Deep Entity Classification: Abusive Account Detection for Online Social Networks

Teng Xu *Gerard Goossen* *Huseyin Kerem Cevahir* *Sara Khodeir* *Yingyezhe Jin*
Frank Li *Shawn Shan* *Sagar Patel* *David Freeman* *Paul Pearce*

Facebook, Inc *University of Chicago* *Georgia Institute of Technology*

Abstract

Online social networks (OSNs) attract attackers that use *abusive accounts* to conduct malicious activities for economic, political, and personal gain. In response, OSNs often deploy abusive account classifiers using machine learning (ML) approaches. However, a practical, effective ML-based defense requires carefully engineering features that are robust to adversarial manipulation, obtaining enough ground truth labeled data for model training, and designing a system that can scale to all active accounts on an OSN (potentially in the billions).

To address these challenges we present *Deep Entity Classification (DEC)*, an ML framework that detects abusive accounts in OSNs that have evaded other, traditional abuse detection systems. We leverage the insight that while accounts in isolation may be difficult to classify, their embeddings in the social graph—the network structure, properties, and behaviors of themselves and those around them—are fundamentally difficult for attackers to replicate or manipulate *at scale*. Our system:

- Extracts "deep features" of accounts by aggregating properties and behavioral features from their direct and indirect neighbors in the social graph.
- Employs a "multi-stage multi-task learning" (MS-MTL) paradigm that leverages imprecise ground truth data by consuming, in separate stages, both a small number of high-precision human-labeled samples and a large amount of lower-precision automated labels. This architecture results in a single model that provides high-precision classification for multiple types of abusive accounts.
- Scales to billions of users through various sampling and reclassification strategies that reduce system load.

DEC has been deployed at Facebook, where it classifies all users continuously, resulting in an estimated reduction of abusive accounts on the network by 27% beyond those already detected by other, traditional methods.

1 Introduction

Online Social Networks (OSNs) connect billions of users around the globe. The largest social network, Facebook, has more than two billion active users sharing content each month [45]. The vast scale of these networks in turn attracts adversaries that seek to exploit the platforms for economic, political, and personal gain. While most OSN activity comes from legitimate users, attackers invest significant resources in signing up fake accounts (i.e., accounts not representative of a real person), creating accounts that impersonate real people, or compromising the accounts of real users. These *abusive accounts* are used to drive a range of negative behaviors including spam, fake engagement, pornography, violence, and terrorism—all actions which violate community norms [12] and are widely studied forms of abuse [1].

A core challenge faced by OSNs is how to identify and remediate abusive accounts in such a way that is both *scalable* and *precise*. Scalability requires approaches that can operate on billions of users and tens of billions of daily actions to detect dozens of different abuse types. Systems that prioritize precision are necessary because abusive accounts are relatively rare [44, 45] and thus a drop in precision would lead to the OSN taking errant actions against a large number of benign users.

OSNs use a broad set of techniques ranging from rule-based heuristics [49] to modern machine-learning algorithms [26, 48] to classify and remediate abusive accounts at scale. Rule-based heuristics act as a first line of defense [4], identifying basic or common attacker tools, techniques, and resources. These heuristics however lack power: they focus on precision rather than recall, they often do not capture the complexity of account *behaviors*, and they are by definition *reactive* [25]. Machine learning systems overcome some of these problems: they generalize from past labeled data in order to improve recall, and they can be iterated on over time to adapt to adversarial evolution [8]. However, precise machine learning systems require a large amount of high-quality labeled ground truth data, can be costly to deploy (in both engineering effort and computational resources), and can be evaded by adversaries who learn how to mimic the appearance of real accounts [17]. Rule-based heuristics and traditional machine learning systems can identify and remediate

the vast majority of abuse [4], but identifying the remaining hard-to-classify accounts—those that closely resemble real users and/or evade OSN defenses—requires fundamentally different and more complex solutions.

A critical insight is that while attackers can produce abusive accounts that appear legitimate *in isolation*, those accounts' embedding in and engagement with the social graph are fundamentally difficult to forge. For example, the number of friend requests sent by a given user is easy for an attacker to control, but the number of friend requests sent by all of that user's friends is outside of the attacker's control.[1] Although attackers can attempt to camouflage their accounts by connecting to legitimate nodes in the graph, this strategy not only is prohibitive to implement at scale, but also creates side effects (e.g., large numbers of rejected friend requests) that are detectable by traditional means.

Leveraging this insight, we develop *Deep Entity Classification (DEC)*,[2] a method and supporting system for OSN abusive account detection. Instead of classifying accounts based on "direct" features and behaviors, DEC leverages social network structure, extracting more than 20,000 features for each account, by operating across the graph. These features are used to train supervised machine learning models that *classify accounts across many different kinds of abuse.* The DEC system consists of label generation and feature extraction, as well as model training, deployment, and updating. Ultimately DEC produces per-account abusive classification results that are robust to adversarial iteration (Section 7).

The large number of features generated by DEC's graph traversal imposes two challenges in terms of model training. First, if applied naïvely, the large feature space could dramatically increase the underlying model complexity, resulting in poor generalization and degraded performance. Second, obtaining proper generalization across so many features would require a prohibitively large training set in a problem space where high-quality human-labeled data is difficult to obtain at billion-user scale.

The second key DEC insight is that in addition to small-scale, high-quality human-labeled data, we can utilize the results of rule-based heuristics as additional "approximate labels." The classifications from such rules are not human reviewed and thus have lower precision than human-reviewed data, but the absolute quantity is much higher.

Building on this insight, we design a "multi-stage multi-task learning" (MS-MTL) framework. Our framework extracts low-dimensional transferable representations via a deep neural network trained using the high-volume approximate labels, then fine-tunes dedicated models given the learned representations and the high-quality human-labeled data.

Model training occurs in two separate stages. The first stage trains a *multi-task* deep neural network [6] on the collected features using the large number of lower-precision approximate labels. Since accounts identified by these lower-precision signals exhibit a multitude of different abuse types (e.g., spam, objectionable content, or malware), we formulate a learning "task" for each abuse type. We then extract the penultimate layer of the neural network as a low-dimensional feature vector [22]. This vector is input to the second stage of the model, which is trained using *per-task* high-precision human-labeled data with a standard binary classifier.

MS-MTL allows DEC to learn the underlying common representations of different abuse types in the first model stage, and then to distinguish different abuse types using high-precision data with separate models in the second stage, resulting in a score for each abuse type for each account. In this way we can use a single model to label as "abusive" accounts exhibiting any of a multitude of abuse types (e.g., scams, spam, adult content, etc.).

Our DEC design is deployed at Facebook, where it has run in production for more than two years. During that time DEC led to the identification and remediation of hundreds of millions of abusive accounts. By comparing the number of accounts actioned by DEC with an unbiased estimate of the number of abusive accounts remaining on the platform, we infer that DEC is responsible for reducing the volume of abusive accounts by approximately 27%.

In summary, our contributions include:

- The algorithmic design, system architecture, and implementation of DEC. Extracting more than 20,000 features per entity, across multiple hops, for billions of active users, presents a unique set of systems challenges (Section 4).
- A novel feature extraction process that produces "deep features" (Section 5) that, over our evaluation, showed no signs of adversarial adaptation (Section 7.4).
- The MS-MTL classification paradigm, which allows us to use a single model architecture to produce high-precision classifiers for each abuse class (Section 6).
- A quantitative evaluation of DEC and MS-MTL vs. other approaches, as well as a qualitative assessment of the impact DEC has had on the overall state of abusive accounts not caught by other systems (i.e., those hardest to classify) at Facebook (Section 7).
- A discussion of the lessons learned from two years of production deployment at Facebook (Section 8).

2 Background

Here we present an overview of abusive accounts on OSNs, existing defenses, and relevant machine learning terminology.

2.1 Abusive Accounts

We define an *abusive account* to be any account that violates the written policies of a given OSN (e.g., [12]). Attackers use abusive accounts for various reasons, including for financially motivated schemes (e.g., spreading spam, scams, objection-

[1] See Section 8.4 for consideration of the case where attacker creates groups of abusive accounts that are connected to each other.

[2] In this context "deep" refers to the features generated via network fanout from each account, not neural network structure.

able content, or phishing links [13–15]) and for causing user harm (e.g., online harassment or terrorism [16]). Abusive accounts can be broadly broken down along two dimensions:

1. **Account Provenance.** An abusive account can be *fake*, where the account does not represent an actual person or organization, or *real*, where it is a legitimate user account, though potentially hijacked by an attacker.[3]
2. **Abusive Behavior.** An abusive account can be characterized by the type of abuse it conducts, such as spreading *scams* or *spam*.

2.2 Defenses

There are multiple types of defenses against abusive accounts on OSNs. Rule-based heuristics, such as rate limits on particular user actions, are straightforward, easy to design and evaluate, and can be quite powerful in practice. However, they are often reactive, permitting some amount of abuse before a threshold is crossed and a rule is triggered. In addition, they conservatively focus on precision rather than recall to avoid false positives.

Another large-scale detection technique is machine learning-based classification, which affords increased complexity of the detection algorithm through digesting more features. However, adversaries can adapt (sometimes quickly) in response to classifier actions [10], making it challenging to properly design features that are difficult for adversaries to discover and evade. Another challenge of this approach is to collect enough high-precision training data. Human labeling is typically the most reliable source but can be expensive in terms of time, money, and human effort.

