section 4.5 we introduce a fifth model designated 'TtD_{Temp}' which revises the TtD_2 architecture by replacing the final set of fully connected layers in TtD_2 with recurrent layers. The input to 'TtD_{Temp}' is a stack of data with the TtD_2 feature set but taken from the previous n game time steps. Table 3 provides an overview of the different models.

4.4. Hyper-parameter selection

Naturally deep learning models are extremely sensitive to hyper-parameters. As a result, extensive hyper-parameters tuning was carried out across all models. There are a number of approaches for hyper-parameter optimisation, such as: random search (Bergstra & Bengio, 2012), Tree Parzen Estimator (TPE) (Bergstra, Yamins, & Cox, 2013) and Adaptive TPE (ATPE) (Wen, Ye, & Gao, 2020). Following on from Katona et al. (2019), we carried out a random search based hyper-parameter fitting process (Bergstra & Bengio, 2012). Search was carried out over several generations of randomly selected hyper parameters, with the search space updated after each iteration to narrow upon a high-quality set of parameters. This was carried out independently for each model, the original Time to Die model, TtD_+, and TtD_2. However we found that in reality, because most models are architecturally similar, they converged to a similar set of hyper-parameters. The final stage of the hyper-parameter selection process was to A–B test slight modifications to the selected parameters where there were two models which disagreed, e.g. two models had independently found slightly different learning rates. In this case both models were tested with both hyper-parameter options and the best one selected. Perhaps surprisingly this resulted in a relatively uniform hyper-parameter set across all models. Because these hyper-parameters are very different from the original hyper-parameter set we experiment with two base Time to Die models, TtD_A which used the original hyper-parameters, and TtD_B with our new parameters. These hyper-parameters are detailed in Table 4.

Table 2
Complete feature set extracted from the data for each hero on both teams that are presented as inputs to the models. The features are grouped into broad categories relating to which parts of the game they describe. Features highlighted in italics are calculated during the processing stage and are not native to the Dota 2 telemetry data. Features highlighted in bold are represented as embeddings for the models which feature embedding layers (i.e. 'TtD_+' and 'TtD_2') and as one hot encoded vectors otherwise.

Hero statistics	Hero status effects	Hero state	Positional data	Tower data	Ability data	Item data
Experience	Is Broken	Health	Hero Position X	*Nearest Ally Tower Proximity*	Level 1–7	Item ID 1-18
Level	Is Disarmed	Max Health	*Hero Position X Delta*	*Nearest Ally Tower Proximity Delta*	Cooldown 1–7	Item Cooldown 1–18
Number of Last Hits	Is Hexed	Health Regen	Hero Position Y	Nearest Ally Tower Health	Mana Cost 1–7	Blink Dagger Owned
Deny	Is Magic Immune	Mana	*Hero Position Y Delta*	*Nearest Enemy Tower Proximity*	Activated 1–7	*Black King Bar Owned*
Net Worth	Is Muted	Max Mana	*Ally Proximity 1–4*	*Nearest Enemy Tower Proximity Delta*		*Magic Wand Owned*
Gold	Is Silenced	Mana Regen	*Ally Proximity Delta 1–4*	Nearest Enemy Tower Health		*Quelling Blade Owned*
Kills	Is Smoked	Hero Damage	*Ally Proximity 1-5*			*Power Treads Owned*
Deaths	Is Stunned	Damage Min	*Ally Proximity Delta 1-5*			*Hand of Midas Owned*
Assists		Damage Max				*Hurricane Pike Owned*
Wards Placed		Damage Bonus				*Force Staff Owned*
Wards Destroyed		Base Strength				*Abyssal Blade Owned*
Runes Activated		Base Agility				*Mask of Madness Owned*
Camps Stacked		Base Intelligence				*Nullifier Owned*
Support Gold Spent		Total Strength				*Travel Boots Owned*
Ultimate Ready		Total Agility				*Dagon 5 Owned*
Is Radiant		Total Intelligence				*Lotus Orb Owned*
Life State		Armour				*TP Scroll Owned*
Hero ID		Magic Resistance				Smoke of Deceit Owned
		Movement Speed				Clarity Owned
		Visible by Other Team 1–10				

Table 3
Comparison of functionality and features of each model. The hyper-parameters are detailed in Table 4. The features groups are listed in Table 1. We introduce TtD_{temp} in Section 4.5.

Model	Hyper-parameters	Original	Embeddings	Status effect	Temporal
TtD_A	From Katona et al. (2019)	Yes	No	No	No
TtD_B	Fit in this work	Yes	No	No	No
TtD_+	Fit in this work	Yes	Yes	No	No
TtD_2	Fit in this work	Yes	Yes	Yes	No
TtD_{temp}	Fit in this work	Yes	Yes	Yes	Yes

4.5. Recurrency

Finally, it is not clear if the Markov property, i.e. future states depends only on the current state and not past states, holds for our data. The previous Time to Die paper made this assumption (Katona et al., 2019) and the previous models presented in this paper have operated under the assumption. Certainly, there are a set of features, e.g. historic visibility and distance deltas, which allow a single sample to detail the short term history of the game, at least to some extent. However, this may not be sufficient to satisfy this assumption.

There may be historic features or information, e.g. movement patterns, which are not modelled in our single-frame data. Therefore we implement a fifth model designated 'TtD_{Temp}' which is a new version of the TtD_2 architecture. 'TtD_{Temp}' replaces the final set of fully connected layers in TtD_2, after concatenation, with a set of layers containing Gated Recurrent Unit (GRU) (Cho et al., 2014) neurons. GRU neurons, rather than the more popular Long Short-Term Memory neurons, are used because they are significantly simpler and have fewer weights, thus will be able to perform inference faster which is vital in a live broadcast setting. This new temporal model then receives as input a stack of data taken from the previous n game time steps. If the Markov property is not sufficiently satisfied with our single frame models then we would expect a temporal model to perform well in comparison.

At this stage it is also useful to compare our TtD_{Temp} architecture with a model, designated 'Standard RNN', which only features the recurrent layers and does not feature the core two stage approach of the Time to Die models. To achieve this the raw input features are concatenated with hero and item embeddings for the 10 players and then this single feature vector is passed to several GRU layers. The architectures of these two approaches are visually demonstrated in Fig. 2 and the hyper-parameters are show in Table 5. Interestingly, while there are fewer neurons in the 'Standard RNN' model it has many more total weights. This is a result of the fact that the entirety of the input is concatenated before the first layer and thus there are many more connections, and therefore weights, between input and the first layer of neurons.

5. Experiments

In total we present three experiments which show the impact that embeddings, status effects and using a recurrent architecture have on in-game death prediction.

5.1. Experiment 1 — Model comparisons

We train a set of models with different characteristics and then compare their performance. To evaluate the models four metrics are employed. **Precision** which is the fraction of deaths detected by the model from all positive predictions made, and **Recall** which is the fraction of deaths which were correctly detected. Additionally, as in Katona et al. (2019) the **Average Precision** (AP) is reported. This is the area under the Precision-Recall curve and represents the average precision value across all possible recall values. AP is often used in imbalanced detection tasks because it does not require a threshold decision boundary to be set, although in practice often this choice need to be made before a model can be deployed and thus AP can make models appear more performant than they will be in a operational setting. Finally, **F1-Score** ($F1$), the harmonic average between Precision and Recall, is reported. While precision is naturally important — the system would fail in its purpose if it often predicted deaths that did not occur, recall is also extremely important because one of the motivating reasons for exploring this type of prediction is that broadcasters often miss events due to the fast-paced and complex nature of the game. A model with low recall would also miss important moments, negating the usefulness of the model. In fact, while AP was the focus of our previous study upon reflection $F1$ is likely be more useful as both precision and recall are equally important, especially given the focus in this work on the suitability of such a model to be operationalised. Note, accuracy is not reported because our test sets are imbalanced, there are many more negative samples than positive ones, and as such reporting accuracy across both of these labels is likely to be misleading.

