Sapienza
Università di Roma

From Scraping to Bot Detection: Navigating Mastodon and the Fediverse with APIs

Faculty of Information Engineering, Computer Science and Statistics
Bachelor's Degree in Computer Science

Nicolo Mariano Fragale
ID number 2017378

Advisor
Prof. Angelo Spognardi

Academic Year 2023/2024

From Scraping to Bot Detection: Navigating Mastodon and the Fediverse with APIs
Bachelor's Thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: ZioNicky@protonmail.com

«Non è il fisico, ma l'allenamento
Non è l'esame, ma l'argomento
Non è la cima, ma il sentiero»

Abstract

The Fediverse is emerging as a decentralized, non-commercial alternative to traditional social media, giving users more control and promoting community engagement. Built on open-source platforms and interconnected via the ActivityPub protocol, this architecture allows for transparency, user autonomy, and ethical governance among thousands of independent instances.

However, the growing presence of automated social bot accounts is affecting conversations and information dissemination.

In response, this paper presents Mastoanalyzer, a versatile tool designed to analyze user behavior and detect automation on Mastodon, one of the Fediverse's prominent platforms. This tool not only addresses the challenges posed by the increasing presence of bots but also helps in understanding human behavior and digital dynamics in decentralized contexts. It uses REST APIs, MySQL databases, and JSON formatting, supporting scalable data collection and long-term analysis. Its modular design allows for seamless adaptation to varied research needs and integration of machine learning for real-time bot detection.

What distinguishes Mastoanalyzer is not only its technical power but also its ethical commitment. It presents itself as an ally for anyone who wants to understand and contribute to the evolution of decentralized social networks, fostering greater awareness about the impact of automation and the quality of online interaction. By promoting transparent analysis, this tool is a valuable resource for better understanding the dynamics of digital communities while supporting responsible data management and respecting the autonomy of federated platforms such as Mastodon.

Contents

Chapter 1

The Fediverse

1.1 Decentralized Online Social Networks

Online social networks (OSN) are digital platforms that allow people to connect, interact and share content with other people or groups.

Over the years OSNs have evolved into a variety of forms with specific functions depending on the target and interests of users. Examples include:

- **Facebook and X**: Offer a wide range of content sharing, including text, photos, and videos, and facilitate communication between users.
- **Instagram, Pinterest, and TikTok**: Focus more on visual content.
- **LinkedIn**: Connects professionals, companies, and aspiring workers.
- **WhatsApp, Messenger, and Telegram**: Provide direct messaging services.

Over the past decade, OSNs have experienced an explosive growth, becoming a central part of our digital lives.

This surge in popularity, however, prompted platform owners to focus on new goals, particularly around content personalization and advertising strategies, aiming to maximize engagement and drive social marketing efforts.

As OSNs evolved, the strategies of personalization and monetization came under scrutiny, sparking debates on user autonomy, data privacy, and corporate control.

These issues fueled interest in developing new OSN paradigms, shifting from *"company-centric"* to *"user-centric"* models that emphasize user ownership, transparency, and control over personal data.

On the other hand, Decentralized Online Social Networks (DOSN) are social platforms that operate in a decentralised manner, avoiding a single centralized system that acts as an access point for all interactions, unlike traditional social networks such as Facebook or Instagram. DOSNs utilize a distributed network often based on technologies like blockchain or peer-to-peer, to ensure greater user autonomy and control over personal data. In this way, there is no central server with full control over data usage, preventing massive data leaks or speculation on user data.

The idea of decentralization in social networks has historical roots within the *Free/Libre and Open Source Software (FLOSS)* movements of the early 2000s, where developers and users envisioned platforms free from corporate oversight and driven by community interests.

DOSNs are built on two key principles:

1. The use of **open-source software**, which empowers individuals to create and manage their own servers, helping to prevent centralization.

2. The implementation of **communication protocols** that allow seamless interaction and data exchange between different servers.

Independent nodes help to keep the platform active and maintain control over their own data, which is not concentrated in a single place vulnerable to violations or abuses. In OSNs, moderation is centralized, leading to arbitrary censorship or restrictions on freedom of expression. In contrast, in DOSNs, moderation depends on the server itself, which can rely on decentralized communities or algorithms, making censorship selective but more challenging to limit inappropriate or harmful content. Given the autonomy of each server, users own their data and decide on its use, reducing the risks of privacy violations, massive hacking, and failures or blocking attempts.

However, DOSNs require some familiarity with decentralized technologies, making these platforms less accessible to less experienced users. Without centralized moderation, it may be difficult to manage inappropriate or illegal content, which poses a risk of attracting spam or malicious activity.

Early efforts to create open, federated platforms mainly attracted developers and privacy advocates, drawing only niche interest. However, the launch of **Mastodon** in 2016 marked a turning point, bringing decentralized networking into the mainstream and sparking the expansion of what we now call the *Fediverse*. In this federated model, servers are called *instances* and communicate with each other through a shared protocol. This setup allows users registered on one server to interact seamlessly with users on other servers, much like how email services operate across different providers. Each instance typically centers around a particular community or interest, whether it's based on shared practices, ideologies, or hobbies. This structure demonstrates that decentralized networks are not only technically feasible but also meet a genuine need for independent digital spaces where communities can establish their own guidelines and norms.

Together, these instances form a vast social network known as the Fediverse.

Thanks to this interconnected system, users can follow and interact with people across different DOSN platforms without needing separate accounts on each one, maximising interoperability and facilitating data transfer between different platforms.

1.2 A view into the Fediverse

Following recent crises at major social media platforms like Twitter and Reddit, the **Fediverse** has gained considerable attention as a compelling, non-commercial alternative. Individuals, communities, and organizations are increasingly turning to the Fediverse to create open channels for public communication, dialogue, and debate outside of the commercial social media ecosystem.

The Fediverse is a decentralized network composed of open-source, largely non-profit social media sites, apps, and services that interconnect using the ActivityPub protocol. This framework supports a diverse ecosystem of linked, independent platforms that foster a more user-centered, community-driven approach to social networking.

The Fediverse has rapidly expanded across nearly 5,000 *instances*, running various software such as Friendica, Funkwhale, Lemmy, Hubzilla, Misskey, PeerTube, PixelFed, and Pleroma. [1]

The popularity of the Fediverse has been driven by a mix of technical features and shifting user preferences. Technically, the Fediverse relies on open protocols and a transparent platform structure, free from corporate control. Socially, it aligns with a growing trend of users seeking more control and autonomy over their social media interactions, rather than being subject to the algorithms and decisions of large tech companies. This unique combination of user empowerment and open architecture makes the Fediverse an appealing alternative to traditional, centralized social media networks.

The Fediverse stands out from traditional online social networks (OSNs) through its strong emphasis on **user agency** and **community-driven governance**. This approach empowers users not just to participate but to shape their social spaces in alignment with their own values and community norms. As a result, the Fediverse is more than a technical network; it embodies a social movement dedicated to digital autonomy, ethical governance and freedom of expression. [2] Its flexible structure allows individuals and communities to build and manage independent instances, creating diverse spaces that support a range of interests and ideologies while promoting user control and transparency.

A key aspect of the Fediverse's functionality lies in its technical foundation, which allows **interoperability** between various platforms. Many Fediverse services, such as Mastodon and PixelFed, operate using the **ActivityPub protocol**, a standardized protocol designed to facilitate communication between different social media platforms.

The Fediverse is also a social and political structure where users and admin-

[1] It's difficult to measure the size of the fediverse because it is noncentralized. Several projects track it: https://fe-didb.org/, https://fediverse.party/en/fediverse/, https://the-federation.info/, and https://mastodon.social/@mastodon-usercount. Each varies, because each observes different sets of servers. All reflect explosive growth in accounts during 2022-23. Since then, users have dropped but remain far above historical levels. In October 2021, the fediverse had 4.1 million accounts [1]. Today it's 14 million.

[2] Art. 21, Italian Constitution: Everyone has the right to freedom of expression. This right includes freedom of opinion and the freedom to receive or communicate information or ideas without interference by public authorities and without frontier restrictions.

istrators make conscious decisions about whom to interact with or block. This approach allows communities to **self-regulate** based on their standards and values. For example, when Gab.com, an ActivityPub-based platform known for hosting alt-right content, joined the network, many Fediverse communities chose to block it, reflecting a commitment to human centric moderation, prioritizing community values and user safety [2].

Gab.com is a social networking platform launched in 2016 with the stated goal of promoting freedom of expression but has been controversial from the start for its community content, which included extremist discussions, hate speech, and movements linked to the radical right. In 2019, it decided to adopt the Mastodon software, avoiding centralized censorship, and remained online autonomously. This allowed it to maintain its features of unmoderated content and attract many users excluded from other social networks due to extremist or hate speech. However, most instances of the Fediverse chose to "de-federate" it by preventing interaction with users in other instances and those in Gab. This decision was motivated by the desire to prevent extremist or hate content from spreading in the decentralized network while maintaining the principle of decentralization. This case highlighted the difficulty of balancing freedom of expression with the need to protect social platforms from harmful or extremist content and illustrated how each instance is free to decide with which other instances to federate or defederate.

Thanks to a set of social and practical choices made by early Mastodon developers and admins, thousands of connected Fediverse instances now use very similar codes, prohibiting actions such as hate speech and harassment and encouraging local, human moderation and collective decision-making.

Beyond social and ethical considerations, the Fediverse also presents unique **technical benefits** that appeal to a diverse range of users, including privacy-conscious individuals, organizations, and journalists. Unlike traditional OSNs, which often deprioritize external links or limit discoverability of content that leads users away from the platform, the Fediverse's interoperability supports seamless cross-platform interaction. This openness enables journalists, researchers, and content creators to directly reach and connect with their audiences without being restricted by algorithms that might obscure or deprioritize their work. As a result, the Fediverse fosters a transparent and accessible environment where information can circulate freely and users can interact across different networks, enhancing both the visibility and engagement of shared content.

However, there are no guarantees that this covenantal, federalized network of small, locally-operated social servers running on an open protocol will remain viable, democratically controlled, or ethical. The Fediverse has three key modalities of governance, each with its own unique practices and needs:

- The W3C standard ActivityPub protocol, which maintains and extends that standard.
- The maintenance and development of Fediverse software packages, such as Mastodon, Pixelfed, and Lemmy.
- The governance of each specific Fediverse instance, which involves internal

content moderation as well as diplomatic relations between instances.

Since each instance is free to set its own policies and moderation, this last point is a crucial aspect for ensuring its functionality and security. Each instance is managed by one or more administrators and moderators who establish and enforce moderation rules by blocking, silencing, or warning those who violate them. These are mentioned in the instance policy, which clearly defines what content is allowed and what is not. Some instances are highly permissive, while others have strict regulations against hate speech, pornography or extremist political content. Therefore, when users choose to join instances that reflect their views and preferences, they are morally obliged to accept and abide by this set of rules.

Despite its promise, the Fediverse faces several notable challenges. As a decentralized ecosystem, it relies on substantial **collaboration** among instances, especially when it comes to **content moderation**. Since each instance can set its own guidelines and rules, maintaining consistent moderation across the network can be complex and resource-intensive. Another key challenge is **discoverability**. Without algorithms designed to promote content—as seen in traditional OSNs—users may find it difficult to access a broad range of posts, trends, or communities, which can limit the Fediverse's reach and visibility for new or diverse voices. This absence of algorithmic promotion represents both a benefit, in terms of user agency, and a limitation, in terms of content discovery.

1.2.1 Privacy and Surveillance

The Fediverse has evolved from treating privacy and surveillance as purely technical challenges—focused on encryption and secure messaging, to viewing them as social issues as well. In the early days, tech communities were primarily concerned with protecting privacy through robust security measures aimed at preventing government or corporate surveillance. However, with the rise of the Fediverse, the concept of privacy has expanded. It now includes managing personal boundaries between users, controlling who can see one's posts, and using tools like content warnings and moderation to shape interactions. This shift is partly driven by the needs of marginalized communities, particularly queer users, who face privacy concerns that go beyond government or corporate oversight. For example, these users often need to navigate personal relationships or explore their identities in ways that ensure their safety. As a result, while the Fediverse still incorporates technical measures for privacy, it increasingly emphasizes creating a space where communities can establish their own rules and feel secure. For example, many instances only collect the information necessary for the platform to function, such as data for profile registration (username, email). Others explicitly state that they do not share user data with third parties for marketing, advertising, or analytics purposes. Users also have the right to request the deletion of their data at any time, including published content, and the ability to permanently delete their account and associated data. Some instances adopt a limited log policy, where data is deleted after a certain period. However, the deletion of federated content in other instances cannot always be guaranteed.

With this in mind, the Fediverse offers an alternative to corporate social media platforms, which often operate under the model of *surveillance capitalism* collecting and selling user data and interactions. These platforms may give users the illusion of control through privacy settings, they continue to gather data in ways that are hard to regulate. In contrast, the Fediverse seeks to break free from this model by prioritizing transparency in how data is handled and giving users a more active role in the process. When users join an *instance*, they are typically informed about how the federation works and how data flows across the network—something that is rare in corporate social media environments.

However, the Fediverse is not without its privacy risks. Being built on an open and public network means that it can be easier for third parties to collect data from it. This could pose a threat to groups such as activists or politically exposed individuals, who place trust in server administrators to enforce respect and privacy. However, this may not be sufficient for more sensitive matters, such as the privacy of direct messages, which could benefit from stronger encryption to decrease risks due to the visibility of their data.

In essence, the Fediverse is experimenting with treating privacy as a social issue and, while this approach is still evolving, the ongoing conversation seeks to find a balance between social solutions and technical safeguards.

