(1) Overview

Title
`PyMstdnCollect`: A Python package for data collection on Mastodon

Paper Authors
1. Chalkiadakis, Ioannis
2. Chavalarias, David

Paper Author Roles and Affiliations
1. CNRS / Institute of Complex Systems of the City of Paris
2. CNRS / Institute of Complex Systems of the City of Paris

Abstract
`PyMstdnCollect` is a Python wrapper for Mastodon's public API. Mastodon is a novel online social medium which has been gaining in popularity over alternatives (e.g. X, formerly known as 'Twitter') both for users and for social and political science researchers. Mastodon has attracted the attention of the latter, particularly since its more well-known alternatives stopped offering open access APIs. Contrary to existing low-level Python wrappers of Mastodon's API, `PyMstdnCollect` is built as a higher-level software that goes beyond being an API wrapper on top of which further code needs to be developed, but rather it is a readily usable tool for the social science researcher who needs network data (interactions and text) but does not have the time or expertise to develop the data collection process from scratch. We expect that the current software package will facilitate research with the Mastodon community and will help towards safeguarding and preserving healthier interaction between user communities on the platform. `PyMstdnCollect` is freely available on GitHub and Zenodo[1].

Keywords
Mastodon; decentralised social media; social science; political science; text mining; natural language processing; NLP; networks analysis; complex systems

Introduction
Online social discourse platforms constitute a rich environment for social and political science research that utilises quantitative methods of network (graph) analysis, text-as-data methods and natural language processing. 'X' (former 'Twitter') has been prevalent in such research due to its vast amount of users and produced text data on a daily basis, which has facilitated the analysis of social phenomena and their expression on the online public discourse. The recent changes in X's ownership, however, have resulted in a significant migration of users from X to alternative online social media, and, in addition, they have posed challenges for researchers to have access to its data, which are now only available through a costly API subscription.

[1] `https://github.com/ichalkiad/PyMstdnCollect`, DOI: 10.5281/zenodo.13119144

The Mastodon federated platform has been the go-to social medium for many former X users and it since recently been evolving into a promising data source, as a social medium gaining in popularity and user base. Its ad-free, decentralised nature gives increased power to the users over the content they are exposed to and the protection of their personal data (Gehl & Zulli 2023), which make Mastodon a potentially healthier space of online discourse, where phenomena such as polarisation, disinformation and echo chambers are expected to be less likely to occur. However, the same decentralised nature, which is built around the concept of 'instances', i.e. specific user community- or topic-based servers that set individual content rules, makes working with Mastodon and collecting data non-trivial. This is because the mechanisms that govern the accessibility of users to posted content are not easy to understand for a non-expert user to subsequently develop a tool that would support research data collection, despite the comprehensive open access API that the platform offers.

At the time of writing this article, there already exist some comprehensive wrappers around Mastodon's API. These include the low-level wrappers `Mastodon.py` in Python (Diener & contributors 2016), `Rtoot` in R (Schoch & hong Chan 2023), and the higher-lever tools `Communalytic` (Gruzd & Mai 2020) and `mastodon-toolbox` (Schatto-Eckrodt 2023). `Mastodon.py` and `Rtoot` are comprehensive wrappers around Mastodon's API, namely they provide Python and R functions over the API methods, so that the user can then build their application utilising these libraries. However, such wrappers are still difficult for social science researchers to utilise, given that they still need significant programming experience to set up the query mechanism for the collection of data, as well as the storage procedure for building a persistent dataset. Furthermore, the large codebase, particularly of the former, renders its extension cumbersome, requiring users that are well-versed in Python and have experience developing code within large codebases. This need is addressed by the higher-level tools `mastodon-toolbox` and `Communalytic`. The former is built on the `Mastodon.py` Python library, providing functionality for collection of toots based on hashtags and keywords, methods to collect 'interaction data' on the network, i.e. 'boosts' (reposts, equivalent to re-Tweet on X) and replies to 'toots' (user posts, equivalent to Tweets on X), as well as the option to store the collected data in text form (.json/.csv files). Finally, `Communalytic` is a proprietary platform for social science research on social media, which is free (not open-source) for educational purposes but not for research. It recently incorporated the ability to collect and process data from Mastodon, yet the user would have to trade-off the ease of utilisation with the cost of using the platform and lack of access to its code. As a result of the aforementioned characteristics, most existing research on Mastodon has relied on custom data collection pipelines (La Cava et al. 2021, 2022).