Table 4
Hyper-parameters for original model (TtD_A), original model with new hyper-parameters (TtD_B), a model that uses embeddings (TtD_+), and the model which uses both embeddings and status effects (TtD_2).

Hyperparameter	TtD_A	TtD_B	TtD_+	TtD_2
Hero Embedding	–	–	{150, 24}	{150, 24}
Item Embedding	–	–	{1500, 8}	{1500, 8}
Shared Layers Dims	{256, 128, 64}	{512, 256, 128}	{512, 256, 128}	{512, 256, 128}
Joined Layers Dims	{1024, 512, 256, 128, 64, 32}	{512, 256, 128}	{512, 256, 128}	{512, 256, 128}
Numb. Parameters	1,458,474	1,115,658	1,156,346	1,162,490
Learning Rate	6.15e−5	2.89e−05	2.89e−05	2.89e−05
Batch Size	128	1024	1024	1024
Early Stopping Patience	–	13	9	10

Table 5
Hyper-parameters for recurrent architectures.

Hyperparameter	TtD_{Temp}	Standard RNN
Hero Embedding	{150, 24}	{150, 24}
Item Embedding	{1500, 16}	{1500, 8}
Shared Layers Type	MLP	–
Shared Layers Dims	{512, 256, 128}	–
Joined Layers Dims	{512, 256, 128}	{256, 128, 64}
Numb. Parameters	3,837,562	6,369,018
Learning Rate	5.35e−05	2.89e−05
Batch Size	1024	1024
Early Stopping Patience	12	10

Table 6
Results for Experiment 1. Precision, Recall and F1 Score used a out threshold of $>= 0.5$ to indicate positive class labels.

Model	Precision	Recall	AP	F1 Score
TtD_A	0.09	0.87	0.46	0.17
TtD_B	0.23	**0.96**	0.75	0.37
TtD_+	0.34	0.93	**0.76**	0.50
TtD_2	**0.38**	0.90	0.71	**0.54**

5.1.1. Experimental results

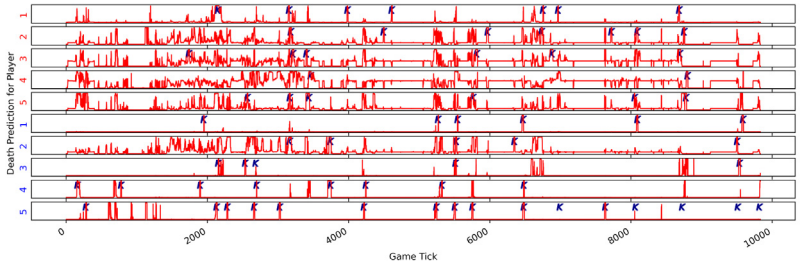
Table 6 shows the Precision, Recall, Average Precision (AP), and F1 Score for the four models experimented with. All models show very high recall and much worse precision. This finding suggests that, regardless of model, it is often very clear when a player is at risk of dying, hence the high recall but there are many times where the models predict that a player is going to die, but they are actually able to escapes, hence the general lack of precision. It is also clear from looking at the results that the original Time to Die model (TtD_A) performs very poorly. In fact, a simple re-tuning of hyper-parameters (TtD_B), which produced a network with a very different architecture, resulting in a large improvement in both AP and F1 score. When discussing the original model, it is important to note that we found an AP of 0.46 compared to 0.54 as reported in the original paper. We expect this variation is due to using a new dataset which contains data from only professional games, uses slightly different features, and was gathered using a different patch. Different patches alter various features of the game which may make prediction easier or harder, and past research has shown that it is more difficult to build predictive models for high level play because player skill begins to have a larger impact on the outcome of an event (Semenov et al., 2016).

As well as the re-tuned parameters we see that the use of embeddings, present in 'TtD_+' and 'TtD_2' also has a huge impact on performance, at least when considering F1 Score, with scores of 0.50 for 'TtD_+' and 0.54 for 'TtD_2', compared to 0.37 for 'TtD_B'. There is also a large impact to Precision, where embedding models scored 0.34 and 0.38 compared to 0.23 for the best performing non-embedding model. It is likely that modelling hero IDs in a smart way, as well as modelling all items in the game, allows the model to better understand when normally risky situations are not actually risky, e.g. due to an ability or item a player has. Finally, the additional features, such as status effect, provide a small but noticeable improvement to Precision and F1 Score.

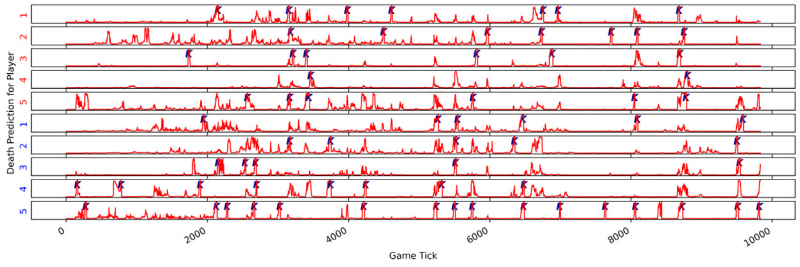
There is a clear improvement in terms of F1 score as the four models add extra functionality. However, other than the original Time to Die model (TtD_A), Average Precision (AP) scores are similar across models, with our most complex model, TtD_2, performing slightly worse than the other two models, 0.71 compared to 0.75 and 0.76. The TtD_2 model also scarifies some Recall, scoring 0.9 compared to TtD_B's 0.96 and TtD_+'s 0.93. It is possible that these results signify a model which is more cautious and less likely to predict a death, hence the drop in Recall but improvements in Precision and F1 Score. As discussed before, F1 Score is likely the best metric for evaluating models for our use case and thus it seems that the TtD_2 model is most suitable for deployment.

Fig. 4 shows the death predictions for all four models on a randomly selected game, ID 5510998077. For each player in the game a prediction for each game time step is displayed, along side the time steps where a kill occurred, marked with a K. From this plot we can see that the two models which do not use embeddings (TtD_A and TtD_B) show significantly more 'noise', i.e. propensity to predict non-zero values in seemingly safe conditions. In fact, looking at the average prediction for all players TtD_A predicted an average chance of death at 0.12 (i.e. 12%), compared to 0.06 for TtD_B, 0.04 for TtD_+ and just 0.03 for TtD_2. Furthermore, there appears to be a lot of variance between how likely a player is to die based only on their position in the input data vector, for instance all players on the Radiant (red) side are often predicted as being in a risky situation, whereas for many players on the Dire (blue) side the model is much less likely to erroneously predict risk. Furthermore, for Dire player 5 (the last row) we see 4 situations where the TtD_A model completely missed their death. This sort of predictive failure is catastrophic for our system, as while risky situations which do not result in a death provide an interesting discussion point for commentary, missing a death results in no opportunity for the commentary to discuss the situation.

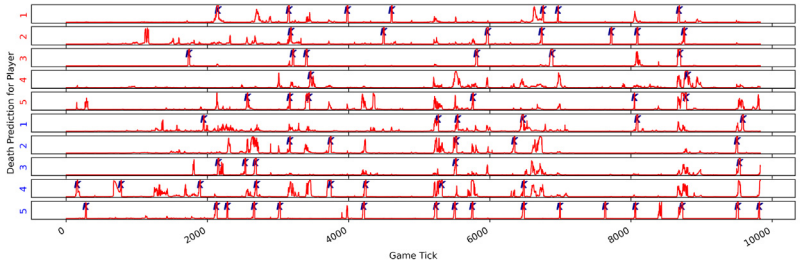
Finally, Fig. 5 shows the distribution of the average predicted probability of death for the model based on how long until the player dies, binned into 1 seconds time windows. As expected, the average output as a player nears death tends towards 1 (i.e. certain death). The most interest aspect of these plots is to observe how the distributions changes over time. For example, the TtD_A model tends to be either certain a player is going to die, or certain it they will survive. Machine learning models are often trained with binary $[0, 1]$ training labels but are able to output scalar values which are interpreted as a probability that the sample is of the positive class, i.e. has the label 1. However, the distributions of outputs suggests that using TtD_A outputs as probabilities are not likely to be interesting because it lacks graduation between its extreme outputs i.e. 0 or 1. This is not the case for the other 3 models where we see a much more gradated change in probabilities as we move further from the death. Interestingly, our TtD_2 model, while trained to predict death in less than 5 seconds has a median output of 0.5 (the decision threshold) at four seconds, whereas TtD_B and TtD_+ both had a medium output of 0.5 somewhere between 4 and 5 seconds. This is further evidence that the TtD_2 model is slightly more cautious, it appears at least for deaths this cautiousness occurs when deaths occur further in time. These values are only gathered in the 20 seconds before a player dies, and therefore the distributions of these values do not



