Characterizing and Detecting Livestreaming Chatbots

Shreya Jain[1*], Dipankar Niranjan[1*], Hemank Lamba[2], Neil Shah[3], Ponnurangam Kumaraguru[4]
[1]IIIT-Hyderabad, [2]Carnegie Mellon University, [3]Snap Inc., [4]IIIT-Delhi
{shreya.jain, dipankar.niranjan}@research.iiit.ac.in, hlamba@cs.cmu.edu, nshah@snap.com, pk@iiitd.ac.in

***Abstract*—Livestreaming platforms enable content producers, or streamers, to broadcast creative content to a potentially large viewer base. Chatrooms form an integral part of such platforms, enabling viewers to interact both with the streamer, and amongst themselves. Streams with high engagement (many viewers and active chatters) are typically considered engaging, and often promoted to end users by means of recommendation algorithms, and exposed to better monetization opportunities via revenue share from platform advertising, viewer donations, and third-party sponsorships. Given such incentives, some streamers make use of fraudulent means to increase perceived engagement by simulating chatter via fake "chatbots" which can be purchased from shady online marketplaces. This inauthentic engagement can negatively influence recommendation, hurt streamer and viewer trust in the platform, and harm monetization for honest streamers. In this paper, we tackle the novel problem of automating detection of chatbots on livestreaming platforms. To this end, we first formalize the livestreaming chatbot detection problem and characterize differences between botted and genuine chatter behavior observed from a real-world livestreaming chatter dataset collected from Twitch.tv. We then propose SHERLOCK, which posits a two-stage approach of detecting chatbotted streams, and subsequently detecting the constituent chatbots. Finally, we demonstrate effectiveness on both real and synthetic data: to this end, we propose a novel strategy for collecting labeled, synthetic chatter dataset (typically unavailable) from such platforms, enabling evaluation of proposed detection approaches against chatbot behaviors with varying signatures. Our approach achieves $.97$ precision/recall on the real-world dataset, and $.80+$ F1 scores across most simulated attack settings.**

I. INTRODUCTION

In recent years, livestreaming platforms such as Twitch, YouTube Live, Facebook Live, and Ustream have grown to become dominant players in the content broadcasting space, commanding *millions* of broadcasters and *tens of millions* of daily active users [1]. These platforms provide avenues for broadcasters, or *streamers*, to share creative content of various forms (e-sports gameplay, live events, art, etc.) to a large audience. Each broadcasting session, or *stream*, consists of two key components – the content being shared live to viewers, and a chatroom (Figure I(left)), where viewers can chat and interact amongst themselves, and with the streamer. These chatrooms provide a completely different community experience to viewers in contrast to traditional media, providing an increased sense of participation and gratification [2].

Most livestreaming platforms recommend streams to would-be viewers based on prior and current engagement metrics, which is effectively a function of viewership and chatroom activity. Specifically, streams that garner high viewership and have active chatrooms are considered to be likely interesting and engaging to new viewers, and are thus recommended to draw new viewers, amplifying preferential attachment effects. Moreover, streamers who produce such content and draw such engagement are prime candidates for on-platform and off-platform monetization via advertising revenue share, donations from viewers, and sponsorships from third-parties (i.e. computer hardware companies for e-sports professionals). Such incentives lead some streamers to resort to fraudulent methods to increase their viewership [3] and increase chatroom activity [4]. Numerous online marketplaces like streambot.com and youtube-livebot.com offer streamers the ability to increase their chatroom activity over sustained period of time, via *chatbots* which simulate human-like chatter. Such fraudulent engagement has several adverse effects: (a) honest streamers may not be as highly recommended as fraudsters and lose out on potential engagement they may have otherwise garnered via preferential attachment, (b) viewers and streamers have reduced trust in the platform to recommend and prioritize good content, (c) the platform and third-party sponsors may lose money by partnering with fraudulent streamers who reach much lesser human eyes than their metrics suggest. Despite these concerns, prior work in mitigating chatbot abuse on livestreaming platforms is minimal – we seek to bridge the gap in this work.

There are numerous challenges in this problem setting: (a) *noisy data*: livestreaming chatter is full of messages with ill-formed sentences, containing spelling errors, "legitimate" spam messages (copypasta), and emotes, limiting efficacy of text-based features to identify fraudulent activity, (b) *user-controlled fraud*: most chatbotting services available on online marketplaces allow streamers to control the bots (Figure I), giving them the ability to decide when and how much fake chatter should be introduced, and thereby complicating the attack space and hurting detection generalizability, and (c)

∗ Both authors contributed equally.

ASONAM '19, August 27 - 30, 2019, Vancouver, BC, Canada

ACM ISBN 978-1-4503-6868-1/19/08...$15.00
https://doi.org/10.1145/3341161.3345308

