Look Ahead at the First-mile in Livecast with Crowdsourced Highlight Prediction

Cong Zhang[§,†], Jiangchuan Liu[†], Zhi Wang[‡¶], Lifeng Sun[*]
[§]School of Computer Science and Technology, University of Science and Technology of China, China
[†]School of Computing Science, Simon Fraser University, Canada
[‡]Tsinghua Shenzhen International Graduate School, China
[¶]Peng Cheng Laboratory, China
[*]Department of Computer Science and Technology, Tsinghua University, China

***Abstract*—Recently, data-driven prediction strategies have shown the potential of shepherding the optimization strategies for end viewer's Quality-of-Experience in practical streaming applications. The current prediction-based designs have largely focused on optimizing the last-mile, i.e., viewer-side, which 1) need the real-time feedback from viewers to improve the prediction accuracy; and 2) need quick responses to guarantee the effectiveness of optimization strategies in the future. Thanks to the emerged crowdsourced livecast services, e.g., Twitch.tv, we for the first time exploit the opportunity to realize the long-term prediction and optimization with the assistance derived from the first-mile, i.e., source broadcasters.**

In this paper, we propose a novel framework *CastFlag*, which analyzes the broadcasters' operations and interactions, predicts the key events (i.e., highlights), and optimizes the transcoding stage in the corresponding live streams, even before the encoding stage. Taking the most popular eSports gamecast as an example, we illustrate the effectiveness of this framework in the game highlight prediction and transcoding workload allocation. The trace-driven evaluation shows the superiority of CastFlag as it: (1) improves the prediction accuracy over other learning-based approaches by up to 30%; (2) achieves an average of 10% saving of the transcoding latency at less cost.

I. Introduction

Fueled by today's high-speed networks (e.g., LTE/5G) and high-performance personal devices (e.g., iPhone 11 Pro), crowdsourced livecast[1] has emerged as one of the most fashionable applications. According to the statistics from Alexa.com, Twitch's global traffic ranking just experienced a huge boost from 100th to 30th in recent three years. Moreover, crowdsourced livecast services also occupy a large number of Internet users' daily life. Recent Twitch's report reveals that there are more than 1.2 million concurrent visitors and more than 3 million unique monthly livecasters in 2019. The "Daily Time on Site" of Twitch's viewers is over five minutes per day, and this index on DouyuTV [1] is about nine minutes[2].

To improve the viewers' Quality-of-Experience (QoE), more and more proposals attempted to use data-driven prediction strategies to analyze the viewers' streaming sessions, then employ the prediction results to shepherd the corresponding optimizations [2]–[4]. Yet the existing prediction-based solutions have largely focused on optimizing the last-mile (i.e., viewer-side) performance, which still suffers from severe limitations, as they mainly collect viewer's feedback and need quick response to usher the prediction and optimization in real-time [6], [8], [9]. Thanks to today's crowdsourced livecast applications, in which livecasters broadcast daily performances, creative shows, and game playthroughs[3] conveniently to a large number of viewers who discuss livecast-related topics at the same time, it is known that the first-mile (i.e., livecaster-side) is equally, if not more, important as the last-mile.

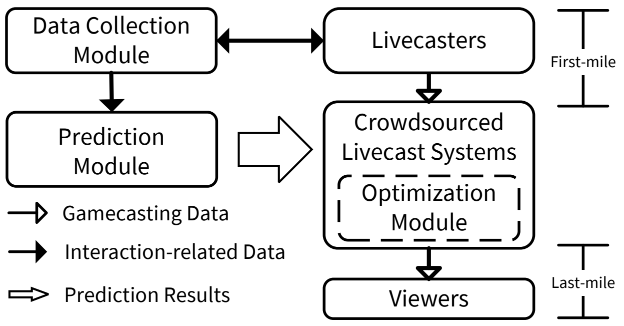


Fig. 1: Illustration of CastFlag

In this paper, we target a livecaster-centric framework *CastFlag*, as shown in Figure 1, to capture their traces, predict the highlights, and optimize the livecasts. The key challenges towards designing the framework lie in: 1) how to determine a proper prediction lag and improve the accuracy based on the collected data from livecasters; and 2) how to optimize crowdsourced livecast services with the assistance of the highlights prediction. Considering that: 1) the number of viewers watching eSports has leapfrogged that of traditional sports broadcasting [10]; and 2) the eSports multiplayer's interactions furnish a large number of first-mile data which contribute to evaluating the effectiveness of our proposed framework, we take eSports gamecast (i.e., game-related livecast) as an example to explore the opportunities and challenges in designing such a framework.

We closely examine the impact of the highlights in eSports gamecast services. The measurement result reveals that a

[1]In this paper, we use terms "livecast" and "livecaster" to represent "live broadcast" and "live broadcaster" for convenience.

[2]The index of YouTube.com and Netflix.com are 11m51s and 3m14s, respectively.

[3]Game playthrough is the act of playing a game from start to finish.

highlight in a game largely impacts the complexity of the corresponding game scene, which in turn determines the transcoding latency. Moreover, by analyzing the official video statistics from Twitch.tv, we find that the broadcast latency, which is defined as the time lag between a game scene on the gamer side and the corresponding gamecast content received on the viewer side, is mainly derived from the transcoding period. Motivated by these observations, we design the prediction and optimization modules in the framework CastFlag to capture the unique features in eSports games, i.e., the multiplayer's real-time operations, extract game-related strategies, predict the game highlights (i.e., key events), and optimize the transcoding task allocation.

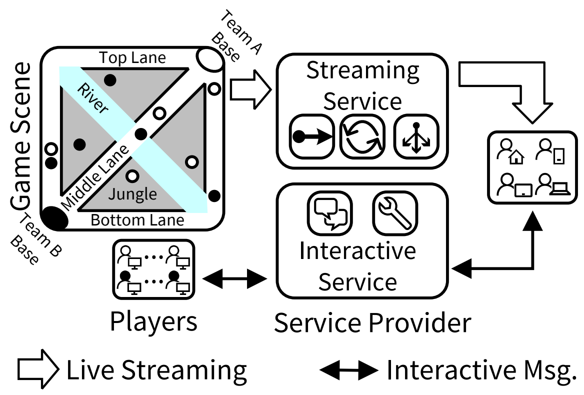


Fig. 2: A typical crowdsourced livecast service

To the best of our knowledge, our study is the first to explore the potential from the multiplayer's operations in the eSports gamecast services. Our contributions can be summarized as follows: (1) We conduct the cloud-based measurements to reveal the challenges and motivate our design; (2) To usher the optimizations for transcoding workloads, the framework is designed with full knowledge of multiplayer's game strategies; (3) The performance evaluation demonstrates that the proposed prediction module achieves more than 90% accuracy in the highlight prediction based on the collected game replays. In addition, the trace-based evaluation shows that the proposed optimization approach reduces the broadcast latency cost-effectively.