`PyMstdnCollect` was developed in the context of the NODES project [2]. Its functionalities and benefits are as follows:

1. friendly to the non-expert user, concise, documented library for collecting and storing Mastodon data of interest,

[2] `https://nodes.eu/`

2. readily prepared scripts to collect and store network/interaction and text data based on hashtags, keywords, user statistics, timeframes,

3. easily extensible by users with basic programming familiarity,

4. easily scalable to medium to large dataset collection due to interaction with SQL Lite database

5. free and open source.

Implementation and architecture

This section will briefly present some of the particular characteristics of the Mastodon federated social medium (valid at the time of writing of this manuscript), before presenting the package structure. This is to facilitate understanding of the network (as acquired from various online sources (Issues - Mastodon GitHub repository 2022*b*,*a*, Eden 2022*a*)) and the package code, hence allowing for easily extending or modifying it if necessary. Mastodon is one microblogging service of the Fediverse, namely a federated network connecting several services, see Figure 1), e.g. some focusing on micro- or macroblogging, visual or audio content etc. The fact that Mastodon is decentralised and part of the Fediverse means that each remote server ('instance') may have its own Terms of Service and rules, yet they all comply to the underlying protocol (*ActivityPub*[3]) that allows the different instances and Fediverse services to communicate. Consequently, Mastodon users and content are endowed with particular properties of which the researcher should be aware of before deciding to work on such platforms.

The Mastodon structure: users, toots, replies and boosts

Users and toots are uniquely identified by a pair of a Snowflake ID [4] (Eden 2022*b*) and the instance name within which the ID uniquely identifies the user/toot. Note that there is no global unique identifier for users/posts, although the researcher may create one by combining these pieces of information. Mastodon users may or may not share their social graph (followers/accounts they follow), and have increased access control over each of their posts individually (4 levels: public, unlisted, followers-only, direct message). Furthermore, note that instances may be renamed or deleted over time. Data that are accessible in a straightforward manner are the instance name and followers of the user who is logged in and aims to launch an API query, the instances of users that have reposted ('re-blogged'/'boosted') toots of the logged in user, as well as the followers of such users that are on the same instance as the logged in user. However, third user (not the logged in user launching the query) followers from other instances are not accessible, unless the logged in user has access (e.g. has been approved) by the other instances as well.
Regarding toots, an instance knows about a particular toot if any of these is true:

- some user on the instance follows the person who posted, at the time of posting;

[3] `https://activitypub.rocks/`
[4] `https://en.wikipedia.org/wiki/Snowflake_ID`

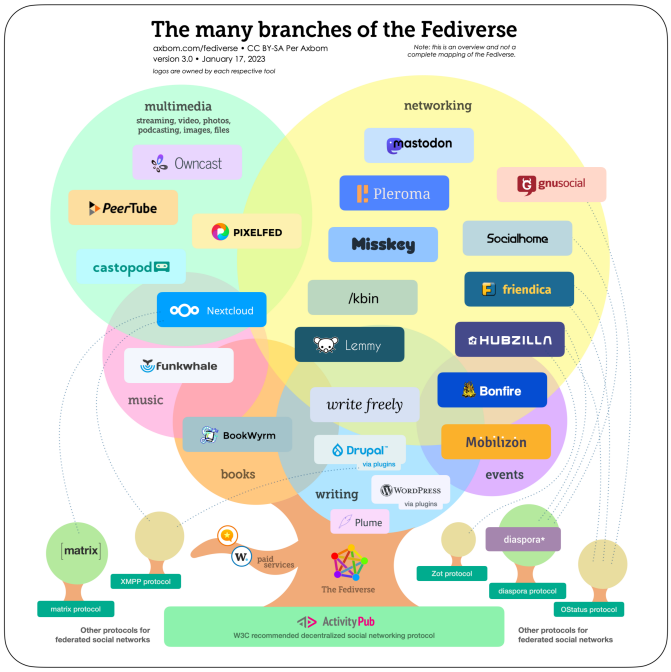

Figure 1: The Fediverse. Source: Per Axbom - `https://axbom.com/fediverse/`, CC BY-SA 4.0, `https://commons.wikimedia.org/w/index.php?curid=125076684`

- a user who is followed by someone on the instance 'boosts' or replies to the toot in question (the toot will be visible to their followers);
- the toot contains a 'mention' (tag) of any other user on the instance;
- the toot was a pinned toot when its author had been discovered by the instance (namely the author was visible within the instance) from where the query for the toot was launched;
- some user on the instance manually fetches the toot by placing the URL of the toot in the search bar of the instance.