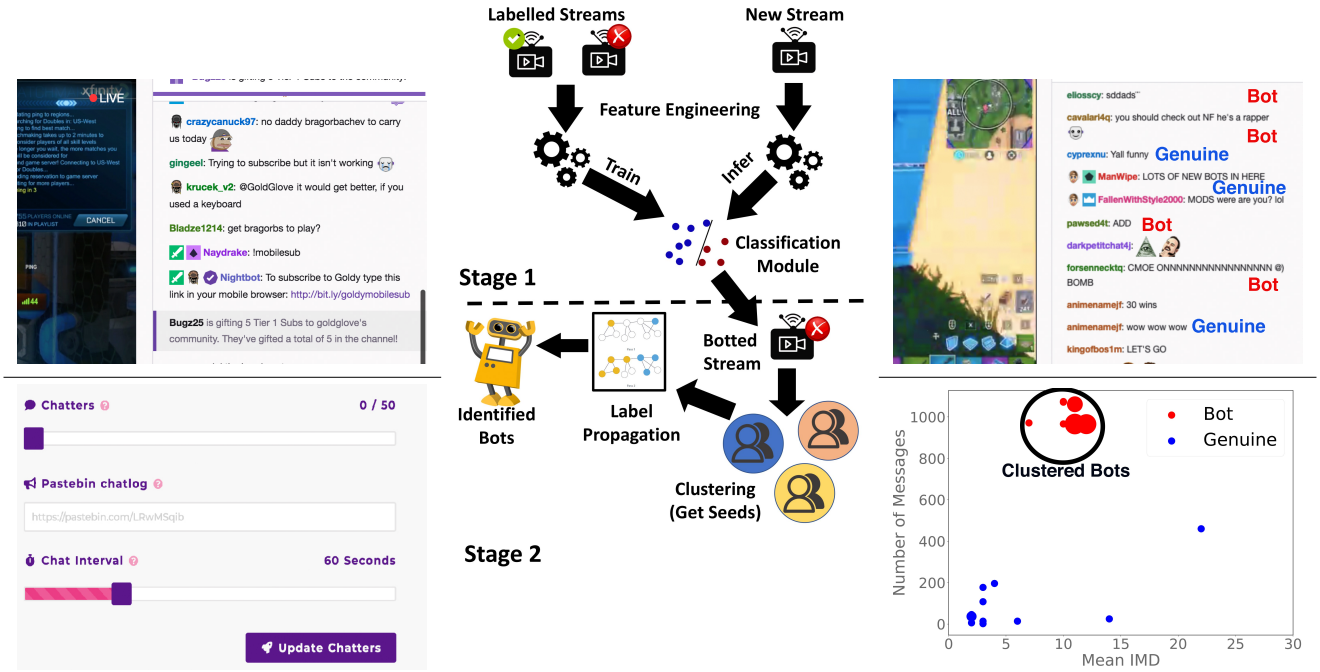


Fig. 1: (**Left**) Livestreaming platforms offer chatrooms (top), which streamers can manipulate via chatbotting tools (bottom) that enable customization of chat interval, number of chatters and even message contents. (**Center**) We propose SHERLOCK, a two-stage chatbot detection approach based on stream (top) and user-level classification (bottom). (**Right**) We enable discovery of chatbotted streams (top – notice genuine users asking for moderators to handle the bots), and the constituent chatbots via discriminative features (bottom – large points indicate high user density).

lack of ground-truth: as livestreaming platforms operate at an extremely large-scale and do not reveal the chatbots they proactively ban from the service, obtaining reliable ground truth for building machine learning models is non-trivial.

In this work, we tackle these challenges and more. To our knowledge, we are the first to study the chatbot detection problem in the livestreaming setting. Specifically, our contributions are as follows:

1) **Problem formulation:** We formalize the chatbot detection problem in the context of chatrooms of livestreaming platforms.
2) **Dataset collection and characterization:** We obtain real livestreaming chatlog data, and compare the behaviors of chatbots and real users. We also discuss how to construct *labeled* synthetic chatter datasets from livestreaming platforms, for a variety of attack models.
3) **Proposed framework:** We propose SHERLOCK which tackles chatbot detection in a two-stage approach: detecting botted livestreams using a classification model (stage I), and detecting constituent chatbots using a seeding and label propagation approach (stage II). Overview of approach given in Figure I(center).

We conduct several experiments to demonstrate that our proposed method is (a) *effective*: we show that our approach outperforms alternatives in detection performance on real chatlog datasets (.97 precision/recall) (Figure I(bottom-right))(b) *robust to different attacks*: we show consistently good performance in detecting chatbots across many attack configurations ($\geq$ .80 F1 against most attack settings), and (c) *scalable*: our approach scales near-linearly on large datasets, especially due to our two-stage task formulation. We make the code for SHERLOCK available at `https://github.com/shreya-03/Sherlock`.

II. Related Work

We discuss prior work in (a) detecting chatbots, and (b) astroturfing in social media.

Detecting chatbots. Most prior work on chatbot detection consider chatbots as accounts that spread malicious or spammy URLs [5], [6], [7]. [5] proposed a classifier based on entropy-based features (message length, and inter-message delay) to detect chatbots on Yahoo chat systems. [6] used similar features to differentiate between bot and genuine users on various instant messaging platforms in IM (instant messaging) settings (i.e. human is chatting only with one user (bot/genuine)). Additionally, [7] proposes detecting chatbots based on the links they post, using cues from spam classification literature to detect malicious URLs. However, all of these methods are based on IM platforms, where chat messages are more directed towards other chatters, and are primarily concerned with delivering a payload of a malicious URL. Our work studies chatbots on livestreaming platforms, where bots are used with an alternative purpose of increasing perceived chatter, and hence vary in their design and motive. Though many works tackle bot detection on popular social media platforms, such as Twitter [8], [9], Facebook [10], and software marketplaces [11], [12], they are characteristically different from our work

as they do not focus on detecting chatbots. Besides this, there is a lot of work in designing conversational agents [13], which is beyond the scope of this work as they aim at coming up with creating realistic chatbots not for malicious purposes.

Astroturfing in social media. Social media websites have become a common target for astroturfing, where users artificially inflate engagement to increase perceived popularity. Graph-based factorization approaches to group nodes based on similarity or dense connectivity implying suspicious, large clusters have shown considerable success in detecting fraudulent activities [14], [10], [15]. Random-walk based methods have also been used to detect abnormal cuts between suspicious and legitimate parts of a social graph [16]. Content-based methods use textual features [11] or local engagement features (i.e. based on egonets) [17] to detect spam and fraud. [18] also propose temporal methods focusing on finding anomalous patterns in multivariate time series. The closest work to ours is by [19], in which the author proposes an unsupervised method to detect livestreaming viewbots. Despite rich literature in this space, none of the prior works have focused on the problem setting of detecting chatbots on livestreaming platforms.

