Towards Extracting Highlights From Recorded Live Videos: An Implicit Crowdsourcing Approach

Ruochen Jiang*
Simon Fraser University
ruochenj@sfu.ca

Changbo Qu*
Simon Fraser University
changboq@sfu.ca

Jiannan Wang
Simon Fraser University
jnwang@sfu.ca

Chi Wang
Microsoft Research
wang.chi@microsoft.com

Yudian Zheng
Twitter
yudianz@twitter.com

ABSTRACT

Live streaming platforms need to store a lot of recorded live videos on a daily basis. An important problem is how to automatically extract highlights (i.e., attractive short video clips) from these massive, long recorded live videos. One approach is to directly apply a highlight detection algorithm to video content. While various algorithms have been proposed, it is still hard to generalize them well to different types of videos without large training data or high computing resources. In this paper, we propose to tackle this problem with a novel implicit crowdsourcing approach, called Lightor. The key insight is to collect users' natural interactions with a live streaming platform, and then leverage them to detect highlights. Lightor consists of two major components. Highlight Initializer collects time-stamped chat messages from a live video and then uses them to predict approximate highlight positions. Highlight Extractor keeps track of how users interact with these approximate highlight positions and then refines these positions iteratively. We find that the collected user chat and interaction data are very noisy, and propose effective techniques to deal with noise. Lightor can be easily deployed into existing live streaming platforms, or be implemented as a web browser extension. We recruit hundreds of users from Amazon Mechanical Turk, and evaluate the performance of Lightor on real live video data. The results show that Lightor can achieve high extraction precision with a small set of training data and low computing resources.

1 INTRODUCTION

Video data is booming and will account for 90% of all internet traffic by 2020 as predicted by Cisco [11]. Applying data science to improve video-related services is of growing interest in the data mining and database community [13, 17, 18, 27]. As an important type of video service, live streaming platforms such as Twitch, Mixer, YouTube Live, and Facebook Live fulfill the mission of democratizing live video broadcasting. With these platforms, anyone can be a broadcaster to record a video and broadcast it in real time; anyone can be a viewer to watch the live video and chat about it in real time. This unique experience makes these platforms more and more popular nowadays. For example, by 2018, Twitch has reached 3.1 million unique monthly broadcasters, and over 44 billion minutes of videos are watched each month [3].

Once a live stream is complete, the recorded video along with time-stamped chat messages will be archived. A recorded video is often very long (from half an hour to several hours). Many users do not have the patience to watch the entire recorded video but only look for a few *highlights* to watch. A highlight represents a small part of the video that makes people feel excited or interested, and it typically lasts from a few seconds to less than one minute. For example, a highlight in a Dota2 game video could be an exciting battle or a critical knockdown.

We study how to automatically extract highlights from a recorded live video. The impact of this work on live-streaming business is two folds. First, it save users' time in manually finding the highlights, potentially increasing the user engagement of a live streaming platform. Second, it is a fundamental task in video processing, enabling a live streaming platform to improve other profitable video applications (e.g., video search, video recommendation).

One approach is to apply an existing highlight detection algorithm to video content [5, 7, 21, 30, 31]. However, these algorithms either only work for a certain type of video (e.g., Baseball [21], Soccer [5]), or require large training data and high computing resources [7, 30, 31]. See Section 2 for a more detailed discussion of related work.

Unlike these existing works, we propose a novel *implicit crowdsourcing* to tackle this problem [2]. Implicit crowdsourcing is the idea of collecting implicit feedback from users (i.e., user's *natural* interactions with the system) and then leveraging the feedback to solve a challenging problem. It has achieved great success in many domains. For example, reCAPTCHA [25] leverages this idea to digitize old books. Search engines collect implicit clickthrough data to optimize web search ranking [12]. To apply this idea, we face two challenges. The first one is how to design an implicit crowdsourcing workflow so that video viewers interact with the system naturally but provide useful feedback implicitly. The second one is how to use the implicit (and noisy) feedback to detect and extract video highlights. We address these two challenges as follows.

Implicit Crowdsourcing Workflow. We design a novel implicit crowdsourcing workflow, called Lightor. Lightor consists of two components. i) Highlight Initializer takes a recorded live video as input and uses its time-stamped chat messages to detect which part of the video could have a highlight. For example, when a large number of chat messages pop up within a short period of time, users may talk about a highlight that has just happened. Note that Highlight Initializer can only get an approximate position of a highlight. It is still not clear about the exact boundary (i.e., exact start and end points) of a highlight. ii) Highlight Extractor is designed to address this problem. At each approximate position, it puts a "red dot" on the progress bar of the video, which informs users that there could be a highlight at this position. Note that users will not be forced to watch this highlight. Instead, they can watch the video as usual. Highlight Extractor collects user interaction data w.r.t. each red dot to identify the exact boundary of each highlight.

*Both authors contributed equally to this research.

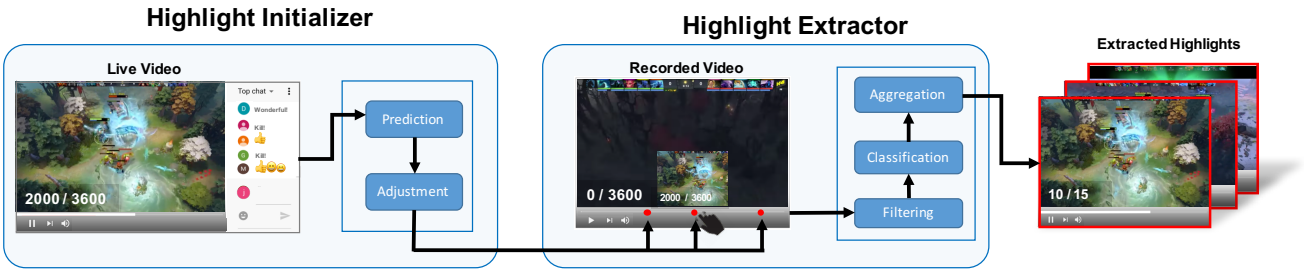


Figure 1: Lightor: An implicit crowdsourcing workflow for extracting highlights from a recorded live video.

Noisy User Data. One major challenge in our implicit crowdsourcing design is how to handle the high noise in the implicit feedback from users. For example, in Highlight Initializer, when a user leaves a chat message, she might not comment on the video content but chat with other users. In Highlight Extractor, when a user watches a certain part of video, she might not be attracted by the video content but check whether this part of video has something interesting. Therefore, we have to be able to separate noise from signal. We analyze real-world user data and derive a number of interesting observations. Based on these observations, we develop several effective techniques to deal with noisy user interaction data.

Lightor can be easily deployed on an existing live streaming platform. The only change is to add red dots to the progress bar of recorded videos. Based on a user survey, we find that most users prefer this change since red dots help them find more interesting highlights. Furthermore, Lightor can be implemented as a web browser extension, which has the potential to support any platform.

We recruit about 500 users from Amazon Mechanical Turk and evaluate Lightor using real live video data from Twitch. The results show that (1) our proposed data-science techniques make Lightor achieve very high precision (up to 70% – 90%) in the returned top-k highlights, which changes the system from unusable to usable, and (2) Lightor requires 122× fewer training examples and over 10000× less training time compared to the state-of-the-art deep learning based approach, thus it is much preferable when there is a lack of training data or computing resources.

To summarize, our contributions are:

- We study how to leverage implicit crowdsourcing to extract highlights from a recorded live video. We propose Lightor, a novel workflow to achieve this goal.
- Lightor consists of Highlight Initializer and Highlight Extractor. For each component, we analyze real-world user data and propose effective techniques to deal with the noisy user data.
- Lightor can be easily deployed on an existing live streaming platform or implemented as a web extension to support any platform. We evaluate Lightor using real data and real users. The results show that Lightor can achieve high precision with a small set of training data and low computing resources.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents the Lightor workflow. We discuss how Highlight Initializer and Highlight Extractor are built in Section 4 and Section 5, respectively. Section 6 discusses how to deploy Lightor in practice. Experimental results are presented in Section 7. We discuss our findings and lessons learned in Section 8, and present conclusions and future work in Section 9. We provide a reproducibility report in the Appendix, and release all the code and datasets at the project page: http://tiny.cc/lightor.

