On Twitter Bots Behaving Badly: Empirical Study of Code Patterns on GitHub

Andrea Millimaggi and Florian Daniel[0000−0003−3004−8702]

Politecnico di Milano
Via Ponzio 34/5, 20133 Milan, Italy
andrea.millimaggi@mail.polimi.it, florian.daniel@polimi.it

Abstract. Bots, i.e., algorithmically driven entities that behave like humans in online communications, are increasingly infiltrating social conversations on the Web. If not properly prevented, this presence of bots may cause harm to the humans they interact with. This paper aims to understand which types of abuse may lead to harm and whether these can be considered intentional or not. We manually review a dataset of 60 Twitter bot code repositories on GitHub, derive a set of potentially abusive actions, characterize them using a taxonomy of abstract code patterns, and assess the potential abusiveness of the patterns. The study does not only reveal the existence of 31 communication-specific code patterns – which could be used to assess the harmfulness of bot code – but also their presence throughout all studied repositories.

Keywords: Bots · Harm · Abuse · Code patterns · GitHub · Twitter

1 Introduction

Social networks, microblogging or instant messaging services like Facebook, Twitter, Instagram, LinkedIn, WhatsApp, Telegram, and similar are the foundation of the Web 2.0, that is, the Web made of content and services provided by the users themselves. Over the last 15 years, these applications have enabled users all around the world to stay informed, share ideas and discuss opinions. In short, they revolutionized online communication to billions of humans.

In the recent years, a new phenomenon has arisen: *bots*, i.e., algorithmically driven entities that behave like humans in online communications and increasingly participate in conversations without the human participants necessarily being aware of communicating with a machine [8]. State-of-the-art artificial intelligence, speech technology and conversational technology enable the implementation of software agents whose communications are only hardly distinguishable from those by human agents. Combined with generally low transparency about the true nature of bot accounts, humans are easily fooled.

In [6], we have started asking ourselves whether the increasing presence of bots may lead to harmful human-bot interactions that may hurt the human participant in the conversation, and by searching for papers, news, blog posts, and similar we found a variety of anecdotal evidence that this may indeed happen.

Of course, bots are not harmful in general. But sometimes, intentionally or unintentionally, software-driven conversations may just break common conversational rules, etiquette, or even the law. It is important to acknowledge the problem, to be able to provide countermeasures and to prevent people from getting hurt.

As we show in our discussion of related works, most of the literature today focuses on the detection of bots and on telling bots and humans apart starting from the evidence (e.g., posts, comments, likes) that is observable and accessible online. There is very little information on assessing the harms caused by this presence of bots, even less so on the reasons that lead to harm. This paper studies this latter aspect and aims to identify how harm is caused by bots to understand the likely, underlying intentions. Doing so requires looking behind the curtain, away from the content shared online and into the actual code implementing the bots' communication logic. The contributions of this paper are:

- The construction of a *dataset* of social bot GitHub code repositories for Twitter; the analysis focuses on code written in Python and on project metadata.
- An *abuse-oriented classification* of bot code repositories according to how the developers themselves advertise their projects.
- A *qualitative, systematic code review* that identifies 31 potentially *abusive code patterns* that may lead to harmful interactions with human users and a discussion of the possible *intentions* underlying these patterns.
- A *qualitative analysis* of the potential harmfulness of each identified pattern.

Next, we elaborate on the background of the work, then in Section 3 we describe the dataset we use for our study and report on a preliminary analysis of the data. In Section 4, we detail the method underlying the analysis and describe the respective results: actions, patterns and possible consequences. After overviewing related works, we conclude the paper and outline future works.

2 Background

2.1 Harm and abuse in human-bot interactions

Harm occurs when someone suffers an injury or damage, but also when someone gets exposed to a potential adverse effect or danger. In prior work [6], we identified the following types of harm caused by bots:

- *Psychological harm*: it occurs when someone's psychological health or wellbeing gets endangered or injured. An example of a bot causing psychological harm is Boost Juice's Messenger bot that was meant as a funny channel to obtain discounts by mimicking a dating game with fruits but used language that was not appropriate for children (http://bit.ly/2zvNt0E).
- *Legal harm*: it occurs when someone becomes subject to law enforcement or prosecution. A good example is the case of Jeffry van der Goot, a Dutch developer who had to shut down his Twitter bot generating random posts, after it sent out death threats to other users (http://bit.ly/2Dfm71P).

- *Economic harm*: it occurs when someone incurs in monetary cost or loses time that could have been spent differently. For example, in 2014 the bot wise_shibe provided automated answers on Reddit and users rewarded the bot with tips in the digital currency Dogecoin, convinced they were tipping a real user (http://bit.ly/2zu2b6r).
- *Social harm* occurs when someone's image or standing in a community gets affected negatively. An example of a bot causing social harm was documented by Jason Slotkin whose Twitter identity was cloned by a bot, confusing friends and followers (http://bit.ly/2Dfq4DH).
- *Democratic harm* occurs when democratic rules and principles are undermined and society as a whole suffers negative consequences. Bessi and Ferrara [2], for instance, showed that bots were pervasively active in the on-line political discussion of the 2016 U.S. Presidential election.