III. Problem Statement

Each stream on a livestreaming platform generally consists of a chatroom panel located adjacent to the live video player (Figure I(Left)). Viewers must be signed in to participate in chat, and the messages typed by any of the signed-in viewers appears in realtime as the user sends each message. Each message is associated with the username of the author, as well as its timestamp. All chat messages are textual (i.e. text, emojis, URLs). Messages are typically short, and have a length cap to prevent single users from dominating the community chatroom with spam. Such chatrooms typically allow users to reply to one another (via an "@handle" mechanism), inducing a conversational aspect to the room. In this work, we leverage all available sources of information above: Specifically, we collect data pertaining to a set of livestreams $\mathcal{S}$. For each stream $s \in \mathcal{S}$, we collect the set of all messages $\mathcal{M}_s$. We refer to messages on stream s that were posted by user/chatter i as $\mathcal{M}_{s,i}$, and the timestamp of the jth message from user i on stream s as $t_{s,j,i}$. Given these information sources, we aim to detect chatbots.

We note that considering all users in all streams is a computationally heavy and expensive task. Moreover, it is difficult to claim in isolation whether any given message is from a chatbot or real user, and even if a single user is a chatbot or not. We take a step back to consider that instead of gauging whether each message or user is legitimate or not, we should first consider the aggregate behavior of the parent stream. This is because it is unlikely to observe a single chatbot in isolation, but far more likely to observe a number of chatbots orchestrating a coordinated activity inflation effort on a given stream. By focusing on a stream-level first, we can leverage aggregate behaviors from many messages from many users jointly to infer whether the stream appears to be botted or not. We formally define this task as follows:

Table I: Dataset Statistics

# of chatlogs	690
# of messages	439,650
# of streamers	168
# of chatters	8,885
Median stream duration	2.7 hours

Problem 1 (**Chatbotted Stream Identification**). ***Given*** *a set of streams $\mathcal{S}$, and corresponding set of chatters $\mathcal{C}_s$ for each $s \in \mathcal{S}$,* ***find*** *the set of chatbotted streams.*

Upon obtaining the set of suspected chatbotted streams $\mathcal{S}_{cb}$, we can next focus only on this subset to discern suspected chatbots from real chatters. We argue that while it is conceivable that chatbots may exist in isolation in other streams, it is unlikely, and at best ineffective from the streamer's point of view. Moreover, since $\mathcal{S}_{cb}$ is likely to be much smaller than $\mathcal{S}$, we can dramatically improve scalability by avoiding chatbot detection for determined "low-suspicion" streams, and only focusing on the high-suspicion ones. The task that we pose for these is as follows:

Problem 2 (**Chatbot Identification**). ***Given*** *a suspicious chatbotted stream $s \in \mathcal{S}_{cb}$, and corresponding set of individual chatters $\mathcal{I}$,* ***label*** *each chatter $i \in \mathcal{I}$ as being part of the (disjoint) set of real users $\mathcal{I}_r$ or chatbotted users $\mathcal{I}_{cb}$.*

IV. Data Description

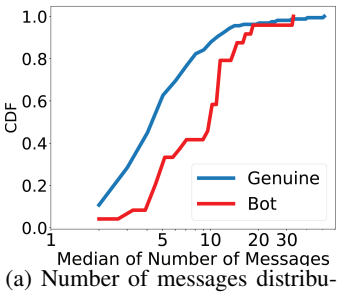
In this work, we study Twitch, a dominant livestreaming platform with over $2.2M$ streamers and $15M$ unique daily viewers reported in 2018[1]. Note that due to limitations on data collection and labelling cost, it is unfeasible to work with ther platforms. However we assume that a similar method of providing chatbots is also used for other livestreaming platforms. We collected chatter of $439K$ messages over a period of three months from August to October 2018 from chatrooms of 690 randomly chosen Twitch streams. A brief description of the dataset collected is given in Table I.

Annotation. We manually annotated 183 chatlogs out of the 690 collected. The annotators used cues such as relevance of text to the context, number of messages posted by accounts, metadata and other similar signals to identify if a particular livestream was chatbotted or not, as per knowledge from prior literature [19] and a survey of chatbotting services. The annotators found 24 botted and 159 seemingly genuine streams. While annotation was possible, it took each annotator roughly 104 hours to complete the task, clearly making annotation of the entire dataset infeasible. Thus, for our further analysis, we use the 183 streams, with 78,124 messages from 6,167 genuine users and 23,236 messages from 2,739 chatbots.

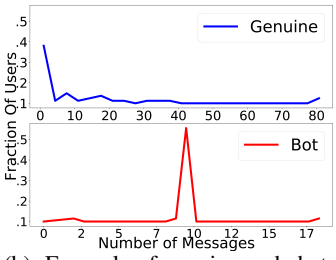
V. Initial Observations

Before proposing our approach, we conduct preliminary exploration of the dataset and try to identify key statistics that can help us differentiate the genuine and suspicious streams/bots. In this section, we describe the potential features

[1] https://twitchadvertising.tv/audience/

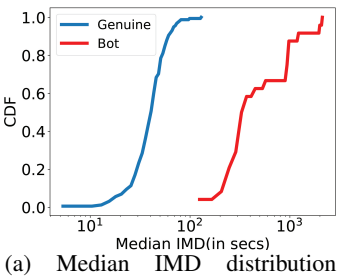


(a) Number of messages distribution ECDF.

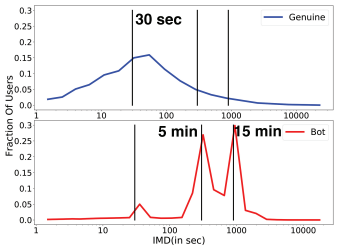


(b) Example of genuine and chatbotted streams.

Fig. 2: (a): ECDF for median distribution on number of messages for genuine and chatbotted streams. (b): Distribution of number of messages posted for randomly selected genuine and chatbotted streams.

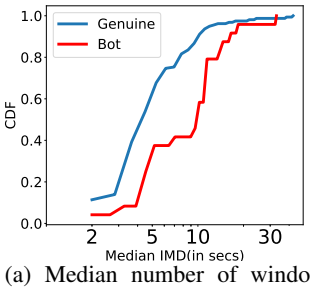


(a) Median IMD distribution ECDF.