1.2.2 Sustainability and Funding

The Fediverse challenges the common misconception that social media is "free." Unlike large corporate platforms, which profit from user data and interactions without compensating the users themselves, maintaining servers and developing software in the Fediverse comes with real costs. While some projects within the Fediverse are funded through donations and voluntary contributions, this funding model remains fragile. Larger projects, like Mastodon, have begun to pay some contributors, but more consistent and stable public support is needed to ensure the long-term sustainability of non-commercial platforms.

At its core, the Fediverse is a collection of alternative social media projects that, despite their differences, share common values around user freedom, self-management, and privacy. Unlike corporate platforms where user data is monetized and users have limited control over how their information is used, the Fediverse emphasizes active user participation. Users are encouraged to contribute to both the development of the code and the governance of their own communities.

The platforms within the Fediverse support a wide range of usage patterns, from privacy protections for queer communities to political subversion initiatives, all adaptable to the unique needs of each group. These models trace their roots back to early free software communities, where the first users were also developers who actively shaped the software. However, with the rise of the tech industry, the role of the user has shifted. Web 2.0 introduced interactivity but also gave rise to surveillance capitalism, where users became passive participants with limited control over their own data. In contrast, the Fediverse aims to empower users by not just allowing them to engage but also to contribute and, in some cases, manage entire network nodes.

In summary, the Fediverse represents an experimental space for exploring new models of user engagement, promoting an approach that balances active participation with fair compensation. It offers an alternative to both the exploitative business models of data-driven platforms and the unpaid, sometimes oppressive models of free but labor-intensive work. This evolution is influencing both the alternative social media landscape and the Free/Libre Open Source Software (FLOSS) culture, setting the stage for a new way of thinking about user governance and involvement.

1.3 Mastodon

Mastodon was created in 2016 by Eugen Rochko, a German programmer, with the aim of providing a decentralized alternative to existing large social media platforms such as Twitter and Facebook. It is mainly designed to address the problems associated with centralized social media, such as control over data and content moderation, censorship of content by central entities, and freedom of expression by exploiting the possibility of creating independent bodies.

Mastodon is a decentralized, open-source, and federated social media platform that allows users to interact, publish content, and create a social network without depending on a single central entity. Similar to Twitter, it allows users to post short messages (up to 500 characters) called *toots*. The functionality analogous to a retweet is referred to as a *boost*. Users can also share images, videos and links, and interact with other users through comments, likes and shares. Additionally, it emphasizes *niche communities* and robust *content moderation* similar to Reddit's sub-communities, but with each instance maintaining autonomy over its policies and moderation strategies.

Furthermore, users can mark certain toots with *content warnings*, accompanied by a *spoiler* text summarizing obfuscated content, enabling viewers to decide whether to access potentially disturbing content.

Mastodon is a network of independent servers known as *instances* that can vary widely in size, from large networks with millions of accounts to smaller communities, even hosting just a single organization. These istances can federate or be linked, allowing users to interact through theme without being bound to a single server creating a global and distributed network. Instance administrators can optionally close registration for their instance, creating a private community or enhancing moderation efficiency. They have control over content policies, setting specific rules regarding permitted and prohibited content. Importantly, closed registrations do not restrict user interactions, which remain possible across the entire network due to the interoperability afforded by the ActivityPub protocol.

This social enjoys all the privileges of DOSNs described in the previous section: privacy and data control, no advertising, flexibility in moderation and customizable communities. However, it also has flaws: a learning curve, complex moderation, and incompatibility between instances.

A defining feature of Mastodon is its use of the *ActivityPub protocol*[3], which supports both client-to-server and server-to-server communications. Through this protocol and a *subscription-based mechanism*, users can seamlessly interact with

[3]`https://www.w3.org/TR/activitypub/`

others, even across different instances. Mastodon's unique followship system also structures its timelines as follows:

- **Home Timeline**: Displays toots from followed users.
- **Local Timeline**: Shows toots created within the user's instance.
- **Federated Timeline**: Contains public toots from all users known to the instance, regardless of the instance origin.

Research on recent open-source DOSNs indicates that Mastodon has been uniquely impactful within the *Fediverse*, the collection of federated social media platforms. Notably, studies such as [3, 4, 5, 6, 7] have examined various aspects of Mastodon. In particular, [7] conducted a qualitative analysis through interviews with instance moderators, highlighting how Mastodon's federative structure enables diverse content, community autonomy, and horizontal growth across instances rather than vertical growth within them.

From a *network science* perspective, [5, 8] conducted foundational analyses of the Mastodon user network, exploring characteristics such as *degree distribution*, *triadic closure*, and *assortativity*, comparing these metrics to those of Twitter. Their analysis found that Mastodon exhibits a more balanced relationship between followers and followees, with 95% of users showing a follower-followee difference within the range of $(-250, 250)$. Additionally, they observed a lower presence of social bots on Mastodon (around 5%) compared to Twitter (15%) as reported by [9].

In terms of clustering coefficient, Mastodon's values fall between those observed on Facebook and Twitter. Degree as while *degree assortativity* analysis—considering metrics like Source In-Degree (SID), Source Out-Degree (SOD), Destination In-Degree (DID), and Destination Out-Degree (DOD)—revealed unique patterns. A lack of correlation between (SOD, DOD), (SOD, DID), and (SID, DOD) suggests that users with high follow counts connect to users with varying popularity, both in terms of who they follow and who follows them. Notably, the negative correlation (-0.1) between SID and DID indicates that popular users tend to follow less popular accounts.

Furthermore, [8] explored the impact of decentralization on user relationships, observing that each instance exhibits unique structural characteristics (e.g., degree distribution and clustering coefficient) which influence user interactions within the instance.

1.4 ActivityPub Protocol

The **ActivityPub protocol** is a decentralized social networking protocol built on the *ActivityStreams 2.0* data format. It offers both a **client-to-server API**, which allows for creating, updating, and deleting content, and a **server-to-server API**, which facilitates the delivery of notifications and content across different servers. This design allows for interoperability between services on different servers, similar to email protocols where users can communicate seamlessly across different providers.

In ActivityPub, users on one server can reach users on other servers throughout the network. For example, if *user1* is on *serverA* running Mastodon (a microblogging platform) and *user2* is on *serverB* running Pixelfed (a photo-sharing platform), each can follow the other to view, like, and comment on posts. This interaction is akin to how open-source email protocols enable users to send messages across different services (e.g., Gmail and Hotmail), creating a non-centralized technical infrastructure distinct from the traditional, centralized social media model [10].

1.4.1 Protocol Structure

The ActivityPub protocol operates on two distinct layers:

- **Server-to-Server Federation Protocol**: This protocol enables decentralized websites to exchange information, facilitating the distribution of activities among users on different servers. This interaction helps to create a unified and interconnected social graph, allowing users on separate instances to communicate and share content seamlessly.
- **Client-to-Server Protocol**: This protocol allows a client, such as a mobile application or web interface, to act on behalf of the user. It provides the functionality to interact with the user's social feed and perform activities like posting, liking, following, and more, ensuring that users can fully engage with the social network regardless of the platform they use.

1.4.2 Actors and Messaging Mechanisms

In ActivityPub, a user is represented by an **actor**, an account on a server that interacts within the network. Each actor has the following components:

- **Inbox**: Receives messages from other actors.
- **Outbox**: Sends messages to others [11].

For example, if *Alyssa* wants to communicate with her friends, the *inbox* and *outbox* mechanisms facilitate these interactions:

- **POST to an inbox**: Sends a message to another actor's inbox (server-to-server, federated).
- **GET from an inbox**: Allows a user to retrieve their latest messages (client-to-server).
- **POST to an outbox**: Sends messages to the broader network (client-to-server).
- **GET from an outbox**: Allows viewing messages an actor has posted (client-to-server or server-to-server) [11].

To ensure efficiency, servers typically distribute messages to the inboxes of other servers' actors, creating a robust federation mechanism [11].

1.4.3 Objects and Activities

In ActivityPub, central elements include **objects**, which represent various types of content or actions (e.g., posts, comments, or likes). These objects can be "wrapped" in **activities** and are often organized into **collections** (or streams of objects), which themselves are subclasses of objects. To maintain security, servers receiving content should validate these objects to prevent spoofing attacks by verifying digital signatures and ensuring the object originates from the declared server.

Each object is assigned a **unique identifier** (typically an HTTPS URI), making it accessible across the network. For example:

- **Public Objects**: Retrieved via HTTP GET on their unique URI.
- **Private Objects**: May return error codes (e.g., 403 or 404) to restrict access.

If an object is created through server-to-server communications, the ID is mandatory; in client-to-server interactions, the server assigns an ID if one is not specified.

1.4.4 Actor Types and Flexibility

ActivityPub supports various actor types as defined in ActivityStreams:

- **Person**: Represents a human user, often associated with a personal profile.
- **Group**: Represents a collective of users.
- **Organization**: Represents an organizational entity.
- **Service**: Represents an automated service or platform.

Additionally, ActivityPub's flexibility permits actors to be custom object types, such as profiles or extended ActivityStreams types, allowing complex interactions across the network:

- **Multiple Profiles**: A single user may have multiple actors (e.g., different accounts for personal and professional use).
- **Bots and Automated Processes**: Actors can represent bots that interact automatically.

This flexibility allows ActivityPub to support not only individual users but also more complex systems such as communities, organizations, and automated software. Through ActivityPub, actors can perform actions like posting, liking, following, and commenting, and communicate both within and across servers, making decentralized interactions possible [11].

Chapter 2

Social Bots Detection

There is mounting evidence that more and more social media content is being created by automated entities known as **social bots** or **sybil accounts**. Recent studies have highlighted cases where these social bots mimic human behavior to influence conversations, manipulate user popularity, flood platforms with irrelevant content, spread misinformation, and even engage in propaganda and recruitment for terrorist activities.

Social bots (short for *software robots*) or automated software, operate via algorithms ranging from simple scripts to complex AIs, engaging in tasks or human-like conversations online [12]. A notable example of bots is the **chatbot**, an algorithm specifically designed to engage in conversation with a human, an idea originally proposed by *Alan Turing* in the 1950s [13]. The goal of creating a computer algorithm capable of passing the *Turing test* has fueled artificial intelligence research for decades. This ambition is evident in initiatives like the **Loebner Prize**, which awards advancements in natural language processing [14].

Benign social bots help automate content sharing, news aggregation, customer service, and support, thus aiding businesses in managing their social media presence and enhancing user engagement [15, 16, 17]. The sophistication level of bots varies widely. Some social bots are relatively simple, merely aggregating information from news sources, weather updates, or blog posts and reposting this content on social networks. Others, however, can be highly sophisticated, capable of infiltrating human conversations. These capabilities have both pros and cons for users of *Online Social Networks (OSNs)* and can be employed for positive or negative purposes.

On the positive side, bots can be designed with good intentions. For example, they can be used to protect user anonymity or to automate and perform tasks much faster than humans could manage manually. However, one significant malicious capability of social bots is their potential to rapidly disseminate misinformation, manipulate public sentiment, promote online conspiracy theories, and expose private information, such as phone numbers and addresses. This can lead to a potential erosion of trust in social media platforms. Another concern is **astroturfing**, where a company group organises an apparent popular support campaign. It is a protest, but in reality, it is all orchestrated by one party to promote its own objective by exploiting public perception to make it appear that there is broad consensus on a certain cause, when in fact this consensus is artificial and created ad hoc.

The main question is: How can we effectively detect malicious activities on Online Social Networks (OSNs)? Broadly, detection techniques can be classified into three categories:

1. **Bot detection systems based on social network topology**: These systems analyze the structure-based information of the network.
2. **Crowdsourcing Systems**: These systems utilize crowdsourcing to analyze user posts and profiles.
3. **Feature-Based Systems**: These systems leverage machine learning methods.

The sophistication of social bots has progressed from early rule-based, keyword-driven interactions to the incorporation of machine learning and natural language processing, allowing for more realistic human-like exchanges [18, 19]. For example, some bots employ **Markov chain-based models** [1] to generate text, mimicking human-produced content [20, 21].

2.1 Evolution of Social Bot Detection Techniques

To effectively detect social bots, we need a comprehensive approach that accounts for the wide range of tactics and behaviors these bots employ is necessary. Since bots can be designed to mimic human interactions, spread misinformation, or impersonate real users, it is essential to use a variety of detection methods. By combining different techniques, we can improve accuracy and reliability, making it easier to spot bots that might otherwise go unnoticed.

In this section, we'll take a closer look at several detection methods and how integrating them can lead to better results, while also referencing key research that has shaped these approaches.

Heuristic Model Based

In the early stages of social bot detection, researchers primarily relied on simple heuristics and rule-based systems to identify bots. These techniques focused on exploiting easily observable patterns and features indicative of bot-like behavior. Commonly used heuristics included:

1. **Account Activity:** Bots often exhibit high-frequency messaging and overall activity levels compared to human users. By analyzing metrics like the number of posts, researchers could identify patterns typical of bots.
2. **Account Metadata:** Certain profile metadata, such as account creation date, number of followers, and the following-to-follower ratio, can also signal bot behavior.

[1] This principle is known as the Markov property or zero memory. A Markov chain model is a type of mathematical model used to describe systems that evolve over time, where the next state of the system depends only on the current state, not how you got to that state.

3. **Content-based Features:** Bots often generate content that is repetitive, contains specific keywords, or is derived from a limited set of sources. By analyzing the diversity of content, the presence of certain keywords, and the distribution of sources, researchers could identify potential bots [22, 23].

4. **Network-based Features:** Bots often exhibit distinct network patterns, such as forming tightly connected groups or having few reciprocal relationships. By analyzing the structure of the follower and friend networks, researchers could detect potential bot accounts [24].