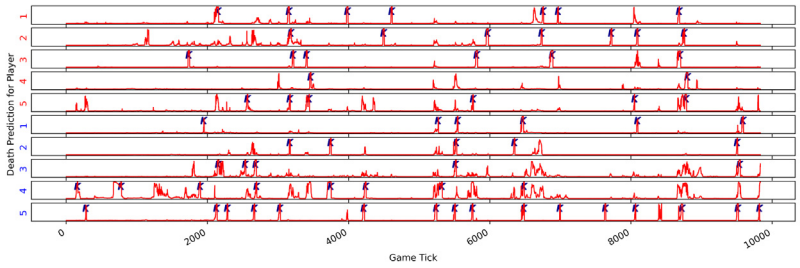
(a) TtD_A



(b) TtD_B



(c) TtD_+



(d) TtD_2

Fig. 4. Model outputs for each of the core models across the course of a randomly selected game. For each figure, the 10 time series plots represent the output probability for each player. Time steps (ticks) labelled with a blue 'k' are when a player died. TtD_A, i.e. the original model, is much more likely to output a high probably of dying across this game, and seemingly does so especially often for player on the red 'radiant' team. By contrast our best performing TtD_2 model is much less likely to output high values unless a death actually occurs within 5 seconds. While there are still regions of high probability which ultimately do not result in a death this are more likely to correspond to a time where another player from the opposing team dies and correspond to battles between the players. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

describe the reasons for false positives predictions. As a result, while TtD_B may appear as if it performs the best, because it recognises danger for these true positive cases sooner, this is because in general the model is more likely to predict that a player is going to die, which, while not represented in these plots, is the reason for worse precision for that model. The key takeaway from these plots is that, excluding TtD_A, models are trained only on labels relating to death in five seconds or less but we actually see that they tend to learn a gradual sloped output and thus are capable of early warning of a death before five seconds. This gradual slope is likely to be very useful to broadcaster as they are able to use it detect developing situations and thus focus on where the action is likely to happen.

5.2. Experiment 2 — Feature importance ablation

The models presented in Experiment 1 use large feature sets. However, there are likely certain features which have significantly more impact on performance than others. Furthermore, a model with fewer inputs has the advantage that it reduces the number of model parameters and thus is likely to speed up inference, which is important

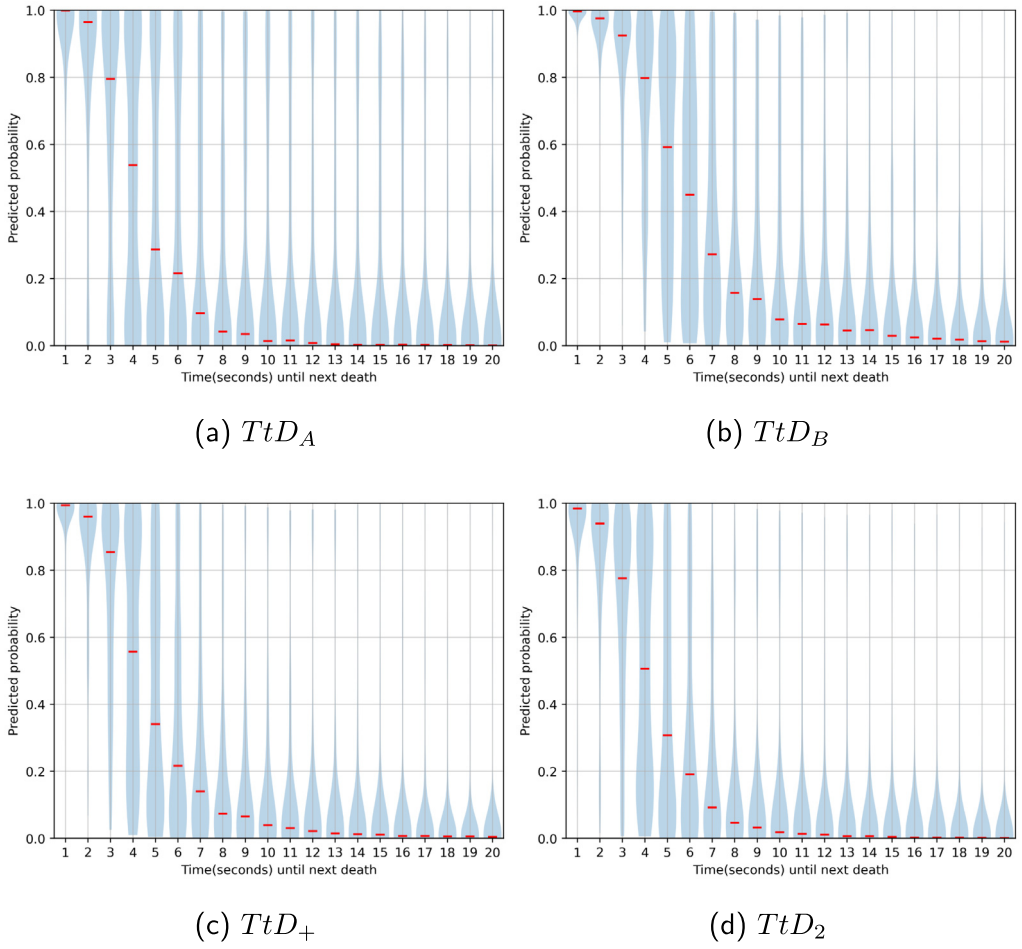


Fig. 5. Violin plots describing the distribution of the average predicted probability of death at different timesteps before a death occurs.

if a Time to Die model is to be deployed in a live system. Finally, explainable artificial intelligence is becoming increasingly important, especially for black-box models such as neural networks. Performing a study on which features are most important to the model also offers insights into why the model is making the decisions that it is. To understand which features are the most important we perform a random permutation feature importance test and then train several models which use only the most important features. These models are then compared to the TtD_2 from Experiment 1 to explore if a model with fewer features has acceptable performance.

To perform the feature importance calculation the testing process is carried out once for each feature, but that feature is randomly shuffled. This provides test set samples which have the correct values for all features bar one. Test sets in which the test error is very large suggest that the feature which has been shuffled is very important to prediction, and likewise low error indicated features that are unimportant to the model. For this experiment the features which use embeddings, i.e. hero ID and item ID, are not included. This is because it is no possible to shuffle these features, because the hero ID which is constant throughout the game and item IDs often do not change once an item has been purchased.

A second feature ablation study is carried out to explore how important the embeddings are to the models performance. To achieve this two additional models are trained based off the TtD_2 model. Both of these models are TtD_2 models but one omits the hero embedding feature and the other omits the item embeddings. Therefore, the first model is trained with no knowledge of which hero is being used and the second is trained with no knowledge of which items are owned.

5.2.1. Experimental results

The first step for the Feature Importance Ablation study was to perform the feature importance test detailed above, the results of which are shown in Fig. 6. From this we can see that if the player is alive and how much health they have are both very important to the likelihood that a player is going to die. This is unsurprising given that if a player is alive or not likely acts as a very simple filter to filter out situations where the player is going to die soon — they cannot die if they are already dead. Health is also a very clear indicator as running out of health is the in-game mechanism which causes death. Surprisingly, the number of deaths a player has is also fairly important, it may because a high number of deaths indicates that a player is not doing well in a game and so is more likely to perform poorly in the future, especially considering that MOBA games often punish players who die with a loss of gold and reward those who have killed other players with experience.