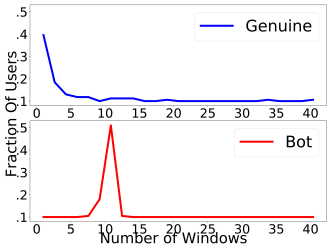


(b) IMD distribution for all genuine (top) and botted (bottom) streams

Fig. 3: (a): ECDF for distribution of median on IMD for genuine and chatbotted streams. (b): Distribution of IMD.



(a) Median number of windows distribution ECDF.



(b) Number of windows per user distribution for all genuine (top) and botted (bottom) streams

Fig. 4: (a): ECDF for distribution of median on number of windows per user for genuine and chatbotted streams. (b): Distribution of number of windows per user.

we considered and point out the key insights we obtained about genuine and fraudulent behavior.

Message frequency. Since bots are created with the purpose of increasing chatroom activity, it is natural to assume that they will post more messages than the genuine users in a stream. However, it could be contrary as well, that is if the users are fooled by the bots into believing that bots are actually genuine accounts, it might happen that genuine users might keep up the end of conversation and end up creating similar or more messages than the bot accounts. For each stream, we compute the entropy of the number of messages posted distribution. We observe that the entropy for chatbotted streams is higher than that of genuine streams. We show this by plotting the empirical cumulative frequency distribution (ECDF) in Figure 2(a). Additionally, we observe that the entropy statistic is able to differentiate between chatbotted streams and genuine streams with a Kolmogrov-Smirnov test p-value of 4.34×10^{-8}. Based on the above statistics, we make the following key observation:

Observation 1 (MESSAGE FREQUENCY). *Chatbots tend to post more messages than genuine users, with most chatbots posting messages with similar frequency.*

Inter-message delays (IMD). IMDs have been used previously in literature to identify bot behavior [18]. They have proved to be useful in identifying footprints of automation by scripting, which tends to be regular and deterministic. We define IMDs for an entire stream as the difference in time between each pair of consecutive messages from the same user, across all users for the duration of the stream.

We plot the ECDF of median IMDs for each stream in Figure 3(a). We can observe that ECDF differs significantly for genuine and chatbotted streams (KS Test p-value: 1.93×10^{-19}). We also plot the PDF across all IMD for users in genuine streams and users in botted streams, and show this in Figure 3(b). Based on the above plots, we make the following observation:

Observation 2 (INTER-MESSAGE DELAYS). *Chatbotted streams have a higher IMD than genuine streams. Chatbots have a consistent IMD showing that they are automated.*

Message Spread. Since bots are designed to maintain engagement for extended periods (rather than specific times), we hypothesize that they post throughout the duration of most chatbotted streams. We investigate this empirically by counting the number of equal-duration time intervals in which a particular user posts during the duration of the stream. To compute this, we partition the stream into equal-duration intervals, and count the number of windows in which each user posts a message. Intuitively, users who post consistently will appear in more windows. Figure 4(a) shows the ECDF of the median of the number of windows per user distribution. We note that the distribution for chatbotted and genuine stream is significantly different, corroborated by a KS test with p-value of 7.34×10^{-7}. Figure 4(b) shows examples of these distributions for a chosen bot and genuine stream. We have the following observation:

Observation 3 (MESSAGE SPREAD). *Chatbots' message distribution is more spread out, and on average, they post consistently throughout the stream.*

Textual Cues. We additionally experimented with various features from text mining literature to determine if language used by chatbots is significantly different from that used by genuine users. We compared tf-idf scores, conversational dynamics, and similarities in term usage within chatbot and genuine user groups. Interestingly, we were not able to find any distinguishing patterns for chatbots. This is likely due to

(a) message text being extremely noisy and short, (b) too much sparsity for conversation threads via "@handle" mechanism, and (c) most chatbot marketplaces enable customers to upload a text file of quotes used by chatbots, making the text customizable and seemingly relevant to legitimate chatter on the stream. An illustrative example of bot messages being short, noisy and indistinguishable from genuine messages is shown in Figure I(top-right).

VI. Proposed Framework: SHERLOCK

We next propose SHERLOCK, a two-stage framework which solves Problems 1 and 2 as discussed below.

A. Stage I: Detecting Chatbotted Streams

Given the set of streams $\mathcal{S}$, we first aim to detect the chatbotted streams $\mathcal{S}_{cb}$. Based on observations from Section V, we aim to featurize streams in a space that can best differentiate chatbotted and genuine streams. We discuss our features below:

Number of messages. Observation 1 shows that chatbotted streams tend to have higher numbers of messages than genuine ones. Though many summary statistics can be extracted from the number of messages distribution, we found that weighted top-k modes (most frequent values) worked well empirically, as they represented the k largest "peaks" in the distribution. We were interested in capturing (possibly multiple) spikes in the distribution (for example, see Figure 2), which are generally associated with chatbotting activities. We used $k = 3$ to avoid introducing noise. Further, we weighed each of the k peaks with the associated fraction of users, allowing us to capture the intensity and overall contribution of the peak. Intuitively, peaks at large number of messages, and with high fraction of users are the most suspicious. This produces 3 features.

IMD quantiles. Observation 2 reflects that chatbotted streams tend to have higher IMDs than genuine ones. Moreover, many chatbots have spiky behavior which involves long lulls between chat messages. To capture the spikes and the overall higher IMD of chatbots, we used higher quantiles of the stream IMD distribution ($60\%^{ile}$, $70\%^{ile}$, $80\%^{ile}$, $90\%^{ile}$).

Number of windows. Observation 3 posits that since chatbots send messages atypically, and spread throughout the chat (rather than in quick conversations), they appear in higher numbers of windows than genuine users. Thus, a stream with many chatbots will likely have a number of window distribution with peaks associated with chatbot behaviors. Following the same rationale as before, we take the weighted top-k modes, again using $k = 3$ to avoid noise.