Rule-based heuristics and typical machine-learning based classifiers are able to identify the vast majority of abusive activity in online services [4]. Identifying those accounts that are able to evade the primary detection systems presents a especially difficult challenge, as they represent the *hardest* to classify accounts. For example, such accounts may be those that adversaries have iterated on while adapting to OSN defenses, or they may very closely resemble real users. The system we present in this paper is designed to mitigate these issues by employing sparse aggregated features on the social graph that should be difficult for attackers to manipulate, and by using a multi-stage training framework.

2.3 Machine Learning Terminology

In this section we describe the machine learning terminology relevant to DEC.

2.3.1 Deep Neural Networks

The first stage of DEC uses a deep neural network (DNN) architecture [31]. It is a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input. In deep learning, each layer learns to transform its input data into a slightly more abstract and composite representation, with the last layer outputting a single score.

2.3.2 Embeddings

In the context of neural networks, *embeddings* are low-dimensional, continuous, learned vector representations of a discrete feature vector. Neural network embeddings are useful because they can reduce the dimensionality of categorical variables and meaningfully represent categories in the transformed space [28]. A common usage of embeddings is to serve as input features for machine learning models. In each layer of a deep neural network, a low-dimensional vector can be extracted as the embedding of the layer.

2.3.3 Gradient Boosted Decision Trees

The embedding of the last layer of deep neural network in DEC's first stage is used as the input feature vector for the second stage of DEC training, which uses a model of gradient boosted decision trees (GBDTs). GBDTs are a machine learning approach that iteratively constructs an ensemble of weak decision tree learners through boosting. It is a widely used algorithm in classification and regression [20].

3 Related Work

The problem of detecting abusive accounts in OSNs has received a great deal of attention in the literature. We split the published efforts into three categories based on technique, and also describe the relevant machine learning literature.

3.1 Detecting Abusive Accounts

Several works have explored using graph structure and the features of neighboring nodes to detect abuse. Yang et al. examined the effectiveness of graph and neighbor-based features to identify spammers on Twitter [58]. Their work formalized 24 detection features—including four graph-based and three direct neighbor properties—showing how these features could identify spammers better than prior state-of-the-art solutions [32, 49, 53]. Our work creates a generalized machine-learning framework (utilizing these features among many others) based on graph, direct, and indirect neighbor features (the "deep entity") which scales to billions of social network users.

Other work has focused exclusively on graph structure, with the goal of identifying groups or connected components. Stringhini et al. produced EVILCHORT, a system designed to identify accounts with common networking resources (e.g., IP addresses) and ultimately generate groups of malicious actors [50]. Earlier, Zhao et al. created BotGraph, which creates an activity graph from user actions and uses that graph to identify tightly connected components indicative of abuse [64]. Instead of focusing on the structure of the graph, Nilizadeh et al. observed how spam moved through the graph to identify common propagation patterns [38]. Compared to these works, we

[3] Real user accounts that violate OSN policies *without* having been compromised are outside the scope of this work, as they are relatively small in volume and are actioned on by other systems.

focus on a generalized framework which leverages such features, as well as a scalable machine learning approach which is utilized continuously at Facebook.

An alternative approach uses "honeypot" accounts to ultimately yield features which could be used for detection. Stringhini et al. used honeypot Twitter accounts to collect direct account, behavior, and content signals which could be used to identify spammers [49]. Similarly, Lee et al. also used honeypot Twitter and myspace accounts to collect direct account, content, and timing signals, also identifying abuse [32]. The features from both these works were later formalized and further analyzed (along with other features) by Yang et al. [58].

3.2 Sybil Accounts

A *Sybil attack* refers to an attack where individual malicious users join the OSN multiple times under multiple fake identities. Many algorithms and systems have been proposed to defend against Sybil attacks.

Yu [61] conduct a comprehensive study comparing various Sybil defenses on social networks as of 2011. A typical graph theory-based Sybil defense systems is SybilGuard [63]. The protocol is based on the social graph among user identities, where an edge between two identities indicates a human-established trust relationship. The key observation is that malicious users can create many identities but few trust relationships. Thus there is a disproportionately small "cut" in the graph between the sybil nodes and the honest nodes. However, there are two downsides of SybilGuard: it can allow a large number of sybil nodes to be accepted, and it assumes that social networks are fast mixing, which has not been confirmed in the real world. Yu et al. [62] propose a SybilLimit protocol that leverages the same insight as SybilGuard but offers near-optimal guarantees. Yang et al. claim that sybils do not form tight knit communities, as other work has explored [59]; instead, linkages are formed between sybils and normal users "accidentally" and therefore tight linkage-based defenses in isolation are problematic.

SybilInfer, proposed by Danezis and Mittal [9], is another sybil detection system. It uses a probabilistic model of honest social networks and a Bayesian inference engine that returns potential regions of dishonest nodes. SybilRank [5] is a detection framework that has been deployed in Tuenti's operation center. It relies on social graph properties to rank users according to their perceived likelihood of being fake, and has been shown to be computationally efficient and scalable. Wang et al. [54] take a different appproach, instead focusing on user actions as a stream and making the observation that the stream of actions for some types of attacks will be different than that of regular users.

While most Sybil defense algorithms and systems focus on exploring connections inside the social graph, this approach may fail to detect some types of abuse such as compromised accounts since they are not distinguishable on the social graph. DEC instead operates by combining information from the social graph with direct user features to conduct general abuse classification, irrespective of Sybil properties.

3.3 User Footprint

A "user footprint" is a signal that can be used to identify the behaviors of a same user across different OSNs. Malhotra et al. [37] propose the use of publicly available information to create a digital footprint of any user using social media services. This footprint can be used to detect malicious behaviors across different OSN platforms. Xiangnan et al. [29] study the problem of inferring anchor links across multiple heterogeneous social networks to detect users with multiple accounts. The key idea is that if a user is abusive on one platform, they are likely to be abusive on other platforms. However, the user footprint is not helpful when a user is only dedicated to spreading abuse in a single platform, which is the focus of DEC.

3.4 Machine Learning

In this section we describe the relevant machine learning works that DEC draws inspiration from.

3.4.1 ML for Abuse Detection

Machine learning-based classification is widely used in abuse detection. Stein et al. [48] proposed one of the first machine learning frameworks for abuse detection, applied to Facebook in 2011. The system extracts users' behavioral features and trains a machine learning model for classification. A similar spam detection system using content attributes and user behavior attributes has been deployed on Twitter as described by Benevenuto et al. [3]. These efforts laid the groundwork for our "behavioral" model described in Section 7.2.

Fire et al. [18] propose the use of topological anomalies on the social graph to identify spammers and fake profiles. Their approach uses only four features per user, all of which are related to the degree of graph connection of the user and their friends. The approach is proven to be useful in various OSNs. For DEC we employed a similar approach for feature extraction, however with a greatly expanded feature space.

In terms of classification algorithms, Tan et al. [51] designed an unsupervised spam detection scheme, called UNIK. Instead of detecting spammers directly, UNIK works by deliberately removing non-spammers from the network, leveraging both the social graph and the user-link graph. In the context of supervised learning, Lin et al. [35] conducted experiments on a Twitter dataset to compare the performance of a wide range of mainstream machine learning algorithms, aiming to identify the ones offering satisfactory detection performance and stability based on a large amount of ground truth data.

3.4.2 Other Relevant ML Work

Recent advances in machine learning, especially in graph learning, transfer learning, and online learning, can also be applied to ML-based abusive account detection.

Graph learning seeks to learn a node embedding or make predictions using relations in the graph. Variants of the techniques have been applied to modeling social networks [40], object interactions [24], citation networks [27], and abstract data structures in program verification [34]. Perozzi et al. [40] proposed an unsupervised graph learning technique to learn node embeddings using random walks in the local graph. Recent works on graph neural networks (GNNs) [27, 33, 55] extend convolutional neural networks to perform node classifications. However, none of the existing graph learning approaches has been shown to scale to billions of nodes as in a typical OSN social graph. We are actively experimenting with GNNs for DEC and have encountered numerous technical challenges in getting the system to work on a graph as large and diverse as that of an OSN. Our exploratory work does suggest potential improvements in model performance, but at a much higher computational cost for training.

Transfer learning uses existing pre-trained models or embeddings as a basis for training models for new tasks. The technique is commonly used to improve the performance of ML models (e.g., facial recognition or image segmentation [39, 60]), especially in cases where little labeled training data is available. In DEC, we leverage transfer learning to boost our model performance by training the first-stage embedding on a second set of labels.

Online learning, first proposed by Saad et al. [43], is a technique to tune existing ML classifiers in real time using newly available training data. Classified samples are sent for labeling, which updates the training set to better capture potential adaptive behaviors; retraining then strengthens the classifier against such behaviors [2]. In theory DEC could be adapted to incorporate online learning; however, our human labels are expensive and take a long time to collect, so the benefit of online learning over our current approach of regular offline retraining would be minimal.

Active learning [7, 46], similar to online learning, is a technique to retrain the model with new data. In active learning, only the data points in which the model has low confidence are assigned to human labellers for review. This approach is intended to achieve maximum model performance improvement with limited labeling resource. In our work we select accounts *at random* for expert labelling. While active learning is a potential avenue for improvement, we have been unable to test it because of labeling constraints: random-sample labeling is used not only for training DEC but also for other applications across Facebook, so any active learning experiments would require additional labellers.

4 DEC System Overview

DEC extracts features from active Facebook accounts, classifies them, and then takes actions on the classified abusive accounts. In order to deploy such a system in a scalable way, we need to address multiple challenges, including scalability, latency, variety of abuse types, and false positives. DEC uses multiple components in order to handle these challenges separately.