Furthermore, note that instances do not keep track of users that received or fetched (copied the URL of a toot into the search bar of the instance) the toots on the instance.
In terms of content interactions, one may have access to 'favourited' toot counts (how many users 'like' the toot), boosts of the toot, as well as toots that may be posted as replies to the original. Note, however, that on small instances, it may be the case that the complete interaction is not visible to the user launching the query (e.g., user A); this may happen if the users that interacted with the post are not visible to user A, e.g. they are registered on an instance invisible to A or A's instance. Such design choices of the Mastodon network have been often debated by the community (Issues - Mastodon GitHub repository 2022*b*,*a*).
The above and the specific data interactions that *PyMstdnCollect* captures are visualised in Figure 2, where $I_i, i = 1, 2, 3$ denote instances that may (I_2) or may

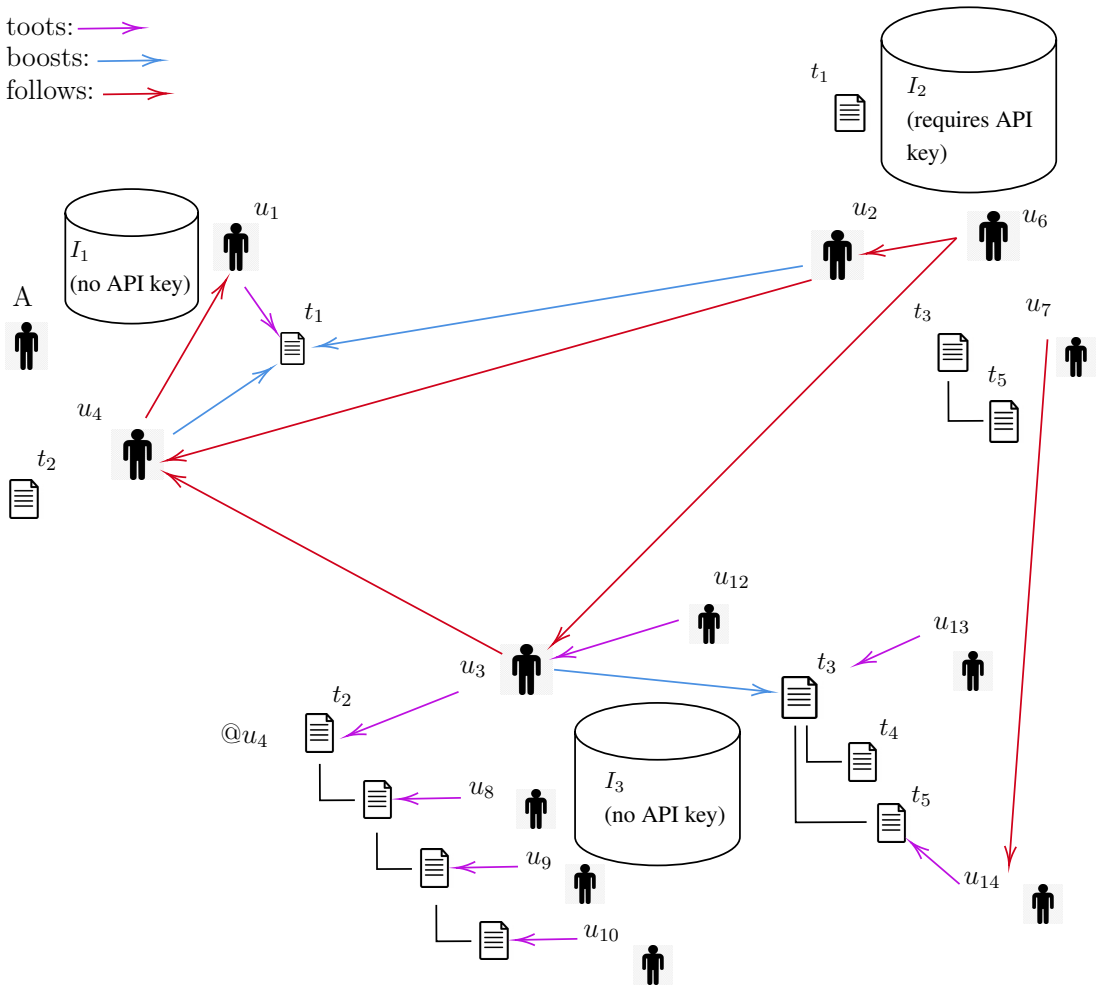

Figure 2: Mastodon federated network interactions and data accessibility. Italic font denotes that the toot is accessible on the instance but has originated from a user on another instance.