Although these early detection methods laid the groundwork for identifying social bots, they struggled to keep pace with the evolving sophistication of bot behavior. More advanced bots could easily circumvent these heuristic-based methods by mimicking human behavior, adjusting their activity patterns, or generating diverse content [19]. Moreover, the reliance on manually crafted rules and features made these methods prone to both false positives and false negatives. Legitimate users could exhibit bot-like behavior, and bots could mimic human actions.

This limitation led to the development of more advanced techniques that incorporated **machine learning** and **natural language processing** to improve detection accuracy and adaptability [25, 12].

Today, many social bots are capable of automatically generating responses through advanced *natural language processing algorithms*. These bots can create contextually relevant replies and often include references to media, such as images or videos, as well as links to external resources. This makes them highly effective at blending into online discussions, often making it difficult for users to distinguish between automated and human interactions. In addition to these communication capabilities, other bots focus on more malicious activities, such as tampering with the identities of legitimate individuals. Some of these bots act as **identity thieves**, adopting slight variants of real usernames to trick others into believing they are legitimate users. These bots may then steal personal information, including photos, links, and even private messages, further compromising the security of individuals on social platforms. Furthermore, more sophisticated bots are being developed with the ability to *clone* the behavior of legitimate users. These bots can interact with a user's friends, post topical and contextually coherent content, and replicate the temporal patterns of real users. By doing so, they create the illusion of authenticity, making it even harder for users and automated systems to detect them. In some cases, these bots can establish a presence on social media that is nearly indistinguishable from that of a real person, contributing to the growing complexity of bot detection and cybersecurity challenges on social networks.

2.1.1 Multimodal Based

Multimodal approaches, which combine different types of data such as **text**, **images**, and **network features**, offer a more complete and reliable way to understand user behavior, making it easier to detect social bots. By using a variety of information sources, these methods can pick up on complex and subtle patterns in the data that might be missed by approaches that rely on just one type of data.

For instance, analyzing **text** can reveal language patterns typical of bots, while looking at **images** might help spot fake or reused photos, and studying **network connections** can highlight unusual relationship patterns, like one-sided connections or rapid follower growth.

Recently, there has been growing interest in using multimodal data for social bot detection. This approach allows for a more well-rounded view of user behavior across different aspects, making it easier to identify bots that blend in by mimicking human activity. Combining data from multiple sources can uncover bot behavior that would be harder to catch with just one method. As bots continue to evolve and become more sophisticated, multimodal techniques are seen as a key tool for improving detection accuracy and keeping up with these challenges.

One promising direction is the integration of **text** and **image analysis**. For example, [26] proposed a multimodal approach to detecting fake news by jointly modeling textual and visual information. Their approach used a hierarchical attention network to capture the dependencies between textual and visual features, enabling the model to identify bots that disseminate misleading information through both text and visual content. Similarly, [27] proposed a model that combined textual and visual cues to detect social bots on Twitter, demonstrating the effectiveness of multimodal approaches in capturing complementary information.

Another direction for multimodal approaches is the incorporation of **network features**. Social bot detection can benefit from analyzing user interactions and network structures, which can reveal suspicious interaction patterns or network structures that may be indicative of coordinated bot activity [28, 29, 30]. [30] demonstrated the value of temporal features by proposing a method that combined them with content-based features to detect coordinated influence campaigns on social media platforms.

Multimodal approaches can also be extended to other data types, such as **audio** or **video**, to further improve social bot detection capabilities. For example, audio or video analysis can be used to identify bots that generate or distribute *deepfake* content, which poses a significant threat to online platforms [31].

By combining different types of data, such as **text**, **images**, and **network features**, detection methods for social bots become more accurate and robust, helping them better keep up with the constantly evolving nature of bot behavior. Since social bots are becoming more sophisticated, it's important that detection methods are flexible and capable of adapting to these changes.

Multimodal approaches, those that pull in a variety of data sources, are especially useful because they allow researchers to see a broader picture of user activity. This way, they can catch more subtle and complex patterns that might go unnoticed in simpler, single-data approaches. To put it simply, multimodal methods hold a lot of potential for improving how we detect social bots. Examining at different layers of data, researchers can identify on clues that might otherwise be overlooked. Whether it's the **text** people write, the **images** they share, or their interactions with others in their network, each piece of information adds another layer of insight.

When all these pieces are combined, they create a more comprehensive and accurate picture of online activity, which helps in reliably identifying bots. This is crucial for preventing the negative effects that bots can have on online platforms, such as spreading misinformation or undermining user trust.

In the end, multimodal approaches represent a crucial step forward in developing more adaptable, comprehensive, and effective detection systems to address the challenges posed by social bots.

2.1.2 NLP Model Based

One of the most widely used strategies in social bot detection is the combination of machine learning and natural language processing (NLP). Machine learning algorithms excel at identifying patterns in large datasets, helping to pinpoint behaviors typical of bots. At the same time, NLP analyzes the language used by bots, allowing for the detection of unnatural or repetitive phrasing indicative of automation.

When combined, these techniques provide a powerful toolkit for recognizing bots based on both their actions and their communication styles.

By incorporating both supervised and unsupervised learning algorithms, as well as natural language processing methods, researchers can analyze content generated by social bots and identify patterns and features that differentiate them from human users [12]. For instance, [32] proposed a deep learning-based method that combined convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to analyze content and account features for bot detection, achieving high accuracy and recall.

Network analysis is another crucial technique for detecting social bots. It involves studying social network structures and the interaction patterns of bot accounts. By examining how bots connect with other users and behave within a network, researchers can identify unusual patterns that are typical of automated accounts. For instance, bots might form tightly connected groups, engage in one-way relationships, or exhibit rapid and unnatural follower growth. This type of analysis provides valuable insights into bot behavior and is often used in conjunction with other detection methods for more accurate results. By investigating these patterns, researchers can identify anomalous behavior and uncover coordinated activities, such as astroturfing campaigns and disinformation efforts [22].

[33] introduced a cross-platform framework called FakeNewsNet to study the diffusion of fake news and the role of social bots in its spread. By collecting a large dataset of news articles labeled as fake or real and analyzing the activity of social bots across platforms, they identified patterns of fake news dissemination and the characteristics of bots involved in spreading it.

In summary, combining a range of detection techniques such as machine learning, natural language processing, network analysis, temporal analysis, and cross-platform analysis can significantly improve the accuracy and reliability of social bot detection. Each of these methods brings a unique strengths, allowing for a more comprehensive approach. Machine learning and NLP help analyze content and language patterns, network analysis uncovers unusual interaction structures, temporal analysis tracks activity over time, and cross-platform analysis identifies bots that operate across different social media platforms. Together, these methods create a more robust system capable of detecting even the most sophisticated social bots.

2.1.3 Crowdsourcing Model Based

[34] explored the idea of human involvement in social bot detection, suggesting that crowdsourcing the task to large groups of workers could be an effective solution. As a proof-of-concept, they developed an Online Social Turing Test platform, based on the assumption that humans are still better at detecting bots than machines. Humans can pick up on subtle conversational cues, like sarcasm or persuasive language, and recognize emerging patterns or anomalies in social interactions skills that machines have yet to fully replicate.

To test their approach, the authors used data from Facebook and Renren (a popular Chinese social network) and had both expert annotators and online workers attempt to detect social bot accounts based solely on the profile informations. The results showed that while the detection accuracy of the hired workers declined over time, it remained high enough to be used in a majority voting system: each profile was shown to multiple workers, and the majority opinion determined the final verdict. This strategy resulted in a very low false positive rate, which is particularly important for service providers looking to minimize errors.

Despite the promising results, bot behavior has become increasingly sophisticated. Bots can now build realistic social networks and create content that mimics human activity, including natural temporal patterns in their posts.

As detection systems improve, we can expect an ongoing "arms race" similar to what we saw with spam in the past [35]. One key challenge, however, is the need for a large volume of labeled data for training machine learning models. In such a dynamic environment, techniques like active learning—where the system can adapt and learn from new data on the fly—might be helpful in addressing the continuously evolving nature of social bots.

2.1.4 The Importance of Updating

To effectively address the constantly changing landscape of social bots, it's crucial to take a proactive approach when developing and deploying detection models. This involves continuously updating and refining these models to adapt to the evolving tactics and strategies used by bot operators. For example, [36] demonstrated the importance of using diverse and recent datasets for training and evaluating social bot detection models. They conducted a systematic comparison of multiple datasets and found that the performance of models trained on older datasets was significantly lower than those trained on more recent and diverse data.

Adapting to changes in social media platforms is another key aspect of maintaining effective detection models. These platforms are constantly evolving, with updates to APIs, user interfaces, and new features that can impact how bots behave or how detection systems interact with the platform. As a result, it's essential to regularly adjust detection models to account for these changes and the new patterns of bot behavior they may introduce. [37] examined the impact of Twitter API rate limits on social bot detection and found that the imposed restrictions reduced the accuracy of certain detection features, highlighting the need to adapt models to accommodate platform changes.

Fine-tuning model parameters is a critical step in ensuring the ongoing effectiveness of detection models. Regularly assessing how well these models perform and adjusting their parameters to improve accuracy, recall, and other relevant metrics can lead to more reliable results. This process might include experimenting with different feature sets, algorithms, or parameter values, as well as conducting cross-validation to test the robustness of the models under various conditions.

Incorporating feedback loops is another essential practice for maintaining and enhancing detection models. By setting up mechanisms to collect user feedback on the accuracy of bot detection, models can be continuously refined and improved. This could involve crowdsourcing, expert validation, or other forms of user engagement to gather valuable insights and evaluate the real-world performance of the detection methods. Such feedback helps ensure that detection systems evolve alongside the tactics used by bots, leading to more effective mitigation of bot-related issues. For instance, [38] developed a system called Botometer, which combined machine learning-based bot detection with user feedback to improve its performance. The system allowed users to report false positives and false negatives, and the input was used to fine-tune the underlying detection models.

In conclusion, regularly updating and fine-tuning social bot detection models is essential to stay ahead of the ever-evolving landscape of social bots. By implementing strategies like updating training data, adapting to platform changes, fine-tuning model parameters, and incorporating feedback loops, researchers and practitioners can create more effective and robust detection systems. These efforts will ultimately help safeguard online information ecosystems and improve the detection and mitigation of social bot activities.

2.2 Other research projects

2.2.1 TwiBot20

TwiBot20 is an innovative dataset created for Twitter bot detection, incorporating semantic, proprietary, and neighborhood information about users to leverage the diversity of the Twittersphere. The dataset includes information such as recent tweets, user characteristics (e.g., number of followers), and the follow relationships between users. Users were selected using a Breadth-First Search (BFS) to generate a directed graph, with 40 seed users from various sectors, including politics, business, entertainment, and sports, chosen to represent a wide range of interests. Semantic information (recent tweets), ownership information (numerical and categorical user data), and neighborhood information (follow relationships) were collected directly from the Twitter API.

To label users as bots or humans, a crowdsourcing process was employed. Experienced annotators on Twitter assessed users based on specific criteria of automated behavior. Through data analysis, TwiBot20 revealed interesting differences between bots and human users. For example, human users tend to follow reputable or "famous" users, while bots generally generate fewer tweets than genuine users. TwiBot20 also highlighted unique patterns in bots, such as the use of random strings in usernames, further distinguishing them from human accounts.

2.2.2 TwiBot22

TwiBot-22 represents a significant advancement in Twitter bot detection, distinguished by its size, heterogeneous graph structure, and robust evaluation framework. As the largest dataset of its kind, TwiBot-22 is five times larger than previous benchmarks and utilizes heterogeneous graphs to represent Twitter's complex network. This broader, more comprehensive structure enhances the dataset's ability to accurately capture anomalous bot behavior.

TwiBot-22 includes four types of entities (e.g., users, tweets, hashtags) and fourteen types of relationships (e.g., follow, retweet). The dataset contains over 92 million nodes and more than 170 million edges, making it the largest available benchmark for bot detection on Twitter. A Breadth-First Search (BFS) strategy was used for data collection, starting with “seed users” and expanding through follow relationships. In addition to the follow network, other entities and relationships, such as tweets, lists, and hashtags, were also collected, resulting in a heterogeneous graph structure.

For data annotation, TwiBot-22 relied on 1,000 bot detection experts rather than crowdsourcing, as was done in TwiBot-20. Probabilistic labels generated with the Snorkel system were used to train an MLP classifier, achieving an accuracy of 90.5% on test annotations. This approach improved label quality and consistency, offering a higher standard of reliability than prior methods.

2.2.3 Fox8

This study analyzes a Twitter social botnet called "fox8," consisting of 1,140 accounts, some of which are partially controlled by automation, interacting primarily through retweets generated by advanced language models (LLM) such as ChatGPT. These bots have been used to promote suspicious websites and disseminate malicious content.

The fox8 botnet was discovered by analyzing tweets containing the phrase "as an AI language model," which revealed automated behavior patterns. Researchers collected 12,226 tweets from 9,112 unique accounts between October 1, 2022, and April 23, 2023. After a manual review, 76% of these accounts were classified as humans interacting with ChatGPT-generated content, while the remaining accounts were bots using LLM. During the annotation, researchers identified patterns connecting some of these bots to three suspicious websites: fox8.news (an imitation site), cryptnomics.org, and globaleconomics.news. This led to the identification of a botnet of 1,140 accounts likely using ChatGPT for content generation.

Fox8 bot accounts have an average of 74.0 followers (SD=36.7), 140.4 friends (SD=236.6), and 149.6 tweets (SD=178.8). These bots interact in three main ways: following, retweeting, and replying. The follower network is dense, with an average in-degree of 13.7 and out-degree of 13.4, while response and retweet networks are sparser, with much lower in-degree and out-degree.