2 RELATED WORK

Computer Vision. There is a recent trend to apply deep learning to highlight detection [7, 24, 31]. For example, a frame-based CNN model [24] was trained to detect the frames with significant visual effects for e-sports. In [7], a joint model of CNN on video and LSTM on chat was trained to detect highlights in game videos. While these deep-learning based approaches achieve good performance, they require large training sets and high computing resources. Unlike these studies, we focus on the use of implicit crowdsourcing which requires much less training data and computational cost. In addition to deep learning, there are some other algorithms proposed for highlight detection, but they are mainly designed for a certain type of video (e.g., Baseball [21], Soccer [5]) rather than focus on a general approach that works for any video type. There are also many studies on video summarization [19, 31], which aim to generate a condensed video to summarize the story of the entire video. Highlight detection often serves as the first step of video summarization and generates a small number of candidate highlights.

Explicit Crowdsourcing. There are some works using explicit crowdsourcing for video analysis [10, 14, 26, 29]. That is, they ask crowd workers to do a certain video-related task explicitly, e.g., video segmentation [14], video tagging [29]. However, none of these studies attempt to apply implicit crowdsourcing to video highlight detection, which is a more monetary-cost efficient and natural way to collect essential data.

Implicit Crowdsourcing (User Comments). There are some works on the use of user comments [20, 27] for video analysis. A LDA model was proposed to generate video tags from time-stamped comments [27]. Another work uses word embedding to extract highlights from time-stamped comments for movies [20]. They are different from Lightor in three aspects. (1) They only focus on user commenting data while Lightor considers both user commenting data and user viewing behavioral data (see Section 5). (2) They use bag of words or word embedding as features while Lightor use more general features (see Section 4.2). (3) They use time-stamped comments rather than live chat messages, thus they do not face the challenge that there is a delay between video content and the comments (see Section 4.3).

Twitter data has been leveraged to detect events in some studies [6, 9, 22, 28]. However, live chat messages are usually shorter and more noisy, thus requiring the development of new techniques.

Implicit Crowdsourcing (User Viewing Behaviors). HCI and Web researchers have designed systems using click-through or interaction data to measure user engagement. For example, the research on MOOC videos or how-to videos leverage interactions as engagement measurement to detect interesting or difficult parts of videos (e.g., [4], [15]). Some studies have also leveraged interaction data to predict audience's drop-out rate and analyzed the cause of interaction peak [16, 23]. These works simply sum up all users' watching sessions along the video and get curves between watched frequency and video timestamps. We have tried this simple method, but found that it did not perform well on our collected user interaction data since when users interact with a casual video, their viewing behaviors are much more unpredictable.

3 THE LIGHTOR WORKFLOW

Figure 1 depicts the Lightor workflow. The workflow consists of two major components: Highlight Initializer and Highlight Extractor. The former determines which part of the video could be a highlight, and the latter identifies the exact boundary (start and end time) of each highlight. We will use a simple example to illustrate how they work as well as the challenges that they face. Consider a one-hour video $\mathcal{V} = [0, 3600]$, which starts at 0s and ends at 3600s. Suppose the video has a highlight between 1900s and 2005s, denoted by $h = [1990, 2005]$. The goal is to extract h from $\mathcal{V}$.

Highlight Initializer. Highlight Initializer aims to identify an approximate start position of each highlight so that if a user starts watching the video from this position, she can tell that there is a highlight nearby. For example, 2000 is a good position since it is within the highlight range [1990, 2005] but 2100 is a bad one since it is very far away from the highlight. We observe that most live videos allow users to leave chat messages in real time. This can be leveraged as a kind of implicit feedback. However, time-stamped chat messages are short and noisy, thus it is challenging to use them to implement an accurate Highlight Initializer. We will discuss how to address this challenge in Section 4.

Highlight Extractor. Suppose the above component returns 2000 as a result. We will add a "red dot" at this position on the progress bar of the recorded video (see Figure 1). A red dot can be seen as a hint, which informs users that there could be a highlight nearby. Users can click on the red dot and start watching the video. If they indeed see a highlight, they may want to watch the highlight again by dragging the progress bar backward. This user interaction data can be leveraged as another kind of implicit feedback since users help revise the start and end positions of the highlight implicitly. However, users behave quite differently. It is challenging to leverage the user interaction data to implement an accurate Highlight Extractor. We will discuss how to address this challenge in Section 5.

4 HIGHLIGHT INITIALIZER

This section presents the design of our Highlight Initializer component. We first define the design objective, then discuss the design choices, and finally propose the detailed implementation.

4.1 Design Objective

There could be many highlights in a video, but most users are only interested in viewing the top-k ones. Highlight Initializer aims to find an approximate start position for each top-k highlight.

Next, we formally define what is a *good* approximate start position (i.e., what is a good red dot). The goal is to make users see a highlight shortly after they start watching the video from a red dot. Let $h = [s, e]$ denote a highlight and r denote the red dot w.r.t. h. We call r a good red dot if it meets three requirements.

First, the red dot should not be put after the end of the highlight (i.e., $r \leq e$). Otherwise, a user is very likely to miss the highlight. This is because a user typically clicks the red dot r and starts watching the video for a short period of time. If nothing interesting happens, she may skip to the next red dot. Second, the red dot should not be put at more than 10s before the start of the highlight (i.e., $r \geq s - 10$). Based on existing studies (e.g., [1]), people can accept less than 10s delay, but may lose their patience when the delay is longer. Third, it is not useful to generate two red dots that are very close to each other. Thus, we require that there does not exist another red dot r' such that $|r - r'| \leq \delta$, where δ is a system parameter and is set to 120s by default.

With the definition of good red dots, we formally define the design objective of Highlight Initializer.

Objective. *Given a recorded live video along with time-stamped messages, and a user-specified threshold k, Highlight Initializer aims to identify a set of k* good *red dots.*

4.2 Design Choices

We face different choices when designing the Highlight Initializer. We will explain how the decision is made for each choice.

Video vs. Chat Data. We choose to only use chat data instead of video data to identify red dots. This design choice has two advantages. First, we can use a small set of training data (e.g., 1 labeled video) to train a good model over chat data. But, it is hard to achieve this for video data. Second, processing video data often requires high computing resources. Since chat data is much smaller in size than video data, this limitation can be avoided. On the other hand, a chat-data based approach may not work well for videos with few chat messages. Nevertheless, as will be shown in the experiment, our model performs well on the videos with 500 chat messages per hour. We find that the majority (more than 80%) of popular videos in Twitch meet this requirement. For the remaining unpopular videos, there may not be a strong demand to generate highlights for them.

General vs. Domain-specific Features. We seek to build a Machine Learning (ML) model to identify red dots. There are two kinds of features that can be used by the model. General features are applicable to any domain. For example, *message number* can be seen as a kind of general feature because we can extract this feature for any type of video and use it as a strong predictor for highlights. In contrast, domain-specific features are highly dependent on the selected domains. For example, the keyword "Goal" is a domain-specific feature since it can be used to detect highlights in a Soccer game, but not in a Dota game. We choose to use general features rather than domain-specific features. This will allow us to handle a large variety of video types in live streaming platforms.

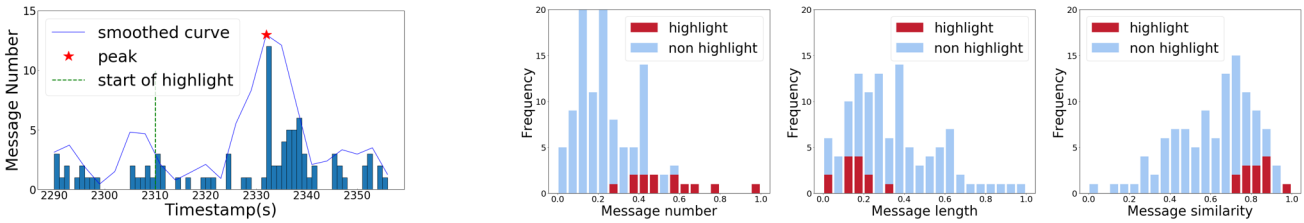
4.3 Implementation

We implement the Highlight Initializer component based on the above design choices. In the following, we first present a naive implementation and identify its limitations. We then propose our implementation to overcome these limitations.