The remainder of this paper is organized as follows. Section II introduces the background. Section III illustrates the measurement results based on a cloud-based streaming testbed, which also motivated our work in this paper. In Sections IV and V, we propose the design of each module in the framework CastFlag. We then evaluate the performance of our framework in Section VI. We present the related work in Section VII. Section VIII concludes the paper with further discussion.

II. Background

A. Crowdsourced Livecast

Crowdsourced livecast has emerged as one of the most popular live streaming applications in recent years [11]. Such services as Twitch, Periscope [12], and DouyuTV, have contributed to a significant amount of today's Internet traffic and got into fellow viewers' daily life. It offers two parallel services: *Streaming Service* and *Interactive Service*, as shown in Figure 2. Monitor scenes are first captured and encoded by a streaming software (e.g., Open Broadcaster Software [13]) deployed on a livecaster's device. Then, the streaming service ingests game sessions, assigns transcoding tasks, and distributes contents to viewers. In the meantime, the interactive service provides an embedded chatting platform for the livecasters and viewers, creating lots of novel streaming scenarios, e.g., crowdsourced gaming *TwitchPlaysPokemon*.

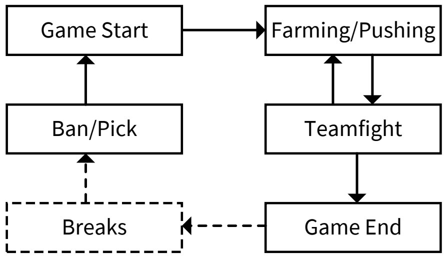
B. eSports Gamecast

As the mainstay of crowdsourced livecast applications, eSports gamecast continuously attracts much attention from both gamers and viewers. According to the statistics from Twitch, more than 60% of concurrent viewers are contributed by eSports gamecasts every day, such as Dota2, League of Legends (LoL), and CS:GO. Here, Dota2 and League of Legends are two multiplayer online battle arena (MOBA) video games and Counter-Strike: Global Offensive (CS:GO) is a multiplayer first-person shooter video game. Similarly, DouyuTV, the twenty-eighth of top sites in China, also reported that more than 70% of the most attractive livecasters are interested in LoL game matches [14].

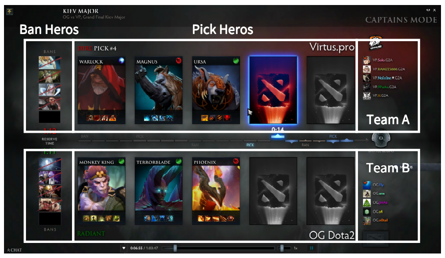
It has been reported that the global games market generated \$120.1 billion in revenues and the number of the viewers for gaming videos grew by 5% to reach \$944 million in 2019 [15], which further elevates the growth of eSports gamecast. As the source of eSports gamecast, a typical eSports match contains several gamers who are divided into different teams to compete for various resources, e.g., golds, weapons, or controlled regions. The left part in Figure 2 briefly demonstrates the design of an eSports game "Dota2", in which ten gamers are grouped into two teams. The battle area consists of several types of regions: three lanes (top, middle, and bottom), a river, four jungle areas, etc. When a match starts, every gamer first chooses one hero, then upgrades the hero's level and equipment by fighting with the enemies and the other gamers in another team. Like most eSports games, Dota2 also highlights the gamers' cooperation in one team, therefore, how to fight enemies together, called "teamfight", becomes the most important event, which may determine the final result of a match. In a Dota2 teamfight, most of the gamers participate in casting their heroes' skills and attacking others, which also attracts the attention from fellow viewers during the gamecast.

In a Dota2 match, the game flow mainly includes the following five parts, as shown in Figure 3:

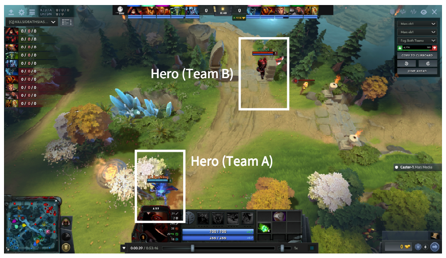
- Pick/Ban (P/B): The captain in each team bans certain heroes, preventing either team from picking the heroes; every captain also chooses five heroes for the whole team alternately;
- Game Start: After choosing preferred heroes, the gamers in a team start the game from their base and choose the combat lanes according to the heroes' abilities;



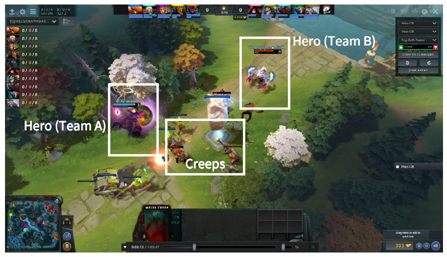
(a) Playing flow of MOBA games



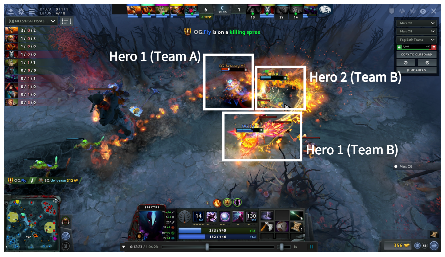
(b) Pick/Ban



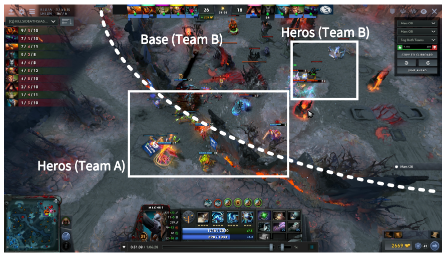
(c) Game start



(d) Farming/Pushing



(e) Teamfight



(f) Game end

Fig. 3: Illustrations in the playing flow of MOBA games.