Concatenating these, we arrive at a 10-dimensional feature space. Next, we train a supervised model over this feature space and use the classifier to predict chatbotting propensity for any new, unseen stream. We add those with a sufficiently confident predictions to $\mathcal{S}_{cb}$.

B. Stage II: Detecting Constituent Chatbots

Upon obtaining a set of chatbotted streams $\mathcal{S}_{cb}$, our goal for each stream $s \in \mathcal{S}_{cb}$, is to label each user $i \in \mathcal{I}$ (relevant chatters) as belonging to real users $\mathcal{I}_r$ or chatbots $\mathcal{I}_{cb}$. We use a semi-supervised learning approach for this stage; such approaches have been demonstrably useful in tasks for which ground truth is limited. In the livestreaming case, collecting ground-truth for individual users as chatbots is highly challenging, time-consuming and unscalable. Thus, we employ a label propagation approach to identify chatbots.

Generating seeds. The success of our label propagation approach for classifying users naturally depends on the goodness of the seed labels. If a stream $s \in \mathcal{S}_{cb}$ has a sufficiently high prediction score, we conjecture that $\mathcal{I}_{cb}$ will be large compared to $\mathcal{I}_r$. With this key assumption, we consider certain regions of our feature space to identify *seed users* for whom we have "high confidence" *seed labels*. We use heuristics based on our earlier observations to obtain these seed labels. Specifically, our approach begins by bootstrapping seed sets using empirically observed highly discriminative features (i.e. high confidence seeds):

Number of messages. Observation 1 notes that chatbots tend to post more messages than genuine users. We denote number of messages sent by chatter i as $\mathbf{m}_i$.

Mean IMD. Observation 2 notes that chatbots tend to have longer IMDs than genuine users. We denote chatter i's mean IMD as $\mathbf{d}_i$.

Subscription status. Many livestreaming platforms offer paid subscription models, where users can pay to subscribe to a streamer. We assume that subscribers are genuine chatters, and can thus be exonerated. We use $\mathbf{r}_i$ to indicate chatter i's subscription status.

Next, we refine the seeds by exploiting synchronicity over less discriminative features to gain confidence in seed veracity; we use the following features:

Number of windows. The message spread of a chatter provides a strong signal if a particular chatter is a bot or not. We count the number of unique windows a chatter i posts a message in and denote it by $\mathbf{w}_i$.

IMD entropy. In addition to computing mean IMD, we also compute entropy of IMD. For each chatter i, entropy of it's inter message delay distribution is given by $\mathbf{h}_i = H(\text{IMD}_i)$.

This approach is summarized in Algorithm 1, which we describe next. We first consider consider all users in $\mathcal{I}$ on $\mathbf{m}$ (number of messages) and $\mathbf{d}$ (mean IMD) (Line 1), as we empirically observed that these features are highly discriminative. In this $(\mathbf{d}, \mathbf{m})$ space, we first remove outliers (Line 2) in sparse regions due to low confidence about their status. Next, we initialize sets $\mathcal{R}_{cb}$ and $\mathcal{R}_r$ with users who have jointly high, and jointly low values on the features; these sets represent candidate bots, and candidate genuine chatters respectively (Lines 3-4). For users in each $\mathcal{R}_{cb}$ and $\mathcal{R}_r$, we next identify the largest cluster of candidate bots and genuine users (we use X-Means clustering [20] as it automates choice of cluster count using information theoretic measures), and add them to the seed set with respective labels (Lines 6-7). We further refine the seeds by exonerating users where $\mathbb{1}(\mathbf{r}_i)$.

Next, we refine $\mathcal{R}_{cb}$ and $\mathcal{R}_r$. To do so, we first construct a bounding box $\mathbf{B}_{cb}$ around $\mathcal{R}_{cb}$ (Line 8), which captures nearby

Algorithm 1: SEEDUSERS

Input: Number of messages vector $\mathbf{m}$, mean IMD vector $\mathbf{d}$, number of windows vector $\mathbf{w}$, IMD entropy vector $\mathbf{h}$, subscriber indicator vector $\mathbf{r}$, synchrony threshold n_{sim}
Output: Refined seed sets $\mathcal{R}_{cb}$, $\mathcal{R}_r$

Project all users into a subset feature space: $\{\mathbf{m}, \mathbf{d}\}$
Remove outliers chatters in this subset feature space.
/* Initialize candidate bot region. */
$\mathcal{R}_{cb} \leftarrow \{i \in \mathcal{I} \,|\, \mathbf{m}_i > \mu(\mathbf{m}) \text{ and } \mathbf{d}_i > \mu(\mathbf{d})\}$
/* Initialize candidate genuine user region. */
$\mathcal{R}_r \leftarrow \{i \in \mathcal{I} \,|\, \mathbf{m}_i < \mu(\mathbf{m}) \text{ and } \mathbf{d}_i < \mu(\mathbf{d})\}$
/* Exonerate users with paid subscriptions. */
$\mathcal{S} \leftarrow \{i \in \mathcal{I} \,|\, \mathbb{1}(\mathbf{r}_i)$
$\mathcal{R}_{cb} \leftarrow$ (largest cluster in $\mathcal{R}_{cb}$) $\setminus \mathcal{S}$
$\mathcal{R}_r \leftarrow$ (largest cluster in $\mathcal{R}_r$) $\cup \mathcal{S}$
/* Track # windows and IMD entropy in candidate bot region. */
Create bounding box $\mathbf{B}_{cb}$ around cluster $\mathcal{R}_{cb}$.
$\mathcal{W} \leftarrow \{\}$ // multiset with freq. $m_W(\cdot)$
$\mathcal{H} \leftarrow \{\}$ // multiset with $m_H(\cdot)$
for *chatter* i *in* $\mathbf{B}_{cb}$ **do**
  $\mathcal{W} \leftarrow \mathcal{W} \cup \{w_i\}$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{h_i\}$
end
/* Augment chatbot seeds with too-synchronous users. */
$\mathcal{W}_{sync} \leftarrow \{w \in \mathcal{W} \,|\, m_W(w) \geq n_{sim}\}$
$\mathcal{H}_{sync} \leftarrow \{h \in H \,|\, m_H(h) \geq n_{sim}\}$
for *chatter* $i \in \mathcal{I}$ **do**
  if $w_i \in \mathcal{W}_{sync}$ *and* $h_i \in \mathcal{H}_{sync}$ **then**
    $\mathcal{R}_{cb} \leftarrow \mathcal{R}_{cb} \cup \{i\}$
    $\mathcal{R}_r \leftarrow \mathcal{R}_r \setminus \{i\}$
end
return $\mathcal{R}_{cb}, \mathcal{R}_r$