Further analysis compared the response behavior of fox8 bots with 7,972 human accounts not associated with the botnet. Results showed that fox8 bots have a response probability of 0.2% among each other, compared to 0.016% in the control group, indicating stronger in-group interactions among bots.

Detection

The fox8 botnet was detected using the following approach:

1. The "fox8-23" dataset was used, containing tweets from both bot and human accounts.

2. Tweets from each account were scored. A high score suggests the account may be a bot.

3. The standard deviation of tweet scores was analyzed. A low deviation indicates that the account may be a bot, as bots tend to generate consistent content.

4. A test was conducted on a random sample of tweets posted between May 8 and May 14, 2023.

The test revealed numerous false positives, especially among accounts with few short tweets. For instance, phrases like "thank you" and "amen" were mistakenly classified as generated by LLMs.

High Score: A high score indicates a high probability that the text was generated by an LLM, based on features such as sophisticated language, thematic coherence, and typical LLM syntax.

Standard Deviation: This measures the variation in tweet content from an account. Bots tend to generate uniform content, resulting in a lower standard deviation, whereas human users produce more diverse content.

2.3 Case Studies: Social Bot Detection in Real-World Applications

In this section, we explore a series of case studies that highlight the real-world applications of social bot detection techniques in various domains. These examples highlight the critical role of social bot detection in addressing significant challenges such as election interference, political manipulation, disinformation campaigns, fake news, and financial scams, including cryptocurrency manipulation. By examining these case studies, we aim to provide insight into the practical implications of social bot detection and emphasize the need for ongoing research and development in this area.

2.3.1 Political Manipulation

Social bots have been widely used to manipulate public opinion and interfere with elections by disseminating political propaganda, false information, and polarizing content. The 2016 U.S. presidential election is a well-known example where social bots played a significant role in amplifying and spreading politically biased content, affecting the dynamics of the election [39, 40, 41, 42]. Researchers have developed various social bot detection techniques specifically targeting political bots, aiming to minimize their influence on public discourse and democratic processes [43, 44].

An in-depth analysis of the 2016 U.S. presidential election [40] revealed that social bots were responsible for generating and disseminating a significant portion of the election-related content on social media platforms, particularly Twitter. Bots were found to be responsible for roughly one-fifth of all conversations surrounding the election, with a noticeable bias towards certain political topics and candidates. By applying machine learning algorithms to analyze behavioral patterns, Bessi and Ferrara were able to uncover the widespread presence of bots and assess their potential influence on the election. These findings were later connected to a state-sponsored operation led by Russia to interfere with the U.S. election [41, 45, 46]. [47] proposed a method to detect social bots in the context of political discussions on Twitter, focusing on the 2017 Catalan referendum for independence. They used a combination of unsupervised learning techniques, such as clustering and dimensionality reduction, to identify groups of similar users and detect bots based on their behavioral patterns. By examining the content created by these bots, the researchers [47] were able to reveal coordinated disinformation campaigns and understand the tactics used by bot operators during the election.

In conclusion, social bots have played a major role in shaping public opinion and interfering with elections in various countries. Research into detecting and understanding how political bots operate has provided important insights into their strategies and the potential impact on democratic processes. Developing effective ways to detect and counter these bots can help limit their influence on public discussions and protect the fairness of elections.

2.3.2 Campaign of Disinformation

The spread of disinformation and fake news on social media has become a major concern, as it can fuel misinformation, deepen polarization, and erode trust in both media and institutions. Detecting social bots that participate in these disinformation campaigns is essential to reducing the spread of fake news and preserving the integrity of online information. In a study by [48], the authors proposed a hybrid deep-learning model called CSI (Capture, Score, and Integrate) to detect fake news and the social bots responsible for spreading it. By combining features from both content and social network structures, their model achieved high accuracy in detecting fake news and the bots disseminating it. This highlights the potential of AI-driven techniques in tackling the challenges of disinformation.

In another study, [49] explored the spread of true and false news online and found that false information spread more rapidly and broadly than true information, partially due to the activity of social bots. They employed a range of machine learning techniques to model how news spreads and identify the features that distinguish true information from false. Their findings emphasized the importance of effective social bot detection and countermeasures to curb the spread of disinformation.

Addressing the issue of disinformation and fake news requires a comprehensive approach, including the development of detection methods that can identify and mitigate the impact of bots spreading false information. By utilizing advanced machine learning and natural language processing techniques, researchers can gain deeper insights into the strategies and tactics of disinformation campaigns, enabling them to design interventions that protect the integrity of online information sources.

2.3.3 Financial scams

The rise of social bots on social media platforms has led to a new wave of financial scams and cryptocurrency manipulation, leaving many unsuspecting users vulnerable to digital deception. These bots, often referred to as wolves in sheep's clothing, can mimic human behaviors so convincingly that they can quickly spread false information or even execute schemes like pump-and-dump frauds, phishing, or manipulating stock and crypto prices.

Social bots in the financial sphere are particularly troublesome because of their ability to operate at scale and with alarming speed. They can inundate platforms with fake news, convince users to invest in worthless assets, or artificially inflate the price of cryptocurrency, all while remaining undetected by traditional methods.

However, there is hope on the horizon. While detecting these bots is no easy feat, as they constantly evolve to circumvent traditional detection methods, research is advancing to develop more sophisticated tools. Various studies have proposed methods to track bot behavior through network analysis, machine learning algorithms, and behavioral anomaly detection—essentially, aiming to catch the bots red-handed before they can wreak havoc on financial systems.

Incorporating these techniques into broader anti-fraud strategies could provide a multi-layered defense, giving users and financial platforms the ability to better identify these digital troublemakers before they cause more damage. For example, [50] developed a data-driven framework to identify pump-and-dump schemes in cryptocurrency markets. By collecting and analyzing millions of messages from platforms like Twitter, Telegram, and Discord, their model was able to unveil mechanisms such as pump-and-dump and Ponzi schemes, revealing activity associated with suspicious bot accounts involved in cryptocurrency frauds.

To reduce the impact of social bots in financial scams and cryptocurrency manipulation, it's essential for researchers to create new detection methods that can keep up with the constantly changing tactics used by these bots. Potential research directions include the incorporation of deep learning and reinforcement learning algorithms, as well as the development of models that can analyze multimodal data sources, such as text, images, and videos [51].

Social bot detection has become an increasingly important field of research, as malicious bots continue to disrupt online information ecosystems and create significant challenges across various domains. In this concluding section, we summarize the key challenges and opportunities in social bot detection, explore the future of detection in the age of advanced AI models like ChatGPT, and offer final thoughts along with potential directions for future research.

Chapter 3

Tools and Technologies Used

3.1 Web scraping

The main technique used by our tool to retrieve data is web scraping; This process involves the automatic extraction of data from websites, aimed at collecting specific information from one or more pages for purposes such as data analysis, dataset creation, or monitoring of certain content. To achieve this, scripts or programs known as **scrapers** send requests to websites to extract the desired data (such as text, images, or other structured content) and save it in a reusable format.

Here are the main steps of the scraping process in our project:

- **Request submission**: The scraper accesses one or more web pages by sending HTTP requests, simulating the behavior of a normal user or browser. In our project, access to information is facilitated by the use of server-provided APIs, which allow specific data requests in a structured and optimized way, reducing the risk of errors and overloads.

- **Receipt of content**: The response to requests sent comes in JSON format, a lightweight and easy-to-read format for both computers and humans. Each JSON response includes the data fields of interest, facilitating fast and accurate handling of information.

- **Data extraction**: Only data specifically required for analysis is identified and collected, while non-relevant content is ignored. At this stage, the scraper carefully filters the received JSON content, selecting targeted fields and attributes.

- **Organization and saving**: Once extracted, the data is organized in a structured format, such as an SQL database, enabling subsequent analysis. This is a crucial step to ensure that the data is ready for processing, visualization, or export into data analysis tools.

Web scraping is a powerful technique but requires compliance with website usage policies and data protection laws. Some sites explicitly prohibit scraping in their terms of service or through the `robots.txt` file, which provides bot instructions on which pages are accessible and which are not. In our case, we have respected the server limit, which allows a maximum of 300 requests every 5 minutes. This limit has been managed by inserting automatic pauses in the script and rotating proxy servers.

One of the main tools used is the `BeautifulSoup` library for Python, which facilitates parsing HTML content. Although most of the data in our project comes from JSON APIs, BeautifulSoup was used for situations where it was necessary to access content available only in HTML, such as published user posts and their profile descriptions.

Thanks to web scraping, large amounts of data could be collected quickly and accurately without manual entry. This approach allowed for the creation of up-to-date and consistent datasets, reducing the margin of human error and accelerating the analysis process.

3.2 Application Programming Interfaces (APIs)

We utilized **Application Programming Interfaces (APIs)** to query the Mastodon server. APIs are interfaces which allow different software applications to communicate with each other. An API defines a set of rules and protocols that specify how an application can request and receive data or services from another application, without the need to understand the internal details of the implementation.

Think of an API as a waiter in a restaurant:

- **Request**: You (the client application) place an order with the waiter (API), choosing from the menu (API specification).
- **Intermediary**: The waiter takes your order and sends it to the kitchen (application server), without you having to enter the kitchen or understand how the dish is prepared.
- **Response**: The kitchen prepares the dish and the waiter brings it to you.

This is exactly the role of an API between two applications: the client application sends a request to the API, which then returns the requested data or service.

There are different types of APIs depending on their purpose and operation:

- **Web API**: The most common type, allowing applications to communicate via the Internet (e.g., REST, SOAP).
- **System API**: Enables programs to interact with the operating system (e.g., Windows API).
- **Library API**: Used to access specific features of a library or framework (e.g., OpenGL graphics library).
- **Class or Module API**: Used within the code of a program to access the functionality of a specific class or module.

APIs are a useful feature as they allow existing services and functionalities to be reused without having to recreate them; this enables different systems to easily connect and integrate within the same context.

3.2.1 Mastodon's REST API

In the Mastodon ecosystem, **representational State Transfer (REST) APIs** provide a way to access platform data and functionality in a structured way, using **HTTP** for requests and **JSON** for sent and received data. As explained in the documentation:

> Mastodon provides access to its data over a REST API. REST stands for REpresentational State Transfer, but for our purposes, just think of it as sending and receiving information about various resources based on the request. The Mastodon REST API uses HTTP for its requests and JSON for its payloads.

The logic of REST APIs is based on a set of specific endpoints that can be accessed via HTTP methods, as listed in the Mastodon documentation:

- **GET**: Retrieves information about a resource. For example, you can use GET to view a post or get the details of a user.
- **POST**: Sends information to the server to create a new resource, such as a new post.
- **PUT or PATCH**: Updates an existing resource, such as changing the content of a post or updating a user's profile.
- **DELETE**: Removes a specific resource, such as deleting a post or comment.

Examples of a GET request on Mastodon could be:

- **GET /api/v1/statuses**: To retrieve public posts.
- **GET /api/v1/accounts/:id**: To get the details of a specific account, given its ID.
- **GET /api/v1/statuses/:id**: To view a specific post given its ID.

Mastodon's REST API offers several endpoints to interact with the platform:

- Retrieving public posts.
- Updating user profiles.
- Displaying notifications.
- Creating or deleting posts.

API requests can include additional parameters, such as query strings, form data, or JSON, that enables you to filter or customize the response. By default, each request returns 20 objects, but with the `limit` parameter, you can get up to 40 objects. Many other parameters can be used as filters.

API responses include a JSON body and HTTP status codes, as explained below:

- **200 OK**: The request was successfully handled.
- **4xx Client Error**: The request contains an error, such as:
  - **401 Unauthorized**: The user is not authorized to access the resource.
  - **404 Not Found**: The requested resource does not exist.
  - **422 Unprocessable**: The request is valid but cannot be processed.
- **5xx Server Error**: Indicates a server error, such as:
  - **503 Unavailable**: The server is currently unavailable.

Some Mastodon API methods require authentication via **OAuth**, a protocol that allows secure requests to be made on behalf of a user. To use OAuth-protected APIs, you must include the `access_token` in the HTTP header of the request:

Authorization: Bearer <access_token>

3.2.2 Rate Limits

Mastodon imposes limits on the number of requests a user or machine can make within a given time period to prevent server overloads and ensure the platform remains responsive for all users.

The main rate limits are:

- **Limits per account**:
  - Each API method can be called 300 times every 5 minutes per account.
  - Media uploads (`POST /api/v1/media`) are limited to 30 requests every 30 minutes.
  - Post deletion (`DELETE /api/v1/statuses/:id`) is limited to 30 requests every 30 minutes.
- **Limits per IP**:
  - Each API method can be called 300 times every 5 minutes per IP address.

Additionally, API responses include headers that provide information about the status of rate limits:

- **X-RateLimit-Limit**: Maximum number of requests allowed for the time period.
- **X-RateLimit-Remaining**: Number of requests left before the limit is reached.
- **X-RateLimit-Reset**: Timestamp indicating when the limit will be reset.

3.3 Data annotation and MySql

An annotated dataset built on MySQL was used to record the data collected by the tool.[1] Each annotation enriches the raw data (e.g., the text of a post) with contextual details that can be used to perform more advanced analysis. For example, this allows us to analyze how bot accounts evolve over time by observing changes in their behavior in relation to the evolution of the platform.

In our case, we are collecting posts from Mastodon along with additional details such as the date of creation, the replies received, the number of reblogs, and more. This data and its related annotations are stored in the MySQL database. Each post is a record that contains several columns for each annotation (e.g., "responses", "media", "language").