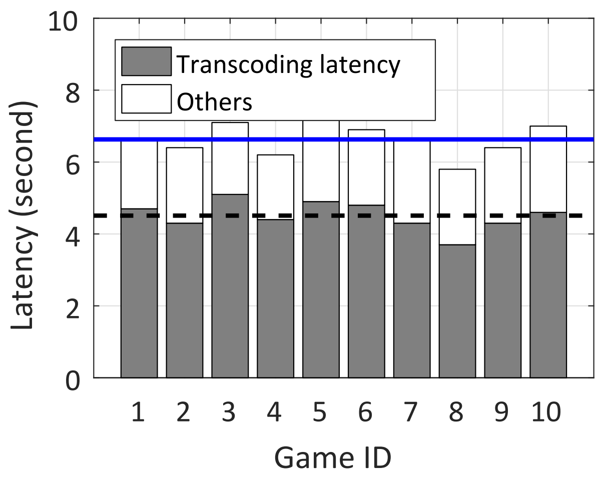


Fig. 4: Transcoding latency

- Farming/Pushing (F/P): All gamers upgrade their equipment and levels through earning golds and experience from clearing enemy creeps on the lanes or combating the neutral creeps at the jungle areas;
- Teamfight (TF): During an eSports gamecast, teamfights are the most attractive events, in which the gamers attack the opponent heroes using their skills. It occurs among several farming/pushing stages. Because the goal of a teamfight is to eliminate as many of the opponent heroes as possible, a good teamfight can determine the final result of a match.
- Game End: After several rounds of farming, pushing and teamfight, the gamers in a team will try to attack the opponents' base, if success, they win the match;

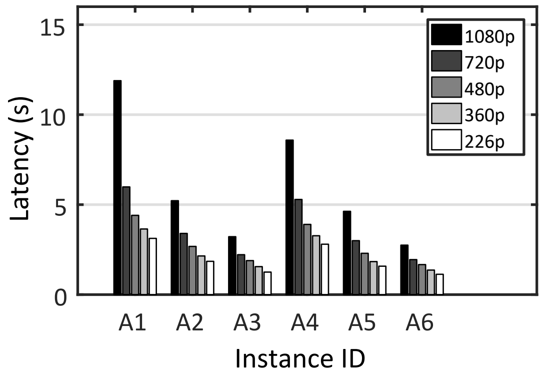
III. Measurement and Observations

In this section, we investigate the characteristics of transcoding tasks on various Amazon EC2 instances. The measurement is mainly based on the game Dota2, which, according to the statistics about the most watched games on Twitch [16], ranks the third in 2018. Besides, Dota2 game client provides game match replays, which contains every player's interaction and all game-related information, to any users friendly. The measurement results also motivate our work in this paper.

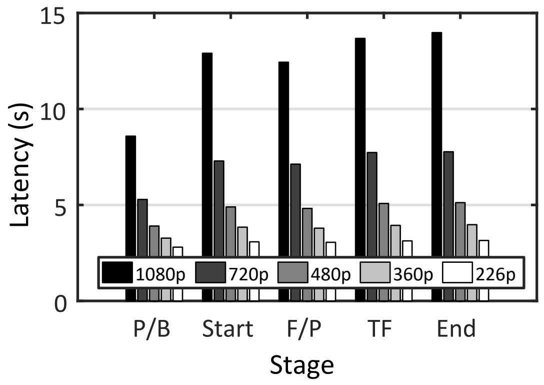
A. Broadcast Latency

The encoded game scenes are continuously streamed from content sources managed by gamers to ingesting servers through proprietary protocols, e.g., RTMP (Real Time Message Protocol [17]). Considering the overhead on the server-side, such proprietary protocols cannot be used to deliver gamecast contents to all viewers in the livecast scenario [18]. The general way, therefore, is to transcode original RTMP streaming to HTTP-based streaming, e.g., HTTP Live Streaming (HLS [19]), but consume a massive amount of computational resources and costs [20]. Besides, eSports gamecast service providers encourage livecasters and fellow viewers to participate in real-time discussions. If there exists a huge difference among the viewer-side latencies, viewers' Quality-of-Experience (QoE) will be largely impaired. For example, a high-latency viewer may know the final result of an eSports match from the low-latency viewers' online discussion before watching the corresponding gamecast content.

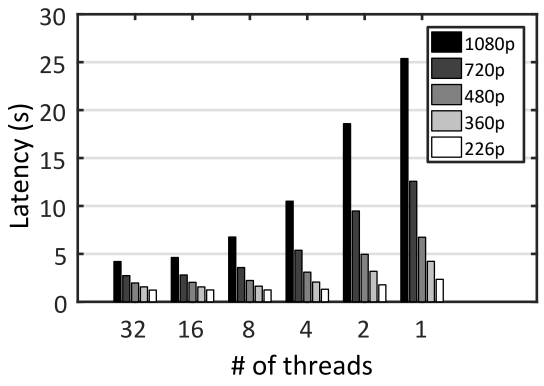
There have been some efforts addressing the aforementioned issues from both industry and academia [6], [21]. Yet the existing solutions either change transcoding settings on the livecaster side or optimize the streaming delivery using the information on the viewer-side. Such strategies, when used in eSports gamecast services, lack full knowledge of game



(a) Instance impacts (P/B stage)



(b) Stage impacts (A4)



(c) Thread impacts (TF stage on A6)

Fig. 5: Measurement results on different instances

events, which lead to the lag and inaccuracy of optimizations [22]. For instance, a strategy first predicts that the highest concurrent viewing number will occur after two minutes in an eSports gamecast. Then, the corresponding optimization strategy reserves extra computational and bandwidth resources to serve this peak workload in the future. Yet this match and gamecast are terminated after one minute, the optimization fails, and the reserved resources also become wasteful.

To explore the challenges and opportunities for addressing these problems, we collect the official latency statistics from the top 100 livecasters in Twitch eSports game to investigate the characteristics of broadcast latency. We observe that

TABLE I: Configuration of Amazon EC2 instances

ID	Type	vCPU	Memory (GB)	Price
A1	m4.xlarge	4	16	\$0.2/h
A2	m4.2xlarge	8	32	\$0.4/h
A3	m4.4xlarge	16	64	\$0.8/h
A4	c4.xlarge	4	7.5	\$0.199/h
A5	c4.2xlarge	8	15	\$0.398/h
A6	c4.4xlarge	16	30	\$0.796/h

more than 68% of the broadcast latency is derived from the transcoding period, as shown in Figure 4. Therefore, if transcoding latency reduces, the viewer's QoE could be improved accordingly.

B. *Transcoding Workload*

We further analyze the transcoding latency by deploying a crowdsourced gamecast service on various Amazon EC2 instances [23]. The measurement for each stage relies on ten videos that are recorded from five randomly selected Dota2 replays.