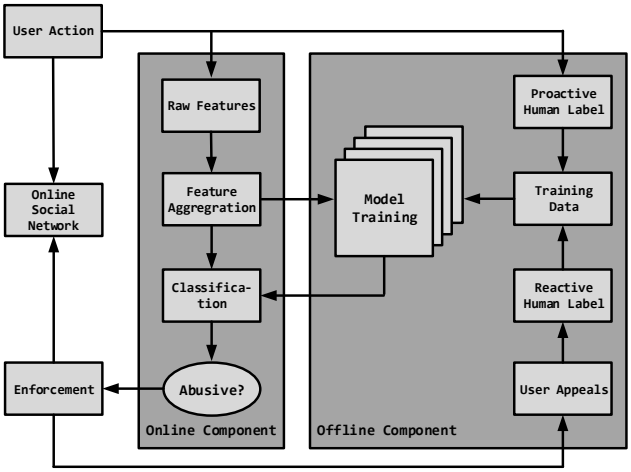

Figure 1: DEC system overview. When an user action occurs on Facebook, the online component will, concurrent with user activity, classify and potentially begin remediation on the user and/or action. Meanwhile, the extracted features from the online component, together with the training data, are used by the offline component to train new models.

Figure 1 shows the DEC architecture. At the highest level, we break down DEC into online and offline components, discussed subsequently.

4.1 Online Component

DEC is triggered by Facebook user actions. When an action occurs, DEC may, based on heuristics (see Section 5.2), schedule a task concurrent with the user activity to start extracting the raw features for the target node and sampled neighboring nodes. For an average account on Facebook, DEC needs to extract hundreds of features for each of hundreds of neighboring nodes, resulting in tens of thousands of raw features to be extracted. Such queries are computationally expensive, and thus the whole process is done asynchronously offline without influencing the user's normal site activity. After feature extraction, DEC aggregates the raw features to form numerical sparse features (further discussed in Section 6). DEC then generates the classification result for the account based on the aggregated features and the in-production model. If the account is classified as abusive, DEC exercises enforcement on the account.

4.2 Offline Component

The offline component of DEC includes model training, and feedback handling.

To classify multiple types of abuse, DEC maintains multiple models, where each model handles a different type of abuse. Each dedicated model is trained on the learned low-dimensional embeddings from the raw features collected as part of the concurrent feature extraction (online component). DEC uses the MS-MTL training framework to simultaneously

train and maintain models for different abuse types (further discussed in Section 6).

As part of our implementation within Facebook, DEC has integrated both human labeling as well as user feedback into the training and enforcement process. Facebook uses a dedicated team of specialists who can label whether an account is abusive. These specialists label accounts both proactively (based on features) and reactively (based on user feedback). For proactive labeling, human labellers check accounts surfaced by various detection signals, take samples, label them, and then take actions accordingly. For the reactive labeling, the process begins when a user appeals an enforcement action (as surfaced through the Facebook product). A human reviewer then investigates the account and either accepts the appeal (false positive from DEC's perspective) or rejects the appeal (true positive). Both proactive and reactive human label results are fed into DEC model training as labeled data. Offline model training uses the human labeled data combined with the extracted features from the online component. After repeated offline and online testing, updated models are deployed into production. DEC is regularly retrained by Facebook to leverage the most recent abuse patterns and signals.

To summarize, DEC:

1. Extracts "deep features" across all active accounts on Facebook to allow classification.
2. Uses classification to predict the level of abusiveness for all active accounts, keeping up-to-date classification results for all users actively engaging with the network.
3. Incorporates user and labeler feedback to iterate classifier models.

5 Methods: Deep Feature Extraction

Feature extraction is a core part of DEC. Compared to traditional abuse detection systems, DEC uses the process of aggregate feature calculations which aims to extract deep features of a "target" account.

5.1 Deep features

In the context of DEC, "deep" refers to the process of fanning out in the social graph. This graph consists of not only users but all *entities* that the platform supports, such as groups, posts, and more. A *direct feature* is a feature that is a function of a particular entity only, such as account age or group size. A *deep feature* is a feature that is a function of the direct features of entities linked to the entity in question. For example, "average age of an account's friends" is a deep feature for the account. Deep features can be defined recursively, as aggregations of deep features on linked accounts; for example, a deep feature on a photo could be "average number of groups joined by friends of people tagged in the photo."

Deep features are useful for classification because they reveal the position of target node in social graph by looking at neighboring nodes. For instance, in the detection of fake accounts, a common pattern that can be revealed by deep features is the batch creation of fake accounts. When classifying fake accounts, deep features include the features from the IP address that registers the account, as well as all the other accounts created from the IP address. When classifying using the above features, the scripted activity of batch account registration can be easily detected.

Table 1: Types of entities with their example direct features and example deep entities in DEC.

Entity Type	Direct Features	Deep Entities
User	age, gender	entities administered, posts
Group	member count, age	admins, group members
Device	operating system	users sharing the device
Photo	like count, hash value	users in the photo
Status Update	like count, age	groups it shared to
Group Post	has a link?	users commenting
Share	number of times shared	original creator
IP Address	country, reputation	registered accounts

A key insight is that deep features not only give additional information about an account, but also are difficult for adversaries to manipulate. Most direct features can easily be changed by the person controlling the entity. For example, account age is controlled by the account owner, and group membership is controlled by the group admin. In contrast, aggregated features that are generated from entities associated with the target account are much more difficult to change. For example, if we consider the age of all of a user's friends, the mean value would be much more difficult to alter by that user, especially when the number of friends is large. Eventually, we can even take a step further by scrutinizing all the friends of friends, and it becomes almost impossible for an adversary to completely change such information.

Table 1 lists some of the entity types considered by DEC, including user, group, device, photo, status update, and group post. For each entity type, we list a few examples of direct features and deep (or fan-out) entities. For direct features, we use features effectively leveraged by other ML classifiers, as well as those found useful during manual investigations.

Figure 2 illustrates an example deep feature. This feature is based on neighboring nodes within two hops from an example account (center, color orange). An edge between two nodes represents the relation of mutual friends. This 2-hop deep feature has exponentially more dependent values comprising the feature than a direct feature.

5.2 Implementation

To extend the above examples to work in production, we have three issues to address: (a) What kind of neighboring nodes do we look at? (b) How can we generate the deep features meaningfully? and (c) How do we keep the computational cost from exploding as we fan out?

The complex and varied nature of OSN products requires us to build our system as generically as possible, allowing us to incorporate a wide variety of entities and edges between them. We also want to be able to add new types of entities or edges as

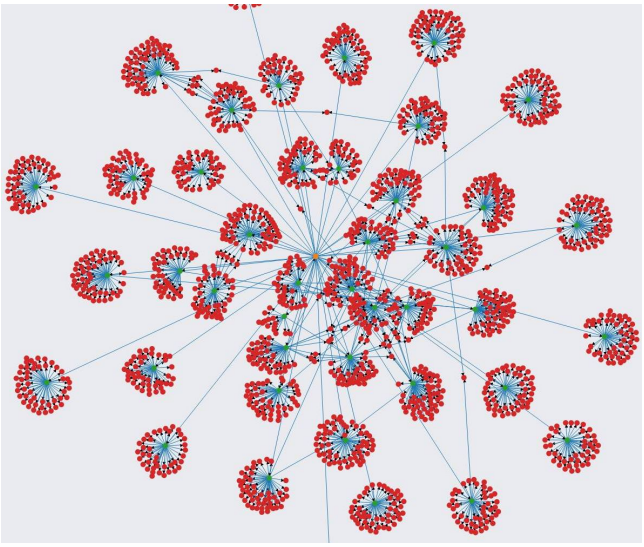

Figure 2: Visualization of the level-2 social graph for a single "target" account in DEC. The centered orange node is the target node to classify. The blue nodes are the neighboring nodes from the first fan-out level. The red nodes are from the second fan-out level. An edge between two nodes represents the relation of mutual friends. For each node visualized in this graph, hundreds of features are extracted and aggregated for classification.

new features and products appear on Facebook. In the social graph, even a single pair of entities can be connected with multiple types of edges. For example, a user can be connected to a group by being the admin of the group. They can also be connected through membership, which is a weaker connection. Even further, a user can be connected by commenting on a post from the group.

To define deep features, we apply aggregation techniques on the set of direct features of nodes, following the lead of Xiao et al. [57], who effectively leveraged aggregated features across clusters of accounts to identify fake ones. As shown in Table 2, we use different aggregation methods for numerical features and categorical features. To aggregate numerical features such as age, we calculate statistics on their distribution such as mean and percentiles. On the other hand, for categorical features such as home country, our strategy is to aggregate them statistically into numerical features. Lastly, we also jointly aggregate numeric features with categorical features by observing the distribution of the numeric features for a given categorical feature. For example, a feature can be the number of accounts that logged in from the same device as the target account, given the device uses the Android operating system.

The use of aggregation has two advantages: first, it produces a dense feature vector, reducing the dimensionality of the model. Second, it helps the model resist adversarial adaptation as discussed in Section 5.1 above. Note that we do not need to define each deep feature explicitly: we can define various graph traversal steps (e.g., user → user, or user → group → photo) and automatically apply all aggregation methods to all the direct features of the target entity. In practice, this method produces thousands of distinct deep features.

Table 2: Example aggregation methods for deep features. Here $p25$ and $p75$ refer to the 25th and 75th percentiles, respectively.