These types of harm may happen while bots perform regular *actions*, such as posting a message or commenting a message by someone else, that are not harmful per se and that also human users would perform. What needs to happen in order to cause harm is the abusive implementation of these actions. *Abuses* we found are: *disclosing sensitive facts*, *denigrating*, *being grossly offensive*, *being indecent or obscene*, *being threatening*, *making false allegations*, *deceiving users*, *spamming*, *spreading misinformation*, *mimicking interest*, *cloning profiles*, and *invading spaces* that are not meant for bots. Some of these may be subject to legal prosecution (e.g., threatening people), others only breach moral, ethical or social norms, yet they still may be harmful to unprepared human users.

2.2 Platform policies and permissions

In order to properly assess the behavior of a bot, it is important to understand the position of the platforms targeted by bots in relation to automation through bots. For this purpose, we manually surveyed the *usage policies* of a selection of social networks (Facebook, Twitter, Thumblr), instant messaging platforms (Telegram, Whatsapp, Facebook Messenger), platforms for media sharing (Instagram, Pinterest), a professional network (LinkedIn) and Reddit.

All platforms provide developers with *programmable interfaces* (APIs) that can be used for the development of bots; Messenger and Telegram even come with APIs specifically tailored to bots, more specifically, chatbots. Whatsapp is the platform that is most restricted: its Business API allows the implementation of bots, but it seems limited to company use only; however, Android intents (https://bit.ly/2RwjScE) can be used locally on the mobile phone to interact with Whatsapp programmatically. Where an API is provided, it typically allows programmatic access to essentially *all functionalities* that would also be available to users via the platforms' user interfaces. Users of the APIs must *authenticate* with the platforms (the preferred protocol is OAuth) and obtain a *token* enabling programmatic access; only Telegram gives tokens without authentication. All of the studied APIs are *REST APIs*; Facebook and Twitter also provide access to *streaming, live data.* To ease development, some platforms (Facebook, Twitter, Messenger, LinkedIn) are equipped with developer-oriented *software development kits* (SDKs), even in multiple programming languages. Others (Twitter, Instagram) provide more basic programming *libraries*.

As for the usage policies, almost all platforms impose some kind of *limitation*. For instance, "200 calls per hour per user" per app on Facebook. Twitter uses message-level limits, e.g., to prevent aggressive following practices. Only Messenger does not explicitly limit usage and instead even states "you can safely send 250 requests per second." Some platforms impose specific *requirements*, such as "keep your app's negative feedback below our threshold" (Facebook) or "automated bots must respond to any and all input from the user" (Messenger). An explicit *code review* is needed for Facebook, Instagram and Messenger. *Automation* is generally allowed, although commonly limited to actions the target users have explicitly granted permission to; Twitter, for instance, disallows "sending messages in an aggressive or discriminate manner." Most policies even include *content restrictions* like "don't create fake accounts" (Facebook) or "don't send tweets containing links that are misleading." All surveyed platforms explicitly state that they may *suspend* accounts or apps if they violate their policies.

3 Dataset: Twitter Bot Code Repositories

This paper follows a Data Science methodology [9] to extract new knowledge from data. We thus describe here the dataset underlying our study and provide a first analysis of how developers themselves describe their own bot projects.

3.1 Data sources and retrieval

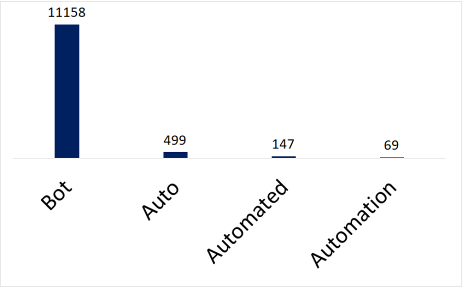

Fig. 1: Distribution of GitHub search results by searched keywords (includes all programming languages).

In this paper, we specifically focus on Twitter (https://twitter.com) and bots written in Python. The former is an opportunistic choice, shared by most literature on the topic (see Section 5 for related works) and is motivated by the openness of Twitter compared to other platforms. The latter stems from the observation that Python is the most used language for Twitter bot implementations in GitHub (35.4% of the repositories we analyzed for Twitter use it). GitHub (https://github.com) is the code hosting service we use for data collection; the choice is again driven by adoption: with about 31M users and 100M projects (or "repositories"), GitHub is today's most used code hosting service (https://www.alexa.com/topsites/category/Computers/Open_Source/Project_Hosting).