4.3.1 *Naive Implementation.*

A naive implementation is to count which part of the video has the largest message number and put a red dot at that position. Figure 2(a) shows a real-world example (without otherwise specified, the reproducibility of all the experimental figures can be found in the Appendix). It plots a histogram along with the smoothed curve of the message number in a Twitch live video. We can see that 2332s has the largest message number, thus this naive implementation will put a red dot at 2332s.

Unfortunately, this implementation does not perform well in practice due to two reasons. The first reason is that having the largest message number does not always mean that users are chatting about a highlight. For instance, there could be advertisement chat-bots which post quite a few messages in a very short period of time. The second reason is that in a live video, users



(a) The change of message number over time

(b) The comparison of the feature-value distributions between highlights and non-highlights

Figure 2: Analysis of the Chat Data in a Twitch Video.

will only chat about a highlight *after* they have seen a highlight. Thus, there is a delay between the start position of a highlight and its comments. For example, in Figure 2(a), we can see the delay (the distance between the green dotted line and the red dot) is around 20s. This naive implementation fails to capture the delay.

4.3.2 *Our Implementation.*

Our implementation consists of two stages.

Prediction. The prediction stage aims to address the first issue mentioned above. Given a set of chat messages within a short sliding window (e.g., 25s), we build a predictive model to determine whether the messages in the sliding window are talking about a highlight or not. We propose three general features for the model.

- *Message Number* is the number of the messages in the sliding window. The naive implementation only considers this feature.
- *Message Length* calculates the average length of the messages in the sliding window, where the length of a message is defined as the number of words in the message. We observe that if viewers see a highlight, they tend to leave short messages. If their messages are long, they typically chat about something else.
- *Message Similarity* measures the similarity of the messages in the sliding window. If the messages are similar to each other, they are more likely to chat about the same topic, instead of random chatting. We use Bag of Words to represent each message as a binary vector and apply one-cluster K-means to find the center of messages. The message similarity is computed as the average similarity of each message to the center.

To make these features generalize well, we normalize them to the range in $[0, 1]$ and build a logistic regression model to combine them. We examine the effectiveness of each feature on Twitch chat data. Figure 2(b) shows the analysis results of a random video. The video contains 1860 chat messages in total. We divide them into 109 non-overlapping sliding windows, where 13 are labeled as highlights and 96 are labeled as non-highlights. For each feature, we compare the feature-value distributions of highlights and non-highlights. We can see that their distributions are quite different. For example, for the message-length feature, all the highlights are between 0 and 0.4, but non-highlights can be any length.

Adjustment. The adjustment stage aims to overcome the second limitation of the naive implementation. Given a set of messages in a sliding window which are predicted to be talking about a highlight, we aim to estimate the start position of the highlight.

The key observation is that people can only comment on a highlight after they have seen it. Based on this observation, we first detect the *peak* in the sliding window, where a peak represents the time when the message number reaches the top. After that, we train a model to capture the relationship between the peak's position ($\text{time}_{\text{peak}}$) and the highlight's start position ($\text{time}_{\text{start}}$).

The current implementation considers a simple linear relationship, i.e., $\text{time}_{\text{start}} = \text{time}_{\text{peak}} - c$, where c is a constant value. We can learn the optimal value of c from training data. Specifically, for each labeled highlight i, the highlight's start position is denoted by $\text{time}_{\text{start}\,i}$. Since it is predicated as $\text{time}_{\text{peak}\,i} - c$, the red dot will be put at $\text{time}_{\text{peak}\,i} - c$. Our goal is to identify as many good red dots as possible. Thus, we aim to find the best c such that

$$\arg\max_{c} \sum_{i} reward(\text{time}_{\text{peak}\,i} - c,\ \text{time}_{\text{start}\,i}),$$

where $reward(\cdot) = 1$ if it is a good red dot; $reward(\cdot) = 0$, otherwise.

Once c is obtained, we can use it to get the red dot positions. For example, suppose the learned c = 20s. It means that we will move the peak backward by 20s. Imagine $\text{time}_{\text{peak}}$ = 2010s. Then we will select $\text{time}_{\text{peak}}$ - 20s = 1990s as a red dot's position. This simple linear relationship leads to good performance as shown in later experiments. We defer the exploration of more complex relationships to future work.

Algorithm Description. To put everything together, Algorithm 1 shows the pseudo-code of the Highlight Initializer component. The input consists of the set M of all the time-stamped messages of video v, the video length t, the number of one's desired highlights k, the sliding window size l, the adjustment value c and Trained Logistic Regression Model, $LRmodel$. The output is the highlight sliding window list, $H = \{(s_j, e_j) | j = 0, ..., k-1\}$, where (s_j, e_j) is respectively the start and end time of a sliding window j.

In line 1, we initially generate the sliding window list $W = \{(s_i, e_i) | i = 0, ..., n\}$. When two sliding windows have an overlap, we keep the one with more messages. From line 2 to line 6, for each sliding window $w_i = (s_i, e_i)$, we apply the trained logistic regression model on the feature vector $f_i = (num_i, len_i, sim_i)$ which is extracted from the massages whose timestamps are in the range of (s_i, e_i). Lines 7 and 8 retrieve the top-k highlight sliding windows H. In Top function, we make sure that H does not contain too close highlights. From line 9 to 11, we adjust the start time by c for each sliding window in H. Finally, we return H as an output.

Algorithm 1: Highlight Initializer for one video v.

Input : M: all the messages; t: video length; k: # of desired highlights; l: sliding window size; c: adjustment value; $LRmodel$: trained logistic regression model

Output: H: top-k sliding window list.

```
1  W ← get_sliding_wins(M, l)
2  foreach sliding window W[i] do
3  |  f_i ← Feature_vec(W[i], M) // Normalized f=(num, len, sim)
4  |  p_i ← LRmodel.predict(f_i) // Get predicted probability
5  |  W[i] ← W[i].append(p_i)
6  end
7  W_sorted ←Sort W by p
8  H ←Top(k, W'_sorted)
9  foreach highlight window H[j] do
10 |  H[j] ← (s_j − c, e_j) // Adjustment
11 end
12 return H
```

5 HIGHLIGHT EXTRACTOR

This section presents the design of our Highlight Extractor component. We first define the design objective, then discuss the challenges, and finally propose the detailed implementation.

5.1 Design Objective

The Highlight Extractor aims to identify the boundary (start and end positions) of each highlight using user interaction data.

User Interaction Data. While watching a video, a user may have different kinds of interactions with the video (e.g., Play, Pause, Seek Forward, and Seek Backward). We analyze user interaction data, and find that if users often watch the same part of video again and again, this part is very likely to be a highlight. Based on this observation, we transform user interaction data into *play data*, where each record is in the form of: $\langle user, \text{play}(s, e)\rangle$. For example, $\langle$Alice, play(100, 120)$\rangle$ means that the user Alice starts playing the video at 100s, and stops at 120s. If the context is clear, we will abbreviate it as play(s, e) and call play(s, e) a *play*.

We leverage the play data to extract highlights. Note that if a play is far away from a red dot, it may be associated with another highlight. Thus, we only consider the plays within $[-\Delta, \Delta]$ around a red dot (Δ = 60s by default).

The following formally defines the objective.

Objective. *Given the play data play$(s_1, e_1), \cdots,$ play(s_n, e_n) w.r.t. a red dot, Highlight Extractor aims to identify the start and end positions of the highlight of the red dot.*

5.2 Challenges

There are several challenges in order to achieve the objective.

How to filter play data? Play data could be very noisy. For example, a user may randomly pick up a position s, and watch for a few seconds (e.g., 5s) to check whether this part of video is interesting or not. If uninteresting, she may jump to another position. Obviously, we should filter this play$(s, s + 5)$ since it cannot be interpreted as the user enjoying watching $[s, s + 5]$. Note that this example only shows one type of noisy play. There could be many others that need to be identified and filtered.