The next most important features are visibility features and proximity to allied players. Visibility is very important because the game has a 'fog-of-war' feature where it is possible for players to avoid dangerous situation by hiding in parts of the map where the other team cannot see them. However, if the other team knows that the player is likely to be in a part of the map because that player have been seen recently the opportunity for hiding and escaping is reduced. Additionally, being near allied players often provides a lot of protection because it makes it more risky for the opposing team to engage in a fight.

The next step of the feature importance ablation experiment is to train a set of models using just the most important features to test if it is possible to build a model with fewer features that still performs well. Three feature sets are used for this, one with a single feature, one with

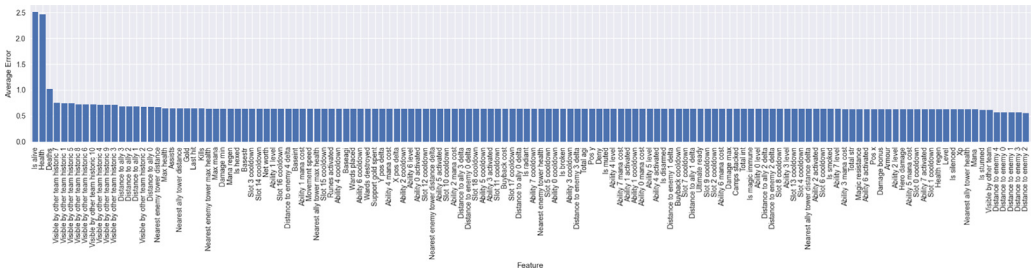


Fig. 6. Feature Importance Test results ordered by average error when the feature was shuffled. Larger error suggests the feature is more important to the model.

Table 7
Results for Experiment 2.

Feature set	Precision	Recall	AP	F1 Score
One Feature	0.08	0.89	0.12	0.14
Three Features	0.20	0.91	0.72	0.32
18 Features	0.11	0.96	0.71	0.19
No Hero ID	0.25	0.95	0.74	0.40
No Item IDs	0.26	0.95	0.74	0.41
Full Feature Set	0.38	0.90	0.71	0.54

Table 8
Results for Experiment 3.

Model	Timesteps	Precision	Recall	AP	F1 Score
TtD_{Temp}	3	0.19	**0.87**	0.60	0.32
TtD_{Temp}	15	**0.52**	0.77	**0.67**	**0.62**
TtD_{Temp}	30	0.30	**0.87**	0.66	0.45
Standard RNN	3	0.09	0.84	0.28	0.16
Standard RNN	15	0.06	0.42	0.10	0.10
Standard RNN	30	0.0	0.0	0.01	0.0

three features, and one with 18 features. For the single feature model we select health rather than if the player is alive because if the player is alive or not only has predictive power after the player has died, which is not particularly useful as a broadcast aid. The three feature set uses both the players health and alive state as well as the number of deaths that player has. The 18 feature set adds allied proximity and enemy visibility features. Models that omited hero IDs as well as item IDs are also presented.

Table 7 shows the core findings for this experiments. Clearly, removing features has a significant negative impact of performance. This is not entirely unexpected especially given that our original work came to similar conclusions. Models with very few features often retain high recall, but lack precision. This is likely because certain features, e.g. health, are really strong indicators of if a player is unlikely to die, i.e. they have high health as so are unlikely to die, but are really poor indicators of when a player may escape a seemingly risky situation. Imagine two situations, one where the player has used all of their abilities and possesses no relevant items, and another where the player has the ability to dash or teleport away from a situation and additionally has an item which reduces the amount of damage they take. Clearly the hero in the first situation is more likely to die even if both heroes have the same amount of health. This anecdotal situation is reflected in the poor performance of models with fewer features. The main potential benefit of using fewer features is to reduce the amount of computation needed and thus enable a model to be deployed more easily. However, it seems that the performance of these small feature sets is such that they would not be suitable even if they are more easily deployed.

With regards to ablating features which utilise embeddings, i.e. hero IDs and item IDs, we see that both are fairly important to predictive power of the model, evidenced by the large drop in F1 score when removing them. Interestingly we see slightly higher AP and Recall, at the expense of Precision given a fixed decision boundary. This is likely a reflection of models which learn to be more 'pessimistic' about the chances a player is going to die, i.e. the model is more likely to predict that a player is going to die than when including these features. It is likely this is because both hero IDs and items often describe abilities or features that increase the survival chances for a player i.e. by giving them a shield or the ability to dash out of danger. We also see that both hero IDs and item IDs have a similar impact on the performance of the model, showing that item IDs are important, despite not being fully represented in our original work.

5.3. Experiment 3 — Recurrency

The final experiment is to compare our models using temporal data and recurrent GRU layers with both the best performing single frame model as well as the standard RNN model. It is not initially clear what the correct number of time steps to use is. Therefore, three models are experimented with for both recurrent models, one that looks at the past three times steps, i.e. one second of game play, one that looks at 15 time steps, i.e. five seconds, and one that looks at 30 time steps, i.e. 10 seconds.

5.3.1. Experimental results

The key results from the recurrency experiment are shown in Table 8, once again detailing Precision, Recall, AP, and F1 Score for each model. These results are very surprising. These results show how useful the Time to Die architecture is compared to a standard RNN network, given that these networks struggled to learn effectively, whereas each of the Time to Die architectures performed better. Perhaps the most interesting discovery from this experiment is that with the Time to Die architectures there seems to be a 'sweet-spot' in terms of the number of time steps compared to performance. A TtD_{Temp} model with 15 time steps, equating to 5 seconds, performed extremely well compared to the other two time step hyper-parameters. It may be that this sweet-spot is the result of two effects. Firstly, a small number of time steps may not aid predictive power too much, given that some features relating to previous frames, e.g. movement deltas and visibility information, are already contained within the feature set. However, by using GRU neurons the model is significantly more complex, thus leading to a model which is more difficult to train and without the benefit of a lot of hindsight information. Secondly, it would be expected that at some point extra information from past frames is going to be detrimental to the models performance. This is because *Dota 2* is a very fast paced game and players can move into and out of danger multiple times within 30 frames, i.e. 10 seconds, thus information from frames many timesteps ago may provide noise and thus impair performance.

The non-recurrent TtD_2 model achieved an AP of 0.71 and a F1 Score of 0.54. Our best recurrent model, the TtD_{Temp} model with 15 times-steps, achieved an AP of 0.67 and a F1 Score of 0.62. This suggests that the recurrent model is more likely to make false positive predictions across all recall values, i.e. a lower AP score, but has a better F1 Score at a threshold of 0.5, and is in fact less likely to make a false positive prediction given a fixed threshold value, i.e. higher precision. It does appear that because the model is more 'pessimistic'

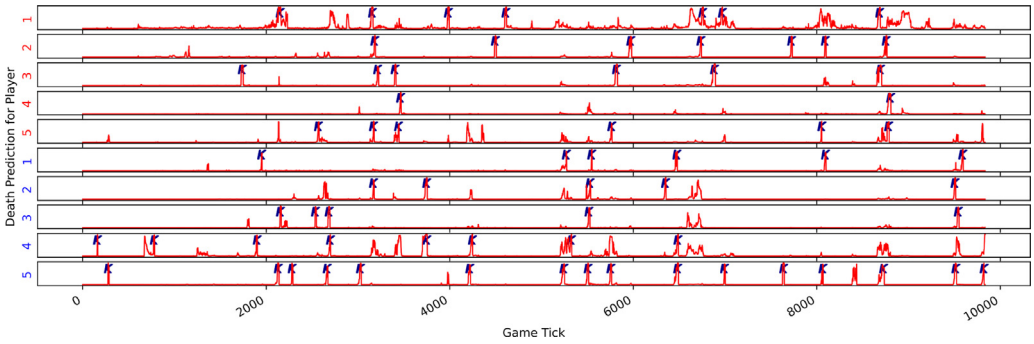


Fig. 7. Model outputs across time for a randomly selected game for the TtD_{Temp} model with 15 time steps (ticks).