In order to identify candidate repositories for our analysis, we used GitHub's search API with a combination of two terms, "Twitter" and any among "bot," "automation," "auto" and "automated." Figure 1 shows the distribution of results obtained by the search considering still all programming languages. As the result of the query "Twitter bot" shows, the term "bot" is highly used for Twitter (we performed similar searches for all platforms mentioned in Section 2.2, and the results distributions do vary from platform to platform). The search represents the state of GitHub as of October 29, 2018, the date the search was performed. For each identified repository, we collected all code files included in the repository as well as a subset of the respective project metadata: URL, programming language, description (a short line of text), and fork/subscriber/watcher counts.

3.2 Preliminary analysis

As the purpose of this paper is to understand how bots implement their interactions with humans, the analysis necessarily requires a manual inspection. This, in turn, requires a careful selection of repositories, in order to keep the size of the dataset manageable and the selected repositories meaningful. Before choosing which repositories to keep and which not, we thus run a simple analysis based on the textual descriptions of the projects in order to obtain a preliminary understanding of which actions the repositories implement.

The analysis followed a top-down approach: We took as starting point the actions identified in our previous work [6], i.e., *talk with user*, *redirect user*, *write post*, *comment post*, *forward post*, *like message*, *follow user*, and *create user*, and matched the retrieved repositories with these action labels. In order to match repositories with action labels, we manually inspected the descriptions of the first 100 items as returned in order of relevance by the GitHub search API and extracted textual keywords from the descriptions. Examples of keywords are: *send messages*, *reply to messages*, *chat*, *post*, *tweet*, *tag*, *poke*, and similar. Then we mapped all keywords to respective action labels, such as $\{$*send messages, reply to messages, read messages, direct message, chat*$\}$ → *talk*.

The mapping exercise produced evidence for the existence in the dataset of all the actions above, plus the addition of 3 new action labels: some projects explicitly claimed to implement a *spam* functionality; others implemented a *poke user* and a *recommend user* functionality.

According to [6], spamming is actually an abuse of the actions *write post* or *forward post*, but we kept it as the descriptions explicitly use the keyword. Poking and recommending users are not functionalities of Twitter: the former is a specific action of Facebook and the latter of LinkedIn, but they appeared anyway in the classification. Very likely the two actions refer to bots that provide cross-platform functionalities, starting from Twitter, which are however out of the scope of this paper.

The goal of this inspection was to enable the automatic labeling of the repositories with

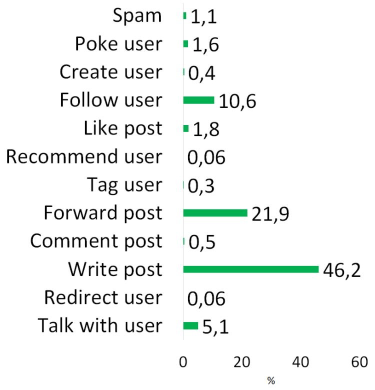

Fig. 2: Labels of repositories.

action labels by analyzing the keywords found in their descriptions and the informed selection of repositories for manual inspection. The automation was achieved by transforming all keywords (and their variants) into regular expressions that could easily be searched for in the repository descriptions. The results of the classification of all retrieved Twitter repositories is shown in Figure 2. It is evident that the most popular action labels are: *follow user*, *forward post*, *write post*, and *talk with user*. Interestingly, the label *like post* is not as important, while all other actions have very little support in the dataset.

3.3 Final dataset

With the goal of maximizing the likelihood of being able to identify recurrent patterns in the code while guaranteeing diversity in the dataset, we applied the following criteria for the selection of the code repositories to be included in the study:

- Selection of repositories that use as main programming language *Python*.
- Exclusion of all those repositories that, after manual inspection, were considered *out of scope*, e.g., because not implementing bots at all or because not implementing any direct communication with other platform users.
- For each of the four most used actions (according to the preliminary analysis), selection of 5 repositories randomly chosen from the respective *best* repositories, according to the ranking provided by GitHub. The respective scores account for the number of forks, clones, likes, and similar. This choice assures that there is a minimum number of popular projects in the dataset for which we can assume to find code patterns with reasonable support.
- For each of the four most used actions, random selection of 5 repositories from the *rest* of the respective retrieved repositories. This choice aims to include also examples that are less popular, while still useful for our analysis.
- Selection of 10 repositories randomly chosen from the *best* repositories we could *not classify* automatically in the preliminary analysis. This assures the presence of a-priori unknown but popular repositories.
- Selection of 10 repositories randomly chosen from the *rest* of the *not classified* repositories, again to assure a representative selection of generic, a-priori unknown repositories.

The final dataset selected for analysis in this paper is thus composed of 60 GitHub Twitter bot repositories whose main programming language is Python. In average, each repository comes with 3 files (standard deviation of 2.02) with an average number of lines of code of 192, an average size of 21.39 KBytes, an average number of subscribers of 3, and an average number of watchers of 13. The most popular repository (twitter-contest-bot, https://github.com/kurozael/twitter-contest-bot) has been forked 99 times, the least popular one (tweetpix, https://github.com/mseri/tweetpix) 0 times, with an average of 5 forks per project across all projects included.