As shown in Table I, we use six types of instances, which have various settings and prices. A1, A2, and A3 are m3 instances, which provide a balance of computing, memory, and network resources. A4, A5, and A6 are c4 instances, which are optimized for compute-intensive workloads and delivers very cost-effective high performance at a low price/compute performance in EC2. We measure the transcoding performance using FFmpeg to convert the original videos to different resolutions and bitrates. The broadcast latency is measured by deploying an RTMP-Nginx module [24] on each platform. The instance with this module can play a practical streaming server that ingests an original video as a livecast and transcodes it to various quality versions.

Figure 5 shows the measurement results of the transcoding tasks on different instances. As shown in Figure 5a, we observe that the computation-optimized instances A4, A5, and A6 achieve lower latencies than the general instances A1, A2, and A3, even they have the similar cost. But the superiority of computation-optimized instances is decreased in the transcoding tasks with the low-resolution quality versions. Figures 5b indicates the performance of transcoding the game scenes in different stages on A4. We find that the latency in the P/B stage is clearly lower than the other stages. Besides, the transcoding tasks in the TF and End stages need more time to process complex game scenes, which introduce extra latencies. We also observe the huge disparity among different streaming qualities. For example, in the TF stage, the average transcoding latency of the 1080p videos is 3 seconds higher than that of the 720p videos in Figure 5b. Figure 5c shows the impacts of the number of the threads on A6 instance. By changing the number of the used CPU threads to a reasonable setting, we can dynamically adjust the transcoding latencies for the different quality versions in a gamecast. These measurement results motivate us to design a framework that not only predicts the highlights in a gamecast but also balances the transcoding latency and cost in different settings according to the prediction results. Our measurement results reveal that game highlights, i.e., the key events in a game, largely impact the complexity of the corresponding game scenes, which in turn determines the transcoding latency in a gamecast.

IV. CastFlag: An Overview

Motivated by the previous measurement results in eSports gamecast services, we propose the detailed design concepts in the CastFlag framework. The two modules in the previous generic framework are designed to facilitate the existing eSports gamecast services. As shown in Figure 6, it consists of a Strategy-based Prediction (SBP) module and a Highlight-aware Optimization (HAO) module. The SBP module is a new part of this framework compared with that of the existing eSports gamecast services. Upon receiving the gamers' interactions in a match from the game server in real-time, the SBP module will be aware of the team strategies, predicting the highlights, e.g., teamfights, in this match. The HAO module receives the predictions from the SBP module as the hints guiding the gamecast transcoding. Combining these hints and the viewing demands, the HAO module allocates resources ahead of the emergences of the predicted highlights cost-effectively.

There are however a number of critical practical and theoretical issues to be addressed in this generic framework. First, the gamers' interactions change over time, which generates the varying strategies of cooperations in a team and competitions between two teams. Although the existing study in [25] have predicted the highlights in the next few seconds, the accuracy of its prediction is not very high. As such, after receiving the interactions, CastFlag must well forecast the highlights with a higher accuracy under a longer prediction gap, which can early shepherd the corresponding optimization strategies. In addition, after predicting the highlights, how to cost-effectively allocate the transcoding tasks and strategically design the content distribution will significantly impact the viewers' QoE. These problems are further complicated given the heterogeneous devices and network connections of viewers and that of livecasters (i.e., gamers). More importantly, such a framework should be transparent to the livecasters and viewers, that is, the whole design should not suspend the match, impair the gamecast, or reduce the viewers' QoE, so as to complement with existing eSports gamecast services.

V. Framework Design

In this section, we illustrate the design of the framework CastFlag and further propose the solution to optimize the gamecast. We first illustrate the strategy-based prediction (SBP) module based on gamers' interactions.

A. Strategy-based Prediction

In an eSports game match, the replay records the gamers' interactions into a proprietary file; we therefore have the opportunity to retrieve every gamer's data by parsing a game replay. To better understand these interactions and preprocess them, we first classify them into several groups according to the interaction types that are pre-defined by the game developers. As shown in Table II, we show the details of different types of data in a replay. We observe that about 88% of data show the changes in gamers' locations, and 6.5% are combat logs, which illustrate how gamers attack their enemies, e.g., damage values during a teamfight. Besides, other data consist of the changes of gamers' golds, experience, skill levels, equipment, etc. These interactions can naturally be classified into two types of interactions: (1) spatial interactions, e.g., every gamer has a fixed location at a certain time; (2) temporal interactions, e.g., the golds owned by every gamer will be changed along with the time of a match. The CastFlag framework, therefore, must capture the characteristics of these two types of interactions. To fulfill this target, we extract global strategies and local strategies. The global strategies contain the location changes of all gamers, the unit changes of the whole battle arena, and the competition data of two teams. The local strategies include the changes of the gold, experience, buy/sell items of every gamer along with match time.

TABLE II: Statistics of the data in a replay sample

Type	Description	# of logs	Percentage
4	Hero's movements	392,268	87.9%
24	Damage in combats	29,544	6.6%
3	Hero's local statistic, e.g., gold and experience	5,810	1.3%
27	Ability used in combats	1,801	0.4%
26	Death in combats	3,903	0.9%
29	Item buy/sell	600	0.1%
others	Other info. in combats, gamers' statistics, etc.	12,434	2.8%

Finally, we constructed the SBP module using a deep learning network, as shown in Figure 7. It consists of several 2dCNN (Convolutional Neural Network) layers [26], [27] and stack LSTM (Long Short-Term Memory) layers [28], capturing the correlations between the spatio-temporal features in an eSports game and the highlights in the corresponding gamecast. In deep learning research, CNN is a class of deep, feed-forward neural network that has successfully been applied to processing videos/images and decreasing the dimensions of input data. The LSTM is a type of RNN (Recurrent Neural Network) [29], well classifying, processing and predicting time series. Through unique gating units, LSTM avoids the long-term dependency problem, having the ability to remove or add information when learning time series. The output shows whether the input data in a time slot will induce a highlight or not after a few times. Considering the eSports gamecast scenario, we define the game teamfight as the highlight in this paper. Because the SBP module is expected to predict highlights before their occurrences, we assume that the interactions between time slot T and $T + t_{gap}$ will trigger a highlight, i.e., teamfight, if they occur t_{gap} seconds ahead of this highlight.