not (I_1, I_3) require approval and an API key to be accessible. *PyMstdnCollect* iterates over a user-defined selection of instances and launches the data collection from within each instance. The package offers the option to collect all public posts between a starting and an end date, or to collect posts that contain user-defined hashtags or keywords. Due to the network's federated structure, however, it is not immediately obvious that posts will have to satisfy a number of additional platform requirements that will make them accessible to the data collector. Referring to Figure 2, we make the following assumptions:

- users that publish toots are not bots, i.e. the library does not collect a toot if it is originating from a bot account (i.e. the account's 'bot' flag is set to 'True')
- toots are collectible by *PyMstdnCollect*, i.e. they are public and contain hashtags/keywords of interest or are within the specified timeframe
- the data collection process is initiated by user A on I_1

Given the aforementioned assumptions, user A has access to their instance and followers by construction of the network. Furthermore, all toots on user A's public timeline (e.g. one of their followers posted or boosted a toot) will be collectible by *PyMstdnCollect*. To illustrate the operation of the network, assume that u_1 toots t_1, which will appear on u_4's public timeline, who boosts it. Note that because u_2 follows u_4, t_1 will appear on their public timeline on I_2 (u_1 is notified about that) and u_2 may boost it, thus making it accessible to their followers (u_6). Hence, if u_2 initiates the data collection on I_2, t_1 will be collected from that instance as well. Furthermore, u_1 can see the instances and followers of users who boosted t_1 and are on the same instance or an instance that does not require authentication (u_3), but would nevertheless need to be authenticated by I_2 in order to see u_2's followers. In addition, we remark that t_2 will be collectible from I_1. This is because u_4 is mentioned in t_2 as u_3, who posted the toot, is one of their followers.
Regarding the accessibility of conversations, we refer to I_3. To begin with, u_3 will see all replies under t_2 since they are the initial poster. At the time of posting t_5, u_7 follows u_14, and I_2 has access to t_3 because u_6 follows u_3. Due to u_7 following u_14, the reply t_5 is accessible to I_2 - not t_4 however as I_2 has no way of accessing it. Hence the perspective of the conversation that I_2 has is limited to only posts accessible to it.

PyMstdnCollect

Figure 3 shows the fields contained by the toot and user data structures in Mastodon. These are the main data structures collected and stored by `PyMstdnCollect`. Note that the fields `account.globalID` and `toot.globalID` are added by `PyMstdnCollect` at the time of collection and contain a unique identifier per account and toot, constructed according to the following rule:

> toot.id@@collected-instance-name
>
> account.acct@originating-instance-name@@collected-instance-name,

where *collected-instance-name* is the name of the instance that the user/toot was collected from, and *originating-instance-name* is the name of the instance that the user is registered in. In case the *originating-instance-name* and *collected-instance-name* differ, the `account.acct` field will already be in the format *account.acct@originating-instance-name*; otherwise, i.e. if the portion *@originating-instance-name* is not in the field (in which case `account.acct` and `account.username` have the same content), then the *originating-instance-name* and *collected-instance-name* are assumed to be the same instance and the `account.globalID` is constructed accordingly. Furthermore, note that the interaction fields `toot.in_reply_to_account_id` and `toot.in_reply_to_id` are updated to the *globalID* of the account/toot once the interaction data have been collected. Finally, `PyMstdnCollect` adds the field `toot.toottext` which contains the raw toot content, ready for text/NLP analysis, as opposed to the raw HTML content of `toot.content`. In case the toot has been reblogged, the field `toot.rebloggedbyuser` contains a list of users that reblogged the toot.
`PyMstdnCollect` comprises the following three modules:

1. main library source code (src/);
2. ready to use scripts (user_scripts/);
3. testing suite (test/).

The main source code contains the library scripts for the interaction with a SQLite (Hipp 2020) database, collection for network interaction data (conversations, head toot relationships, user outbox, boosts and user followers), toot data collection (hashtags, public timeline toots), as well as an auxiliary library for supporting operations in storing toots and constructing global identifiers for toots and users.
The provided user scripts use the library methods to collect public timelines based on hashtags or keywords, the complete public timeline at specified time intervals and user interactions. They may be used with schedulers of the operating system (e.g. cron on Linux) to automate the data collection. An indicative pipeline for the data collection for a set $\mathcal{T}$ of topics is illustrated in Algorithm 1.