4 Identification and Analysis of Abusive Code Patterns

4.1 Method

To the best of our knowledge, this is the first study that aims to understand and categorize how state-of-the-art social bots implement their interactions with human actors and whether it is possible to identify explicit intentions for the behaviors the bots exhibit in their social communications; no results exist yet. Starting from the dataset described above, we thus perform a manual, systematic review [10] of the code retrieved from GitHub, in order to (i) identify which code passages implement *interactions* with humans, (ii) categorize the concrete *actions* the bots use in their interactions (similar to [6]), and (iii) identify different implementation *patterns* for each categorized action, along with respective *examples* (green field analysis). Actions and patterns were first categorized by one of the authors and then agreed on and integrated by both authors. The result is a conceptual framework composed of actions, patterns and code examples that may allow us to infer the intention behind possible abuses.

4.2 Actions: how bots participate in communications

The preliminary analysis of our dataset in Section 3.2 has shown that Twitter bot developers promise almost all of the typical actions also human users can perform when using social networks. In order to understand which actions are really implemented in the repositories forming our dataset, and how, we reviewed all code files of the dataset manually looking for relevant code fragments. For a code fragment to qualify as *action* it either has to (i) implement some form of interaction by the bot with other users or (ii) implement application logic that manages content or user data fetched from the social network. An action thus represents a self-contained interaction of the bot with content and/or users.

The result of this iteration is summarized in Table 1, which describes the 9 actions that represent a consistent synthesis of all examples identified by this exercise. As expected, the bottom-up analysis brought up a set of typical *social network actions*, declined in Twitter terminology. Bots *follow* other users, *like* their tweets, *tweet* own content, *mention* other users in their tweets, or *retweet* tweets by others. Inside private chat rooms, they also *talk to* other users using instant messages. This result is in line with the actions identified in [6].

But there is more. Looking at the code of the bots further produced three *internal actions* that support their social network actions: bots heavily *search* Twitter for users or tweets, in order to harness accounts and content to work with; they intentionally *pause* or delay their interactions, in order to impersonate users; and they may *store* content they retrieve from the network for later use. These internal actions are observed only in the code of the bots and would not be identifiable by looking at the externally visible communications of the bots only, as done by most literature on the topic. Later in this paper, we will see that also internal actions that are not immediately visible to users may lead to abuses and harm.

Table 1: Synthesis of online communication actions implemented by Twitter bots

Action	Description
Search	Search users or tweets using names, keywords, hashtags, ids or similar or by navigating social network relationships (e.g., friends of friends, followers of friends, friends of followers, followers of followers)
Follow	Follow users to establish social relationships
Like	Like tweets by other users to endorse them
Tweet	Post a new tweet to communicate content
Mention	Mention other users in tweets using @ to attract attention
Retweet	Re-post tweets by other users to endorse them
Talk to	Send direct messages to users to converse with them
Pause	Pause the conversation flow of the bot
Store	Store content retrieved from the social network for later use

4.3 Code patterns: how bots implement their actions

Focusing on the code fragments considered relevant as communication actions, a second iteration of the code review aimed at synthesizing all examples of action implementations into a taxonomy of recurrent code patterns that explains how actions are implemented in practice. For a code fragment to qualify as a *pattern*, two requirements must be met: (i) it must be possible to abstract the fragment and to associate it to at least one action, and (ii) it must recur at least two times in the dataset. A pattern thus represents the intended function of a set of instructions, not their syntactic manifestation in the code.

Even accounting for different names of identifiers in the code, without this type of semantic abstraction it would be necessary to perform a purely syntactic similarity search. However, given the diversity of the repositories and developers that characterize our dataset, only unlikely it would have been possible to spot two fragments that are syntactically equivalent.

For instance, it is possible to interact with the Twitter API using direct, low-level HTTP requests, or one can use a dedicated API wrapper library, such as (in order of use in our dataset): tweepy (http://www.tweepy.org), Twitter libraries (https://bit.ly/2Gg3WJC), TwitterAPI (https://bit.ly/2UwSZri), Twython (https://bit.ly/2aOjCnT), or own, proprietary libraries. Similarly, there are different options for the automatic generation of text for tweets or instant messages, such as nltk (https://www.nltk.org/) or seq2seq (https://bit.ly/2Ry2FQt). Patterns abstract away from these implementation choices and aim to capture the essence of what the developer wanted to implement.

The result of this analysis is reported in Table 2, which names and summarizes the identified patterns. These 31 patterns concisely represent the different interpretations of the 9 actions as implemented in the approximately 140 code examples collected and analyzed.

Example 1. Let us inspect the following two lines of code to understand the logic of the proposed patterns:

Table 2: Taxonomy of code patterns used for the implementation of actions.