In the SBP module, the first part is two 2dCNN layers for global strategy and local strategy, respectively. For each time

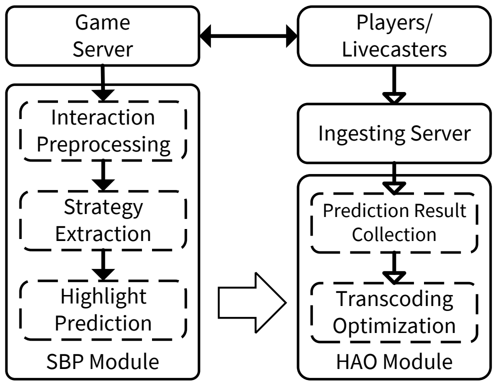


Fig. 6: Illustration of the workflow

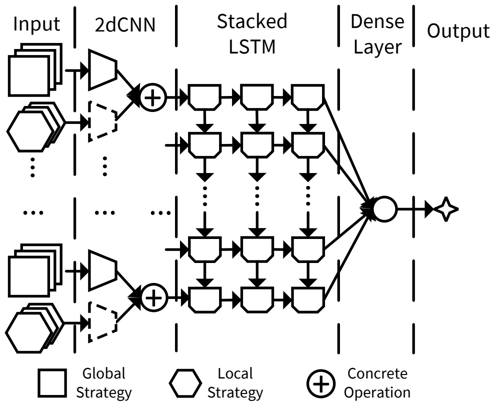


Fig. 7: Design of highlight prediction

slot, the 2dCNNs can reduce the dimensions of the two types of input data. Then, we concrete the outputs of two 2dCNNs as the input of the stacked LSTM Networks. The stacked LSTM layers output the prediction results, indicating that the highlights will occur. To reflect the prediction lag, we train the model using different input data labeled by various time gaps. The training and testing procedures and the parameter settings of all learning layers will be illustrated in Section VI. In this paper, we utilize the features of eSports games as an example to design the SBP module in this framework, which can be easily revised to fit other livecast scenarios in crowdsourced livecast. For example, livecaster can only use mobile devices to broadcast live contents in Periscope. After collecting the live contents and the corresponding time-series from the sensors, e.g., GPS sensor and accelerometer sensor, on the mobile devices, a similar learning network can be trained to predict the highlights, which will be investigated in our future work.

B. Transcoding Task Assignment Optimization

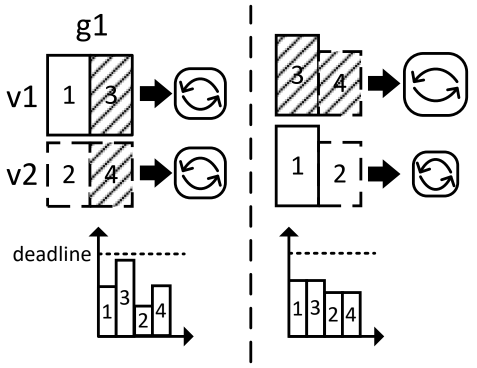


Fig. 8: Example of task assignment

In this subsection, we design the transcoding optimization. We assume a general transcoding framework, where a large number of transcoding tasks need to be assigned to different servers with various computational configurations. Because the highlight prediction module creates an opportunity to reserve and allocate resource earlier, we perform the transcoding task assignment according to the following two rules: 1) the workloads belonging to a predicted highlight should be prioritized, such that the transcoding time for these workloads can be reduced using high-performance servers; (2) other workloads should be finished before deadlines when viewers request them, such that our optimization does not impair viewers' QoE. Figure 8 illustrates the basic concept of the transcoding task assignment optimization. Gamecast $g1$ needs to be transcoded into two versions: $v1$ and $v2$. Based on the predictions, we already know the time when a highlight occurs (shadow areas); therefore, we first assign the workloads 3 and 4 on the high-performance server s1, and then assign the workloads 1 and 2 on the low-performance server $s2$. Finally, all transcoding tasks will be finished before their deadlines and the total transcoding latency is lower than the previous assignment. Yet how to allocate these tasks cost-effectively still need to be addressed. Next, we formulate this problem and propose a solution.

We denote (g, v) as a task that transcodes a gamecast g from the original quality version to quality version v. Without loss of generality, we use $S^{(T)}_{(g,v)}$ to denote the transcoding server that is assigned to task (g, v). Let $E^{(T)}[(g, v), s]$ denote the transcoding cost if task (g, v) is allocated to server s in time slot T. It can be calculated as follows:

$$E^{(T)}[(g,v),s] = \frac{c^{(T)}(g,v)}{\mathbb{C}(s)}\mathbb{P}(s)$$

where $c^{(T)}(g, v)$ is the amount of computation resource that is required by task (g, v) in time slot T. $\mathbb{C}(s)$ and $\mathbb{P}(s)$ are the computation capacity and unit price of a server s, respectively. We also denote the profit of transcoding task (g, v) on server s in time slot T as $F^{(T)}[(g, v), s]$, which can be calculated as follows:

$$F^{(T)}[(g,v),s] = W^{(T)}(g,v)log(\alpha - \beta l^{(T)}[(g,v),s])$$

where $W^{(T)}(g, v)$ is the predicted viewing number of the version v of gamecast g, which can be acquired according to the viewing number in previous time slots using regression models (e.g., ARIMA [30] used in this paper); $l^{(T)}[(g,v),s])$ is the transcoding latency of task (g,v) on server s. Base on the definitions of the transcoding cost and profit of tasks, we define the transcoding gain $TG^{(T)}[(g,v),s]$ when assigning task (g,v) on server s as follows:

$$TG^{(T)}[(g,v),s] = \frac{F^{(T)}[(g,v),s]}{E^{(T)}[(g,v),s]} \tag{1}$$

As such the optimization of transcoding tasks in time slot T is then formulated as follows:

$$\max \sum_{g} \sum_{v} TG^{(T)}[(g,v), S^{(T)}_{(g,v)}] \tag{2}$$

subject to:
Transcoding Latency Constraint:

$$\begin{aligned} l^{(T)}(g,v)H^{(T)}_{(g,v)} &\leq \frac{\sum_{t\in[0,T-1]} l^{(t)}(g,v)(1-H^{(t)}_{(g,v)})}{\sum_{t\in[0,T-1]}(1-H^{(t)}_{(g,v)})} \\ &\leq \gamma, \forall (g,v) \in G^{(T)} \end{aligned} \tag{3}$$

$$H^{(T)}_{(g,v)} = \begin{cases} 1, & \text{if task } (g,v) \text{ belongs to a highlight} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Resource Availability Constraint:

$$\sum_{g} \sum_{v} c^{(T)}(g,v) \leq \sum_{s} C^{(T)}(s) \tag{5}$$

where $H^{(T)}_{(g,v)}$ shows whether task (g,v) belongs to a highlight, as shown in equation (4). The optimization is to assign the transcoding tasks to different servers, so that the overall transcoding gain can be maximized. The rationale of transcoding constraint is as follows: (1) the tasks that belong to highlights should enjoy a similar transcoding time compared with other tasks; (2) all tasks should be completed in a period γ, which proposes a basic QoE requirement for the transcoding tasks in eSports gamecast services. The resource availability constraint guarantees that there exist enough computational resources for all transcoding tasks.