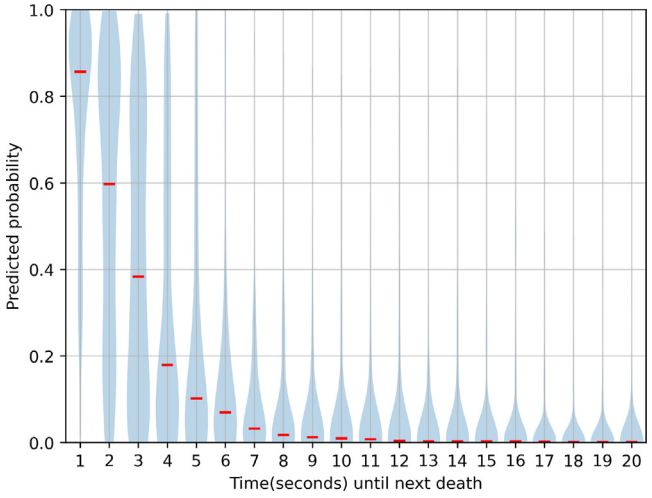


Fig. 8. Violin plot describing the distribution of the average predicted probability of death at different time steps before a death occurs for TtD_{Temp} model with 15 time steps.

with its positive predictions the model has a lower recall score than the non-temporal TtD_2 model. All of this results in a model which has a much better F1 score and is in general likely to be a more useful model for broadcast analysis. Fig. 8 shows the violin plot for the average predicted probability of death at different times to death for the TtD_{Temp} model trained with 15 time steps, mirroring Fig. 5 for the non-temporal models. Similarly, Fig. 7 shows the output of the model across a game, and is the equivalent of Fig. 4. These plots demonstrate the improved performance of the model, e.g. there are only limited situations where the model suggests someone is going to die and they do not, and for this model mis-classifications more often align with other players dying. This is unsurprising, when two heroes are fighting the likelihood that both are going to die increases because it is rarely clear exactly who will win a fight. That said, it does seem from Fig. 8 that the temporal model is less able predict player death many seconds in the future compared to non-temporal models, likely due to the fact it is less likely to predict deaths in general. It is not clear if, in practice, this would make the tool more difficult to use, i.e. because the broadcasters need more time to build up play, or if this look-ahead is sufficient.

While the TtD_{Temp} temporal model is better at prediction it has a computational overhead, having greater than 3 times more parameters than the TtD_2 model. This may make it impossible to deploy the temporal model unless the broadcast team has access to significant compute infrastructure. To test this we took a computer fitted with consumer-grade hardware and experimented with the time it takes both models to process a sample. The computer had a Ryzen 5 1600 CPU, 16 gb of RAM, and a Nvidia 2080ti GPU. The average time taken

Table 9
Inference time (in seconds) to process a single sample for both the non-temporal TtD_2 model and the temporal model TtD_{Temp} when processing the data using a GPU or CPU.

Model	GPU	CPU
TtD_2	9.39e−05	1.02e−03
TtD_{Temp}	3.48e−03	1.35e−02

to process a single sample, averaged from 256 samples, is shown in Table 9. From this we can see both that the non-temporal model is much faster and also that performing inference on the GPU is much faster than the CPU for both temporal and non-temporal models. Data is sampled at a rate of three times per second and thus a model would have to be able to perform inference at faster than 0.33 seconds per sample in order to run in real time. From Table 9 we see that all models would be able to operate in real time on either GPU or CPU with computation time to spare for e.g. data processing, when using consumer grade computing hardware. Therefore, it is highly likely that either the standard TtD_2 model or the TtD_{Temp} temporal model would be suitable for deployment in a broadcast setting with only minimal infrastructure investment.

6. Discussion

Generally, the proposed modifications to the Time to Die architecture show improvements compared to the original work (Katona et al., 2019). In particular utilising hero ID and item ID embeddings, optimising the hyper-parameters as well as changing the architecture to use recurrent layers had a large impact of predictive performance. We achieved an F1 score of 0.54 for our new model TtD_2 with embeddings, status effect features and optimised hyper-parameters compared to 0.17 for our previous model called TtD_A (when both models are trained and predict using the new data). We were able to improve this further by experimenting with the length of the time-series in the input data and found that generating the input vector using 15 time steps further improved the F1 score to 0.62 for the TtD_{Temp} model. This compares to F1 of 0.1 for a standard RNN on the same task.

That said, there are still examples where the model mis-classified a sample, see Fig. 4. It may be that there are factors not currently modelled by our data which explain the situations where the player escapes a risky situation. For example, our model is currently player agnostic, mainly due to the difficulty in gaining enough data about individual players to sufficiently model this. Alternatively, it could be that the model becomes more sure a player is going to die closer to their time of death and thus is more likely to make mistake when there is a larger amount of time between sample and death. This would not be particularly surprising given the complex, chaotic nature of *Dota 2*, nor is it necessarily unwanted given that the model outputs are intended to be utilised as probabilities that a player dies in a certain situation. Thus

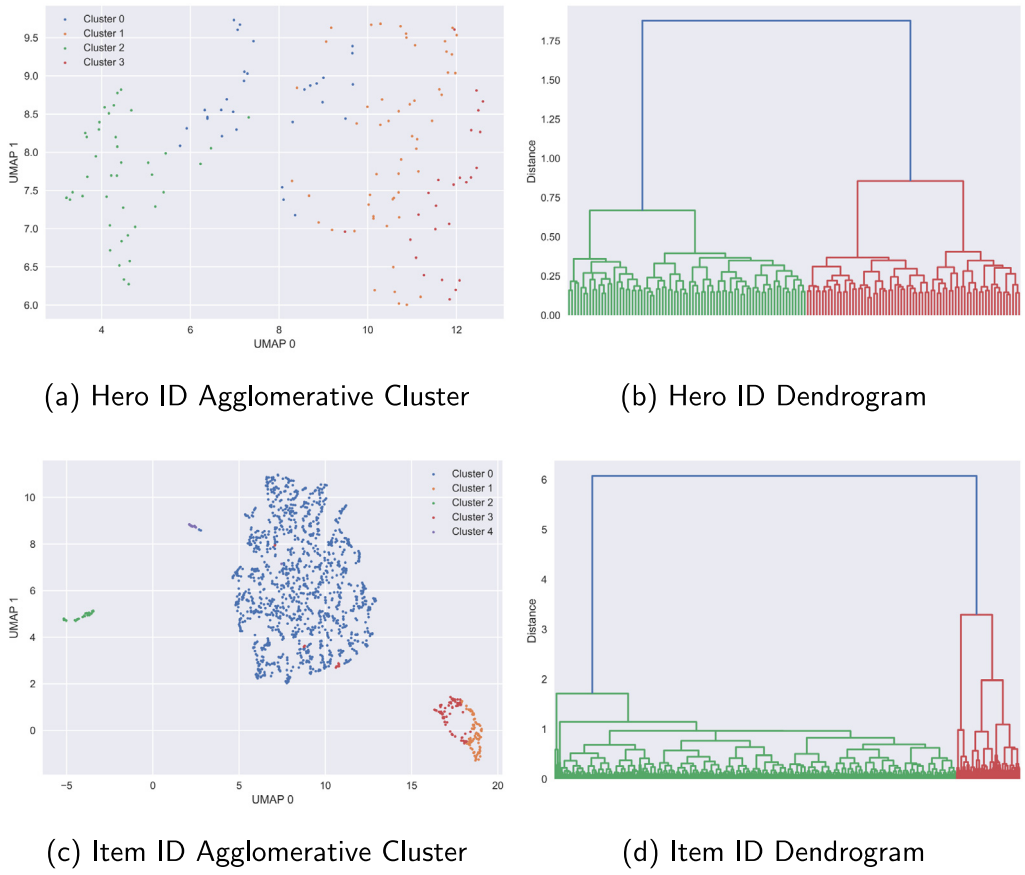


Fig. 9. UMAP Decomposition of clustered embedding spaces using agglomerative clustering and associated dendrogram for both hero and item embedding spaces (TtD_2 model).

probabilities greater than 0.5 which do not result in a death can still be useful because they highlight to the broadcast team that a player may have made a lucky or skilful escape. Further to this the model has the ability to begin recognising dangerous situations developing ahead of the five seconds training window, Fig. 5. This functionality may allow the model to aid in post match analysis, e.g. by searching for infliction points where the likelihood of a player dying suddenly increases, and using these to discuss the developing situation.