Action	**Pattern**	**Description**
Search	*User search*	Search user account by name, keyword, id or similar
	Tweet search	Search tweets by keyword or hashtag
	Trend search	Search trending topics or hashtags by location
Follow	*Indiscriminate follow*	Follow users without checking suitability of users, usernames or content shared
	Whitelist-based follow	Follow only users whose attributes or tweets match some element of a given whitelist
	Blacklist-based follow	Don't follow users whose attributes or tweets satisfy one or more criteria specified in a blacklist
	Phantom follow	Follow users and unfollow them as soon as a given condition is satisfied, e.g, a limit of friends reached or being followed back
Like	*Indiscriminate like*	Like tweets without checking suitability of content, user or username
	Whitelist-based like	Like only tweets by users whose attributes or content match some element of a whitelist
	Blacklist-based like	Don't like tweets whose attributes or users match an element of a blacklist
	Mass like	Aggressively like tweets of given users
Tweet	*Fixed-content tweet*	The content of the tweet is taken from a fixed, static collection of predefined messages
	AI-generated tweet	The text of the tweet is automatically generated using AI/NLP tools
	Trusted source tweet	The content of the tweet is taken from a source that can be considered trusted
	Tweet with opt-in	Tweets are sent only to people who ask to interact with the bot, sending it a message or mentioning it in a tweet
Mention	*Indiscriminate mention*	Mention other users without checking suitability of username or content shared
	Targeted mention	Classify users on the basis of their tweets and mention them in targeted messages
	Whitelist-based mention	Mention only users whose attributes match some element of a whitelist
	Blacklist-based mention	Don't mention users whose attributes match elements of a blacklist
Retweet	*Indiscriminate retweet*	Retweet tweets without checking content or username for suitability
	Whitelist-based retweet	Retweet content only from users whose attributes match some element of a whitelist
	Blacklist-based retweet	Don't retweet tweets whose attributes or users satisfy some condition expressed in a blacklist
	Mass retweet	Aggressively retweet multiple tweets by selected users
Talk to	*Indiscriminate talk*	Send direct, instant messages to users without checking their suitability
	Talk with opt-in	Reply only to messages sent to the bot (passive behavior)
	AI-generated talk	Generate messages using AI/NLP tools
	Fixed-content talk	Take message from a fixed list of predefined phrases
	Targeted talk	Classify users based on their tweets or attributes and target message accordingly
Pause	*Mimic human*	Use pauses in instant messages to deliver human-like conversation experience to other humans
	Satisfy API constraints	Use as short as possible pauses just to avoid being blocked by API usage limitations
Store	*Store persistently*	Store retrieved content or user information for later use

```
for tweet in tweepy.Cursor(api.search, q=QUERY).items():
     tweet.user.follow()
```

The code uses the tweepy library to interact with Twitter and implements two actions: *search* and *follow*. The *search* action is reified by the *search user* pattern (which exact feature is used for the search is unknown as the content of QUERY is not visible). The *follow* action is reified by the *indiscriminate follow* pattern, as line 2 follows all users without applying any filter on the users. ◁

Example 2. The following three lines of code show a concrete implementation of the *blacklist-based mention* pattern:

```
def mentions(count, max_seconds_ago, id_blacklist) :
   return [mention for mention in api.mentions_timeline(count=count)
      if not mention.id in id_blacklist ]
```

The code defines a function that returns all the ids of the users that have mentioned the bot in prior tweets (expressing some form of interest in the bot) and whose ids are not contained in the list of banned ids id_blacklist. ◁

Incidentally, these examples are also representative of two recurrent types of patterns across multiple actions: for all those actions that somehow endorse a user or a tweet (follow, like, mention, retweet), the analysis identified patterns that do so *indiscriminately* or that do so by first checking if the involved user is *blacklisted* or not. Independently of these examples, the analysis also identified other recurrent types of patterns for these actions that endorse users only if they are *whitelisted*. Other notable patterns implement massively repeated actions like *mass like* and *mass retweet*, which aggressively endorse content by given users, or specially targeted actions like *targeted mention* and *targeted talk*, which instead carefully select the users to interact with (e.g., suicide candidates) and send them particularly tailored messages (e.g., to point to help and prevent suicide).

4.4 Effects of actions: assessing potential harmfulness

Considering again the indiscriminate, blacklist and whitelist patterns, it is important to acknowledge that they implement different levels of sensibility of risk as perceived by the developer. Indiscriminately retweeting content expresses either a high level of trust in the users who produce the retweeted content, or it expresses a lack of awareness of the risks that retweeting for example offensive, denigrating or obscene content may have on the reputation of the bot owner. Either way, it becomes evident that each pattern may have a different effect or impact on the users a bot interacts with.

In our prior work [6], we identified 12 major types of abuses bots have committed in the past (see the top-right list in Figure 3) and that have produced harm (remember Section 2.1). The first half of these abuses are legally prosecutable in most democratic countries (see, for example, New Zealands Harmful Digital Communications Act of 2015 [14]). The typical question that remains unanswered when harm occurs is *why* the respective abuse was committed.