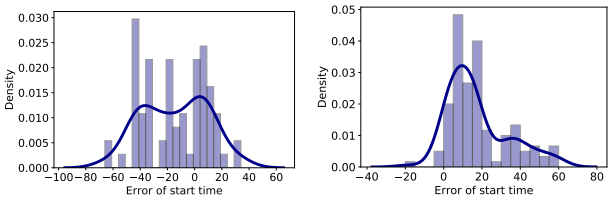
How to aggregate play data? Let play$(s'_1, e'_1), \cdots,$ play(s'_m, e'_m) denote the play data after the filtering. Each play can be considered as a vote for the highlight. For example, play(1990, 2010) means that the user votes 1990s and 2010s as the start and end positions of the highlight. Users may have different opinions about the highlight. We can aggregate their opinions using median because it is robust to outliers. Thus, the new start and end positions are computed as median$(s'_1, s'_2, \cdots, s'_m)$ and median$(e'_1, e'_2, \cdots, e'_m)$.

Unfortunately, when applying this idea to real-world user interaction data, it does not always work well. We have a very interesting observation: whether this idea works well or not strongly depends on the relative position between the red dot and the highlight. There are two possible relative positions:

Type I: the red dot is *after* the end of the highlight;

Type II: the red dot is *before* the end of the highlight.

Since many users start watching the video from a red dot, if they do not find anything interesting, they may skip to the next red dot. Imagine the red dot is put after the end of the highlight (i.e., Type I). Many users may miss the highlight, thus their play data are not reliable indicators of the highlight. Imagine the red dot is put before the start of the highlight (i.e., Type II). Many users will watch the same highlight, thus their play data follow a similar pattern.



(a) Type I **(b) Type II**

Figure 3: Distribution of the difference between each play's start position and the ground-truth start position.

To further examine this observation, we calculate the difference of each play's start position and the ground-truth start position. Figure 3(a) shows the distribution of all plays of Type I. We can see the curve approximately follows a uniform distribution between -40 and +20. It shows that the play activities for Type I are quite diverse. Users may either play back randomly in order to find the highlight or skip to the next highlight. In comparison, Figure 3(b) shows the distribution of all plays of Type II. We can see the curve approximately follows a normal distribution. It implies that most plays for Type II correspond to highlight watching.

This observation poses two new questions. The first one is that given a red dot, how to determine whether it belongs to Type I or Type II? The second one is that after a red dot is classified as Type I (or Type II), how to aggregate its play data?

5.3 Implementation

We propose a series of techniques to address these challenges. The following shows our detailed implementation.

Filtering. The main idea is to filter the plays that are not about watching the highlight but about doing something else (e.g., looking for a highlight). We observe that if a play is far away from the red dot, it typically does not cover the highlight. Thus, we remove such plays from the data. We also notice that if a play is too long or too short, it tends to have little value. A too short play could indicate that viewers watch for a few seconds and find it uninteresting, while a too long play means that viewers may be watching the entire video. Thus, we remove such plays from the data. Third, there could be some outliers, i.e., the play that is far away from the other plays. We adopt an outlier detection method to find the outliers and then remove them. More details about this filtering process can be found in the Appendix.

Classification. Given a red dot, we build a classification model to determine whether it belongs to *Type I* or *Type II*. More specifically, we need to classify the relative position between the red dot and the end of the highlight into *Type I* or *Type II*. We find that this (unknown) relative position has a strong correlation with the (known) relative position between the red dot and observed play data. Therefore, we identify the following three features.

- *# Plays after red dot* computes the number of plays which start at or after the red dot.
- *# Plays before red dot* computes the number of plays which end before the red dot.
- *# Plays across red dot* computes the number of plays which starts before the red dot and ends after the red dot.

Figure 4 shows an example to illustrate the three features. For Type I, since the highlight ends before the red dot, some users play before or across the red dot in order to find the highlight.

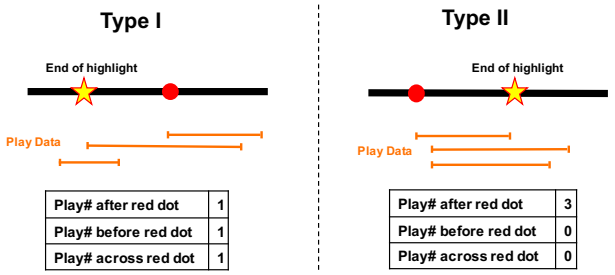


Figure 4: An illustration of three features for classifying the relative position between red dot and the end of highlight.

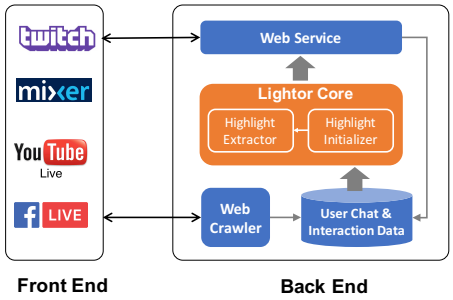


Figure 5: Lightor Web Browser Extension.

In comparison, there is no such play in Type II since if a user starts watching the video at the red dot, she will see the highlight. Our experiments show that our classification model achieves high accuracy (around 80%).

Aggregation. Different aggregation strategies are proposed for Type I and Type II, respectively.

For Type II, as Figure 3(b) shows, the play patterns of most users are similar. The median of the start time offsets is between 5 and 10. This is because that the most exciting part of the highlight usually happens a few seconds after its start point, which causes users to skip the beginning of the highlight. This kind of error is tolerable. Therefore, we can use median to aggregate their play data.

For Type I, as seen from Figure 3(a), the distribution of start time offsets is rather random. Therefore, we need to collect more play data. Our main idea is to convert a red point from Type I to Type II. Given that Type II can collect high-quality play data, once a red dot is converted to Type II, we can get high-quality play data as well. Specifically, once a red dot is classified as Type I, we will move it backward by a constant time (e.g., 20s) and collect new interaction data based on the new red dot location. If the new red dot is classified as Type II, we apply the Type II's aggregation approach; otherwise, we move it backward by another 20s.

Algorithm Description. To put everything together, Algorithm 2 shows the pseudo-code of the entire Highlight Extractor component. The input consists of a highlight $h = (s, e)$, and a moving duration m for $Type_I$, which is a constant described above to convert a $Type_I$ to a $Type_{II}$. The output is the updated $h = (s', e')$.

In lines 2 and 3, we get the user interactions I for the current h and filter them to get a list of plays, P. In lines 4 and 5, we extract the feature f from P and perform the binary classification to decide h's *label*. From lines 6 to 14, as we describe above in Aggregation, we update $h = (s', e')$. If *label* is Type II, it means the red dot is before the end of the highlight. From lines 7 to 10, we remove the plays whose ends are before the red dot. Then we calculate the median to update h. If *label* is Type I, it means the red dot is after the end of the highlight. So we move $h.s$ backward by m. We iterate this procedure until the red dot position is stable (e.g., $|h.s - h.s'| < \epsilon$).

Algorithm 2: Highlight Extractor for one highlight h

Input : Highlight $h = (s, e)$;
Moving duration for Type I, m.
Output: Highlight refined boundaries, $h = (s', e')$.

```
1  repeat
2      I ← get_interact()
3      P ← filter(I) // Filtering
4      f ← feature_vec(P) // f=(before_red, after_red, across_red)
5      label ← classification(f) // Classification
       // Aggregation
6      if label = Type II then
7          foreach p in P do
8              if p.e < h.s then
9                  Remove(p) // Remove plays before red dots
10         end
11         h.s' ← median(P.s) ; h.e' ← median(P.e)
12     else
13         h.s' ← h.s − m
14     end
15 until Red dot position converge
16 return h
```

6 DEPLOY LIGHTOR IN PRACTICE

In this section, we discuss two ways to deploy the Lightor workflow: one is to wrap it as a web browser extension and the other is to integrate it into existing live streaming platforms.

6.1 Web Browser Extension

Figure 5 depicts the architecture of our web browser extension. It has the potential to support any live streaming platform. We will use Twitch as an example to illustrate how it works.