Feature Type	Aggregation Method
Numeric	min, max, mean, variance, $p25$, $p75$
Categorical	percentage of the most common category, percentage of empty values, entropy of the category values, number of distinct categories
Both Numeric & Categorical	max of numeric A from category B, p75 of numeric A from most common category

Ideally, we would trigger a new feature extraction and classification *every* time a user action happens on Facebook. This is not possible at billion-user scale given the necessary computational resources. DEC relies on heuristics to decide when to begin the process of feature extraction and (re-)classification. The core idea is the use of a "cool-down period" between reclassifications, where the length of the cool-down period increases as the account spends more time active on the platform. Our motivating intuition is that accounts that have been active for longer have gone through many previous checks and are generally less likely to be abusive, while newly registered accounts are more likely to be created to abuse.

While (re-)classification is triggered in production in real time, feature extraction and aggregation are computed asynchronously without interfering with an account's experience on Facebook. Given the expense of extracting all deep features, especially for an account with many connections in the social graph, we restrict the amount of computational resources used per account. Specifically, we place a limit on the number of neighboring nodes used to compute a deep feature, and sample randomly if the number is over the limit. The random sample is different on each reclassification; our goal is to capture the position of the entity in the graph from many different angles. This sampling procedure allows us to limit computational cost without reducing the diversity of features.[4]

5.3 Feature selection

We only use deep features of a target account, and not direct features, for classification in DEC. The primary motivation for this choice is that we observed that direct target account features are extremely likely to become dominant features in the model. This undesired dominance is caused by the bias inherent in our training data. For example, one of our

[4] In our implementation, we use up to 50 neighboring nodes to compute a deep feature, downsampling if the number of neighboring nodes exceeds that threshold. On average, two fan-out levels of neighboring entities are used for feature computations.

experimental spam detection models used whether a user posts a URL as a feature; it turns out that this feature easily becomes the dominant one in the model because spammers are much more likely to include URLs in their posts than benign users. However, it creates a huge number of false positives as it classifies almost all users posting URLs as abusive. In addition, direct features are easy for the attacker to manipulate; once the attacker learns that "has posted URL" is a feature, they can switch from directly posting URLs to putting URLs as overlay in a photo in order to avoid detection.

5.4 Feature modification

As adversaries adapt and as we gain new insights about their behavior, we will wish to add new features to DEC and/or retire poorly performing features to save computation cost. There are two issues to consider when modifying features. The first is the influence on the current detection model. Once we add or remove any feature, the classification result from the original DEC model will be influenced as the model is still trained using the original list of features. Our solution is to split the feature logging into two pipelines: experimental features and production features. We can log (or not log) newly added (or removed) features into the experimental group, from which we can train a new model. Meanwhile, the production classifier still uses the production list of features. When the new model is pushed to production, we switch the experimental feature set into the production pipeline.

A second problem with adding features is the computational cost of re-computing across the entire graph. When we add a new direct feature to an entity A, it not only influences A, but also all the connected entities because they use features from A to calculate their own deep features. Conversely, most direct features have multiple dependent deep features, and multiple levels of fan-out can easily require the recomputation of the whole feature space when a single feature is added. For example, DEC needs to extract `new_feature` from all of the friends of friends in order to compute 75th percentile, $p75$(`friends.friends.new_feature`). Traversing through other features along with friends ultimately results in re-extracting features of any active entity. To limit the impact of the re-computation overhead, we define isolated *universes of features*. The old and new versions of features will run in parallel universes, with existing models using the old universe of features, until feature generation for the new universe is complete. At that point the functionality of the old universe is subsumed, and it can be discarded as new models will be trained using the new universe of features.

Again referring to Figure 2, we see the potential computational impact of feature changes. In this example a change or addition of a new direct feature with dependent deep features has exponentially more dependent computations than the direct feature.

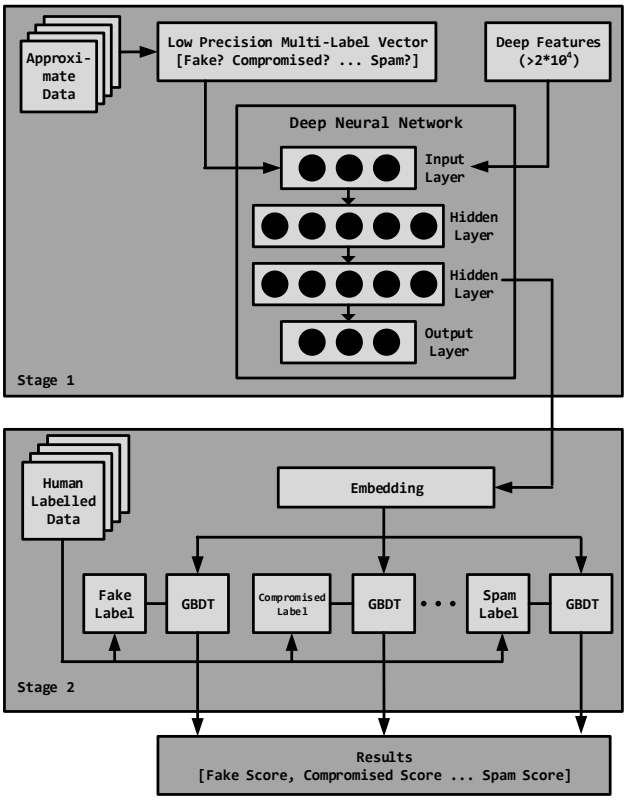

Figure 3: MS-MTL model training flow. Stage 1 uses the raw deep features with low precision labels to train a multi-task deep neural network. By extracting the embedding from the last hidden layer of the deep neural network, we train dedicated GBDT models for each task in stage 2 with human labeled data.

6 Methods: Multi-Stage Multi-Task Learning

Multi-task learning [6] (MTL) is a type of transfer learning [41] used to improve model generalization. MTL trains multiple related "tasks" in parallel using a single neural network model. The core idea is that what the model learns for each task can boost the performance of other tasks. In our context of abusive account classification, we define "task" and "label" as follows:

- A *task* refers to the classification of a specific category of abusive accounts on an OSN (e.g., fake accounts, spamming accounts).
- A *label* of a training sample is a boolean value indicating whether or not the sample falls into an abusive account category. Each training example has multiple labels, one for each task. This *multi-label* is represented by a vector of boolean values.

As a concrete example, if we take four tasks in DEC model training to be classifying fake, compromised, spamming, and scamming accounts, the label vector of one account might be $[1,0,0,1]$. This vector indicates the account was identified as fake and carrying out scams, but is not identified as compromised or spreading spam.

6.1 Motivation

We employ a multi-stage framework to detect abusive accounts on Facebook. Our framework addresses three key challenges in abusive account classification: simultaneously supporting a variety of abuse types, leveraging a high-dimensional feature space, and overcoming a shortage (relative to billions of accounts) of high quality human labels.

First, since there are many different ways in which an account can be abusive, we use different tasks to represent different sub-types of abuse, and multi-task learning to increase the amount of information encoded in the model. The underlying assumption is that the features distinguishing abusive accounts from benign ones are correlated between abuse types. As a result, the knowledge learned for one abuse type can be beneficial for determining other abuse types because an account exhibiting one abuse type is more likely to show other abusive behaviors. As compared with splitting labeled data based on abuse types and training a separate model for each type, multi-task training gives us a full picture of the account by collectively looking at all associated abusive behavior. We expect that this knowledge sharing across tasks will allow us to achieve better prediction accuracy using multi-task learning, especially for smaller tasks.

Second, the multi-stage framework addresses the "curse of dimensionality" [23] by reducing the high-dimensional raw feature vector to a low-dimensional representation. Specifically, our two stages of training reduce the number of features from more than 10^4 (raw deep feature space) to around 10^2 (learned low-dimensional representation space). We achieve this reduction by using the embedding from the last hidden layer of the multi-task deep neural network as input features for the second stage of training.

Finally, a practical engineering problem is that human labeled data is very expensive, and particularly so in the domain of account labeling. In order to label an account as abusive or benign, a human reviewer needs to look at many aspects of the account and consider multiple factors when making a decision. On the other hand, we have a large amount of lower-confidence labeled data in the form of machine-generated labels. This scenario is ideal for multi-task leaning as it has proven to be successful to extract useful information from noisily labeled data [52].

6.2 Training Data Collection

We have two sources of data labels on abusive accounts in DEC. The first consists of human reviewers, who are shown hundreds of signals from each account and asked to provide a judgment on whether the account is abusive. Labels provided in this manner have high accuracy, but are also computationally expensive, and therefore can only be obtained in low volume (relative to the billions of accounts on Facebook).

The second label source consists of automated (non-DEC) algorithms designed to detect abusive accounts, as well as user reported abusive accounts. These algorithms may be focused on a specific attack or abuse type, or may be previous versions of global abuse detection models. We consider the accounts identified by these algorithms to be *approximately labeled* abusive accounts. We then split the labels into different tasks based on the type of abuse per each account. To obtain approximately labeled non-abusive accounts, we randomly sample accounts that have never been actioned on. Our approximate labels have lower precision than human reviewed data, but are much cheaper to obtain and can be obtained in high volume. For example, in our evaluation the training dataset has over 30 million approximate labels and only 240,000 human labels (Table 3).