Some bots intentionally create spam messages, e.g., to influence political elections [2], but then there are bots like Microsoft's AI-based chatbot Tay that got trained by multiple colluding users, e.g., to offend Jews (http://bit.ly/2DCdqM4). Evidently, the bot was not ready for orchestrated attacks. From the outside, it is generally not possible to tell why abuse happens. This paper provides a look inside the logic that drives bots, and attempts a technical explanation for some of the abuses. In fact, patterns may have the following *effects*:

- *Enable an abuse*, if they implement logic that by design performs an abuse. For example, the *phantom follow* pattern enables mimicking interest for opportunistic reasons, e.g., to be followed back by users, or the *mass retweet* pattern enables artificially boosting the visibility of a user.
- *Prevent an abuse*, if they implement logic that aims to prevent the bot from performing an abuse. The *blacklist-based follow* pattern, for instance, prevents interactions with unwanted users, while the *tweet with opt-in* pattern prevents spamming users not interested in the bot.
- *Be vulnerable to content abuse*, if they implement interactions with users and/or content that may be inappropriate. The *indiscriminate follow* pattern, for instance, causes the bot to follow users that may have inappropriate usernames or spread inappropriate content. The vulnerability may arise when endorsing content or users or when feeding user-provided content to AI algorithms without proper prior checks (see the example of Tay).
- *Be vulnerable to trust abuse*, if they forward, store or analyze content retrieved from users. The *store persistently* pattern is an example of this threat. A user sharing, for instance, sensitive information via personal messages is vulnerable if stored data are leaked to unintended audiences.

These four effects may translate into human users of the social network getting harmed or not. But harm in this context has two sides: if a user gets harmed through interaction with a bot, this may also affect and possibly harm the owner of the bot himself. If a bot threatens someone or discriminates people, the owner may become subject to legal prosecution. If it leaks private data, it may be suspended by the social network, as this violates the usage policies.

Figure 3 graphically summarizes for the patterns in Table 2 (except the *search* patterns without side-effects) which abuses they may enable, prevent or risk to commit. It is meant to create awareness in bot developers of the effects the code they write may have once their bot is deployed and interacting with people.

Coming back to the *why* question and the technical considerations on the possible abuses, it seems reasonable to conclude: (i) that bots that explicitly enable abuses *intentionally* try to do harm or at least accept the possibility to do so; (ii) bots that are vulnerable to content abuse by other users may *unintentionally* cause harm, while still being responsible for the content they endorse or spread; and (iii) bots that are vulnerable to trust abuse, if they leak data, may do so intentionally (e.g., it they sell data) or unintentionally (e.g., if intruders steel data). Regarding this last case, we did not find any hint for intentional leaks in our dataset.

Action	Pattern	Disclose sensitive facts	Denigrate	Be grossly offensive	Be indecent or obscene	Be threatening	Make false allegations	Deceive	Spam	Spread misinformation	Mimic interest	Clone profile	Invade space
Follow	Indiscriminate follow	—	△C	△C	△C	—	—	—	—	—	GO	—	—
	Whitelist-based follow	—	⊘	⊘	⊘	—	—	—	—	—	GO	—	—
	Blacklist-based follow	—	⊘	⊘	⊘	—	—	—	—	—	GO	—	—
	Phantom follow	—	△C	△C	△C	—	—	—	—	—	GO	—	—
Like	Indiscriminate like	—	△C	△C	△C	—	—	—	—	—	GO	—	—
	Whitelist-based like	—	⊘	⊘	⊘	—	—	—	—	—	GO	—	—
	Blacklist-based like	—	⊘	⊘	⊘	—	—	—	—	—	GO	—	—
	Mass like	—	△C	△C	△C	—	—	—	GO	—	GO	—	—
Tweet	Fixed-content tweet	⊘	⊘	⊘	⊘	⊘	⊘	—	GO	—	—	—	—
	AI-generated tweet	△C	△C	△C	△C	△C	△C	—	GO	△C	—	GO	—
	Trusted source tweet	⊘	⊘	⊘	⊘	⊘	⊘	—	GO	—	—	—	—
Mention	Indiscriminate mention	—	GO	GO	GO	—	—	—	GO	—	GO	—	—
	Opt-in mention	—	GO	GO	GO	—	—	—	⊘	—	—	—	—
	Targeted mention	—	△C	—	—	—	△C	—	—	—	—	—	—
	Whitelist-based mention	—	⊘	⊘	⊘	—	—	—	—	—	GO	—	—
	Blacklist-based mention	—	⊘	⊘	⊘	—	—	—	—	—	GO	—	—
Retweet	Indiscriminate retweet	—	△C	△C	△C	△C	△C	△C	—	△C	GO	—	—
	Whitelist-based retweet	—	⊘	⊘	⊘	⊘	⊘	⊘	—	⊘	GO	—	—
	Blacklist-based retweet	—	⊘	⊘	⊘	⊘	⊘	⊘	—	⊘	GO	—	—
	Mass retweet	—	△C	△C	△C	△C	△C	△C	GO	△C	GO	—	—
Talk to	Indiscriminate talk	—	—	—	—	—	—	—	GO	—	GO	—	—
	Fixed-content talk	⊘	⊘	⊘	⊘	⊘	⊘	—	GO	—	GO	—	—
	AI-generated talk	△C	△C	△C	△C	△C	△C	—	GO	△C	GO	GO	—
	Talk with opt-in	—	—	—	—	—	—	—	⊘	—	⊘	—	—
	Targeted talk	—	—	—	—	—	—	—	GO	—	GO	—	—
Pause	Mimic human	—	—	—	—	—	—	GO	—	—	—	—	—
	Satisfy API contraints	—	—	—	—	—	—	—	GO	—	—	—	—
Store	Store persistently	△T	—	—	—	—	—	—	—	—	—	△T	—