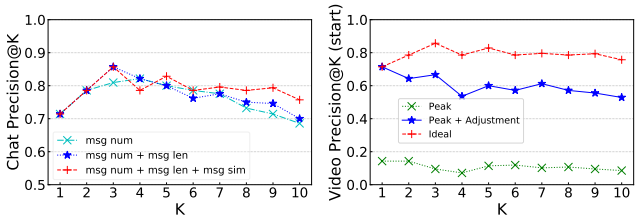
In addition to the Lightor's core components, we need two additional components: Web Service and Web Crawler.

Web Service. When a user opens up a web page in Twitch, if the URL of the web page is a recorded video, Lightor will be automatically activated. It extracts the Video ID from the web page and sends it to the back end server. The server receives the Video ID and checks whether the video's chat messages have been crawled and stored in the database. If not, it will call the web crawler component to crawl the chat messages. After that, it will use the chat data to identify the positions of top-k highlights and return them to the front end. The returned results will be rendered on the original web page by adding red dots on the progress bar of the video. Meanwhile, the user interaction data will be logged. Highlight Extractor will use the data to refine the results. The refined results will be stored in the database continuously.

Web Crawler. The web crawler component crawls the chat messages of recorded videos in Twitch. The crawled chat messages will be stored into the database. The crawling process can be executed both *offline* and *online*. The offline crawling periodically checks a given list of popular channels. If new videos are uploaded in those channels, their chat messages will be crawled accordingly. The online crawling will crawl the chat messages on the fly. It will be triggered if the chat messages of a video do not exist in the database.

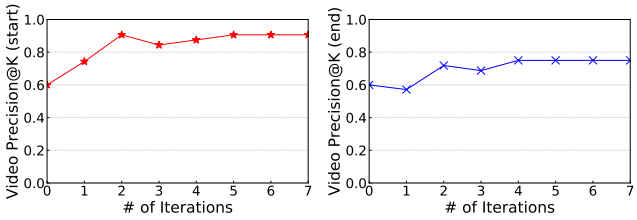
6.2 Integrate Into Live Streaming Platforms

Another way to deploy Lightor is to integrate it into existing live streaming platforms. The only change is to add red dots to the progress bar of recorded videos. It is easy to implement this feature from a technical point of view. Moreover, based on our user study,



(a) Prediction Stage (b) Adjustment Stage

Figure 6: Evaluation of Highlight Initializer.



(a) Start Position (b) End Position

Figure 7: Evaluation of Highlight Extractor.

this new interface is more attractive since it can help users find more interesting highlights.

Lightor is also useful for improving existing features. For example, Twitch allows broadcasters to cut and upload the highlights of their recorded videos manually. Lightor can provide broadcasters with a set of highlight candidates. This will help broadcasters save a lot of time when they need to edit highlights repeatedly.

7 EXPERIMENTS

We evaluate Lightor on real live video data. The experiments aim to answer three questions. (1) How well does Highlight Initializer perform? (2) How well does Highlight Extractor perform? (3) How does Lightor compare with deep-learning based approaches? We will first describe experimental settings and then present the experimental results. We provide a reproduciblility report in the Appendix, including detailed data description, detailed data preprocessing, model parameters and configurations, software versions, hardware configuration, and a jupyter notebook to reproduce all experimental figures.

7.1 Experimental Settings

Video Data. Game videos dominate mainstream live streaming platforms such as Twitch and Mixer. They were also used to evaluate the state-of-the-art highlight detection approaches [7, 24]. We evaluated Lightor on two popular games: Dota2 and LoL.

(1) Dota2. We crawled eight live videos in Dota 2 using Twitch APIs. The length of each video is between 1 hour to 2 hours. We asked experienced game players to watch each video and manually label the start and end positions of each highlight. Each video contains 10 labeled highlights on average. The length of each highlight is between 5s to 50s. (2) LoL. We randomly selected eight live videos in League of Legends (LoL) from [7]. The length of each video is between 0.5 hour to 1 hour. The labels were obtained by matching with highlight collections of a YouTube channel. Each video contains 14 labeled highlights on average. The length of each highlight is between 2s to 81s.

The Dota2 and LoL datasets are different in two aspects. First, the game types are different, and thus raw visual and textual features do not generalize well. Second, the Dota2 videos were from Twitch personal channels, but the LoL videos came from North America League of Legends Championship Series. Thus, their chat data have different characteristics.

User Data. Lightor relies on two kinds of user data: chat data and play data. For chat data, live streaming platforms make the data accessible. We used their APIs to crawl the data. The number of chat messages crawled for each video is between 800 to 4300.

Play data are not accessible from live streaming platforms. So, we recruited game fans from Amazon Mechanical Turk (AMT), and asked them to watch the recorded live videos. Each video's progress bar has a single red dot since we would like to get rid of the influence of nearby red dots and study user interactions on one red dot directly. We collected the user interaction data and then generated the play data. *Note that we did not ask the crowd to enter the start and end positions of a highlight. Therefore, the crowd provided us with the boundary of a highlight implicitly.* There were 492 workers participating our experiments and we spent about $750 to create the dataset. We have published the dataset. To the best of our knowledge, this is the first publicly available play data. More details can be found in the Appendix.

Evaluation Metrics: We used Precision@K to evaluate the performance since most users are only interested in watching a small number of highlights (e.g., k = 5 to 10). We defined three Precision@K metrics in the experiments.

(1) Chat Precision@K is to evaluate the effectiveness of the prediction stage in Highlight Initializer. The prediction stage sorts chat-message sliding windows based on how likely they are talking about a highlight, and returns the top-k sliding windows. Chat Precision@K is defined as the percentage of correctly identified sliding windows out of the k identified sliding windows.

(2) Video Precision@K (start) is to evaluate the precision of the identified start positions of highlights. Since people typically cannot tolerate more than 10s delay, we say a start position x is correct if there exists a highlight $h = [s, e]$ such that $x \in [s - 10, e]$. Video Precision@K (start) is defined as the percentage of correctly identified start positions out of the k identified start positions.

(3) Video Precision@K (end) is to evaluate the precision of the identified end positions of highlights. It is similar to Video Precision@K (start). We say an end position y is correct if there exists a highlight $h = [s, e]$ such that $y \in [s, e+10]$. Video Precision@K (end) is defined as the percentage of correctly identified end positions out of the k identified end positions.

7.2 Evaluation of Lightor

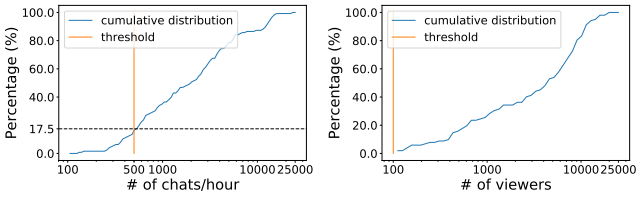
In this section, we first evaluate the Highlight Initializer and Highlight Extractor of Lightor, and then examine the applicability of Lightor on the Twitch platform.

7.2.1 *Evaluation of Highlight Initializer.*

Highlight Initializer consists of prediction and adjustment stages. We evaluated their performance on the Dota2 video data.

Prediction Stage. The prediction stage is designed to get top sliding windows corresponding to highlights. We propose three features, message number (*msg num*), message length (*msg len*), and message similarity (*msg sim*), and build a logistic regression model based on them. To evaluate the effectiveness of the proposed features, we build two additional logistic regression models using *msg num* and *msg num + msg len*. We used 1 video's sliding windows as training data and used other 7 videos' sliding windows as test data.

Figure 6a shows the average Chat Precision@K of the 7 testing videos on different k from 1 to 10. We have two interesting observations. First, *msg num* was an effective feature for small k (≤ 5) but did not perform well as k got larger (e.g., $k = 10$). This is because that as k increased, it would be more and more challenging to detect new highlights. If we only used the *msg num*



(a) CDF for Chat Messages (b) CDF for Viewers

Figure 8: Cumulative distribution over recorded videos.

feature, these messages sometimes were sent because viewers were discussing something on random topics which were not related to the highlights. Second, the ML model using all three features was better at capturing the nature of highlight messages especially when one wants to detect more than 5 highlights. The reason is that when viewers saw a highlight, their messages tend to be in a similar pattern. In addition to actively sending more messages, they would send more short messages such as Emojis or Stickers which make the average length of messages in the sliding windows shorter than common ones. When viewers were talking about something particular in the highlights, the messages would have a higher similarity.