Because the Transcoding Latency Constraint is independent of the Resource Availability Constraint, we can first consider the former to remove the transcoding servers that do not meet it, and then this problem can be transformed into a minimum cost network flow problem. The minimum cost network flow problem [31] is briefly introduced as follows. Let $G = (N, E)$ be a directed network with a set N of n nodes and a set E of m edges. Every node $i \in N$ has an associated number $b(i)$ to denote its supply or demand depending on whether $b(i) > 0$ or $b(i) < 0$. Every edge $(i,j) \in E$ has an associated per unit flow cost c_{ij} and a flow capacity u_{ij}. The minimum cost flow problem can be formulated as follows.

$$Minimize \sum_{(i,j)\in E} c_{ij}x_{ij}$$

TABLE III: Performance of the SBP module

Approaches	Highlight			Non-highlight		
	R	P	F	R	P	F
CNN-based	0.84	0.89	0.86	0.81	0.86	0.83
LSTM-based	0.71	0.79	0.74	0.74	0.83	0.78
CF-G	0.99	0.66	0.79	0.74	0.99	0.85
CF-L	0.50	0.99	0.67	0.50	0.99	0.67
CF-GL	0.98	0.93	0.95	0.92	0.98	0.95

subject to:

$$\sum_{\{j:(i,j)\in E\}} x_{ij} - \sum_{\{j:(j,i)\in E\}} x_{ji} = b(i), \forall i \in N$$

$$0 \leq x_{ij} \leq u_{ij}, \forall (i,j) \in E$$

To transform our optimization to a minimum cost flow problem, we next introduce several transformation rules:

1) We introduce supply node v_{start} and demand node v_{end}.
2) For each transcoding task (g,v), we add a node $v_{(g,v)}$ and an edge $(v_{start}, v_{(g,v)})$, its capacity $u_{v_{start},v_{(g,v)}}$ is equal to the computational requirement of transcoding task (g,v);
3) For each transcoding server s, we add a node v_s and an edge (v_s, v_{end}), and its capacity $u_{v_s,v_{end}}$ is equal to the amount of available computational resource on server s;
4) If task (g,v) can be assigned to server s, we add an edge $(v_{(g,v)}), v_s)$ and its capacity $u_{v_{(g,v)},v_s}$ is equal to the computational requirement of transcoding task (g,v).
5) We set the cost from v_{start} to any task node $v_{(g,v)}$ and the cost from any server node v_s to v_{end} to 0, and set the cost of existing edge $(v_{(g,v)}, v_s)$ to $-[TG[(g,v),s]/c_{v_{(g,v)},v_s}]$[4].
6) To meet our optimization problem, we use the following constraints in the transformed minimum cost flow problem:

$$x_{v_{start},v_{(g,v)}} = u_{v_{start},v_{(g,v)}}, \forall v_{(g,v)} \in N$$

$$x_{v_{(g,v)}),v_s} = u_{v_{(g,v)}),v_s}, \forall v_{(g,v)}, \forall v_s \in N$$

$$x_{v_s,v_{end}} \leq u_{v_s,v_{end}}, \forall v_s \in N$$

After applying the above rules, we can transform our optimization problem (1) into a minimum cost flow problem. Such a problem can be solved by double scaling algorithm [31] in polynomial time. Let C denote the largest magnitude of any edge cost and U denote the largest magnitude of any edge capacity. The time complexity is $O(nmlogUlog(nC))$.

VI. Performance Evaluation

We have conducted trace-driven experiments to evaluate the performances of the prediction module SBP and the optimization module HAO in the framework CastFlag.

[4]Without loss of generality, we omit the time slot index T.

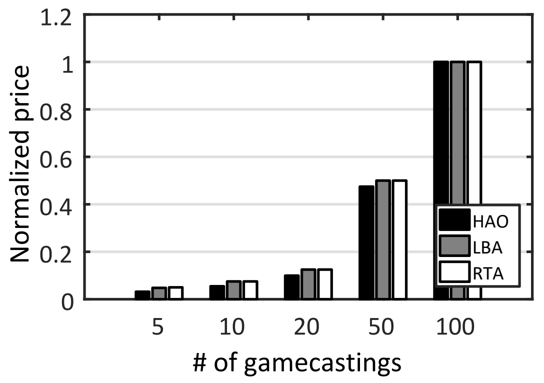


Fig. 9: Comparison of normalized transcoding expense under different # of broadcasters and three strategies

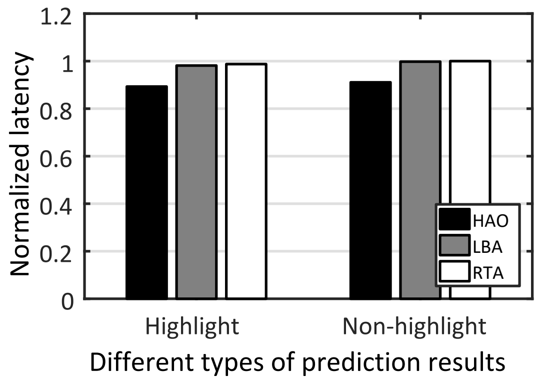


Fig. 10: Comparison of normalized transcoding latency under different types of segments and three strategies

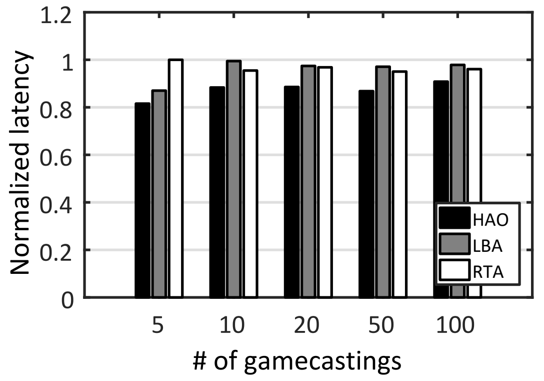


Fig. 11: Comparison of normalized transcoding latency under different # of broadcasters and three strategies

A. Strategy-based Prediction Results

We collect 80 game replays from Dota2. Based on an open-source package [32], we write an offline parser to extract interaction data from every replay into continuous time slots and recognize teamfights based on the damage and multiple kill data. According to the prediction rule mentioned in Section V, we label the data in different time slots using *highlight* or *non-highlight*. We train the proposed learning network using half of these data, and then test the prediction performance of the network using the remaining half. The deep learning based highlight prediction in the SBP module is implemented in Keras [33] with cuDNN on an NVIDIA GTX 1080 GPU. For the comparison of the SBP module, we have also implemented several networks with different settings, as shown in Table III. In our evaluation of this module, we use three typical metrics: Recall, Precision, and F-score [34], which combined together reflect the performance of the SBP module.