GO *Enables* ⊘ *Prevents* △C *Vulnerable to content abuse* △T *Vulnerable to trust abuse*

Fig. 3: Potential effects of actions and patterns on the users in online communications: patterns either enable, prevent or are vulnerable to abuses. For example, following an account with a denigrating or offending username may perpetuate and endorse the denigration or offense.

It is important to note that the analyzed dataset features only a few bots that implement patterns that aim to prevent abuses, which testifies a generally low awareness of the problem and commitment to mitigate risk by developers. Specifically, only 5 repositories implement blacklist-based patterns, 2 control if the user is verified by a whitelist (implementing multiple patterns), 6 use opt-in verification, 4 use a trusted source for tweets, and 8 use fixed content instead. Finally, we did not find any indication of effects of the identified patterns on the abuse *invade space* (it refers to bots invading spaces, e.g., online discussion groups or social networks, that are not meant for bot participation), as Twitter is generally open to bots.

5 Related Works

As already hinted at in the introduction, the topic of social bots has so far been approached mostly from the perspective of telling humans and bots apart, that is, with the intention of *detecting* bots. The work that is most closely related to this aspect is Botometer, formerly known as BotOrNot [7,8], an online tool that computes a bot-likelihood score for Twitter accounts and allows one to tell bots and genuine user accounts apart. The tool builds on more than 1000 features among network, user, friends, temporal, content and sentiment features, and uses a random forest classifier for each subset of features. The training data used is based on bot accounts collected in prior work by Lee et al. [11], who used Twitter honeypots to lure bots and collected about 36000 candidate bot accounts following or messaging their honeypot accounts.

Some works go further and turn their focus to *specific types* of social bots and, thereby, harms. For instance, Ratkiewicz et al. [13] studied the phenomenon of *astroturfing*, i.e., political campaigns that aim to fake social support from people for a cause, and showed that bots play a major role in astroturfing activities in Twitter. Cresci et al. [5] specifically focused on the problem of *fake followers*. They constructed a dataset of human accounts (manually and by invitation of friends) and bought fake followers from online services like http://fastfollowerz.com. The work compares two types of automatic classifiers, classifiers based on expert-defined rules and feature-based classifiers (machine learning), and shows (i) that fake followers can indeed be spotted and (ii) that black-box, feature-based classifiers perform better than white-box, rule-based classifiers. In addition, the work also produced a publicly available, labeled dataset that can be used for research purposes. Varol et al. [15] propose a bottom-up approach to the identification of bots with similar online behavior. The classifier used is the one adopted by Botometer, while the dataset used also included a manually annotated collection of Twitter accounts. After classifying accounts into bot or not, the authors further clustered the bot accounts into three types of bots: *spammers*, *self promoters*, and accounts that *post content from applications*. Chu et al. [4] coined the term *cyborg* to refer to bot-assisted humans in social networks and used a manually labeled dataset of 6000 randomly sampled Twitter accounts and a random forest classifier plus entropy measures to classify accounts into bots, cyborgs and humans.

In terms of *datasets* analyzed for bot detection, Beskow and Carley [1] propose four tiers of data for the classification of Twitter accounts: single tweet text (tier 0), account + one tweet (1), account + full timeline (2), and account + timeline + friends timelines (3). The assumption is that bot detection is achieved using feature-based classification or AI algorithms. In fact, with their tool bot-hunter, the authors study different machine learning techniques for tier-1 datasets. Differently from these classification-based approaches, Cao et al. [3] describe SybilRank, a tool for the detection of sybil accounts (bots) in social networks by analyzing the social graph (of Facebook, in the specific study). The study in this paper focuses on a different type of dataset, i.e., code, to understand the internals of bots, not their externally visible behavior or traces.

Little or no work has been done so far on the analysis of *harms* and *abuses*, as proposed in this paper. Perhaps the work by Varol et al. [15] can be seen as an ethical alarm: it estimates that between 9% and 15% of all accounts in Twitter are likely automated accounts and shows that bots are able to apply sophisticated communication tactics, distinguishing between humans and bots.