Adjustment Stage. Suppose the prediction stage returns k sliding windows as highlights. Then, the adjustment stage aims to find the approximate start positions of the highlights (i.e., red dots). It first finds the peak in each sliding window and subtracts it by a constant value (learned from labeled data) to get the red dot. We used one video as training data to get the constant value, and evaluated Video Precision@K (start) on the other seven videos. The ideal situation of the adjustment stage is to be able to get a correct red dot for every top-k highlight. So the Ideal curve in Figure 6b is the same as the red line in Figure 6a. We can see that without adjustment, precision was below 20%. The adjustment improved the precision by around 3×. It shows that the adjustment stage performs well on capturing the delay between highlights and message-number peaks.

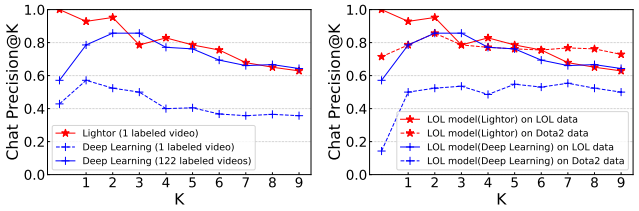
7.2.2 *Evaluation of Highlight Extractor.*

Highlight Extractor aims to leverage user interactions to identify the boundary of each highlight. We selected 35 red dots (5 per video) from 7 testing videos, and created 35 tasks. We first published these tasks to AMT. After receiving 10 responses for each task, we computed the new position of each red dot, and published a set of new tasks with updated red-dot positions to AMT. We repeated this process until users reached a consensus on the extracted highlights.

Figure 7 shows how Video Precision@K (start) and Video Precision@K (end) change over iterations. We can see that Video Precision@K (start) had a big improvement, increasing from 0.6 to 0.9. This improvement came from two sources. On one hand, it removed the red dots that did not talk about a highlight (i.e., improving the prediction stage in Highlight Initializer); on the other hand, it made a better adjustment about where a red dot should be put (i.e., improving the adjustment stage in Highlight Initializer). We also noticed that although Video Precision@K (end) got improved as well, the improvement was not as big as Video Precision@K (start). The main reason is that users tend to watch longer even after the end of highlights to guarantee not missing anything interesting.

7.2.3 *Applicability of Lightor in Twitch.*

Based on our experiments, in order to achieve high precision, Highlight Initializer requires the number of chat messages per hour larger than 500 and Highlight Extractor requires more than 100 viewers per video. We examine the applicability of Lightor with these requirements in Twitch.



(a) Training Data Size (b) Generalization

Figure 9: Comparison of Lightor and deep learning.

We selected the ten most popular channels in Doda 2 and crawled twenty most recently recorded videos from each channel. We plot the CDF of the number of chat messages and the number of viewers, respectively. Figure 8 shows the results. We can see that more than 80% of recorded videos have more than 500 chat messages per hour and all the recorded videos have more than 100 viewers. These results indicate that Lightor is applicable to the majority of popular videos in Twitch.

7.3 Comparison with Deep Learning

We compared Lightor with the state-of-the-art deep learning approach [7]. The approach can use either chat data, video data, or both to train a model. Since Lightor only uses chat data, we adopted their chat model. In fact, if video data are included, the deep-learning model would take several days to train and also does not generalize well [7]. The chat model, L-Char-LSTM, is a character-level LSTM-RNN [8] model. For each frame, it treats all chat messages that occur in the next 7-second sliding window as input for the 3-layer L-Char-LSTM. We used the model to predict the probability of each frame being a highlight, and selected the top-k frames. Note that if two frames are close to each other (within 2 mins), we only pick up the one with higher probability.

We first compared Lightor with Deep Learning in terms of training data size. Figure 9a shows the result. We can see that Lightor only needs to label a single video in order to achieve high precision, but the deep learning model did not perform well with a single labeled video. In the experiment of [7], the deep learning model was trained on 122 labeled videos. As shown in Figure 9a, even with such large training data size, the deep learning model still performed worse than Lightor for $k = 1, 2$. Lightor needs very little training data because it only selects three generic features and adopts a simple logistic regression model. We then compared Lightor with Deep Learning in terms of generalization. Figure 9b shows the result. We can see that for Lightor, the model trained on the LoL data can still achieve high precision on both LoL and Dota2 data. In comparison, for deep learning, the model trained on the LoL data only got high precision on the LoL data but not on the Dota2 data. Lightor has good generalization because the selected features are very general. Finally, we compared Lightor with Deep Learning in terms of computing resources. The training time for Lightor was seconds *versus* hours for the Deep Learning model.

The experimental results indicate that Lightor has great advantages over the deep-learning based approach in terms of training data size, computational cost, and generalization. Nevertheless, we do not argue to totally replace the deep-learning based approach with Lightor. Deep learning has its own advantages. For example, if a deep learning model is trained over video data, it does not need chat messages or user interaction data to detect highlights. An interesting future direction is to explore how to combine Lightor with Deep Learning, where Lightor is used to generate high-quality labeled data and Deep Learning is then applied to train a model.

8 FINDINGS & LESSONS LEARNED

We present interesting findings and lessons learned.

- It is important to do a pilot test and analyze real user interaction data. For example, we originally thought that Seek Backward could be a useful interaction to detect start positions of highlights. However, from real data we find that since there are various reasons to trigger this interaction (e.g., re-watch a highlight, look for a new highlight), it is not easy to infer users' true intent.
- The recorded live videos in Twitch typically attract thousands of viewers on average. In our experiments, we recruited around 500 viewers and showed promising results. Based on these findings, we believe that there is no obstacle for Lightor to collect enough user interaction data in real live streaming platforms.
- Users prefer spreading the red dots over the entire progress bar, instead of cluttering them in a narrow region. They think the former can help them have a broader overview of the whole video, while the latter only shows the content of part of the video.
- Viewers sometimes get excited about the interesting clips that are not related to a video's main theme, such as the break between two games, or the preparation for a game. Lightor may identify these clips as highlights. But it may not work if one only wants to watch highlights in a game. We will study how to overcome this limitation in future work.

9 CONCLUSION & FUTURE WORK

We presented Lightor, a novel implicit crowdsourcing workflow to extract highlights for recorded live videos. Lightor consists of two components. In Highlight Initializer, we explored different design choices and justified our decisions. We proposed three generic features (message number, message length, and message similarity) and built a model to predict highlight positions. We also noticed that there is a delay between a highlight and its comments, and proposed a simple learning-based approach to estimate the delay. In Highlight Extractor, we identified the challenges to use noisy user interaction data to extract highlights, and proposed a three-stage dataflow (filtering → classification → aggregation) to address these challenges. We discussed how to implement Lightor as a web browser extension and how to integrate Lightor into existing live streaming platforms. We recruited about 500 real users and evaluated Lightor using real Dota 2 and LoL data. We compared with the state-of-the-art deep learning approach. The results showed that Lightor achieved very high detection accuracy (Precision@K: 70%-90%). Furthermore, it only needed to label a single video and spend a few seconds on training, and the obtained model had good generalization.

There are many future research directions to explore. First, we were told by the data science team at a well-known live streaming platform that they stored several terabytes of chat data, but have not tried to extract value from the data. We are planning to deploy Lightor on their platform, and conduct more large-scale experiments. Second, we want to further optimize the workflow, especially on the adjustment stage. The current implementation assumes that there is a simple linear relationship between $\mathsf{time}_{\mathsf{peak}}$ and $\mathsf{time}_{\mathsf{start}}$. We plan to relax this assumption and build a more sophisticated regression model. Third, we plan to further evaluate the generalization of our system using data collected from other domains (e.g., celebrity events) and other live streaming platforms (e.g., YouTube Live). Fourth, this paper demonstrates a great potential of the application of implicit crowdsourcing to video highlight detection. It is promising to investigate how to design an implicit crowdsourcing workflow for other video analysis tasks (e.g., video querying, video search, and video indexing).