While 30 million labels may seem significant, it represents less than 2% of the billions of accounts on Facebook. Thus, any adversary attempting a poisoning attack [21, 36, 47] on the training data would need to create thousands of accounts in order to ensure that some of them were sampled for our training set as negative examples (and tens of thousands if trying to poison the second stage). On the other hand, the fact that there are millions of negative samples implies that any one account cannot have outsize influence on the model, thus increasing the required attack size even further. Such large attacks are easy for both rule-based systems and human reviewers to detect and label, and thus the adversary's intention of poisioning the training set will be foiled. Furthermore, even if somehow the adversary obtains enough accounts to poison the training process, they will need to manipulate the features on these accounts to produce very specific values, which (as discussed in Section 5.1) is difficult to achieve with our "deep feature" architecture.

To provide insight into the reliability of this approach, we took a random sample of approximately labeled accounts and sent them through the manual review process described previously. In those experiments the approximate labeling precision varied between 90% and 95%, indicating that the approximate labels still provide significant discerning power.

6.3 Model Training Flow

Figure 3 shows the two stage training flow of the MS-MTL framework. The first stage, trained on a large volume of low precision data, learns the embedding of the raw features. We then apply a transfer learning techique and use the embedding along with high precision labels to train the second stage model. The classification results are generated as the outputs from the second stage.

6.3.1 First Stage: Low Precision Training

The objective of the first training stage is to reduce the high-dimensional vector of aggregated raw deep features to a low-dimensional embedding vector. This dimensionality reduction is done through the training of a multi-task deep neural network model [6] using our approximate label data. Each sample in the training data has a vector of labels where each label corresponds to a task, and each task corresponds to classifica-

tion of a sub-type of abusive accounts on Facebook. After the training has converged, we take the outputs of the last hidden layer of the neural network as the learned low-dimensional embeddings.

For our implementation, we use a neural network model with 3 fully connected hidden layers having 512, 64, and 32 neurons respectively. For each task, the model outputs a probability using a sigmoid activation function. The inputs are normalized using a Box-Cox transformation. We trained the model using PyTorch [42] for an epoch using per-task binary cross entropy and an Adagrad optimizer [11], with a learning rate of 0.01.

6.3.2 Second Stage: High Precision Training

We leverage a technique from transfer learning [41] and extract the last hidden layer's output from the first stage model as the input for the second stage. We train the second stage (GBDT model) with high precision human-labeled data to classify abusive accounts regardless of the sub-types of violations. The scores output by the GBDT model are the final DEC classification scores.

Our implementation of the GBDT model uses an ensemble of 7 trees with a maximum depth of 4. We trained the model with a company-internal gradient boosting framework similar to XGBoost [56], using penalized stochastic gradient boosting, with a learning rate of 0.03 and a feature sampling rate of 0.2.

7 Evaluation

In this section we evaluate the performance of our MS-MTL approach and the DEC system as a whole. Specifically we analyze three abusive account models:

1. A behavioral-only model, which represents traditional detection techniques employed by OSNs;
2. DEC as a single multi-task neural network ("Single Stage," SS), and
3. DEC with MS-MTL.

We performed our evaluation on *active* accounts on Facebook. These accounts have already gone through multiple early-stage security systems such as registration or login-time actioning, but have not yet gone through full behavioral (i.e., activity- and content-based) detection. We also investigate adversarial adaptation, in particular looking at the stability of DEC's precision and recall over time.

7.1 Datasets

Table 3 summarizes the dataset used for our experiments and evaluation of DEC.

Training Data. We test DEC's performance on production Facebook data. We consider four types of abusive accounts (tasks) in our MS-MTL implementation: fake, compromised, spam, and scam. We split the abuse types into these four different categories for two reasons. First, they are violating different policies of Facebook, which causes the detected accounts to be actioned on by separate enforcement systems, each employing distinct appeals flows. Second, the positive samples of different abuse types are not homogeneous by nature. For example, fake accounts are largely driven by scripted creation, while compromised accounts usually result from malware or phishing. The behavioral patterns and social connections of these accounts are distinctive for each abuse type, lending themselves well to different "tasks" in our formulation.

Table 3: Datasets: Number and composition of labels used for our training and evaluation. The longitudinal dataset is measured in # of samples per day.

Training Dataset	Label Type	Training Stage	# Samples
Fake	Approximate	First	3.0×10^7
Comp.	Approximate	First	7.8×10^5
Spam	Approximate	First	6.2×10^5
Scam	Approximate	First	6.2×10^5
Benign	Approximate	First	2.6×10^8
Abusive	Human	Second	1.2×10^5
Benign	Human	Second	1.2×10^5
Evaluation Dataset	**Label Type**	**Evaluation Mechanism**	**# Samples**
Abusive	Human	Offline	3.0×10^4
Benign	Human	Offline	3.0×10^4
Longitudinal	Human	Online	2.0×10^4/day

We maintain separate datasets of approximate (lower-precision) and human labels. The quantity of approximate labels is significantly larger than human labels. The first training stage uses four approximate datasets of abusive accounts and one of benign accounts, while the second stage requires only human-reviewed accounts labeled as abusive or benign. The approximately labeled data comes from three sources:

1. **User reports**: Users on Facebook can report other users as abusive. This source is noisy [19], but appropriate as low-precision labels for the first stage of training.
2. **Rule-based systems**: Outside of DEC, there are other existing enforcement rules on Facebook. We take users caught by these enforcements, categorized by the type of abuse, as an additional approximate label source. Some examples of users labeled by rule-based systems include:
   - Users sending friend requests too quickly;
   - Users with multiple items of content deleted by spam-detection systems;
   - Users distributing links to known phishing domains.

   In total, rule-based systems account for more than half of our abusive account labels.
3. **Discovered attacks**: It's common to have "waves" of scripted attacks on OSNs, such as malware or phishing attacks. When Facebook notices such a wave they can identify a "signature" for the accounts involved and use the signature as an approximate label for our first stage. These discovered attacks comprise approximately 10% of our abusive account labels.

All of the above sources provide noisy, low-precision abuse data. While inappropriate for full system training, they are apt

for the first stage of training. For the first stage, we construct a set of benign users by randomly sampling active users and excluding those contained in approximate abuse dataset.

In contrast, we generate training data for the second stage by having human labellers employed by Facebook manually review randomly sampled users on the platform. Accounts labeled as abusive are used as positive samples for training, and accounts labeled as benign are negative samples.

Evaluation Data. To evaluate DEC's performance, we create an evaluation dataset of accounts by sampling *active* users from Facebook. These are users that have already passed through several early-stage abuse detection systems, and as such contain the *hardest* abusive accounts to classify. We perform manual human labeling of a large number of randomly selected accounts using the same methodology and process that Facebook uses for ground truth measurement. We then randomly select 3×10^4 accounts labeled abusive and 3×10^4 accounts labeled benign for offline evaluation.

7.2 Model Evaluation

We use three different models to evaluate the performance of our DEC approach (single stage and with MS-MTL) both in isolation, and in comparison to traditional techniques. Note that the objective of DEC is to identify accounts committing a wide spectrum of abuse types. This approach goes beyond traditional Sybil defense techniques which primarily focus on detecting fake accounts.

A summary of these models, their training data, and their evaluation data can be found in Table 4. The three models we compare are:

1. **Behavioral**: This GBDT model classifies accounts based only on the direct behavioral features of each account (e.g., number of friends), and outputs whether the account is abusive (regardless of the specific abuse type). Thus, this model does not use deep features and is not multi-task. Since the number of behavioral features is relatively small, we train the model with the human labeled dataset. This model is representative of traditional ML based detection techniques used in OSNs, similar to the system described by Stein et al. [48]. By operating on an evaluation dataset drawn from active accounts on Facebook that have already undergone early-stage remediation, adding this behavioral (later-stage) system is representative of an end-to-end solution. We employ a GBDT architecture with an ensemble of 200 trees of depth of 16, each with 32 leaf nodes.
2. **DEC-SS**: This model uses the DEC approach outlined in this paper to extract deep features, but does not leverage the MS-MTL learning approach. A single deep neural network model is trained by combining all the approximate data across multiple tasks. If a user is identified as violating by any one of the included tasks, we consider this as a positive sample. Because of the huge number of features extracted by DEC, the quantity of human labeled data is too small to be used for training.

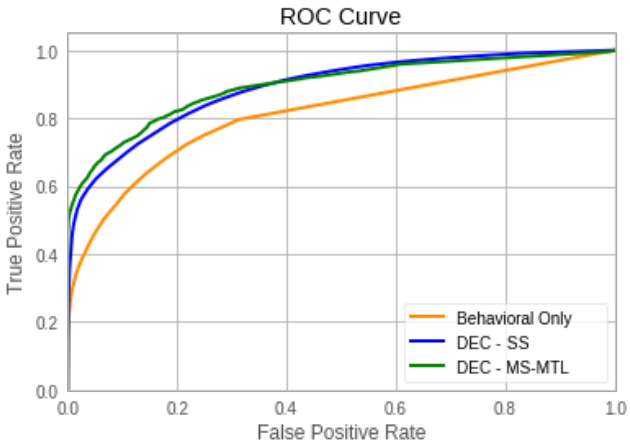

Figure 4: Comparison of ROC curves for different models on evaluation data. Both DEC models (single stage and with MS-MTL) perform significantly better than the behavioral model at all points in the curve.

3. **DEC-MS-MTL**: This is is the complete end-to-end framework and model described in Section 6. It combines the DEC-only approach with MS-MTL.

Outside of this evaluation section, references to DEC without a MS-MTL or SS qualifier refer to DEC MS-MTL.

7.3 Performance Comparisons

We compare various metrics based on the results of above three models.