As part of this thesis, the annotated dataset we are building is not just a simple social data repository but a specialized dataset for *bot detection* within the Mastodon platform. In other words, we are creating a dataset that not only collects information about users and their posts but also includes annotations that will allow us to distinguish between human behaviors and those that indicative of bot activity.

MySQL plays a crucial role in the management and analysis of this annotated dataset. The database structure allows for the efficient collection and storage of user and post data, as well as specific annotations essential for bot detection. Each post is represented by a record that includes columns for various types of annotations (e.g., `replies_count`, `reblogs_count`, `favourites_count`), enabling advanced queries to search for and identify suspicious behaviors. In addition, the use of primary keys (`post_id`, `account_id`) allows relationships to be established between users and their posts, making it easier to collect all relevant information for each account.

An example of a useful query to detect bots could involve looking for accounts with an abnormal number of posts in a short time interval or a high number of posts that receive no interactions (no likes, no replies, no reblogs). Post annotations, such as the `created_at` field and interaction counts, help build these patterns of behavior.

[1]An annotated dataset is a set of data that has been enriched with additional information (the "annotations") that describe, label or categorize the data itself.

3.4 Proxy Server

As described in the next section, proxy servers have been a fundamental component of our project. Before delving into their specific uses, let's first understand what they are and their functions.

A proxy is an intermediate server that acts as a "bridge" between a client and the end server, forwarding requests and responses between the two.

It has multiple functions:

- **Security Enhancement**: Provides protection against direct access and enables anonymization.
- **Caching and Optimization Performance**: Reduces latency via intermediate caches.
- **Access Control and Filtering**: Monitors and restricts access to specific sites.
- **Load Distribution**: Optimizes traffic management on backend servers.

There are several types of proxy:

- **HTTP Proxy**: Handles HTTP requests and responses, typically on standard ports (such as 80 for HTTP and 443 for HTTPS).
- **HTTPS Proxy (SSL Proxy)**: Handles HTTPS traffic, encrypted with SSL/TLS.
- **SOCKS Proxy**: Manages any type of traffic (HTTP, FTP, SMTP, and others) and is often used to bypass network restrictions. SOCKS is a lower-level protocol that works on the session level.[2]
- **Transparent Proxy**: Redirects traffic without the user being aware of its existence and commonly used to monitor traffic in a non-intrusive manner (e.g., in offices or schools).
- **Reverse Proxy**: Manages incoming traffic to the web server (backend) and forwards it to the appropriate internal servers.

Proxies can also be used to ensure anonymity and privacy by hiding the client's real IP address. However, not all proxies offer this service. For example:

- **Transparent proxies**: Forward the user's IP address and do not offer anonymity.
- **Anonymous proxies**: Hide the client's IP by omitting certain types of data, such as client identifiers.

[2]The order of OSI model levels, from level 1 to level 7, is as follows: 1) Physical, 2) Data Link, 3) Network, 4) Transport, 5) Session, 6) Presentation, 7) Application.

- **High-Anonymity Proxies (Elite Proxies)**: Guarantee maximum privacy by camouflaging the user's IP address and the use of the proxy.

While proxies offer privacy benefits, they also have limitations and risks. For example, public or unprotected proxies could intercept and record sensitive data, exposing you to potential privacy breaches or data theft risks. Therefore, private information or credentials should be hidden. Additionally, connection performance may suffer drops, especially if the proxy is geographically far from the target client or server, causing higher latencies and response times.

Choosing the most suitable proxy type and configurations, and meeting some compromises, can balance data security with connection performance.

In our project we used free **Forward Proxy** to implement **proxy rotation**. This is a common tecnique used in web scraping to prevent the program from being interrupted due to exceeding rate limits. When the maximum number of requests is reached, triggering error 429, the client that makes the requests is updated with a new proxy. To implement proxy rotation, we simply update the parameters of the GET request by specifying the proxy server to be used.

3.5 About the efficiency

Initially, we chose to follow a simple and linear approach, adopting a sequential process.

- **Sequential approach**:

  The common practice of making multiple HTTP requests can be done sequentially.

  There are two ways to make these requests: with or without maintaining a persistent session.

  The latter is commonly used by those who are new to it (like me when I started this project). When using requests.get(), you are making an HTTP request without keeping a persistent session open. This means that the requests library opens a new HTTP connection to the server, sends the request, receives the response, and then closes the connection. This method is time-consuming because it requires negotiating a TCP connection and, if it is an HTTPS request, the SSL handshake each time. This can slow down the program, especially if you make many requests to the same server. Additionally, since the connection is not maintained, resources from existing connections (such as cookies or persistent headers) cannot be reused for subsequent requests, limiting session continuity.

  On the other hand, when you use requests.Session(), an HTTP session is created that keeps the TCP connection open for a series of consecutive requests to the server. This session allows requests to reuse the same connection for multiple requests, creating a more efficient and consistent operation. It reduces latency by avoiding the overhead of reopening the connection and it retains cookies, custom headers and other settings between requests. For example if a website uses cookie-based authentication, once the authentication is successful, session cookies will be retained in session, making access continuous in subsequent requests.

We then decided to switch to an asynchronous approach, to further optimize our workflow.

- **Asynchronous approach**:

  Asynchronicity is a programming concept that allows multiple operations to be managed simultaneously in an efficient way by exploiting a single thread which requires minimal memory and computing power from the CPU, unlike multiprocessing. It is particularly useful for operations that require waiting for a response (such as network requests, read/write to files or database interactions).

  Asynchronicity is ideal for Latency network operations, such as retrieving data by exploiting APIs, and interacting with external systems or devices where the program has to wait for a response. However, it is not suitable for complex calculations that require a lot of processing power.

  In a traditional synchronous program, instructions are executed one at a time, with each function completing before the next one starts. Asynchronicity, on the other hand, operates using an **event loop** that decides which operation to perform at any given time. This handles several operations called **couroutine** [3] and, whenever an operation is waiting for a response from the server, moves on to the next task without blocking the whole program. This function is possible via the *await* function. The event loop is therefore a conductor who calls the coroutines, or his orchestral ones, to perform them, but if one is not ready, it moves to the next.

  A synchronous approach would be like having only one waiter take an order, wait for the food to be ready, serve it and then move on to the next table, while the other tables must wait. An asynchronous approach is like having a waiter take the order and, while waiting for food to be ready, take other orders.

- **Httpx**:

  Httpx is a Python library for making HTTP and HTTP/2 requests that combines the simplicity of requests with native support for both synchronous and asynchronous operations. In asynchronous mode, httpx uses *asyncio*, supporting many concurrent requests efficiently without blocking the main thread.

  Designed to be user-friendly, httpx maintains a similar syntax to requests but with powerful support for both synchronous and asynchronous operations, timeouts, advanced authentication and persistent connection management,

[3]The coroutines are defined with the keyword async def. When you call a coroutine, instead of running the code immediately, it returns an object that can be run when it is ready, but does not run immediately.

improving performance when making many requests to the same server. Like aiohttp, httpx is not suitable for CPU intensive tasks.

Httpx uses a persistent client model thanks to Client and AsyncClient, which employ advanced connection management called **connection pooling**. This keeps the connection open for subsequent requests, avoiding additional latency due to the continuous opening and closing of TCP connections.

From the project's perspective, it is important to note that Httpx offers integrated support for working with HTTP and HTTPS proxies, enhancing security, privacy, or network needs.

Chapter 4

Mastoanalyzer

4.1 About Mastoanalyzer

Mastoanalyzer aims to create an analysis tool that facilitates data collection on Mastodon and supports bot detection. This tool is designed to enable researchers and analysts to extract, organize, and analyze relevant informations about users and Mastodon posts to gain meaningful insight into user behavior.

The tool uses the technologies mentioned in the previous chapter, including web scraping, which makes use of the REST APIs provided by Mastodon to access public information safely and reliably. It also leverages annotated datasets stored in MySQL, chosen for its robustness in managing structured, fast queries, and easy integration with analytical tools. With data storage in MySQL, the tool can support long-term analysis, handle larger datasets, and facilitate fast access to data without performance issues. In addition, the JSON format allows integration with other analysis software.

The tool architecture is designed to promote organic growth, allowing additional functionality to be added without rewriting the system from scratch. For example, quick editing of endpoints (instances and timelines of interest) makes it adaptable to different research needs without requiring advanced programming skills for the user. It is also modular, with dedicated spaces for configurations and parameters that promote the future evolution of the tool without compromising its stability. Moreover, it is versatile enough to support the implementation of machine learning algorithms, enabling the creation of real-time bot classification models using the collected data.

This project sought to combine practicality with ethical considerations, offering an accessible solution for analyzing the increasingly complex digital context of decentralized social networks. The flexibility and interoperability of the tool aims meet immediate data collection needs while fostering greater awareness of the impact of automation and the quality of online interaction. By contributing to new possibilities for transparent analysis, this tool is proposed as a valuable resource for better understanding the dynamics of digital communities and supporting responsible data management, respecting the autonomy of platforms such as Mastodon.

4.2 Program architecture

The core of the program revolves around two main functions: the first is to collect all users interacting in a specific timeline, and the second is to record statuses that these users have posted on their profile.

1. **Fetch users (def get_timeline_posts())**:

   During this phase the mastodon.social timeline is queried. By exploiting Mastodon's REST APIs, we can retrieve all the statuses published here. Each request can contain a maximum of 40 statuses[1], so 300 requests[2] were made, collecting a total of 12,000 statuses from the most recent posts. To narrow the field of analysis, we added tags to the *url* from which the posts are collected. The primary tag used was *politics*, with secondary tags such as *technology*, *science*, and *internet*.

   Since the focus of interest shifted to the users, the JSON response of each request contains not only information about the status but also about the account that published it. This way, we collected user information.

   Once the program run correctly and for each post we have the user's information, we can upload each tuple[3] into the table *users*, which contains the following field:

   Table 4.1. users

Field	Type	Null	Key
user_id	varchar(255)	NO	PRI
username	varchar(255)	YES	UNI
bot	tinyint(1)	YES	
url	varchar(255)	YES	
followers	int	YES	
following	int	YES	
statuses	int	YES	
description	text	YES	

   The following graph shows the differences between opening a session for each request and using a single session for all requests.

   As illustrated in figure, the response time gradually increases with the number of requests, from a few hundredths of a second to almost three seconds. The graphs were generated from a total of 350 requests. In *request* case, the total processing time was 697 seconds, while in *session* case, it was 660 seconds. The comparison of the two scenarios shows that the session-based approach is more advantageous, since it reduces overall processing time and also ensures a more stable and predictable response by avoiding abnormal peaks in response time.

[1]Mastodon's REST API set as default number of statuses you can retrieve per request up to 20, but its adaptable from 1 to 40.

[2]Per account and per IP all endpoints and methods can be called 300 times within 5 minutes.

[3]Mastodon allow to self-declare bot, in JSON response is reported and the key has value of 1.

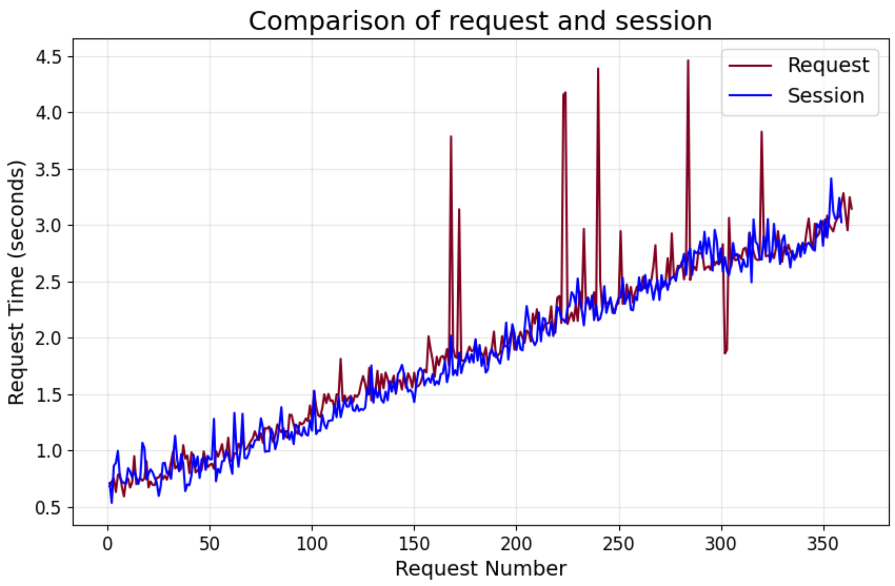


Figure 1. Comparison beetween *request.get()* and *session.get()*

2. **Statuses per user (def fetch_posts())**:

   Once the users table is filled, we can start retrieving posts from these users. Starting with the most active users (those with more statuses), we retrieve a given number of statuses per user, in our case, 200 statuses. This allows us to query up to 60 users due to rate limits.

   However, once the rate limit is reached, the program raises error 429.[4] To remedy this problem, a *time.sleep* is started with a variable duration depending on how many seconds are left until the reset time. The default value for time.sleep is 300 seconds, as the reset occurs every 5 minutes. This automates the program and ensures that all users are queried.

   Another problem occurs because all requests are made from the same IP address. Mastodon slows down the response time until it closes the connection and the program raises error 503.[5]

   The solution to errors 429 and 503 is to use proxy rotation. When the program encounters a rate limit exceed error, instead of going into *time.sleep*, we change the client making the request to a new proxy. This ensures a continuity almost infinite program, preventing the server from slowing down response times and depending only on the proxy efficiency. Requests are distributed across different clients that alternate, reducing the risk of Mastodon closing the connection and blocking the program's execution. By reusing each client over a span of several minutes, we can keep the connections active with minimal interruption.