Fully understanding and analysing the learnt embedding spaces for hero IDs and item IDs is outside of the scope of this work. However, it is possible to perform some high level analysis to begin understanding these spaces and suggest future research questions. For example, both clustering an embedding space as well as decomposing the space into a visualisable number of dimensions, i.e. 2, may start to explain the learnt space. Fig. 9 shows agglomerative clustering applied to both of these embedding spaces as well as the corresponding dendrogram. This dendrogram is used to determine an appropriate distance metric, and thus the number of clusters. The embedding spaces have also been decomposed using UMAP (McInnes, Healy, & Melville, 2018). While UMAP cannot be used to make firm conclusions about high-dimensional spaces, because distances between points are not always faithfully preserved, it does aid in visualising the clusters. The dendrograms suggest that Hero IDs are best partitioned into four clusters while item IDs are best partitioned into five clusters. With regards to items there seems to be one large cluster, cluster 0 in the UMAP plot. This is like because there are many item IDs which are not used in the game in a meaningful way, there are nearly 1500 item ID embeddings but only about 200 usable items, and embeddings are initialised similarly. These IDs have not been trained and therefore occupy a similar space. It is highly likely that many of these embeddings are items IDs which are either unused or not used often, however it is not current possible to verify this given the available data on items.

There is some available information about heroes which detail various character statistics, e.g. how much damage they do for basic attacks, which may help describe the embedding spaces. In fact, if our embeddings spaces correlate to these features this adds further evidence that the embedding space is learning something meaningful. To observe this we can plot Kernel Density Estimation (KDE) for these features across the heroes in each cluster and observe the differences in the distributions. Fig. 10 shows the distribution of a selection of these statistics across the four hero clusters. Several things can be said about these clusters. For example, Cluster 2 features heroes which tend to be Intelligence focused with low base attack damage and low strength. These heroes are possibly casters who focus on activated abilities rather than basic attacks. Conversely, Cluster 3 has heroes who have higher than average maximum base attack damage, higher base strength but lower base intelligence. It may be that heroes in this cluster are base attack focused damage dealers. This analysis is deliberately superficial because there are limitations to these methods, e.g. UMAP plots are not always representative of the underlying space. However it highlights that these embedding spaces are learning to separate out heroes based on their features. Further analysis is suggested in Section 8.

It seems that the Time to Die architecture is extremely sensitive to hyper-parameters, for example, we see that there is an 0.2 improvement in F1 Score between the TtD_A and TtD_B models, where the only change is adjusting hyper-parameters. Furthermore, the number of time steps used in the TtD_{Temp} models has a huge impact of performance. This was also observed during the hyper-parameter fitting process where small changes in hyper-parameters lead to large changes in performance. It is not immediately clear why this is, although it is likely impacted by the huge number of inputs passed to the model. For each hero there are 138 standard features, one hero ID and 19 item IDs. However, as the experimental results show, these features are all relatively important to the predictive power of the model, and attempts at building smaller feature sets resulted in much worse performance.

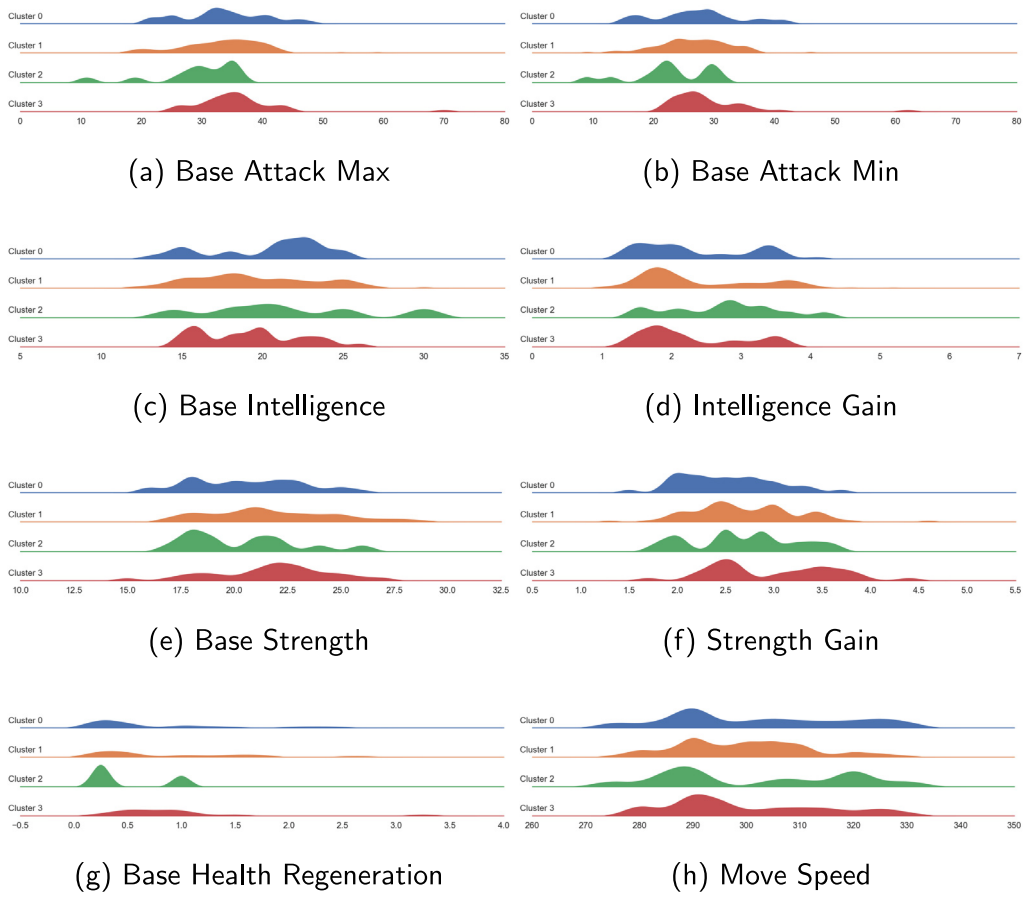


Fig. 10. KDE Plots for distribution of selected hero statistics across the four clusters identified using agglomerative clusters.

7. Conclusion

Esports is a rapidly growing sector of the larger sports market. Esports tournaments are characterised by a complex data space and fast-paced action (Block et al., 2018; Demediuk et al., 2019), and correspondingly complex tasks for commentators and broadcast creators. A central challenge is being able to anticipate where to focus the attention of a broadcast, in real-time. Towards meeting this challenge, moment-to-moment predictions that can navigate the complex data space of esports in real-time are required. Here, we present an extension to the *Time to Die* architecture (Katona et al., 2019), which enables prediction of player death events in the popular MOBA *Dota 2*, towards empowering broadcasters to better observe and process esports matches. The extensions include embeddings for categorical data features which reduces the size of the model input vector compared to one-hot encoding used previously in Katona et al. (2019). This allows us to include the ID of each player's hero and the ID of in-game items owned by each player. We also extended the framework to include new data features reflecting each player's hero's status features such as if the hero is stunned which we expected to correlate with each player's likelihood of dying, as well as new neural architecture using recurrent layers to capture temporal data patterns which were not captured in the original model (Katona et al., 2019). Finally, we investigate different time-series lengths to optimise prediction accuracy.

We describe our data processing and model generation pipeline to transform raw *Dota 2* match data into a training dataset from which we generate a recurrent neural model for conducting micro-predictions. We evaluated a number of data and model configurations for predictive accuracy.