7.3.1 ROC Curves

Figure 4 examines the ROC performance of all three models. ROC curves capture the trade-off in a classifier between false positives and false negatives. For all operating points on the curve, the DEC models (both MS-MTL and SS) perform significantly better than a behavioral-only approach—by as much as 20%, depending on the operating point. From a ROC perspective, both DEC models perform similarly.

While ROC curves are important measures of the effectiveness of models, they are inherently *scaleless*, as the x-axis considers only ground-truth negatives and the y-axis considers only ground-truth positives. If the dataset is being classified is imbalanced, as is the case with abusive accounts (there are significantly more benign accounts than abusive accounts), ROC curves may not capture the actual operating performance of classification systems—particularly precision, a critical measure in accessing abuse detection systems.

7.3.2 Precision and Recall

Figure 5 compares the precision and recall of the models. We find the behavioral model is unable to obtain precision above 0.95 and has very poor recall throughout the precision range. Both DEC models perform significantly better than the behavioral model, being able to achieve a higher precision and have significantly higher recall at all relevant operating

Table 4: Comparisons of the three evaluation models' type, training features, training data, and evaluation data.

Name	Model	Training Features	Training Data	Evaluation Data
Behaviorial	GBDT	Account behavior features ($\sim 10^2$)	Human labels	Human labels
DEC- SS	Multi-Task DNN	DEC deep features ($\sim 10^4$)	Approximate labels	Human labels
DEC- MS-MTL	Multi-Task DNN + GBDT	DEC deep features ($\sim 10^4$)	Approximate labels+human labels	Human labels

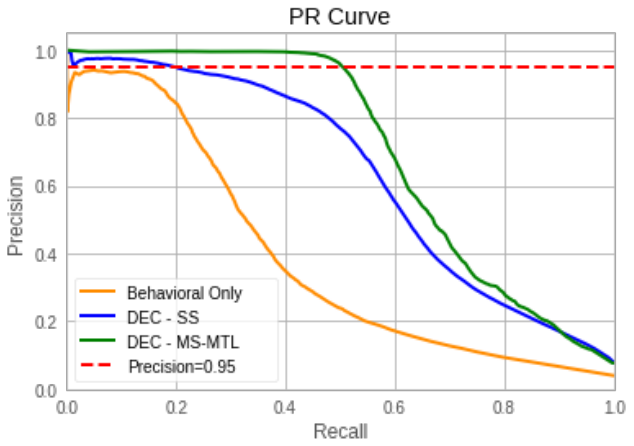

Figure 5: Comparison of precision vs recall curves for different models on our evaluation data. Both DEC models perform significantly better than the behavioral model, and the DEC-MS-MTL has higher recall across the entire operating space. This evaluation is over accounts that have already gone through several stages of security evaluation, and as such this population represents the hardest accounts to classify. Given the difficult classification nature of this sub-population, such recall performance is considered excellent by Facebook.

regions. DEC with MS-MTL significantly improves the system recall over single stage DEC at high precision operating points, improving by as much as 30%.

We note that this evaluation is over accounts that have already gone through other security classifications such as registration time or login-time remediation (i.e., the *hardest* to classify accounts). As such, the overall recall level is expected to be lower than that of a system which operates on all active accounts (Section 7.4).

DEC with MS-MTL's improvement in recall over behavioral models makes it particularly attractive in a real world operating environment where recall over hard to classify accounts is an important operating characteristic.

7.3.3 Quantiative Assessment: Area Under the (AUC) Curve and Precision / Recall

Table 5 shows a comparison of precision, recall, and ROC performance between the three models. ROC performance is calculated as the total area under the curve (AUC). Precision is fixed at 0.95, a common operating point for assessing performance. The behavioral model is unable to achieve a precision of 0.95 at any recall, and is excluded. We find that while

Table 5: Comparison of the area under the curve (AUC) and recall at precision 0.95 for different models on evaluation data. The DEC with MS-MTL model achieves the best result by a significant margin, a nearly 30% absolute improvement. The behavioral model is unable to obtain precision 0.95.

Model	AUC	Recall @ Precision 0.95
Behavioral	0.81	NA
DEC- SS	0.89	0.22
DEC- MS-MTL	0.90	0.50

DEC both single stage and with MS-MTL have similar AUC performance, adding MS-MTL more than doubles the model recall, increasing it from 22% to 50%. This increased performance, both over behavioral and over DEC without MS-MTL, enables significantly better real-world impact when deployed in production.

7.4 Results In Production Environment

Building on our design and evaluation of DEC (with MS-MTL), we deployed the system into production at Facebook. The system not only identified abusive accounts, but also triggered user-facing systems to take action on the accounts identified. To assess the model's real-world impact and longevity, we evaluate our system in production by looking at the stability of precision and recall over time.

Precision Over Time. Figure 6 examines the 3-day moving average of the precision of our DEC with MS-MTL system in production at Facebook. As with our prior evaluation, we obtain ground truth for our measurements by relying on manual human labeling of a random sample of accounts classified as abusive by DEC. We find that the precision of the system is stable, with the precision never dropping below 0.97, and frequently being higher than 0.98.

Recall Over Time. We examine the stability of our production DEC-MS-MTL model's recall by considering its false negative rate (FNR), where FNR $= 1 -$ recall. Using a longitudinal sample of 2×10^4 users randomly chosen and manually labeled each day, we compute an unbiased FNR statistical measure of the volume of abusive accounts on Facebook, regardless of direct detection. This measure is denoted as the "prevalence" of abusive accounts and can be thought of as the false negative rate of all abusive account detection systems (including DEC) combined. If we add to the prevalence measurement the number of abusive accounts caught by DEC

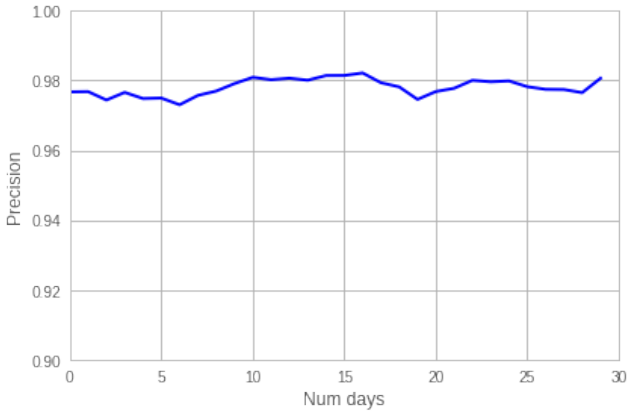

Figure 6: Precision over time: 3-day moving average of deployed (DEC-MS-MTL) model precision on live Facebook production data, spanning one month. Precision is stable, never decreasing below 0.97. The *y*-axis is truncated.

specifically (and not other detection systems), we obtain an estimate of what the prevalence of abusive accounts *would have been* in the absence of DEC.

Figure 7 plots the observed prevalence of abusive accounts (with DEC deployed) and inferred prevalence without DEC, over the period of a month. A loss in DEC's recall (equivalently, an increase in DEC's FNR) would manifest as either an increase in overall abusive account prevalence, or a decrease in the power of DEC compared to non-DEC methods (a decrease in the difference between the two measures). We observed neither of these phenomena over our one-month experiment, indicating that DEC's recall did not meaningfully shift during this period and suggesting that there was not adversarial adaptation to DEC.

Before DEC's launch, Facebook reported instances of adversaries adapting within hours to new detection systems; since the advent of DEC there have been no such reports. Our hypothesis is that the "deep feature" architecture of DEC makes the system more resistant to adversarial adaptation than other abusive account detection systems. As discussed in Section 5.1, an adversary wishing to manipulate a user feature aggregated through the graph must control that feature on *all* of the relevant entities connected to the original user. When we apply this reasoning to the multitude of different entity associations — including but not limited to user friendship, group membership, device ownership, and IP address appearance — we are drawn to the conclusion that manipulating many such features would be far more expensive for an attacker than manipulating "direct" user features such as country, age, or friend count.

Since deployment, DEC has become one of the key abusive account detection systems on Facebook, where it has been responsible for the identification and deactivation of hundreds of millions of accounts. Over our evaluation period the average estimated prevalence without DEC would have been 5.2%, while the average observed volume of abusive accounts

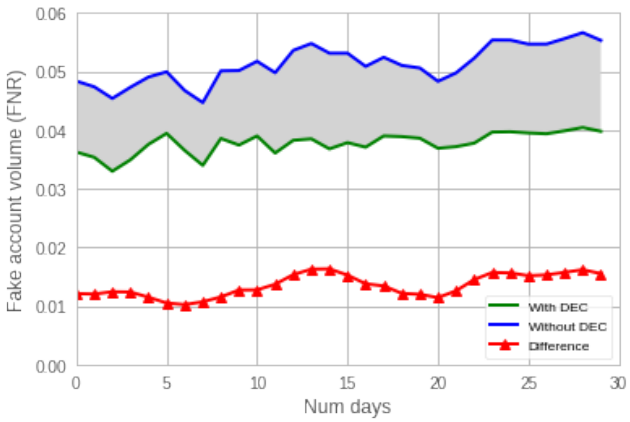

Figure 7: Recall over time: DEC defense over a 30-day window, using 3-day moving averages. The green line is the observed volume (as a percent) of abusive accounts on Facebook, and the red marked line is the volume of accounts taken down by DEC. The blue line is the sum of the other two and estimates what the volume of abusive accounts would have been in the absence of DEC; the gray shaded area thus represents the inferred impact of DEC.

Table 6: Area under the curve (AUC) and recall at precisions 0.95 and 0.99 for DEC over a random sample of all accounts on Facebook.