[4]Error 429: You have sent too many requests in a given amount of time. Please try again later.
[5]Error 503: The server is temporarily unable to service your request due to maintenance downtime or capacity problems. Please try again later.

With the program running and completing successfully, the *posts* table contains 200 statuses for almost each user in the *users* table (with the number of statuses being freely adjustable at any time). Some users may not have all their messages retrieved because they belong to instances with different structures and technologies, in this case they are deleted from the *users* table.

The table is structured as follows:

Table 4.2. posts

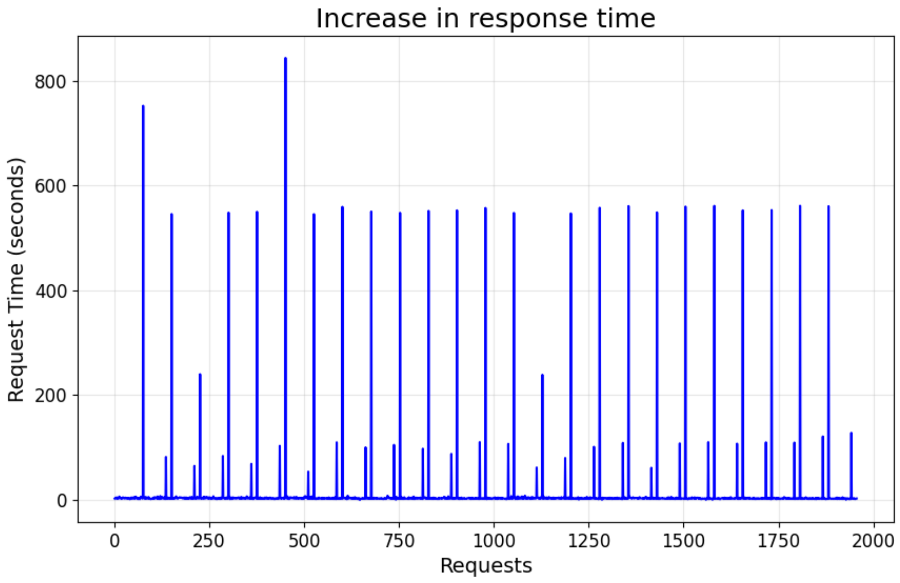
Field	**Type**	**Null**	**Key**
post_id	varchar(255)	NO	PRI
created_at	datetime	NO	
in_reply_to_id	varchar(255)	YES	
in_reply_to_account_id	varchar(255)	YES	
sensitive	tinyint(1)	YES	
spoiler_text	text	YES	
visibility	varchar(255)	YES	
language	varchar(10)	YES	
uri	text	YES	
url	text	YES	
replies_count	int	YES	
reblogs_count	int	YES	
favourites_count	int	YES	
favourited	tinyint(1)	YES	
reblogged	tinyint(1)	YES	
muted	tinyint(1)	YES	
bookmarked	tinyint(1)	YES	
pinned	tinyint(1)	YES	
content	text	YES	
media_attachments	json	YES	
account_id	varchar(255)	NO	PRI
account_username	varchar(255)	NO	
account_display_name	varchar(255)	YES	
account_url	varchar(255)	YES	
reblog_id	varchar(255)	YES	
reblog_content	text	YES	
reblogged_from_account	varchar(255)	YES	

The selection of these fields was based on an analysis of previous work and existing literature, which identified theme as crucial for our analysis, in particular for the detection of bots on social media (social bot detection). By examining the username structure on Mastodon and other relevant features, we confirmed that this data is critical to our research goals.

The users table includes both self-declared bots and real users, and consequently, the posts table contains statuses belonging to both categories. This distinction is fundamental to the analysis, as it allows us to study the differences in behaviour between these two categories.

Let's examine the program's efficiency when employing different techniques.

1. **Effect of *Time.Sleep()* on Request Timing**
   The graph illustrates how the program's time efficiency is influenced by the implementation of *time.sleep()* to pause execution when encountering error 429. Unfortunately, when a single client makes a high volume of requests, the server could eventually terminates the connection after hundred or thousands of users have been queried. In a single session, querying 300 users took the program 3428 seconds, or approximately 57 minutes, before the server aborted the connection. Each point on the graph represents the time taken to perform 5 requests to query one single user and retrieve its last 200 posts. Each request takes a few cents of a second to reach a few seconds, however there are some abnormal peaks, the lower are caused by the *time.sleep* to reach the reset time, while the higher are caused by **connection aborted**, this case is handled repeating *time.sleep* since the it works again.



2. **Effect of *Proxy Rotation* on Request Timing**
   The graph illustrates how the time efficiency of the program is affected by the rotation of eight proxies and our IP address. In a single session, querying 300 users took the program 2646 seconds, or approximately 44 minutes. Theoretically each client can query up to 60 users and execute a total of 1,500 requests to retrieve 60 000 statuses, however, as shown in the graphs, each negative peak represents a replacement of proxy, which are a very insecure and unreliable tool such that they do not work properly even if previously tested (proxy number 1, 2, 3, 5, 8) or are very slow (proxy number 4, 6, 7). However by this

technique, the program can maintain consistent access, increase anonymity, and prevent server-side restrictions that could arise from too many requests coming from a single IP address.

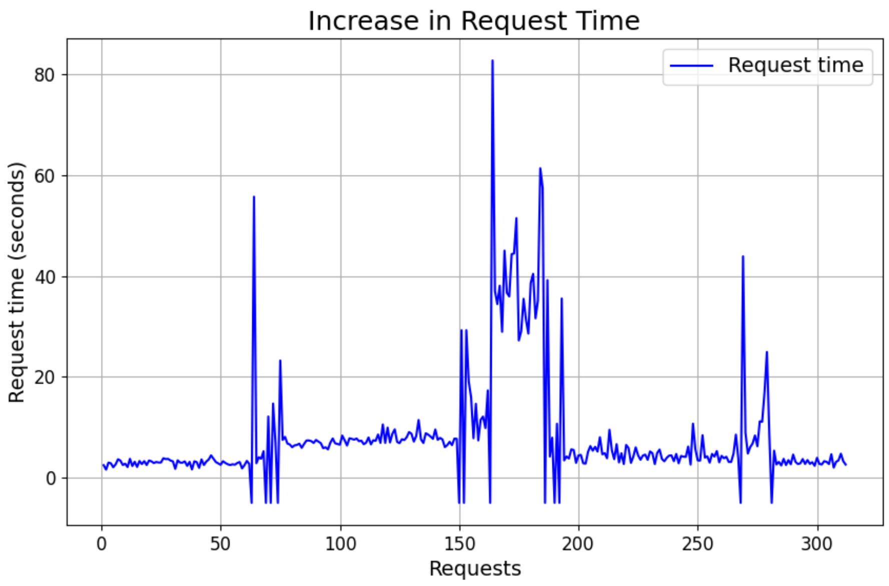


Figure 2

3. **Effect of *Asynchronous task* on Request Timing**

   To maximize the client's performance, we can execute asynchronous requests that query different portions of users in the *users* table. We generate three tasks that simultaneously execute requests to the Mastodon server, each querying 100 users, for a total of 300 users and 1,500 requests. Unfortunately, due to the imposed rate limit, the client encounters **error 429** and to remedy this problem, we can implement *time.sleep* as described above. The median of request time is 2.7 seconds and the peak shown in the graph are caused by *time.sleep* and the amount of execution time is 2783 seconds, 46 minutes. The following graph shows the performance in seconds of each task.

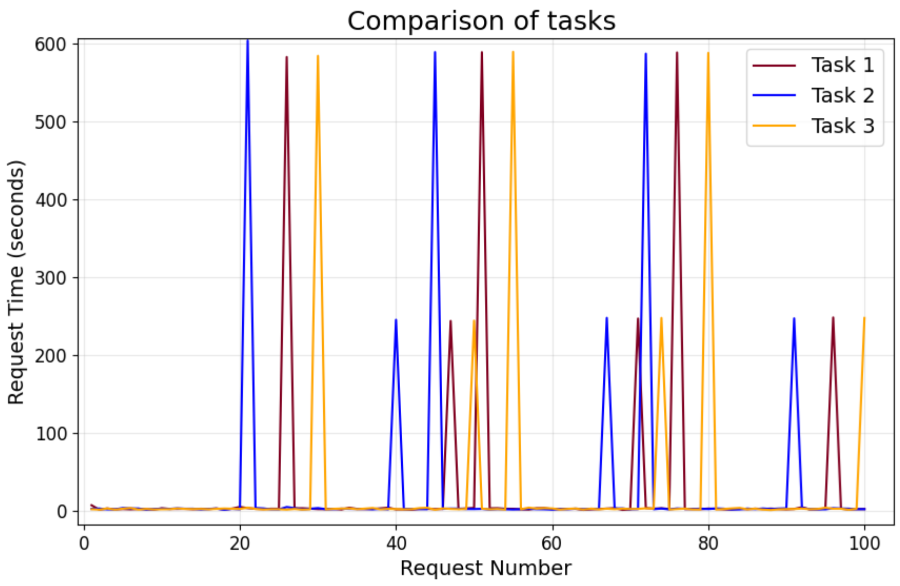


Figure 3

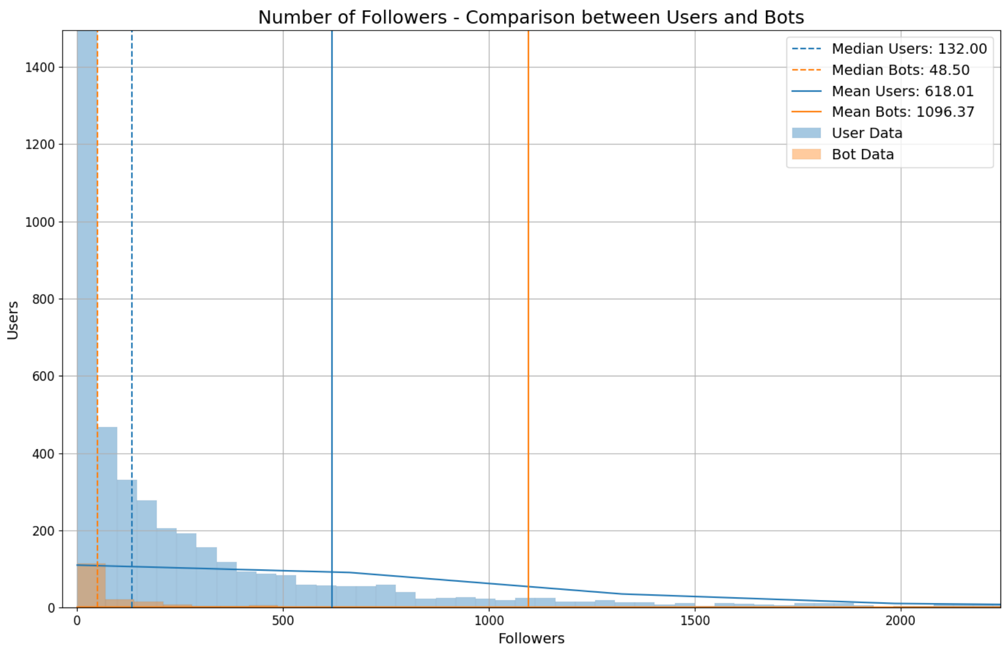
4. **Effect of *Asynchronous proxy rotation* on Request Timing**

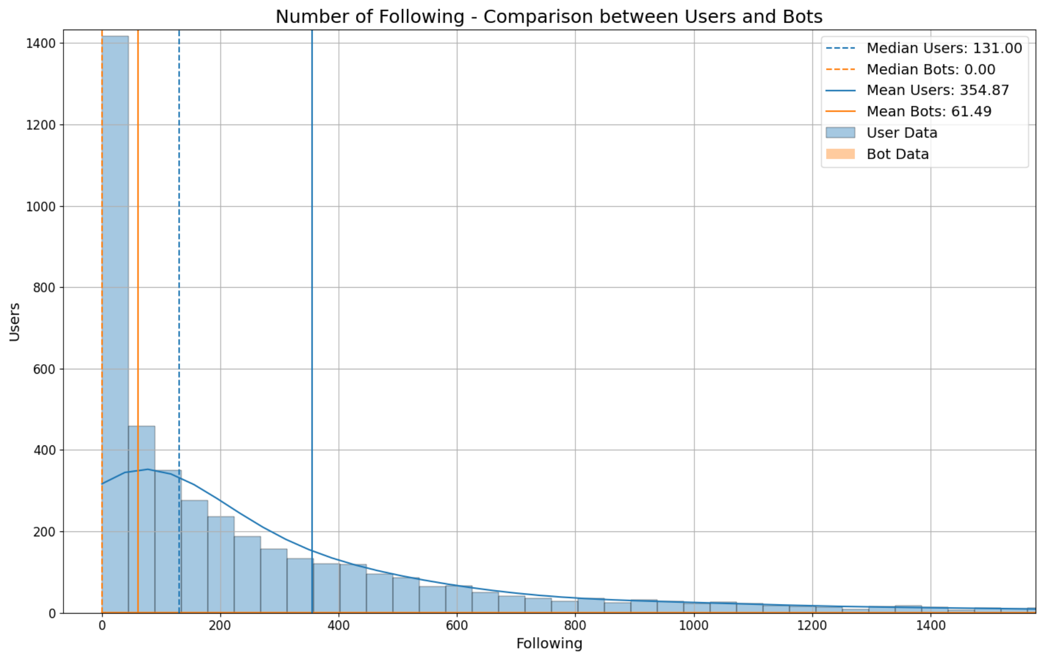
   Unfortunately free proxies tend to be unstable, slow, and overloaded, causing numerous network or timeout issues, leading to frequent connection errors like **httpx.ProxyError** or **httpx.NetworkError** and for this reason we were not able to find enough proxies to conclude the execution, altough the script works.
   In a single session, querying 300 users should take the program one third of the time on the previous approach by the moment we leverage on three tasks working simultaneously. Each tasks queries 100 users, having at its disposal a list of several free proxy that they exchange when they go in rate limit exceeded.