As shown in Table III, we compare the prediction module SBP with two typical approaches: (1) CNN-based approach only adopts CNNs to classify key events with full information; (2) LSTM-based approach only uses LSTM network with the full information. We select the best performance of all three approaches in different settings empirically. For our method, the setting is 1-layer CNN and 3-layer stacked LSTM networks, the prediction gap is 10 seconds. Here, we also investigate the impact of the global (G) and local (L) information in our design. We observe that our design achieves the highest score (up to 30%) in the three metrics.

B. Highlight-aware Optimization Results

For the comparison of the HAO module, we have implemented two strategies: Randomly Transcoding Allocation (RTA) and Load-Based Allocation (LBA). The former assigns a transcoding task on a randomly selected server from all available ones. The latter always selects the transcoding server with the lowest load for the current task. In the evaluation of this module, we adopt two metrics: transcoding latency and expense, which combined together illustrate the performance when using three strategies. We use the transcoding traces collected in Section II and the gamecast traces crawled using Twitch's official API [35] in one hour. We write three simulators to conduct the performance evaluation, which can choose the transcoding servers from three types of instances, i.e., A4, A5, and A6 and elastically lease/release them. Considering the transcoding settings in practical gamecast services, we assume that all gamecasts are transcoded from the original RTMP streaming to HLS segments with four quality versions, i.e., 1080p, 720p, 480p, and 360p [36], using the instances on Amazon EC2.

1) Transcoding Cost on the Cloud: In this experiment, because the instances can be leased elastically, they can satisfy all the transcoding requests of gamecasts. We calculate the cost of leasing Amazon EC2 instances as transcoding servers. In Figure 9, each bar represents the cost when a particular number of gamecasts are transcoded in one hour. For ease of comparison, the results are normalized by the maximum cost. It is noted that transcoding a large number of gamecasts generally consume larger computation resources. The reason is that all gamecasts have to be processed and transcoded, even if some of them do not have any viewer. For any number of gamecasts in this figure, we observe that the transcoding cost with our HAO module is less than or equal to that with the other two approaches.

2) Transcoding Latency: In the following experiments, we will evaluate the effectiveness of our HAO module. We focus on the average transcoding latency, which is the average time when transcoding a large number of gamecasts. For ease of comparison, the results are normalized by the maximum latency in Figures 10 and 11. Because our optimization strategy has knowledge of gamecast highlights, we first compare the effectiveness when using it to transcode the highlight and non-highlight segments. We fix the number of gamecast to 100. In Figure 10, because the HAO module gets the differences of highlight segments and non-highlight segments, the average transcoding latency of the highlight segments is slightly lower than that of the non-highlight segments. Moreover, both of them with our optimization strategy is lower than those with the other two comparisons. We next evaluate the impact of the different number of gamecasts. As shown in Figure 11, each bar represents the average transcoding latency when a particular number of gamecasts are transcoded in one hour. Our optimization approach achieves an average of 10% saving of the transcoding latency. In particular, transcoding latency is reduced by up to 20% when the number of the gamecasts is equal to five.

VII. Related Work

A. Event Prediction in eSports Games

Several pioneer works to detect the highlights and winning team in an eSports match have been presented in recent years. After parsing the replay or video of a match, these studies focus on the changes of the gamers' information, e.g., location, gold, and experience, and the dynamics of visual effects in the videos. Yang *et al.* [37] analyzed the replays using a sequence of event-related graphs and extracted key patterns to predict the successful team of the entire game. Drachen *et al.* [38] examined the spatiotemporal patterns of the gamers with four skill tiers, i.e., normal, high, very high, and professional. They further analyzed the relationship between game skills and match results. The work in [39] detected the highlights from the videos of eSports game matches using CNNs to learn the features of visual effects in the videos. Similarly, Chu *et al.* [25] recognized the designated text displayed on a game screen when key events occur as visual features. Their approach can also predict the emergence of a highlight in the next few seconds. These previous efforts motivate our work, but still face two limitations: (1) because the highlights can only be detected after they appear, existing approaches cannot meet the timeliness requirement of the optimization in eSports gamecast applications; (2) based on pre-recorded eSports videos, current prediction strategies can hardly guide the optimization due to the low accuracy and short forecasting gap. To overcome these limitations, we directly investigate the interactions among the gamers in an eSports game, closely explore the correlations between these interactions and highlights, and timely predict the highlights in the next minutes.

B. Optimization in Livecast Services

Some recent studies have already focused on the optimization in crowdsourced livecast services. Fan-Chiang *et al.* [37] investigated the importance of segment-of-interest in such services and optimized the resource allocation during the gamecast. Wu *et al.* [40] explored opportunities to combine the edge-based and cloud-based network resources. A novel framework to allocate the resources was designed to improve the ingesting performance in crowdsourced gamecast services. He *et al.* [41] proposed a fog-Based transcoding framework for livecast services that allocate the transcoding assignment to the massive viewers. Our work complements them by exploiting the hints that guide the optimization in advance, even before the encoding stage of a gamecast. Our work proposes a new design in the realm of utilizing the gamers' interactions in eSports games, where the cooperation and competition data are extracted into gamer-related, team-related and item-related features. Integrating these features into deep learning models offers a new opportunity to optimize the eSports gamecast services and improve the viewers' QoE.

VIII. Conclusion

In this paper, we have shown a strong evidence that in practical crowdsourced gamecast platforms like Twitch, the transcoding complexity and latency of an eSports gamecast closely depends on the gamers' interactions in the corresponding game match. Motivated by this observation, we explored opportunities and challenges to optimize the transcoding tasks in eSports gamecast services. We presented *CastFlag*, a generic framework that investigates eSports gamers' interactions and predicts game highlights to optimize the corresponding transcoding tasks cost-effectively. Specifically, we first designed a strategy-based prediction (SBP) module, which mainly adopts a deep learning based network to predict highlights in real-time. We further proposed a highlight-aware optimization (HAO) module, which receives the prediction results from the SBP module and select transcoding servers cost-effectively. Extensive evaluations driven by traces from Amazon EC2 and Twitch illustrated the high prediction accuracy of the SBP module as well as the cost-effectiveness and superior transcoding performance of the HAO module.