Population	AUC	Recall @ Prec. 0.95	Recall @ Prec. 0.99
All accts.	0.981	0.981	0.955

on Facebook was 3.8%— an improvement of 27%.

DEC Over All Accounts. Our evaluation of DEC thus far has focused on the *hardest* types of abuse to classify—accounts that were not identified by other production abuse detection systems. A separate question is how effective could DEC be at identifying *all* abusive accounts, including those caught by these other systems. To answer this question we evaluated DEC over 1.6×10^4 active accounts sampled at random from the entire population of accounts on Facebook, including those that had been detected as abusive by other systems. These accounts were definitively labeled by expert human labellers and used as ground truth for our evaluation. Table 6 shows the performance of DEC across this population of all accounts. DEC performs well over this population, with an AUC of 0.981, recall at precision 0.95 of 0.981 and recall at precision 0.99 of 0.955. As expected, both the AUC and recall at fixed precision are significantly higher on the full population than on the sub-population of accounts not detected by other systems (Table 5).

8 Discussion and Lessons Learned

After more than two years of deployment at Facebook, we have learned multiple lessons and identified several limitations from developing and using DEC.

8.1 Reducing Computational & Human Load

It is computationally expensive to extract graph features for all active users at the scale of Facebook. Given our current implementation of feature extraction within two hops from the target node in graph, for each user we might need to reach out to hundreds or thousands of neighboring nodes in order to extract all of their information and aggregate it back to the target node. To mitigate this problem we have developed caching strategies that reuse previous feature extraction results as much as possible. However, because many features have time sensitivity, we still need to update and re-extract a considerable amount of them at each reclassification.

The computational load of DEC is high—equivalent to 0.7% of global CPU resources of Facebook. However, the deployment of DEC actually **reduced** global CPU usage of Facebook. DEC achieved this counter-intuitive result by identifying and removing such a large volume of abusive accounts that the combined CPU usage of those abusive accounts more than accounted for the computation required for feature extraction, training, and deployment of DEC.

DEC also greatly reduced human costs, in terms of human review resources that would have been needed to evaluate and take down abusive accounts manually. DEC's deployment reduced the total review resources needed for abusive account detection by between 15% and 20%.

8.2 Segmentation and Fairness

One key finding is that a single-task classifier performs differently across different segments within the task. For example, if we segment accounts by the self-reported age of their owners, an abusive account classifier might show a higher false positive rate on one age segment than others. Similarly, the performance might vary over different geographies, as we are building a single model to fit a global product that may be used differently across different cultures. Such variation, which can be expected across such a large and heterogenous user base, may be interpreted as the model treating some groups of people unfairly relative to others.[5] In the data set used for this paper we were not able to find any segments on which classifier performance differed to a statistically significant extent, but it is possible that with retraining and/or different segmentation such unfairness may arise. As a result, we have proactively considered several measures to reduce variation across different segments.

Our key insight is that segmentation effects are highly correlated with bias in the training data. Suppose for example that we use the account owner's age as a feature, and that the owners of abusive samples in the training data are younger on average the owners of non-abusive samples. In this case, if we do not adjust the proportions of different segments in our training data, the classifier may reach the conclusion that accounts owned by young people are more likely to be abusive.

As a first step towards preventing such bias, we have removed from the model all "direct" user demographic features, including age, gender, and country. While these features could be helpful in predicting abuse, they could easily introduce unfairness in the model as in the age example above — we don't want to penalize younger benign users just because attackers usually choose to set their fake accounts to have a young age.

The next approach we considered is to sample the labeled data in order to create a training set that reflects overall OSN distributions as closely as possible. In ongoing work, we are experimenting with training DEC using stratified sampling based on attack clustering, in particular downsampling large clusters so as to minimize the influence of a single attack on the ultimate model. This approach would make sure that a large attack from a given user demographic does not teach the model that most users from that demographic are abusive. However, stratified sampling becomes prohibitively costly as we try to match the distribution of more and more segments. In addition, as we add more dimensions the segments get smaller, and statistical noise soon introduces enough error to outweigh the precision gains from sampling.

A final approach is to split particular segments out and create dedicated tasks in the MS-MTL framework for them; however, this approach requires us to collect sufficient training data for each segment, and the maintenance cost increases with the number of models trained. Instead of training and maintaining multiple models, Facebook has chosen to monitor specific high-profile segments for false positive spikes and address any issues by tuning the overall model to reduce segment-specific false positives.

8.3 Measuring in an Adversarial Setting

Since abuse detection systems inherently operate in an adversarial environment, measuring the impact of system changes is a particularly difficult problem. A common adversarial iteration looks like:

1. The attacker finds a successful method to abuse Facebook.
2. Facebook adjusts its detection system and mitigates the attack.
3. The attacker iterates until they either achieve (1) again, or the resource cost becomes too high and they stop.

Assuming constant effort on the part of the attacker and Facebook, the above cycle eventually settles on an equilibrium. Because of this cycle, it is difficult to properly measure the effect of our models using A/B tests during deployment. If our experiment group is too small, we never reach step 3 because the attacker has no incentive to change. Our metrics might look good in the experiment group, but we will hit step 3 when we launch more broadly and performance will decline.

One way to mitigate this problem is to add a "holdout group" to feature launches. The holdout group is a random sample of users that are predicted by the model to be abusive.

[5] Note that the assessment of "fairness" will depend on the metric used, and one may get different results when using, for example, accuracy vs. precision vs. false positive rate.

Instead of acting to block these accounts immediately upon detection, we stand back and confirm the abuse happened as expected before enforcing on these users. Such holdouts help us to more accurately measure the precision of our classifier, but must be carefully weighed against the potential impact, as holdouts can lead to further abuse. For this reason, holdouts are not used for all types of abuse.

8.4 Adversarial Attacks on DEC

An attacker may attempt to *poison* the first stage of low-quality labels by creating numerous colluding accounts that seek to be labelled benign by the rule-based detection systems. Given the scope of DEC's training data and the relatively low sample rate, it would be extremely difficult for attackers to generate such accounts at a scale that would significantly impact the trained model (Section 6.2), especially given that other (non-DEC) systems exist specifically to limit the creation of fake accounts at massive scale.

An attacker may attempt to *evade* the classifier by creating large groups of fake accounts connected to each other so that they can control all of the deep features. This subgraph would have to either be isolated from the rest of the friend graph (which is itself suspicious) or have a reasonable number of connections to the main graph. In the latter case, since DEC operates on second-order connections, almost all of the DEC features would include data from real accounts outside the adversary's control. In addition, while the adversary controls the fake accounts' behavior, they don't know how a similar set of connected *legitimate users* behaves, and the coordinated activity of the fake accounts would be detected as anomalous by DEC.

An attacker could also attempt to *trick* DEC into misclassifying a benign user as abusive, based on features of its neighbors that the victim has no control over. For example, an attacker could create a subgraph of abusive accounts as above and attempt to friend a victim using these accounts. If the victim accepts one or more friend requests, they embed themselves in the abusive sub-graph, which could cause DEC to incorrectly act on the victim. This "forced-embedding" attack is also challenging to execute. First, "attempted" links between entities (e.g., unresolved or denied friend requests) are not features in DEC. Second, a single bad edge between the victim and an abusive sub-graph is insufficient to cause a false classification. A victim would need to be deceived numerous times for there to be a risk of misclassification. Finally, DEC-identified accounts are given the opportunity to complete challenges or request human review as a fail-safe to guard against incorrect classification [30].

8.5 Limitations and Future Directions

While DEC has been highly effective at detecting abusive accounts in practice, its design offers several opportunities for improvement:

- DEC is computationally expensive, particularly due to its use of deep features. However, in Section 8.1 we discussed how this high computational cost is actually balanced by resource savings from identifying more abusive accounts. Reducing the computational cost further is an active area of work that is receiving at least as much attention as improving model quality.
- Intuitively, DEC's classifications are based on an account's position and connections within the Facebook graph. Accounts that exhibit low levels of activity or connections provide fewer signals for DEC to leverage for inference, limiting its effectiveness. However, even if such accounts are abusive, they inherently have less impact on Facebook and its users. We are currently exploring approaches to include features that better capture these low-signal accounts.
- DEC's machine learning model lacks interpretability, as it relies on a DNN to reduce the high-dimensional space of deep features into the low-dimension embedding used for classification decisions. This characteristic makes it difficult to debug and understand the reasoning behind DEC's decisions. Making the model interpretable is an active area of research.
- DEC's approach of aggregating data from many users to produce features for classification is less sensitive to outliers than an approach of using direct features. As a consequence, DEC may be less discriminative of extreme feature values than other model families. We have taken a "defense-in-depth" approach to address this challenge, as extreme outliers can be captured quite effectively by manual rules. It still remains an open question to address such outliers within the DEC framework.
- DEC, like other supervised or semi-supervised machine learning systems, is heavily dependent on the quality of its training data labels. Adversaries that manage to induce inaccurate human labeling at scale may be able to manipulate or interfere with DEC's classifications. We are constantly working to improve our labeling process to address any observed or potential limitations.

Even with these limitations, our evaluation on production data at Facebook indicates that DEC offers better performance than traditional detection approaches.

9 Conclusion

We have presented Deep Entity Classification (DEC), a machine learning framework developed to detect abusive accounts in OSNs. Our framework addresses two problems in the existing abuse detection systems: First, its "deep feature" extraction method creates features that are powerful for classification and (thus far) show no signs of the adversarial adaptation typical for account or behavioral features. Second, it uses a novel machine learning training framework to leverage both high-quantity, low-precision and low-quantity, high-precision training data to improve model performance.