REFERENCES

[1] 1993. Response Times: The 3 Important Limits. https://www.nngroup.com/articles/response-times-3-important-limits. (1993). Accessed: 2018-11-05.
[2] 2018. Implicit Crowdsourcing. https://en.wikipedia.org/wiki/Crowdsourcing#Implicit_crowdsourcings. (2018). Accessed: 2018-11-05.
[3] 2018. Twitch Tracker. https://twitchtracker.com/statistics. (2018). Accessed: 2018-11-05.
[4] Konstantinos Chorianopoulos. 2013. Collective intelligence within web video. *Human-centric Computing and Information Sciences* 3, 1 (2013), 10.
[5] Ahmet Ekin, A. Murat Tekalp, and Rajiv Mehrotra. 2003. Automatic soccer video analysis and summarization. *IEEE Trans. Image Processing* 12, 7 (2003), 796–807.
[6] Atefeh Farzindar and Wael Khreich. 2015. A Survey of Techniques for Event Detection in Twitter. *Computational Intelligence* 31, 1 (2015), 132–164.
[7] Cheng-Yang Fu, Joon Lee, Mohit Bansal, and Alexander C. Berg. 2017. Video Highlight Prediction Using Audience Chat Reactions. In *EMNLP*. 972–978.
[8] Alex Graves. 2013. Generating Sequences With Recurrent Neural Networks. *CoRR* abs/1308.0850 (2013). arXiv:1308.0850 http://arxiv.org/abs/1308.0850
[9] Liang-Chi Hsieh, Ching-Wei Lee, Tzu-Hsuan Chiu, and Winston H. Hsu. 2012. Live Semantic Sport Highlight Detection Based on Analyzing Tweets of Twitter. In *ICME*. IEEE Computer Society, 949–954.
[10] Yun Huang, Yifeng Huang, Na Xue, and Jeffrey P Bigham. 2017. Leveraging Complementary Contributions of Different Workers for Efficient Crowdsourcing of Video Captions. In *CHI*. ACM, 4617–4626.
[11] Cisco Visual Networking Index. 2016. Cisco VNI Forecast and Methodology, 2015–2020. *Cisco white paper* 1 (2016).
[12] Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *ACM SIGKDD*. 133–142.
[13] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Deep CNN-Based Queries over Video Streams at Scale. *PVLDB* 10, 11 (2017), 1586–1597.
[14] Alexandre Kaspar, Geneviève Patterson, Changil Kim, Yagiz Aksoy, Wojciech Matusik, and Mohamed Elgharib. 2018. Crowd-Guided Ensembles: How Can We Choreograph Crowd Workers for Video Segmentation?. In *CHI*.
[15] Juho Kim, Philip J. Guo, Carrie J. Cai, Shang-Wen (Daniel) Li, Krzysztof Z. Gajos, and Robert C. Miller. 2014. Data-driven interaction techniques for improving navigation of educational videos. In *UIST*. ACM, 563–572.
[16] Juho Kim, Philip J. Guo, Daniel T. Seaton, Piotr Mitros, Krzysztof Z. Gajos, and Robert C. Miller. 2014. Understanding In-video Dropouts and Interaction Peaks Inonline Lecture Videos. In *L@S*. 31–40.
[17] Sanjay Krishnan, Adam Dziedzic, and Aaron J. Elmore. 2019. DeepLens: Towards a Visual Data Management System. In *CIDR*.
[18] Joonseok Lee, Sami Abu-El-Haija, Balakrishnan Varadarajan, and Apostol Natsev. 2018. Collaborative Deep Metric Learning for Video Understanding. In *ACM SIGKDD*. 481–490.
[19] Mayu Otani, Yuta Nakashima, Esa Rahtu, Janne Heikkilä, and Naokazu Yokoya. 2016. Video Summarization using Deep Semantic Features. *CoRR* abs/1609.08758 (2016).
[20] Qing Ping and Chaomei Chen. 2017. Video Highlights Detection and Summarization with Lag-Calibration based on Concept-Emotion Mapping of Crowd-sourced Time-Sync Comments. *EMNLP 2017* (2017), 1.
[21] Yong Rui, Anoop Gupta, and Alex Acero. 2000. Automatically extracting highlights for TV Baseball programs. In *ACM MM*. 105–115.
[22] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. 2013. Tweet Analysis for Real-Time Event Detection and Earthquake Reporting System Development. *IEEE Trans. Knowl. Data Eng.* 25, 4 (2013), 919–931.
[23] Tanmay Sinha, Patrick Jermann, Nan Li, and Pierre Dillenbourg. 2014. Your click decides your fate: Leveraging clickstream patterns in MOOC videos to infer students' information processing and attrition behavior. *CoRR* abs/1407.7131 (2014).
[24] Yale Song. 2016. Real-Time Video Highlights for Yahoo Esports. *CoRR* abs/1611.08780 (2016).
[25] Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. 2008. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science* 321, 5895 (2008), 1465–1468.
[26] Carl Vondrick, Donald Patterson, and Deva Ramanan. 2013. Efficiently scaling up crowdsourced video annotation. *International Journal of Computer Vision* 101, 1 (2013), 184–204.
[27] Bin Wu, Erheng Zhong, Ben Tan, Andrew Horner, and Qiang Yang. 2014. Crowdsourced Time-sync Video Tagging Using Temporal and Personalized Topic Modeling. In *SIGKDD*.
[28] Changsheng Xu, Jinjun Wang, Kongwah Wan, Yiqun Li, and Lingyu Duan. 2006. Live sports event detection based on broadcast video and web-casting text. In *MM*. 221–230.
[29] Peng Xu and Martha Larson. 2014. Users tagging visual moments: timed tags in social video. In *Proceedings of the 2014 International ACM Workshop on Crowdsourcing for Multimedia*. ACM, 57–62.
[30] Huan Yang, Baoyuan Wang, Stephen Lin, David P. Wipf, Minyi Guo, and Baining Guo. 2015. Unsupervised Extraction of Video Highlights via Robust Recurrent Auto-Encoders. In *IEEE ICCV*. 4633–4641.
[31] Ting Yao, Tao Mei, and Yong Rui. 2016. Highlight Detection with Pairwise Deep Ranking for First-Person Video Summarization. In *CVPR*. 982–990.

APPENDIX: REPRODUCIBILITY SUPPLEMENT

In this supplement section, we first show how to reproduce all experimental plots in batch, and then present details related to experimental settings and the reproducibility of each individual plot. More detailed explanation and the experiment code can be found at the project page: https://tiny.cc/lightor.

A. Reproducing All Experiments in Batch

We prepared a jupyter notebook[1] to reproduce all experiments in batch. To set up the environment, Python 3.5 and Jupyter notebook are required. Other required packages are listed in *requirements.txt*. One can easily reproduce all of our experiments by:

- Clone the repo: `git clone https://github.com/sfu-db/Lightor_Exp` .
- Install required packages: `pip3 install -r requirements.txt` .
- Open the jupyter notebook file and run the code blocks: `jupyter notebook exp.ipynb` .

For details, please refer to README file in our repo.

B. Detailed Experimental Settings

We will discuss more details about our experimental settings.

Data Description: The Dota2 and LoL datasets are 2 datasets we used in our experiments. They are different in two aspects. First, the game types are different thus raw visual and textual features do not generalize well. Second, the Dota2 videos were from Twitch personal channels, but the LoL videos came from North America League of Legends Championship Series.

- Dota2 dataset: We used an open-sourced tool, twitch-chat-logger[2], to crawl 8 Dota2 videos from Twitch stored as MP4 files and their corresponding messages stored as CSV files. 6 out of 8 videos are 1 hour long, 1 video is 1.5 hours long, and 1 video is 2 hours long. Video file names are in the format of channel name + formatted time-stamp and Message file names are channel_name + formatted_timestamp + raw_timestamp, where channel_name is the name of a Twitch channel. Timestamp is the time that the crawling got started. In the message data, the schema of the message is [Message_id, channel_name, sender_ID, Message_content, timestamp], where Message_id is a unique integer to identify a message; channel_name is a string to indicate the name of the corresponding channel. sender_ID is a string of the user's id. Message_content is a string of the message. timestamp is the time when the message is sent.
- LoL dataset. This dataset comes from [7]. This dataset includes 321 labeled videos from 2 series games (NALCS and LMS) in League of Legends (LoL). Their message format is different from ours. As we know, a video can be transformed into a sequence of frames. The messages are stored in an array of strings as a JSON file and each frame corresponds to a message string or an empty string.