We are currently designing a novel deep learning network to forecast game highlights with various prediction gaps in different game stages. We are also interested in enhancing the HAO module to provide gamecast delivery and cache optimization according to the prediction results from the SBP module. Moreover, we are collecting data traces from other crowdsourced livecast services, e.g., Periscope, to verify the effectiveness of the CastFlag framework in general.

Acknowledgment

This work is supported by the NSFC under Grant No. 61902369 and a Canada NSERC Discovery Grant. The work of Z. Wang is supported by the NSFC under Grant No. 61872215 and the SZSTI under Grant No. JCYJ20180306174057899. The work of L. Sun is supported by the NSFC under Grant No. 61936011 and Beijing Key Laboratory of Networked Multimedia.

References

[1] Douyu. [Online]. Available: https://www.douyu.com

[2] J. Jiang, R. Das, G. Ananthanarayanan, P. A. Chou, V. Padmanabhan, V. Sekar, E. Dominique, M. Goliszewski, D. Kukoleca, R. Vafin, and H. Zhang, "Via: Improving internet telephony call quality using predictive relay selection," in *ACM SIGCOMM*, 2016.

[3] M. Torres Vega, D. C. Mocanu, and A. Liotta, "Unsupervised deep learning for real-time assessment of video streaming services," *Multimedia Tools and Applications*, May 2017.

[4] A. Zhou, H. Zhang, G. Su, L. Wu, R. Ma, Z. Meng, X. Zhang, X. Xie, H. Ma, and X. Chen, "Learning to coordinate video codec with transport protocol for mobile video telephony," in *ACM MobiCom*, 2019.

[5] J. Jiang, S. Sun, V. Sekar, and H. Zhang, "Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation," in *USENIX NSDI*, 2017.

[6] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun, "Comyco: Quality-aware adaptive video streaming via imitation learning," in *ACM Multimedia*, 2019.

[7] Y. Guan, C. Zheng, X. Zhang, Z. Guo, and J. Jiang, "Pano: Optimizing 360 video streaming with a better understanding of quality perception," in *ACM SIGCOMM*, 2019.

[8] J. Bickerton, "Esports set to rise to £1.1bn," https://goo.gl/p41jrc, March 2018.

[9] R.-X. Zhang, T. Huang, M. Ma, H. Pang, X. Yao, C. Wu, and L. Sun, "Enhancing the crowdsourced live streaming: A deep reinforcement learning approach," in *ACM NOSSDAV*, 2019.

[10] Periscope. [Online]. Available: https://www.pscp.tv

[11] O. B. Software. [Online]. Available: https://obsproject.com

[12] Douyu, "Popular game ranking," 2018. [Online]. Available: https://www.douyu.com/directory/rank_list/PCgame

[13] Superdata, "Superdata reports games and interactive media earned a record $120.1b in 2019," 2020. [Online]. Available: https://www.superdataresearch.com/blog/superdata-reports-games-and-interactive-media-earned-a-record-1201b-in-2019

[14] SullyGnome.com, "Most watched games on twitch over the past 365 days," July 2019. [Online]. Available: https://sullygnome.com/games/365/watched

[15] Adobe, "Real-time messaging protocol (rtmp) specification — adobe developer connection," 2012. [Online]. Available: https://www.adobe.com/devnet/rtmp.html

[16] B. Wang, X. Zhang, G. Wang, H. Zheng, and B. Y. Zhao, "Anatomy of a personalized livestreaming system," in *ACM IMC*, 2016.

[17] Apple, "Http live streaming (hls) - apple developer," 2017. [Online]. Available: https://developer.apple.com/streaming

[18] K. Pires and G. Simon, "Youtube live and twitch: A tour of user-generated live streaming systems," in *ACM MMSys*, 2015.

[19] T. Y. Fan-Chiang, H. J. Hong, and C. H. Hsu, "Segment-of-interest driven live game streaming: Saving bandwidth without degrading experience," in *IEEE NetGames, 2015*.

[20] C. Zhang, J. Liu, M. Ma, L. Sun, and B. Li, "Seeker: Topic-aware viewing pattern prediction in crowdsourced interactive live streaming," in *ACM NOSSDAV*, 2017.

[21] Amazon, "Amazon elastic compute cloud," 2018. [Online]. Available: https://aws.amazon.com/ec2

[22] Dotabuff, "Github - dotabuff/manta: Dota 2 source 2 replay parser," 2018. [Online]. Available: https://github.com/arut/nginx-rtmp-module

[23] W.-T. Chu and Y.-C. Chou, "On broadcasted game video analysis: Event detection, highlight detection, and highlight forecast," *Multimedia Tools and Applications*, vol. 76, no. 7, pp. 9735–9758, April 2017.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[25] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 677–691, April 2017.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Interspeech*, vol. 2, 2010, p. 3.

[28] G. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159 – 175, 2003.

[29] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., 1993.

[30] GitHub, "Github - dotabuff/manta: Dota 2 source 2 replay parser," 2018. [Online]. Available: https://github.com/dotabuff/manta

[31] Keras. [Online]. Available: https://keras.io

[32] J. Han, J. Pei, and M. Kamber, *Data Mining: Concepts and Techniques*. Elsevier, 2011.

[33] Twitch, "Twitch developers - home," 2018. [Online]. Available: https://dev.twitch.tv

[34] ——, "Twitch streamers - twitch video encoding/bitrates/and stuff," 2017. [Online]. Available: https://stream.twitch.tv/encoding

[35] P. Yang, B. E. Harrison, and D. L. Roberts, "Identifying patterns in combat that are predictive of success in moba games," in *Proceedings of the 9th International Conference on the Foundations of Digital Games*, 2014.

[36] A. Drachen, M. Yancey, J. Maguire, D. Chu, I. Y. Wang, T. Mahlmann, M. Schubert, and D. Klabajan, "Skill-based differences in spatio-temporal team behaviour in defence of the ancients 2 (dota 2)," in *IEEE GEM*, 2014.

[37] Y. Song, "Real-time video highlights for yahoo esports," *CoRR*, vol. abs/1611.08780, 2016.

[38] C. Wu, Z. Wang, J. Liu, and S. Yang, "Crowdsourced live streaming over aggregated edge networks," in *IEEE GLOBECOM*, 2016.

[39] Q. He, C. Zhang, X. Ma, and J. Liu, "Fog-based transcoding for crowdsourced video livecast," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 28–33, April 2017.