In our evaluations, we found that a number of our enhancements are important and improve performance, namely status effects, embeddings for categorical features, a recurrent architecture and an optimised time-series length. These extensions outperform the original work, and in the case of embeddings and status effect do so without inference-time penalty. Furthermore we show that using embeddings rather than one-hot encoding allows for full modelling of item IDs, which are important to the predictive power of the model. We improved the F1 score from 0.17 using the previous *Time to Die* model to 0.62 using our new data and architectural enhancements. We also show that while recurrent architectures do have a performance penalty they would be feasible for deployment in a broadcast setting as they can run in real time on consumer grade hardware. Thus the improved models presented in this work are likely to be useful for *Dota 2* broadcasters.

8. Future work

It would be very useful to further understand false predictions made by the model, both false positives, i.e., the model predicted a death that did not occur and false negative, i.e., the model predicted a player would survive by they did not. For example, are these false predictions explained by player skill or play style? It is currently infeasible to profile individual players in the network due to the sheer size of the player base and the relative small number of samples for each player. However, if there are consistent features of false predictions, especially false positive predictions then it may be possible to model these and thus establish a set of key performance indicators which improve a player's chance of surviving. These can then be used either by the broadcast team to analyse important plays within a game or used as a training aid for esports teams.

Understanding how the embedding space is learned and if previously unknown information about *Dota 2* can be uncovered by analysing this is outside of the scope of this work, and likely constitutes

a research contribution in its own right. Such work is likely to be very valuable in understanding the complexities of *Dota 2*. Naturally, these embeddings spaces are learnt within a model which is trained to predict if a player is going to die and therefore the spaces are unlikely to perfectly explain the landscapes of hero IDs and item IDs. However, there is the possibility that valuable information can be uncovered, e.g. perhaps offensive items and defensive items are grouped together because they actually have similar impacts to the survivability of a player. Furthermore, there maybe a relationship between the two learnt spaces, e.g. certain items are more often used by a subset of heroes and potentially these items and heroes would then be co-located in their respective spaces.

Finally, this work is concerned with in-game prediction of player death. However, there are many other key in-game events which are useful to predict, e.g. if the teams are going to engage in a team fight or target a certain objectives around the map. Predicting player death within 5 seconds is a useful broadcast aid because team fights are fast-paced (Tot et al., 2021). However, predicting if certain objectives are going to be targeted within 5 seconds is – depending on the specific context – less likely to be useful as it is easier for an expert commentator to detect the lead-up to these types of events. Therefore experimentation for longer form prediction (e.g. 10+ seconds) would likely be necessary for predicting objective- or team fight-based labels.

CRediT authorship contribution statement

Charles Ringer: Conceptualisation, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualisation. **Sondess Missaoui:** Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft. **Victoria J. Hodge:** Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing. **Alan Pedrassoli Chitayat:** Methodology, Data curation, Investigation, Validation, Writing – original draft. **Athanasios Kokkinakis:** Data curation, Investigation, Validation. **Sagarika Patra:** Data curation, Investigation, Validation. **Simon Demediuk:** Data curation, Investigation, Validation, Writing – original draft. **Alvaro Caceres Munoz:** Data curation, Investigation, Validation. **Oluseji Olarewaju:** Data curation, Investigation, Validation. **Marian Ursu:** Data curation, Investigation, Validation, Funding acquisition. **Ben Kirman:** Data curation, Investigation, Validation, Funding acquisition. **Jonathan Hook:** Data curation, Investigation, Validation, Funding acquisition. **Florian Block:** Conceptualisation, Data curation, Validation, Writing – review & editing, Supervision, Project administration, Funding acquisition. **Anders Drachen:** Conceptualisation, Data curation, Validation, Writing – review & editing, Supervision, Project administration, Funding acquisition. **James Alfred Walker:** Conceptualization, Methodology, Resources, Writing – original draft, Writing – review & editing, Visualisation, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

This work is jointly funded by the Digital Creativity Labs (EPSRC/AHRC/Innovate UK, grant no. EP/M023265/1) and WEAVR (Audience of the Future programme by UK Research and Innovation through the Industrial Strategy Challenge Fund, grant no. 104775).

This project was undertaken on the Viking Cluster, which is a high performance compute facility provided by the University of York. We are grateful for computational support from the University of York High Performance Computing service, Viking and the Research Computing team.

References

Agarwala, A., & Pearce, M. (2014). *Learning DOTA 2 team compositions: Tech. rep.*, Stanford University.

Akhmedov, K., & Phan, A. H. (2021). Machine learning models for DOTA 2 outcomes prediction. URL https://arxiv.org/abs/2106.01782.

Ani, R., Harikumar, V., Devan, A. K., & Deepa, O. S. (2019). Victory prediction in league of legends using feature selection and ensemble methods. In *2019 international conference on intelligent computing and control systems* (pp. 74–77). http://dx.doi.org/10.1109/ICCS45141.2019.9065758.

Anshori, M., Mar'i, F., Alauddin, M. W., & Abdurrahman Bachtiar, F. (2018). Prediction result of DOTA 2 games using improved SVM classifier based on particle swarm optimization. In *2018 international conference on sustainable information engineering and technology* (pp. 121–126). http://dx.doi.org/10.1109/SIET.2018.8693204.

Bailey, K. (2020). *Statistical learning for Esports match prediction* (Ph.D. thesis), Pomona: California State Polytechnic University.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*(10), 281–305, URL http://jmlr.org/papers/v13/bergstra12a.html.

Bergstra, J., Yamins, D., & Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th international conference on machine learning* (pp. 115–123).

Block, F., Hodge, V., Hobson, S., Sephton, N., Devlin, S., & Ursu, M. F. (2018). Narrative bytes: Data-driven content production in esports. In *Procs of the 2018 ACM international conference on interactive experiences for tv and online video* (pp. 29–41). http://dx.doi.org/10.1145/3210825.3210833.

Chitayat, A. P., Kokkinakis, A., Patra, S., Demediuk, S., Robertson, J., & Olarewaju, O. (2020). WARDS: Modelling the worth of vision in MOBA's. In *Science and information conference* (pp. 63–81). Springer.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., & Schwenk, H. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing* (pp. 1724–1734). Doha, Qatar: Association for Computational Linguistics, http://dx.doi.org/10.3115/v1/D14-1179, URL https://aclanthology.org/D14-1179.

Cleghern, Z., Lahiri, S., Özaltin, O., & Roberts, D. L. (2017). Predicting future states in DOTA 2 using value-split models of time series attribute data. In *Pros of the 12th int'l conf. on foundations of digital games*. ACM, http://dx.doi.org/10.1145/3102071.3102095, URL http://doi.acm.org/10.1145/3102071.3102095.

Conley, K., & Perry, D. (2013). *How does he saw me? A recommendation engine for picking heroes in DOTA 2: Tech. rep.*, Stanford University.

Demediuk, S., York, P., Drachen, A., Walker, J. A., & Block, F. (2019). Role identification for accurate analysis in DOTA 2. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, *15*(1), 130–138, URL https://ojs.aaai.org/index.php/AIIDE/article/view/5235.

Drachen, A., Yancey, M., Maguire, J., Chu, D., Wang, I. Y., & Mahlmann, T. (2014). Skill-based differences in spatio-temporal team behaviour in defence of the ancients 2 (DOTA 2). In *2014 IEEE games media entertainment* (pp. 1–8). http://dx.doi.org/10.1109/GEM.2014.7048109.

Eggert, C., Herrlich, M., Smeddinck, J., & Malaka, R. (2015). Classification of player roles in the team-based multi-player game DOTA 2. In K. Chorianopoulos, M. Divitini, J. Baalsrud Hauge, L. Jaccheri, & R. Malaka (Eds.), *Entertainment computing* (pp. 112–125). Cham: Springer International Publishing.

Hamari, J., & Sjöblom, M. (2017). What is eSports and why do people watch it? *Internet Research*, *27*(2), 211–232.