Details Related to Dataset Filtering. In Highlight Initializer, we filtered all chat messages starting with '@' and 'http' (web url). Chat messages starting with @ represents a chat between viewers while a web url represents an ad. Both are noisy data. In Highlight Extractor, we firstly filtered too short and too long playing records then adopted an outlier detection algorithm to filter outliers. We set parameters $(\phi, \Delta) = (4, 60)$ for filtering. For outlier detection, we constructed an undirected graph $G = (V, E)$, where V represents all playing records and $(v, w) \in E$ represents that v and w have overlapping part. Then we find the center node $c \in V$ of the graph which has the largest degree. We used c and its neighbor nodes while filtered others to analyze boundaries of highlights.

Play Data Collection: We recruited game fans from Amazon Mechanical Turk (AMT) and asked them to watch the recorded live videos. We collected 7 rounds of crowdsourcing results in this stage. Each round contains 15 tasks for each video. Each task contains a complete recorded video. The video's progress bar has a red dot. We tell the crowd that there could be an interesting highlight around the red dot. They can either write down what the highlight is about or skip this task. Note that we do not ask the crowd to enter the start and end positions of the highlight. And we also do not make use of their input answers. Therefore, the crowd provides us with implicit feedback.

One reason that we ask workers to write down their understanding of a highlight is that we want them to really look for a highlight by watching a video as what real-world game fans do. Otherwise, they may simply submit a task without watching any part of the video. The other reason is to understand the distribution of true game fans. We do not set any qualification test because in the real world anyone can watch a recorded live video even he/she is not a game fan. By reviewing their answers, we found that not all workers are true game fans. There is a small number of malicious workers who submit random answers, which can represent the noisy data in the real world.

Ground-Truth Labelling: We recruited two experienced game players who watch live streaming videos at a regular basis to label the data as follows:

- Dota2 dataset: (1) Sliding windows labels: In Highlight Initializer, they manually labeled all generated sliding windows as *True* or *False* by examining whether the chat messages in a sliding window are talking about a highlight event. They labeled them for evaluating Chat Precision@K. (2) Ground truth labels: They manually labeled the accurate boundaries of each highlight by watching them. They labeled them for evaluating Video Precision@K.
- LoL dataset: Originally this dataset has the ground-truth labels. In [7], each message (empty string if there is no message content) corresponds to a frame in the video. If the frames are within highlights, their messages will be labelled as positive, while the non-highlight messages are labelled as negative. We used their ground-truth labels directly and labeled the sliding windows according to ground truth manually. Specifically, we only examined the content of chat messages in sliding windows near ground truth and labeled them. For other sliding windows, we directly labeled them as *False*.

Software Versions and Hardware Configuration: The Lightor system was implemented using Python 3.5. Logistic regression models were trained using scikit-learn 0.20. The experiments were run over a Ubuntu virtual server with an Intel(R) Xeon(R) CPU E7-4830 v4 @ 2.00GHz processor and 53GB of RAM. The sliding window size was set to 25s. The deep learning model was trained on an Nvidia GTX 1080 GPU.

[1] https://github.com/sfu-db/Lightor_Exp/blob/master/exp.ipynb
[2] https://github.com/bernardopires/twitch-chat-logger

C. Reproducing Each Experiment in Detail

Parameters for Highlight Initializer. We set the following parameters for Highlight Initializer:

(1) $L = 25$: the length of a sliding window.
(2) $N = 200$: the number of sliding windows.
(3) $\delta = 120$ for the interval between two predicted highlights.
(4) bin parameters $(B, W, P) = (3, 7, 4)$ for the smoothing and peak detection algorithm.

Parameters for Highlight Extractor. We set the following parameters for Highlight Extractor:

(1) $(\phi, \Delta) = (4, 60)$ for filtering too short and long plays.
(2) thresholds $(\theta, \lambda) = (20, 40)$ for converging judgement.
(3) feature vectors $f = [[4, 33, 252], [25, 12, 56]]$ for clustering red dots type, which are trained by labeled data.

For more detail, please refer to *parameter.py* file in our repo.

Implementation Details. We implemented *Highlight Initializer* as a data loader class for loading chat messages, ground truth and extracting features of a dataset. A model can be trained by or applied on a instance of such data loader. We encapsulated the process of analyzing user interaction data in *Highlight Extractor*, including filtering, type classification and data aggregation. For details, please refer *highlight_initialzer.py* and *highlight_extractor.py*.

Train/Test Splits. We used the following datasets in our experiments. For more detail, please refer *dataset* folder in our repo.

- $Dota_{train}$ contains 1 Dota2 video's chat messages used for training the Lightor model.
- $Dota_{test}$ contains 7 Dota2 videos' chat messages used for testing.
- LOL_{train1} contains 1 LOL video's chat messages used for training the Lightor model.
- $LOL_{train122}$ contains 122 LOL videos' chat messages used for training the deep learning model.
- *LOLtest* contains 7 LOL video's chat messages used for testing.
- I contains about 560K user interaction records (i.e. playing timestamp and video id) which we collected from Amazon Mechanical Turk.
- A contains 7 files of crowdsourcing answers (i.e. Assignment ID, Worker ID and Anwser), which we used to map interaction records in I.

Trained models. We trained the following models for our experiments (L refers to Lightor and D refers to the deep learning model): (1) $L1_{dota}$ was trained on dataset $Dota_{train}$ using one feature. (2) $L2_{dota}$ was trained on dataset $Dota_{train}$ using two features. (3) $L3_{dota}$ was trained on dataset $Dota_{train}$ using three features. (4) $L3_{lol}$ was trained on LOL_{train1} using three features. (5) D_{122} was trained on $LOL_{train122}$ using the state-of-art deep learning method[7]. (6) D_1 was trained on LOL_{train1} using the state-of-art deep learning method[7]. (7) E were trained on one $Dota_{train}$ for Adjustment stage in Highlight Initializer. We used the default hyper-parameters of scikit-learn.

Reproducing Figure 2(a) includes:

- Load $Dota_{test}$
- Select a time range in one chat file
- Plot the curve of number of chat messages, peak and ground truth of highlight.

Reproducing Figure 2(b) includes:

- Load $Dota_{test}$
- Plot the feature-value distribution histogram

Reproducing Figure 3 includes:

- Load recorded playing interaction file generated by selecting records in I.
- Plot the distribution of error interval time of each type.

Reproducing Figure 6(a) includes:

- Apply models $L1_{dota}$, $L2_{dota}$, $L3_{dota}$ on $Dota_{test}$
- Compute the chat precisions by comparing predicted labels with ground truth.
- Plot the precision curves.

Reproducing Figure 6(b) includes:

- Apply model $L3_{dota}$ on $Dota_{test}$ and get predicted labels.
- Compute the chat precision of predicted labels and video precision of peak points.
- Apply expander model E on predicted labels.
- Compute the video precision of predicted labels after adjustment.
- Plot the precision curves.

Reproducing Figure 7 includes:

- Apply model $L3_{dota}$ on $Dota_{test}$ and get predicted labels and ground truth.
- Use Highlight Extractor to process interaction data.
- Compute video precision of start time and end time.
- Plot the precision curves.

Reproducing Figure 8 includes:

- Load the external json file which we crawled from Twitch, containing the number of chat messages and number of viewers on most recently recorded videos.
- Plot the CDF plots.

Reproducing Figure 9 includes:

- Apply model $L3_{lol}$ on LOL_{test} and compute chat presicion P_1.
- Apply model $L3_{lol}$ on $Dota_{test}$ and compute chat presicion P_2.
- Apply model D_{122} on LOL_{test} and compute chat presicion P_3.
- Apply model D_1 on LOL_{test} and compute chat presicion P_4.
- Apply model D_{122} on $Dota_{test}$ and compute chat presicion P_5.
- Plot precision curves of P_1, P_3 and P_4.
- Plot precision curves of P_1, P_2, P_3 and P_5.