6 Conclusion

This paper proposes an original perspective on bots for online communication: instead of looking at messages or network activity, which is the typical practice in literature, it analyzes the code that produces them. To the best of our knowledge, this is the first study of its kind in this area. The study contributes to the state of the art in a threefold fashion: It extracts *31 patterns* that implement different variants of 9 communication actions from a dataset of 60 GitHub Twitter bot repositories (approximately 75-80 hours of manual code inspection). Then, it discusses the *effects* the patterns may have at runtime and provides a systematic mapping of patterns to potential abuses as a reference for developers. Finally, it proposes a technical interpretation of why abuses may happen by linking the *intentionality* of abuses to the nature of the patterns, distinguishing between intentional and unintentional patterns. These ethical aspects are particularly relevant to web engineering if we consider that many understand bots as the apps of tomorrow. As a possible usage scenario, social network providers that host third-party bot code (e.g., Facebook) could use these patterns to implement early warning systems to prevent harm.

The findings of this paper are empirical and stem from a careful, manual systematic code review. They are limited by nature. As for the *internal validity*, the study suffers of course from the limited size of the dataset; perhaps more repositories would have allowed us to identify more patterns. Also, the open-source nature of the projects may provide a limited view on the possible patterns, as developers of intentionally malicious bots may not share their code. The focus on Python was needed to keep the dataset manageable. The careful, randomized selection of repositories aimed to increase internal validity. As for the *external* *validity*, different programming languages and communication platforms may behave differently. However, the core of the actions and patterns proposed in this paper are similar in other platforms and programming languages. These may differ in platform-specific functionalities (e.g., poking a user in Facebook), but the abstractions of this paper make the actions and patterns portable.

As for the next steps, we are already working on the implementation of a suitable, formal language for action patterns and a respective pattern search engine for the automated retrieval of patterns from large numbers of code repositories based on the approach proposed in [12]. Expanding the horizon of the investigation beyond Twitter and Python is planned next.

References

1. Beskow, D.M., Carley, K.M.: Bot-hunter: A tiered approach to detecting & characterizing automated activity on twitter. In: SBP-BRiMS 2018 (2018)
2. Bessi, A., Ferrara, E.: Social bots distort the 2016 us presidential election online discussion. First Monday, **21**(11) (2016)
3. Cao, Q., Sirivianos, M., Yang, X., Pregueiro, T.: Aiding the detection of fake accounts in large scale social online services. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. pp. 15–15 (2012)
4. Chu, Z., Gianvecchio, S., Wang, H., Jajodia, S.: Detecting automation of twitter accounts: Are you a human, bot, or cyborg? IEEE Transactions on Dependable and Secure Computing **9**(6), 811–824 (2012)
5. Cresci, S., Di Pietro, R., Petrocchi, M., Spognardi, A., Tesconi, M.: Fame for sale: efficient detection of fake twitter followers. Decision Support Systems **80**, 56–71 (2015)
6. Daniel, F., Cappiello, C., Benatallah, B.: Bots Acting Like Humans: Understanding and Preventing Harm. IEEE Internet Computing, in print (2019), https://ieeexplore.ieee.org/document/8611348
7. Davis, C.A., Varol, O., Ferrara, E., Flammini, A., Menczer, F.: Botornot: A system to evaluate social bots. In: WWW 2016. pp. 273–274 (2016)
8. Ferrara, E., Varol, O., Davis, C., Menczer, F., Flammini, A.: The rise of social bots. Communications of the ACM **59**(7), 96–104 (2016)
9. Hey, T., Tansley, S., Tolle, K.M., et al.: The fourth paradigm: data-intensive scientific discovery, vol. 1. Microsoft research Redmond, WA (2009)
10. Kitchenham, B.: Procedures for performing systematic reviews. Keele, UK, Keele University **33**(2004), 1–26 (2004)
11. Lee, K., Eoff, B.D., Caverlee, J.: Seven months with the devils: A long-term study of content polluters on twitter. In: ICWSM. pp. 185–192 (2011)
12. Paul, S., Prakash, A.: A framework for source code search using program patterns. IEEE Transactions on Software Engineering **20**(6), 463–475 (1994)
13. Ratkiewicz, J., Conover, M., Meiss, M.R., Gonçalves, B., Flammini, A., Menczer, F.: Detecting and tracking political abuse in social media. In: ICWSM. pp. 297–304 (2011)
14. The Parliament of New Zealand: Harmful Digital Communications Act 2015. Public Act 2015 No 63 (2015), http://www.legislation.govt.nz/act/public/2015/0063/
15. Varol, O., Ferrara, E., Davis, C.A., Menczer, F., Flammini, A.: Online human-bot interactions: Detection, estimation, and characterization. arXiv preprint arXiv:1703.03107 (2017)