users that may be missing in $\mathcal{R}_{cb}$, but may still be suspicious. We then consider the number of windows $\mathbf{w}$ and IMD entropy $\mathbf{h}$ feature values for these users, as we empirically observed that many chatbots tend to share similar values (motivated by Observations 2-3). We identify the feature values that occur over users in $\mathbf{B}_{cb}$ with greater than a given frequency n_{sim} as supposed "peaks" or bot signatures. Given these, we add chatters in $\mathcal{I}$ who have highly recurring feature values to $\mathcal{R}_{cb}$, and also remove them from $\mathcal{R}_r$ if applicable. In effect, our seeding process is a two-level clustering, where the first-level relies on exploiting knowledge of suspicious regions in the $(\mathbf{d}, \mathbf{m})$ space, and the second-level relies on augmenting this with non-region-specific synchronicity in the $(\mathbf{w}, \mathbf{h})$ space. We note that we considered seeding via a single clustering stage in experimentation, but achieved poor results due to noisiness induced by the less-discriminative features.

Propagating suspiciousness. Upon obtaining the seed sets $\mathcal{R}_{cb}$ and $\mathcal{R}_r$, we constructed a k-nearest-neighbors (kNN) graph between all chatters in $\mathcal{I}$ to represent their proximity in the feature-space. Finally, we utilized a graph-based label propagation algorithm proposed in [21], seeding nodes (users) with labels as applicable. We tuned parameters of the propagation algorithm empirically to maximize performance.

VII. EXPERIMENTS

A. Baselines

Although no prior works are directly related to the problem we tackle on livestreaming chatbot detection, we adapt certain spam detection approaches which use user similarity and textual features for this setting.

Table II: Precision and Recall for SHERLOCK, SSC and SynchroTrap on real data.

Model	Genuine Class		Bot Class	
	Precison	Recall	Precision	Recall
SHERLOCK	**97.4%**	**98.6%**	**97.0%**	**94.4%**
SSC	92.6%	96.2%	90.0%	82.8%
SynchroTrap	74.1%	51.8%	35.4%	59.3%

Supervised Spam Classifier (SSC) [22]: We adapt the original work (used for Twitter spam user classification) to our setting. For each user, various features like *max, min, mean, median* of number of words, characters, URLs and IMDs are used to infer in a supervised fashion if user is a chatbot or not. The method works at user-level and does not consider group effects/information at stream level.

SynchroTrap [23]: SynchroTrap is an unsupervised method that operates on user groups; hence, we apply it for each stream to identify constitutent chatbots. We construct edges between any pairs by measuring a soft Jaccard similarity (values are considered similar if they are within small ϵ) between every pair of users. The similarity is computed on two features – (i) IMD, and (ii) number of messages for each user, for every window. We sum the two similarity scores and construct a pairwise similarity graph. We cluster the matrix into two groups via KMeans, and consider the chatbots as the one associated with the group that maximizes performance.

B. Results on Real Dataset

We evaluate SHERLOCK against the two adapted baselines. We evaluate all three methods at the finest applicable granularity, on their eventual detection performance in detecting chatbots. Stage I is applicable only for SHERLOCK, and we evaluate it's performance using 5-fold cross validation. We discover that SHERLOCK correctly identifies 98.3% of streams, reporting a precision of 0.95. We run Stage II only on those streams that are marked as chatbotted in Stage I; thus, for a misclassified genuine stream, all chatbots are false negatives, and vice versa. For SynchroTrap, we evaluate on all 183 streams in our dataset. Similarly for SSC, we evaluate on all users. We report precision/recall values for each method in their capability to identify chatbots in Table II.

We find that SHERLOCK outperforms both SSC and SynchroTrap in precision and recall, despite SHERLOCK only requiring stream-level labels and SSC requiring much harder to obtain user-level labels. We further conjecture that SSC would perform much worse if the chatbot text was more intelligently generated, while our approach would remain unaffected, due to our text-agnostic feature space. SynchroTrap (unsupervised), works at the stream-level and is unable to leverage information from other streams, hence performing the worst.

C. Synthetic Dataset Generation

As real world data is not exhaustive, we perform a set of experiments on a variety of synthetic datasets to test the performance of our approach in Stage I/II under unseen, adversarial settings. We consider only our performance, given

that SynchroTrap is shown to perform poorly in Table II, and SSC only operates at user-level.