Hodge, V., Devlin, S., Sephton, N., Block, F., Cowling, P., & Drachen, A. (2019). Win prediction in multi-player esports: Live professional match prediction. *IEEE Transactions on Games*, 1. http://dx.doi.org/10.1109/TG.2019.2948469.

Hodge, V., Devlin, S., Sephton, N., Block, F., Drachen, A., & Cowling, P. (2017). Win prediction in esports: Mixed-rank match prediction in multi-player online battle arena games. ArXiv E-Prints (CS:AI) URL https://arxiv.org/abs/1711.06498.

Johansson, F., & Wikström, J. (2015). *Result Prediction by Mining Replays in DOTA 2*. Karlskrona, Sweden: Blekinge Institute of Technology.

Kang, S. -K., & Lee, J. -H. (2020). An E-sports video highlight generator using win-loss probability model. In *Proceedings of the 35th annual ACM symposium on applied computing* (pp. 915–922). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3341105.3373894, URL https://doi.org/10.1145/3341105.3373894.

Katona, A., Spick, R., Hodge, V. J., Demediuk, S., Block, F., Drachen, A., et al. (2019). Time to die: Death prediction in DOTA 2 using deep learning. In *2019 IEEE conference on games* (pp. 1–8). http://dx.doi.org/10.1109/CIG.2019.8847997.

Ke, C. H., Deng, H., Xu, C., Li, J., Gu, X., & Yadamsuren, B. (2022). DOTA 2 match prediction through deep learning team fight models. In *2022 IEEE conference on games* (pp. 96–103). http://dx.doi.org/10.1109/CoG51982.2022.9893647.

Kim, C., & Lee, S. (2020). Predicting win-loss of league of legends using bidirectional LSTM embedding. *KIPS Transactions on Software and Data Engi- neering*, *9*(2), 61–68.

Kinkade, N., Jolla, L., & Lim, K. (2015). *DOTA 2 Win Prediction*: *Tech. rep.*, University of California San Diego.

Lopez-Gordo, M., Kohlmorgen, N., Morillas, C., & Pelayo, F. (2020). Performance prediction at single-action level to a first-person shooter video game. *Virtual Reality*, 1–13.

Luo, Z., Guzdial, M., & Riedl, M. (2019). Making CNNs for video parsing accessible: Event extraction from DOTA 2 gameplay video using transfer, zero-shot, and network pruning. In *Proceedings of the 14th international conference on the foundations of digital games*. New York, NY, USA: Association for Computing Machinery, URL https://doi.org/10.1145/3337722.3337755.

Makarov, I., Savostyanov, D., Litvyakov, B., & Ignatov, D. I. (2018). Predicting winning team and probabilistic ratings in DOTA 2 and counter-strike: Global offensive video games. In *Procs 6th international conference on analysis of images, social networks and texts* (pp. 183–196).

Marshall, S., Mavromoustakos-Blom, P., & Spronck, P. (2022). Enabling real-time prediction of in-game deaths through telemetry in counter-strike: Global offensive. In *Proceedings of the 17th international conference on the foundations of digital games*. New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3555858.3555859, URL https://doi.org/10.1145/3555858.3555859.

Maymin, P. Z. (2021). Smart kills and worthless deaths: Esports analytics for league of legends. *Journal of Quantitative Analysis in Sports*, *17*(1), 11–27. http://dx.doi.org/10.1515/jqas-2019-0096, URL https://doi.org/10.1515/jqas-2019-0096.

McInnes, L., Healy, J., & Melville, J. (2018). UMAP: Uniform manifold approximation and projection for dimension reduction. URL https://arxiv.org/abs/1802.03426.

Pobiedina, N., Neidhardt, J., Calatrava Moreno, M. d. C., & Werthner, H. (2013). Ranking factors of team success. In *Procs of the 22nd international conference on world wide web* (pp. 1185–1194). ACM.

Ringer, C., Walker, J. A., & Nicolaou, M. A. (2019). Multimodal joint emotion and game context recognition in league of legends livestreams. In *2019 IEEE conference on games*. IEEE.

Rioult, F., Métivier, J. -P., Helleu, B., Scelles, N., & Durand, C. (2014). Mining tracks of competitive video games. *AASRI Procedia*, *8*, 82–87.

Schubert, M., Drachen, A., & Mahlmann, T. (2016). Esports analytics through encounter detection. In *Proceedings of the MIT sloan sports analytics conference*.

Semenov, A., Romov, P., Korolev, S., Yashkov, D., & Neklyudov, K. (2016). Performance of machine learning algorithms in predicting game outcome from drafts in DOTA 2. In *International conference on analysis of images, social networks and texts* (pp. 26–37). Springer.

Seo, Y. (2013). Electronic sports: A new marketing landscape of the experience economy. *Journal of Marketing Management*, *29*(2), 1542–1560.

Silva, A. L. C., Pappa, G. L., & Chaimowicz, L. (2018). Continuous outcome prediction of league of legends competitive matches using recurrent neural networks. In *SBC-proceedings of SBCGames* (pp. 2179–2259).

Song, K., Zhang, T., & Ma, C. (2015). *Predicting the winning side of DOTA 2*: *Tech. rep.*, Stanford University.

Tot, M., Conserva, M., Chitayat, A. P., Kokkinakis, A., Patra, S., & Demediuk, S. (2021). What are you looking at? Team fight prediction through player camera. In *2021 IEEE conference on games* (pp. 1–8). http://dx.doi.org/10.1109/CoG52621.2021.9619038.

Wang, W. (2016). Predicting multiplayer online battle arena (MOBA) game outcome based on hero draft data. (Masters thesis), Dublin: National College of Ireland.

Wang, K., Li, H., Gong, L., Tao, J., Wu, R., & Fan, C. (2020). Match tracing: A unified framework for real-time win prediction and quantifiable performance evaluation. In *Proceedings of the 29th ACM international conference on information & knowledge management* (pp. 2781–2788).

Wang, K., & Shang, W. (2017). Outcome prediction of DOTA 2 based on Naïve Bayes classifier. In *2017 IEEE/ACIS 16th international conference on computer and information science* (pp. 591–593). http://dx.doi.org/10.1109/ICIS.2017.7960061.

Wen, L., Ye, X., & Gao, L. (2020). A new automatic machine learning based hyper-parameter optimization for workpiece quality prediction. *Measurement and Control*, *53*(7–8), 1088–1098.

White, A., & Romano, D. M. (2020). Scalable psychological momentum forecasting in Esports. arXiv preprint arXiv:2001.11274.

Xue, H., Pu, H., Hawzen, M., & Newman, J. (2016). E-sports management? Institutional logics, professional sports, emerging esports field. In *2016 north american society for sport management conference*.

Yang, P., Harrison, B. E., & Roberts, D. (2014). Identifying patterns in combat that are predictive of success in moba games. In *Procs of the 9th int'l conf. on foundations of digital games*. ACM.

Yang, Y., Qin, T., & Lei, Y. -H. (2017). Real-time eSports match result prediction. URL https://arxiv.org/abs/1701.03162.

Yang, Z., Wang, Y., Li, P., Lin, S., Shi, S., & Huang, S. -L. (2022). Predicting events in MOBA games: Prediction, attribution, and evaluation. *IEEE Transactions on Games*, 1. http://dx.doi.org/10.1109/tg.2022.3159704.

Yannakakis, G. (2012). Game AI revisited. In *Proc. of ACM computing frontiers conference* (pp. 285–292).

Yu, L., Zhang, D., Chen, X., & Xie, X. (2018). MOBA-slice: A time slice based evaluation framework of relative advantage between teams in MOBA games. arXiv preprint arXiv:1807.08360.

Zhang, C., Liu, J., Wang, Z., & Sun, L. (2020). Look ahead at the first-mile in livecast with crowdsourced highlight prediction. In *IEEE infocom 2020 - IEEE conference on computer communications* (pp. 1143–1152). http://dx.doi.org/10.1109/INFOCOM41043.2020.9155395.