Our evaluation on production data at Facebook indicates that DEC offers better performance than traditional detection

approaches. Moreover, DEC's performance is stable over time, suggesting that it is robust to adversarial adaptation. During DEC's deployment for more than two years at Facebook, it has detected hundreds of millions of abusive accounts. We estimate that DEC is responsible for a 27% reduction in the volume of active abusive accounts on the platform.

10 Acknowledgements

Many individuals at Facebook contributed to the development of DEC and to this publication. Among them we would like to thank Daniel Bernhardt, Scott Renfro, Vishwanath Sarang, and Gregg Stefancik.

We would also like to thank the anonymous reviewers for their valuable feedback that substantially improved this work's quality.

References

[1] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: A survey. In *Data mining and knowledge discovery*, volume 29, pages 626–688, 2015.

[2] Raman Arora, Ofer Dekel, and Ambuj Tewari. Online bandit learning against an adaptive adversary: From regret to policy regret. *arXiv preprint arXiv:1206.6400*, 2012.

[3] Fabricio Benevenuto, Gabriel Magno, Tiago Rodrigues, and Virgilio Almeida. Detecting spammers on Twitter. In *Collaboration, electronic messaging, anti-abuse and spam conference (CEAS)*, volume 6, page 12, 2010.

[4] Elie Bursztein. How to successfully harness AI to combat fraud and abuse. https://elie.net/talk/how-to-successfully-harness-ai-to-combat-fraud-and-abuse/, 2018. RSA.

[5] Qiang Cao, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *USENIX NSDI*, pages 15–15, 2012.

[6] Rich Caruana. Multitask learning. In *Machine learning*, volume 28, pages 41–75. Springer, 1997.

[7] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 4:129–145, 1996.

[8] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *SIGKDD conference on knowledge discovery and data mining (KDD)*, pages 99–108. ACM, 2004.

[9] George Danezis and Prateek Mittal. Sybilinfer: Detecting sybil nodes using social networks. In *NDSS*, pages 1–15, 2009.

[10] Louis DeKoven, Trevor Pottinger, Stefan Savage, Geoffrey Voelker, and Nektarios Leontiadis. Following their footsteps: Characterizing account automation abuse and defenses. In *Internet Measurement Conference (IMC)*, pages 43–55. ACM, 2018.

[11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

[12] Facebook.com. https://www.facebook.com/communitystandards/, 2019.

[13] Facebook.com. https://www.facebook.com/help/287137088110949, 2019.

[14] Facebook.com. https://www.facebook.com/help/166863010078512?helpref=faq_content, 2019.

[15] Facebook.com. https://www.facebook.com/communitystandards/objectionable_content, 2019.

[16] Facebook.com. https://www.facebook.com/communitystandards/safety, 2019.

[17] Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Analysis of classifiers' robustness to adversarial perturbations. In *Machine learning*, volume 107, pages 481–508. Springer, 2018.

[18] Michael Fire, Gilad Katz, and Yuval Elovici. Strangers intrusion detection: Detecting spammers and fake profiles in social networks based on topology anomalies. In *Human journal*, volume 1, pages 26–39, 2012.

[19] David Mandell Freeman. Can you spot the fakes?: On the limitations of user feedback in online social networks. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1093–1102, 2017.

[20] Jerome H Friedman. Greedy function approximation: A gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[21] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[22] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. In *Science*, volume 313, pages 504–507, 2006.

[23] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *ACM symposium on theory of computing*, pages 604–613. ACM, 1998.

[24] Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-RNN: Deep learning on spatio-temporal graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5308–5317, 2016.

[25] Meng Jiang, Peng Cui, and Christos Faloutsos. Suspicious behavior detection: Current trends and future directions. In *IEEE intelligent systems*, volume 31, pages 31–39. IEEE, 2016.

[26] Xin Jin, C Lin, Jiebo Luo, and Jiawei Han. A data mining-based spam detection system for social media networks. In *Proceedings of the VLDB endowment*, volume 4, pages 1458–1461, 2011.

[27] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[28] W. Koehrsen. Embeddings in neural network. https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526, 2018.

[29] Xiangnan Kong, Jiawei Zhang, and Philip S Yu. Inferring anchor links across multiple heterogeneous social networks. In *International conference on information & knowledge management*, pages 179–188. ACM, 2013.

[30] Fedor Kozlov, Isabella Yuen, Jakub Kowalczy, Daniel Bernhardt, David Freeman, Paul Pearce, and Ivan Ivanov. A method for evaluating changes to fake account verification systems. In *RAID*, 2020.

[31] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. In *Nature*, volume 521, page 436, 2015.

[32] Kyumin Lee, James Caverlee, and Steve Webb. Uncovering social spammers: Social honeypots + machine learning. In *Conference on Research and Development in Information Retrieval (SIGIR)*, 2010.

[33] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[34] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[35] Guanjun Lin, Nan Sun, Surya Nepal, Jun Zhang, Yang Xiang, and Houcine Hassan. Statistical Twitter spam detection demystified: Performance, stability and scalability. In *IEEE access*, volume 5, pages 11142–11154. IEEE, 2017.

[36] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *NDSS*, 2017.

[37] Anshu Malhotra, Luam Totti, Wagner Meira Jr, Ponnurangam Kumaraguru, and Virgilio Almeida. Studying user footprints in different online social networks. In *International conference on advances in social networks analysis and mining (ASONAM)*, pages 1065–1070. IEEE Computer Society, 2012.

[38] Shirin Nilizadeh, Francois Labrèche, Alireza Sedighian, Ali Zand, José Fernandez, Christopher Kruegel, Gianluca Stringhini, and Giovanni Vigna. Poised: Spotting Twitter spam off the beaten paths. In *CCS*, 2017.

[39] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[40] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[41] Lorien Y Pratt. Discriminability-based transfer between neural networks. In *Advances in neural information processing systems*, pages 204–211, 1993.

[42] PyTorch. https://pytorch.org/.

[43] David Saad. Online algorithms and stochastic approximations. *Online Learning*, 5:6–3, 1998.

[44] United States Securities and Exchange Commission. Facebook archive form 10-q. https://www.sec.gov/Archives/edgar/data/1326801/000132680117000007/fb-12312016x10k.htm, 2016.

[45] United States Securities and Exchange Commission. Facebook archive form 10-q. https://www.sec.gov/Archives/edgar/data/1326801/000132680118000067/fb-09302018x10q.htm, 2018.

[46] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[47] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*, pages 6103–6113, 2018.

[48] Tao Stein, Erdong Chen, and Karan Mangla. Facebook immune system. In *Workshop on social network systems*, page 8. ACM, 2011.

[49] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Detecting spammers on social networks. In *Annual Computer Security Applications Conference (ACSAC)*, 2010.

[50] Gianluca Stringhini, Pierre Mourlanne, Gregoire Jacob, Manuel Egele, Christopher Kruegel, and Giovanni Vigna. EVILCOHORT: Detecting communities of malicious accounts on online services. In *USENIX Security*, 2015.

[51] Enhua Tan, Lei Guo, Songqing Chen, Xiaodong Zhang, and Yihong Zhao. UNIK: Unsupervised social network spam detection. In *International conference on information & knowledge management*, pages 479–488. ACM, 2013.

[52] Andreas Veit, Neil Alldrin, Gal Chechik, Ivan Krasin, Abhinav Gupta, and Serge J Belongie. Learning from noisy large-scale datasets with minimal supervision. In *CVPR*, pages 6575–6583, 2017.

[53] A. H. Wang. Don't follow me: Spam detection in Twitter. In *Conference on Security and Cryptography (SECRYPT)*, 2010.

[54] Gang Wang, Tristan Konolige, Christo Wilson, Xiao Wang, Haitao Zheng, and Ben Y. Zhao. You are how you click: Clickstream analysis for sybil detection. In *USENIX Security*, 2013.

[55] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*, 2019.

[56] XGBoost. https://xgboost.ai/.

[57] Cao Xiao, David Mandell Freeman, and Theodore Hwa. Detecting clusters of fake accounts in online social networks. In *Workshop on artificial intelligence and security*, pages 91–101. ACM, 2015.

[58] Chao Yang, Robert Chandler Harkreader, and Guofei Gu. Die free or live hard? empirical evaluation and new design for fighting evolving Twitter spammers. In *Conference on Recent Advances in Intrusion Detection (RAID)*, 2011.

[59] Zhi Yang, Christo Wilson, Xiao Wang, Tingting Gao, Ben Y. Zhao, and Yafei Dai. Uncovering social network sybils in the wild. In *Internet Measurement Conference (IMC)*, 2011.

[60] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Proceedings of the Conference on Neural Information Processing Systems*, 2014.

[61] Haifeng Yu. Sybil defenses via social networks: A tutorial and survey. In *ACM SIGACT news*, volume 42, pages 80–101. ACM, 2011.

[62] Haifeng Yu, Phillip B Gibbons, Michael Kaminsky, and Feng Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *Symposium on security and privacy*, pages 3–17. IEEE, 2008.

[63] Haifeng Yu, Michael Kaminsky, Phillip B Gibbons, and Abraham Flaxman. Sybilguard: Defending against sybil attacks via social networks. In *ACM SIGCOMM computer communication review*, volume 36, pages 267–278. ACM, 2006.

[64] Yao Zhao, Yinglian Xie, Fang Yu, Qifa Ke, Yuan Yu, Yan Chen, and Eliot Gillum. Botgraph: Large scale spamming botnet detection. In *USENIX NSDI*, 2009.