To generate a synthetic, labeled chatbotted livestreaming dataset, we performed the following steps. Firstly, we hired a chatbot service provider and had them attack a dummy stream we had setup ourselves. We avoided targeting others' streams to avoid hurting their reputation. We logged all timestamps relative to the beginning of the stream. We then collected chatlogs with timestamps from a variety of popular, Twitch verified profiles which had high subscriber count. Finally, to generate instances of "chatbotted" streams, we superimposed the original ("legitimate") and synthetic ("botted") chatter, while maintaining respective relative timestamps of both sets of messages. This is a reasonable construction strategy since most chatbots behave independently of legitimate conversation dynamics. We additionally vary control parameters configured through the service provider (the number of chatbots active, N_c, and the maximum delay between consecutive messages d_{max}). By varying these two variables, we construct four attack models:

- **Controlled Chatters (CC)**: We fix N_c and vary d_{max}, mimicking an attack mode where streamers use a constant number of chatbots and tweak delays over the stream.
- **Rapid Increase (RI)**: We start with a small N_c and large d_{max}, and rapidly increase the former and decrease the latter, until the former reaches a certain point. This mimicks streamers trying to poorly emulate organic growth and prolonged engagement.
- **Gradual Increase (GI)**: We consider a similar case as RI, but with longer delays between changing N_c and d_{max}, mimicking a more patient attacker.
- **Organic Growth (OG):** We increase N_c over time, but at each increase, we revert to a large d_{max} before decreasing it (in contrast to keeping d_{max} fixed or a given N_c as in RI/GI), and eventually converging. This mimicks an intelligent attacker, trying to prevent sudden growths in number of chat messages.

To create synthetic datasets, we consider the various (a) attack models {CC, RI, GI, OG}, (b) stream duration {0.5, 1, 1.5, 2, 2.5, 3 hours}, (c) ratio of botted to overall messages and (d) ratio of chatbots to real users {40, 60, 80%}. We created multiple simulated attack chatlogs by considering variants of {a,b,c} and {a,b,d}. This labeled dataset is also used to train the Stage I classifier when classifying unseen streams.

D. Results on Synthetic Dataset

By considering various parameters, we generated 945 CC, 180 RI, 149 GI and 939 OG chatbotted streams. For Stage I, we report performance of SHERLOCK using various traditional supervised learning methods, for different attack models. For Stage II, we consider only streams classified as chatbotted in Stage I. We study the effects of the various synthetic chatlog generation parameters mentioned above.

Stage I: We evaluated performance of different supervised classification models over our feature set, and across varying attack models. We used the corrupted versions of legitimate streams as the positive class, and the original legitimate streams as the negative class. All experiments were conducted using 5-fold cross validation – Table III shows F1 score for the different classification and attack models.

Table III: F1 score of SHERLOCK across different classification and attack models (Stage I).

Classifier	CC	RI	GI	OG
Decision Tree	0.884	0.943	0.906	0.881
Random Forest	0.889	0.940	0.922	0.899
SVM	0.775	0.711	0.623	0.781
NN	0.842	0.927	0.902	0.892
NN-MLP	0.852	0.925	0.911	0.833
XGBoost	**0.897**	**0.949**	**0.928**	**0.909**

We found that gradient boosted trees (XGBoost) performed the best amongst the tested methods. Moreover, we discovered that for all classifiers, the CC attack model is the most difficult, while RI is the easiest. We conjecture that this is due to our model's reliance on discriminating IMD features, which are most variant throughout the stream under the CC model (unlike other models, d_{max} never stabilizes in CC).

Stage II: We conduct analysis on all streams marked as botted by the best-performant Stage I classifier. We study the effect of attack model, stream duration, and noise (both ratio of chatbots, and ratio of bot messages). Figure 5 shows the collective results in terms of F1 score.

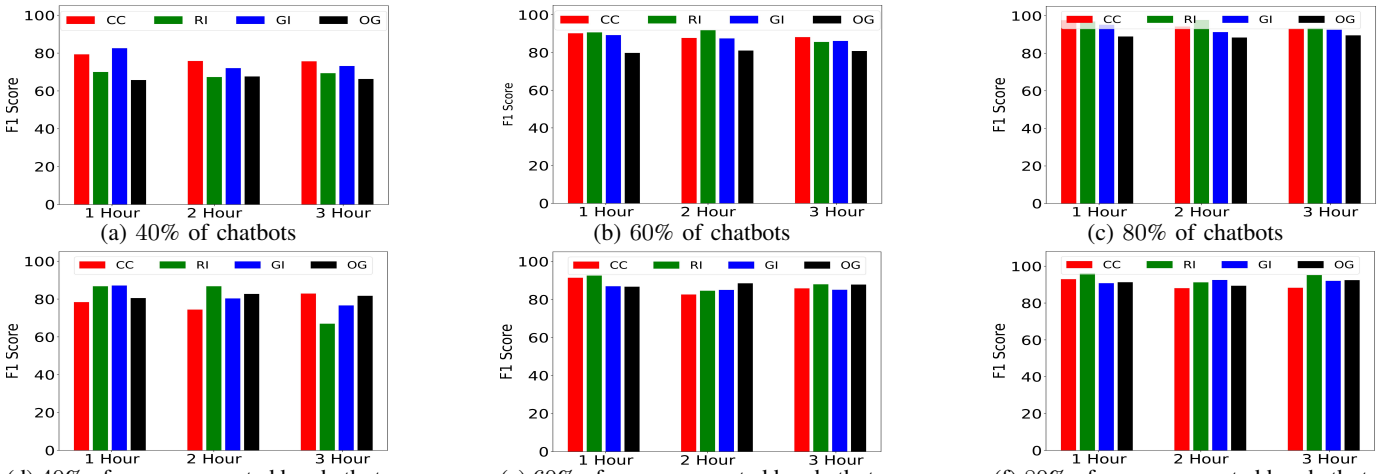
Effect of Attack Model. Unlike in Stage I, we find that the OG model is most challenging. We conjecture that the OG model produces tremendous diversity in the user feature space given many different chatbot configurations, and thus hurts the clustering and propagation steps the most. The GI model proves the easiest to handle; the slow, staggered parameter changes produces several close-by microclusters, which are well-handled by the label propagation.

Effect of Duration. Figures 5(a-c) and (d-f) show that duration impacts performance minimally, with slight reduction for higher durations, likely due to increased IMD variety in genuine behaviors.

Effect of Noise. We alter between two types of noise models, based on the bot message and bot user ratios. In both cases, increasing the chatbot noise percentage improves performance across various attack models and durations for most configurations. For example, F1 score improves from $78.38(40\%)$, to $91.39(60\%)$, and $92.94(80\%)$ for the 2-hour, CC model, bot user noise setting (red bars in (a-c)). Naturally, higher chatbot signal accentuates the features we use for chatbot seeding and label propagation, lending to better separation.