4.3 Euristic Bot Detection

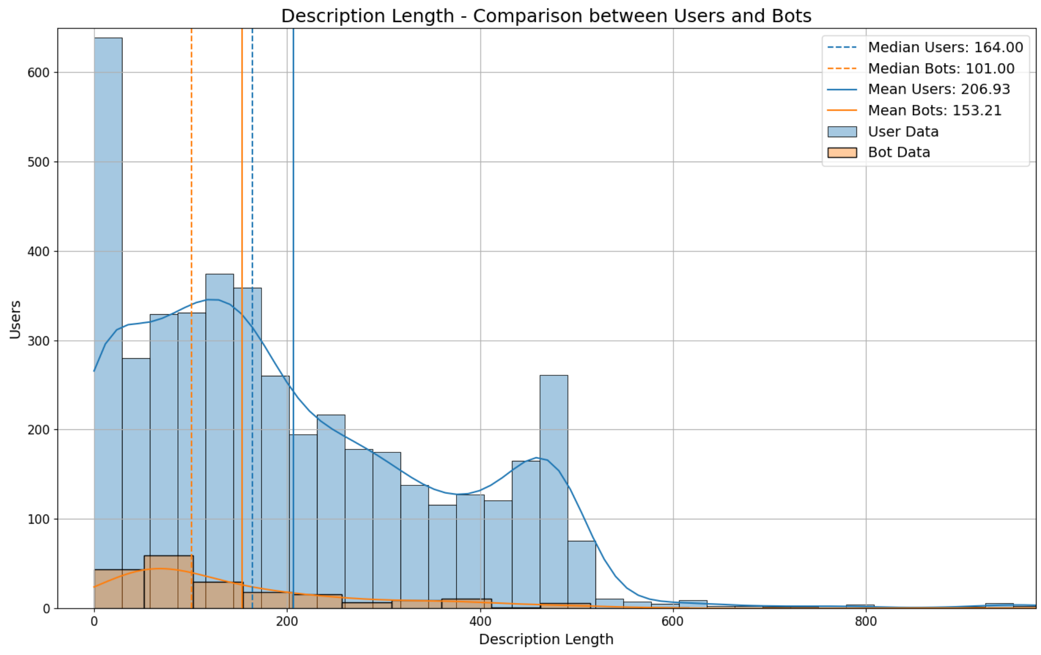
The following sections present a detailed comparison of real users and bots belonging to a dataset counting approximately 4600 users. Based on the analysis of key differences such as *followers*, *following*, *description length*, *username characteristics*, and *number of statuses*, the main conclusions are as follows:

1. **Followers and Following:**
   Users whose follower and following counts are similar to the medians observed for bots will be classified as bots. This pairing of median values serves as a key indicator for classification. The first graph shows how bots tend to be followed by many fewer people than real users, about a few dozen, unlike real users who have a greater following. Whereas the second graph shows that the number of users followed by bots is minimal, and that it includes other bots.

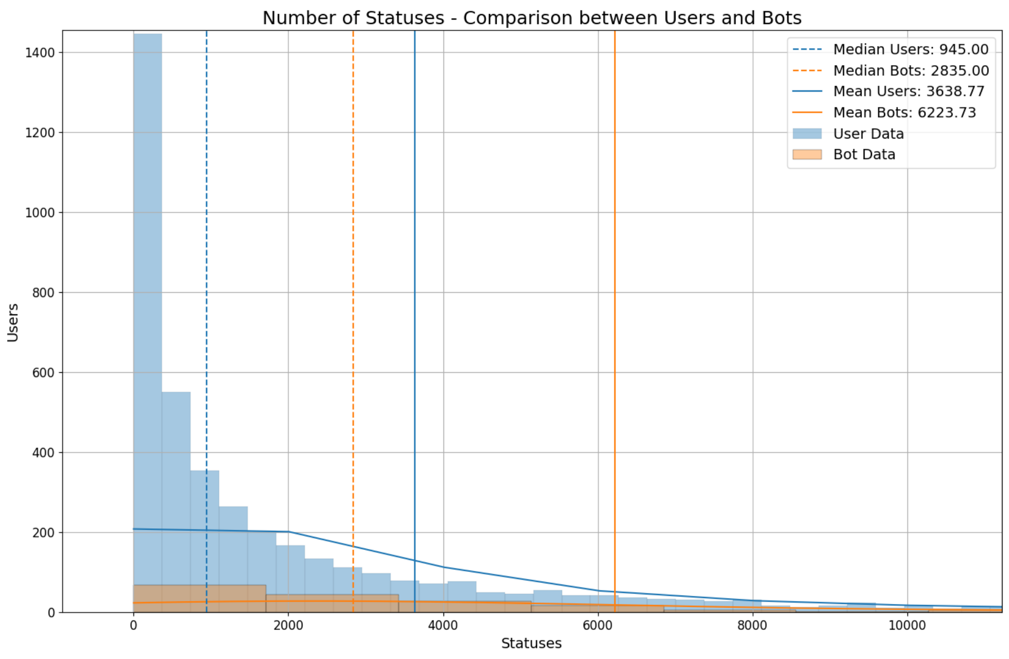




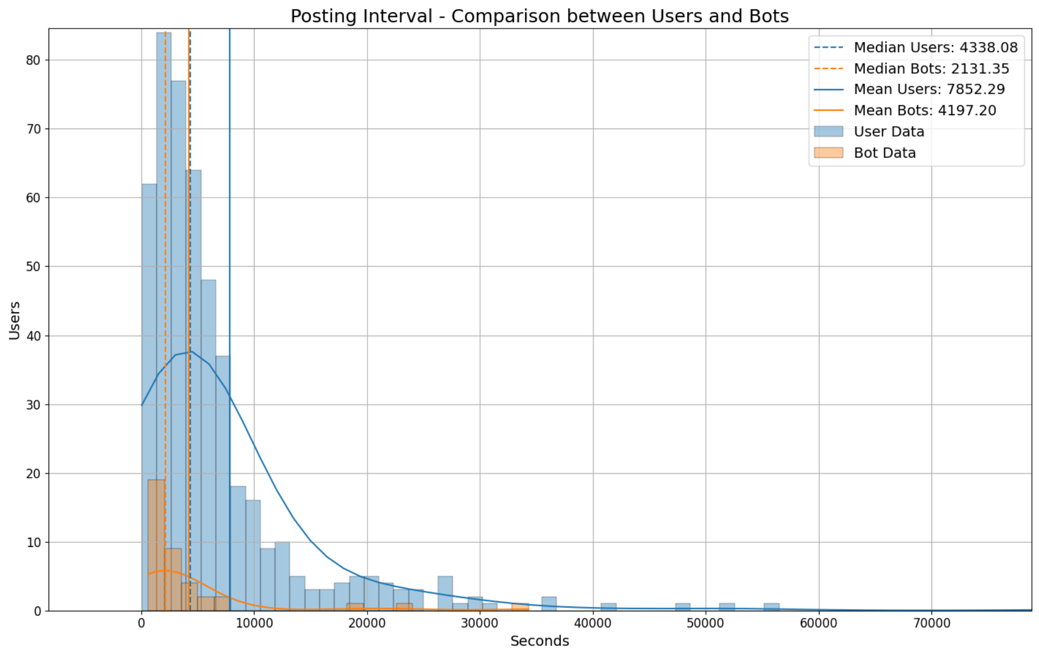
2. **Description Length and Links:**
Users whose description length is close to the median description length of bots, especially if the description contains a link, will be classified as bots. This highlights the tendency of bots to include links in their profiles, unlike real users who write about their lifestyle and hobbies.



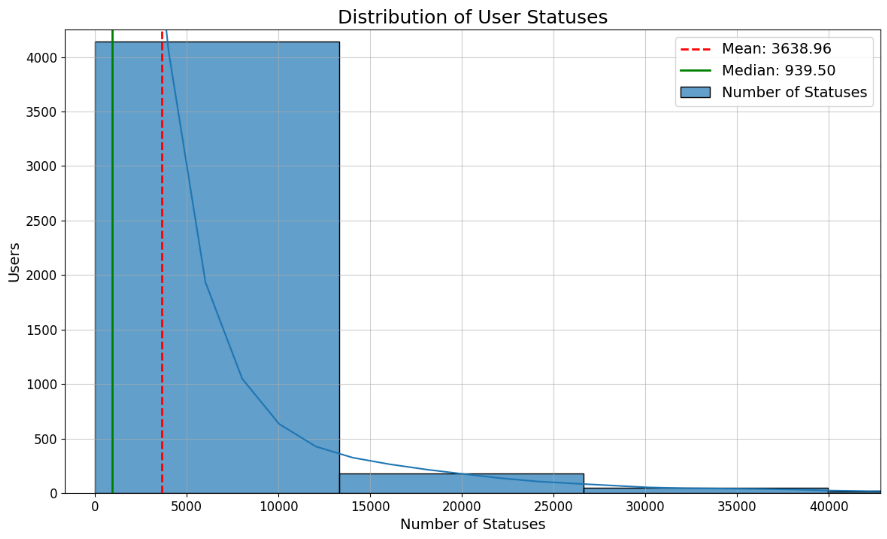
3. **Number of Statuses:**
   Users with a number of posts approximately equal to the median number of statuses for bots will also be classified as bots. This aligns with typical posting behavior observed in automated accounts. The median of number of user's statuses demonstrates that users publish less frequently than bots since they adopt a schematic and algorithmic behaviour.



4. **High-Frequency Posting:**
   Users who post very frequently, approximately every 1500 seconds, will also be classified as bots. This behavior suggests automated scheduling rather than human interaction. High-Frequency Posting is a common beaviour among automated softwares as consequence of spams, promotions and other reasons.



5. **Outlier Activity:**
The graph shows that 41000/4600 users have less than 15,000 posts and someone else has less than 28,000. Users with an exceptionally high number of statuses will be flagged as bots. Precisely those who have a much higher number of posts than the 30 000 statuses. These extreme activity patterns often distinguish automated accounts from human behavior.



By applying these criteria, the program leverages statistical medians across multiple attributes to identify suspicious users with a high likelihood of being bots.

The ultimate goal of the program is to automate the detection of suspicious accounts on Mastodon, using statistical models to compare user behavior with that typical of bot accounts. At the end of the process, the program generates a list of suspicious users, identifying those whose behavior, based on certain parameters, is similar to that of a bot. With the dataset previously generated we tracked down a handful of suspicious users, on 4619 users, 204 are bots, and 4415 are real users. Among these, 118 users were flagged as suspicious.

Conclusions

The work to develop MastoAnalyzer has been a fascinating and challenging journey into the world of the Fediverse and the challenges posed by detecting social bots. This research aimed to create a tool to monitor and understand interactions within the Fediverse, an area of the network still largely unexplored, tackling new complexities and discovering extraordinary opportunities.

Another crucial aspect was addressing the phenomenon of social bots, which can influence public discussions, spread misinformation, or even manipulate people's emotions. MastoAnalyzer demonstrates that by combining various analytical techniques, it is possible to detect bots with good accuracy, even in decentralized environments like the Fediverse.

Unfortunately, bots continue to evolve, becoming increasingly sophisticated and requiring continuous monitoring of ever-larger and more complex data volumes. However, the work done has laid a solid foundation to address these difficulties. MastoAnalyzer is designed to be a versatile and useful tool not only for researchers but also for the communities that populate the Fediverse. The results obtained show that this approach works and that we can improve the way we analyze decentralized platforms.

The structure of MastoAnalyzer is designed to efficiently and purposefully analyze data while respecting user privacy by collecting only publicly available data through APIs. This data is then processed using behavioral detection algorithms, all of which can be graphically visualized using the matplotlib library.

To better understand the dynamics between bots and human users in the context of the Fediverse, it was essential to analyze the data represented in the graphs produced by MastoAnalyzer. We can demonstrate that one of the most evident differences concerns behavior patterns: bots interact with greater regularity and repetitiveness, while human users are characterized by greater variability both temporally and in content. In fact, the graphs related to the temporal distribution of posts highlighted that bots often post at regular intervals, regardless of content or context, unlike human users who follow less predictable patterns based on time, events, or personal emotions.

Another aspect that characterizes bots is their tendency to form isolated clusters or predominantly interact with other bots, generating a less dense and more fragmented network. Human users, on the other hand, participate in more diverse conversations and contribute to more complex and interconnected networks. This data not only helps identify bots with greater precision but also highlights the importance of promoting authentic and dynamic networks within the Fediverse.

MastoAnalyzer is a modular tool, allowing new functionalities to be added or components to be replaced without affecting the entire system. It is also adaptable to other social networks within the Fediverse or even centralized ones.

In the future, we could enhance the capabilities of MastoAnalyzer by integrating more advanced technologies, such as:

- **Artificial intelligence models** to further improve bot detection and expand the analysis to other aspects of online behavior, such as the impact of misinformation or the respect for privacy.
- **Cross platform analysis** to collect and analyse data from different platforms of the Fediverse (e.g. PeerTube, Pixelfed).
- **Coordinated attack detection** to identify cyber attacks or planned disinformation campaigns through bot networks.
- **Advanced time graphs**: to represent the evolution of bot-human interactions over time.