E. Scalability

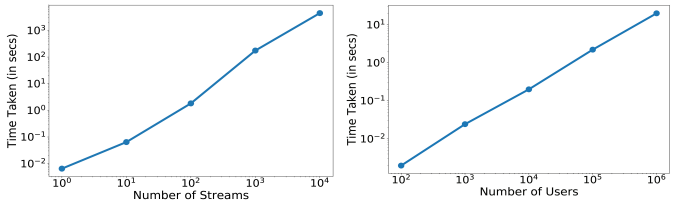
Our two-stage approach is designed to scale naturally, as Stage II (more demanding) works on a significantly reduced set of streams. We evaluate SHERLOCK's scalability in terms of both stages. For Stage I, we generate a synthetic dataset with varying number of streams and show runtime in Figure 6(a). For Stage II, we measure time for seeding and propagation; despite $O(kn^2)$ worst-case complexity for k neighbors and n users, Figure 6(b) shows near-linear convergence in practice.



(a) 40% of chatbots (b) 60% of chatbots (c) 80% of chatbots

(d) 40% of messages posted by chatbots (e) 60% of messages posted by chatbots (f) 80% of messages posted by chatbots

Fig. 5: Performance of SHERLOCK on various attack models (bar colors), stream durations (bar groups), noise levels (columns) and noise types (bot users in (a-c), and bot messages in (d-f)). SHERLOCK is robust to noise and performs consistently well across varying adversarial configurations, with F1 scores generally over 0.80.



(a) Stage I (b) Stage II

Fig. 6: SHERLOCK has near-linear runtime in (a) # streams (Stage I) and (b) # users (Stage II).

VIII. CONCLUSIONS

In this work, we tackle the problem of detecting chatbots on livestreaming platforms. Chatbot detection is important due to its direct impact on recommendation, user trust and monetization for these services. We make several contributions in this paper: We are the first to introduce and formalize the chatbot detection problem in the livestreaming setting. Next, we collect and annotate a real-world livestreaming chat dataset from Twitch.tv and compare and contrast genuine and chatbot user behaviors, by identifying key differentiators. We additionally discuss a strategy for obtaining realistic chatlogs with varying attack types and signatures, and employ it in our experimentation. Based on our observations, we propose SHERLOCK, a two-stage approach for detecting chatbotted streams and users with limited supervision. Finally, we evaluate SHERLOCK's effectiveness on both - a real-world dataset (achieving .97 precision/recall), and a synthetically generated dataset, showing robustness under various intelligent attack models (achieving 0.80+ F1 score across most settings), and also demonstrate near-linear empirical runtime.

REFERENCES

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

[2] E. Chow, "Crowd culture & community interaction on twitch. tv," 2016.

[3] P. Martineau, *Inside YouTubes Fake Views Economy*, 2018.

[4] M. DiPietro, *On Artifical viewers, Followers, and Chat Activity*, 2016.

[5] S. Gianvecchio, M. Xie, Z. Wu, and H. Wang, "Humans and bots in internet chat: measurement, analysis, and automated classification," *IEEE/ACM Transactions On Networking*, 2011.

[6] J. P. McIntire, L. K. McIntire, and P. R. Havig, "Methods for chatbot detection in distributed text-based communications," in *ISCTS*, 2010.

[7] D. Guan, C.-M. Chen, and J.-B. Lin, "Anomaly based malicious url detection in instant messaging," in *JWIS*, 2009.

[8] N. Shah, A. Beutel, B. Gallagher, and C. Faloutsos, "Spotting suspicious link behavior with fbox: An adversarial perspective," in *ICDM*, 2014.

[9] N. Shah, H. Lamba, A. Beutel, and C. Faloutsos, "The many faces of link fraud," in *ICDM*, 2017.

[10] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos, "Copycatch: stopping group attacks by spotting lockstep behavior in social networks," in *WWW*, 2013.

[11] L. Akoglu, R. Chandy, and C. Faloutsos, "Opinion fraud detection in online reviews by network effects." *ICWSM*, 2013.

[12] H. Lamba, B. Hooi, K. Shin, C. Faloutsos, and J. Pfeffer, "zooRank: Ranking suspicious entities in time-evolving tensors," in *PKDD*, 2017.

[13] L. C. Klopfenstein, S. Delpriori, S. Malatini, and A. Bogliolo, "The rise of bots: A survey of conversational interfaces, patterns, and paradigms," in *DIS*, 2017.

[14] M. Jiang, P. Cui, A. Beutel, C. Faloutsos, and S. Yang, "Catchsync: catching synchronized behavior in large directed graphs," in *KDD*, 2014.

[15] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos, "Timecrunch: Interpretable dynamic graph summarization," in *KDD*. ACM, 2015.

[16] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "Sybilguard: defending against sybil attacks via social networks," *ACM SIGCOMM Computer Communication Review*, 2006.

[17] L. Akoglu, M. McGlohon, and C. Faloutsos, "Oddball: Spotting anomalies in weighted graphs," in *PAKDD*, 2010.

[18] A. Ferraz Costa, Y. Yamaguchi, A. Juci Machado Traina, C. Traina, Jr., and C. Faloutsos, "Rsc: Mining and modeling temporal activity in social media," in *KDD*, 2015.

[19] N. Shah, "Flock: Combating astroturfing on livestreaming platforms," in *WWW*, 2017.

[20] D. Pelleg and A. W. Moore, "X-means: Extending k-means with efficient estimation of the number of clusters," in *ICML*, 2000.

[21] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," ser. NIPS'03, 2003.

[22] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida, "Detecting spammers on twitter," in *CEAS*, 2010.

[23] Q. Cao, X. Yang, J. Yu, and C. Palow, "Uncovering large groups of active malicious accounts in online social networks," in *CCS*, 2014.