Acknowledgements

As the cherry on top, I would like to thank everyone who shared their energy with me. To my university friends, key figures who made this journey truly special: I will forever cherish the precious memories we created together, from train rides to nights on Discord, from shared study sessions to exams we faced side by side. Without each of you, achieving this milestone wouldn't have been as easy. Thank you from the bottom of my heart! To my closest friends, who stood by my side even before this journey began and followed my every step, always cheering for me. Your support has been an invaluable gift. To my family, who, with their unwavering support, gave me the opportunity to fully dedicate myself to my studies, comforting me in moments of stress and encouraging me when I needed it most. Your help has been indispensable. And finally, to myself, for never giving up and for the promise that I never will.

Bibliography

[1] G. Gow. "Turning to Alternative Social Media". In: *The SAGE Handbook of Social Media Research Methods*. SAGE Publications Ltd., 2022, pp. 568–580. DOI: 10.4135/9781529782943. URL: https://doi.org/10.4135/9781529782943.

[2] Aymeric Mansoux and Roel Roscam Abbing. "Seven Theses on the Fediverse and the Becoming of FLOSS". In: *The Eternal Network*. Ed. by Kristoffer Gansing and Inga Luchs. Institute of Network Cultures, 2020, pp. 124–140. URL: http://journals.sagepub.com/doi/10.1177/2056305115604338.

[3] C. Cerisara et al. "Multi-task dialog act and sentiment recognition on Mastodon". In: *Proceedings of the COLING Conference*. 2018, pp. 745–754. URL: https://www.aclweb.org/anthology/C18-1063/.

[4] J. Trienes, A. T. Cano, and D. Hiemstra. "Recommending users: whom to follow on federated social networks". In: *CoRR* arXiv: 1811.09292 (2018).

[5] M. Zignani, S. Gaito, and G. P. Rossi. "Follow the "Mastodon": structure and evolution of a decentralized online social network". In: *Proceedings of the International Conference on Web and Social Media (ICWSM)*. 2018, pp. 541–551.

[6] A. Raman et al. "Challenges in the decentralised web: the Mastodon case". In: *Proceedings of the ACM IMC Conference*. 2019, pp. 217–229. DOI: 10.1145/3355369.3355572.

[7] D. Zulli, M. Liu, and R. Gehl. "Rethinking the "social" in "social media": insights into topology, abstraction, and scale on the Mastodon social network". In: *New Media & Society* 22.7 (2020), pp. 1188–1205.

[8] M. Zignani et al. "The footprints of a "Mastodon": how a decentralized architecture influences online social relationships". In: *Proceedings of the IEEE INFOCOM Workshops*. 2019, pp. 472–477. DOI: 10.1109/INFCOMW.2019.8845221. URL: https://doi.org/10.1109/INFCOMW.2019.8845221.

[9] O. Varol et al. "Online human–bot interactions: detection, estimation, and characterization". In: *Proceedings of the International Conference on Web and Social Media (ICWSM)*. 2017, pp. 280–289.

[10] R. W. Gehl and D. Zulli. "The digital covenant: Non-centralized platform governance on the mastodon social network". In: *Information, Communication & Society* 0.0 (2023), pp. 1–17. DOI: 10.1080/1369118X.2022.2147400. URL: https://doi.org/10.1080/1369118X.2022.2147400.

[11] World Wide Web Consortium (W3C). *ActivityPub*. `https://www.w3.org/TR/activitypub/`. Accessed: 2024-11-09. 2017.

[12] Emilio Ferrara et al. “The rise of social bots”. In: *Communications of the ACM* 59.7 (2016). Accessed: 1 November 2024, pp. 96–104. DOI: `10.1145/2818717`.

[13] Alan M. Turing. “Computing machinery and intelligence”. In: *Mind* 49.236 (1950), pp. 433–460.

[14] *Loebner Prize*. `http://www.loebner.net/Prizef/loebner-prize.html`. Accessed: 1 November 2024.

[15] Emilio Ferrara. “What types of COVID-19 conspiracies are populated by Twitter bots?” In: *First Monday* 25.6 (2020). Accessed: 1 November 2024. DOI: `10.5210/fm.v25i6.10633`.

[16] Bjarke Mønsted et al. “Evidence of complex contagion of information in social media: An experiment using Twitter bots”. In: *PLoS ONE* 12.9 (2017). Accessed: 1 November 2024, e0184148. DOI: `10.1371/journal.pone.0184148`.

[17] Petter Bae Brandtzaeg and Asbjørn Følstad. “Why people use chatbots”. In: *Internet Science, Lecture Notes in Computer Science*. Ed. by Ioannis Kompatsiaris et al. Vol. 10673. Accessed: 1 November 2024. Cham, Switzerland: Springer, 2017, pp. 377–392. DOI: `10.1007/978-3-319-70284-1_30`.

[18] Zi Chu et al. “Who is tweeting on Twitter: Human, bot, or cyborg?” In: *Proceedings of the 26th Annual Computer Security Applications Conference*. Accessed: 1 November 2024. 2010, pp. 21–30. DOI: `10.1145/1920261.1920265`.

[19] Howard C. H. Chang and Emilio Ferrara. “Comparative analysis of social bots and humans during the COVID-19 pandemic”. In: *Journal of Computational Social Science* 5 (2022). Accessed: 1 November 2024, pp. 1409–1425. DOI: `10.1007/s42001-022-00173-9`.

[20] Tim Hwang, Ian Pearce, and Max Nanis. “Socialbots: Voices from the fronts”. In: *Interactions* 19.2 (2012). Accessed: 1 November 2024, pp. 38–45. DOI: `10.1145/2090150.2090161`.

[21] Irene Pozzana and Emilio Ferrara. “Measuring bot and human behavioral dynamics”. In: *Frontiers in Physics* 8 (2020). Accessed: 1 November 2024. DOI: `10.3389/fphy.2020.00125`.

[22] Jahan Ratkiewicz et al. “Truthy: Mapping the spread of astroturf in microblog streams”. In: *Proceedings of the 20th International Conference Companion on World Wide Web*. Accessed: 1 November 2024. 2011, pp. 249–252. DOI: `10.1145/1963192.1963301`.

[23] Kyumin Lee, James Caverlee, and Steve Webb. “The social honeypot project: Protecting online communities from spammers”. In: *Proceedings of the 19th International Conference on World Wide Web*. Accessed: 1 November 2024. 2010, pp. 1139–1140. DOI: `10.1145/1772690.1772843`.

[24] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. “Detecting spammers on social networks”. In: *Proceedings of the 26th Annual Computer Security Applications Conference*. Accessed: 1 November 2024. 2010, pp. 1–9. DOI: `10.1145/1920261.1920263`.

[25] V.S. Subrahmanian et al. “The DARPA Twitter bot challenge”. In: *Computer* 49.6 (2016). Accessed: 1 November 2024, pp. 38–46. DOI: `10.1109/MC.2016.183`.

[26] Lei Wu et al. “Different absorption from the same sharing: Sifted multi-task learning for fake news detection”. In: *arXiv* 1909.01720 (2019). Accessed: 1 November 2024.

[27] Christopher Besel, Juan Echeverria, and Sijing Zhou. “Full cycle analysis of a large-scale botnet attack on Twitter”. In: *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM).* Accessed: 1 November 2024. 2018. DOI: `10.1109/ASONAM.2018.8508708`.

[28] Diego Pacheco, Alessandro Flammini, and Filippo Menczer. “Unveiling coordinated groups behind white helmets disinformation”. In: *WWW ’20: Companion Proceedings of the Web Conference 2020.* Accessed: 1 November 2024. 2020, pp. 611–616. DOI: `10.1145/3366424.3385775`.

[29] Diego Pacheco et al. “Uncovering coordinated networks on social media: Methods and case studies”. In: *Proceedings of the International AAAI Conference on Web and Social Media.* Vol. 15. Accessed: 1 November 2024. 2021, pp. 455–466. DOI: `10.1609/icwsm.v15i1.18075`.

[30] Karishma Sharma et al. “Identifying coordinated accounts on social media through hidden influence and group behaviours”. In: *KDD ’21: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining.* Accessed: 1 November 2024. 2021, pp. 1441–1451. DOI: `10.1145/3447548.3467391`.

[31] Andreas Rössler et al. “Faceforensics++: Learning to detect manipulated facial images”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV).* Accessed: 1 November 2024. 2019. DOI: `10.1109/ICCV.2019.00009`.

[32] Zhilin Yang et al. “XLNet: Generalized autoregressive pretraining for language understanding”. In: *NIPS ’19: Proceedings of the 33rd International Conference on Neural Information Processing Systems.* 2019, pp. 5753–5763.

[33] Kai Shu et al. “FakeNewsNet: A data repository with news content, social context and spatial-temporal information for studying fake news on social media”. In: *arXiv* 1809.01286 (2018). Accessed: 1 November 2024.

[34] Gang Wang et al. “Social Turing tests: Crowdsourcing sybil detection”. In: *NDSS.* The Internet Society, 2013.

[35] Paul Heymann, Georgia Koutrika, and Hector Garcia-Molina. “Fighting spam on social web sites: A survey of approaches and future challenges”. In: *Internet Computing* 11.6 (2007), pp. 36–45.

[36] Stefano Cresci et al. “The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race”. In: *WWW ’17 Companion: Proceedings of the 26th International Conference on World Wide Web Companion.* Accessed: 1 November 2024. 2017, pp. 963–972. DOI: `10.1145/3041021.3055135`.

[37] Juan Echeverria and Shi Zhou. “Discovery, retrieval, and analysis of the ‘Star Wars’ botnet in Twitter”. In: *ASONAM ’17: Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*. Accessed: 1 November 2024. 2017, pp. 1–8. DOI: 10.1145/3110025.3110074.

[38] Kai-Cheng Yang, Emilio Ferrara, and Filippo Menczer. “Botometer 101: Social bot practicum for computational social scientists”. In: *Journal of Computational Social Science* 5.2 (2022). Accessed: 1 November 2024, pp. 1511–1528. DOI: 10.1007/s42001-022-00177-5.

[39] Emilio Ferrara. “Manipulation and abuse on social media”. In: *ACM SIGWEB Newsletter* 2015 (Spring 2015). Accessed: 1 June 2023, pp. 1–9. DOI: 10.1145/2749279.2749283.

[40] Alessandro Bessi and Emilio Ferrara. “Social bots distort the 2016 U.S. Presidential election online discussion”. In: *First Monday* 21.11 (2016). Accessed: 1 June 2023. DOI: 10.5210/fm.v21i11.7090. URL: https://firstmonday.org/ojs/index.php/fm/article/view/7090/5653.

[41] Adam Badawy, Emilio Ferrara, and Kristina Lerman. “Analyzing the digital traces of political manipulation: The 2016 Russian interference Twitter campaign”. In: *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. Accessed: 1 June 2023. 2018, pp. 258–265. DOI: 10.1109/ASONAM.2018.8508646.

[42] Hsin-Chang Han Chang et al. “Social bots and social media manipulation in 2020: The year in review”. In: *Handbook of Computational Social Science. Volume 1: Theory, Case Studies and Ethics*. Ed. by Ulrich Engel et al. Accessed: 1 June 2023. London: Routledge, 2021. DOI: 10.4324/9781003024583.

[43] Robert Gorwa and Douglas Guilbeault. “Unpacking the social media bot: A typology to guide research and policy”. In: *Policy & Internet* 12.2 (2020). Accessed: 1 June 2023, pp. 225–248. DOI: 10.1002/poi3.184.

[44] Philip N. Howard and Bence Kollanyi. “Bots, #StrongerIn, and #Brexit: Computational propaganda during the UK-EU referendum”. In: *arXiv* 1606.06356 (2016). Accessed: 1 June 2023. DOI: 10.48550/arXiv.1606.06356.

[45] Alaa Addawood et al. “Linguistic cues to deception: Identifying political trolls on social media”. In: *Proceedings of the International AAAI Conference on Web and Social Media*. Vol. 13. Accessed: 1 June 2023. 2019, pp. 15–25. URL: https://ojs.aaai.org/index.php/ICWSM/article/view/3205.

[46] Luca Luceri et al. “Red bots do it better: Comparative analysis of social bot partisan behavior”. In: *WWW ’19: Companion Proceedings of The 2019 World Wide Web Conference*. Accessed: 1 June 2023. 2019, pp. 1007–1012. DOI: 10.1145/3308560.3316735.

[47] Massimo Stella, Emilio Ferrara, and Manlio De Domenico. “Bots increase exposure to negative and inflammatory content in online social systems”. In: *Proceedings of the National Academy of Sciences* 115.49 (2018). Accessed: 1 June 2023, pp. 12435–12440. DOI: 10.1073/pnas.1803470115.

[48] Natali Ruchansky, Sungyong Seo, and Yan Liu. “CSI: A hybrid deep model for fake news detection”. In: *CIKM ’17: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management.* Accessed: 1 June 2023. 2017, pp. 797–806. DOI: `10.1145/3132847.3132877`.

[49] Soroush Vosoughi, Deb Roy, and Sinan Aral. “The spread of true and false news online”. In: *Science* 359.6380 (2018). Accessed: 1 June 2023, pp. 1146–1151. DOI: `10.1126/science.aap95`.

[50] Luca Nizzoli et al. “Charting the landscape of online cryptocurrency manipulation”. In: *IEEE Access* 8 (2020). Accessed: 1 June 2023, pp. 113230–113245. URL: `https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9120022`.

[51] Emilio Ferrara. “Twitter spam and false accounts prevalence, detection and characterization: A survey”. In: *First Monday* 27.12 (2022). Accessed: 1 June 2023. DOI: `10.5210/fm.v27i12.12872`. URL: `https://firstmonday.org/ojs/index.php/fm/article/view/12872/10749`.