Listing 1: Toot structure

```
toot: struct
 |– globalID: string
 |– toottext: string
 |– rebloggedbyuser: array
 |– account: struct
 |– application: struct
 |– bookmarked: boolean
 |– card: struct
 |– content: string
 |– created_at: string
 |– edited_at: string
 |– emojis: array
 |– favourited: boolean
 |– favourites_count: long
 |– filtered: array
 |– id: string
 |– in_reply_to_account_id: string
 |– in_reply_to_id: string
 |– instance_name: string
 |– language: string
 |– local_only: boolean
 |– media_attachments: array
 |– mentions: array
 |– muted: boolean
 |– poll: struct
 |– reblog: string
 |– reblogged: boolean
 |– reblogs_count: long
 |– replies_count: long
 |– sensitive: boolean
 |– spoiler_clean_text: string
 |– spoiler_text: string
 |– tags: array
 |– uri: string
 |– url: string
 |– visibility: string
```

Listing 2: User account structure

```
account: struct
  |– globalID: string
  |– account_note_text: string
  |– acct: string
  |– avatar: string
  |– avatar_static: string
  |– bot: boolean
  |– created_at: string
  |– discoverable: boolean
  |– display_name: string
  |– emojis: array
  |– fields: array
  |– followers_count: long
  |– following_count: long
  |– group: boolean
  |– header: string
  |– header_static: string
  |– hide_collections: boolean
  |– id: string
  |– indexable: boolean
  |– last_status_at: string
  |– limited: boolean
  |– locked: boolean
  |– moved: struct
  |– noindex: boolean
  |– note: string
  |– roles: array
  |– statuses_count: long
  |– uri: string
  |– url: string
  |– username: string
```

Figure 3: The user and toot structures of Mastodon, with the added fields of `PyMstdnCollect`.

Algorithm 1: Data collection process on topics $\mathcal{T}$.

```
while true do
    day ← 0;
    if end-of-day then
        collect public timeline for last 24h;
        for topic in topics-of-interest do
            collect hashtags from all newly collected toots if their content
             contains words pertinent to topic T ∈ 𝒯
        end
        day ← day + 1;
    end
    // weekly conversation and user outbox collection
    if day == 6 then
        // toots
        collect conversations of messages that were 'heads', i.e. conversation
         starters, and contained hashtags or topic words of interest;
        update toots with links to their parent message id if they were a reply
         in a conversation (fields: in reply to id, in reply to account id);
        extract status boosts for messages in the conversation and update
         corresponding toot entries;
        // users
        determine 'critical'/most active users: extract the posting activity of
         users over the past week and extract those that fall in the 95th (or 98th
         if returned users are too many) percentile of the activity (number of
         overall statuses);
        store critical users of the past week;
        extract the 'outbox' of the critical users;
        update toots with outbox statuses of critical users, and conversations if
         a status is a head - only if the outbox status falls into last 1 week;
        extract status boosts ('reblogs') for messages of critical users and
         update corresponding toot entries - only if the status falls into last 1
         week;
        // hashtags
        for topic in topics-of-interest do
            update topic's hashtag lists: extract top used hashtags of the past
             week for the topic and append to existing hashtag list to be used in
             the following week;
        end
        day ← 0;
    end
end
```

Example

An example run of a user script is show in the current section and more may be found on the GitHub repository of the package. The script collects all public timeline toots from mastodon.social instance of the last 24h hours and stores them

in an SQLite databse:

```
with open("/home/ubuntu/pymstdncollect/authorisations/auth_dict.json", "r") as f:
    auth_dict = json.load(f)

upper = datetime.now(timezone.utc)
max_id_snowflake = datetime2snowflake(upperend)
lower = upper - timedelta(days=1)
min_id_snowflake = datetime2snowflake(lower)
DIR_out = "/tmp/dailycollects/"

database = "{}/toots_db_{}_{}.db".format(DIR_out,
             lower.strftime("%Y-%m-%dT%H:%M:%S"),
             upper.strftime("%Y-%m-%dT%H:%M:%S"))

sql_create_toots_table = """
            CREATE TABLE IF NOT EXISTS toots (
                    globalID text PRIMARY KEY,
                    id text NOT NULL,
                    accountglobalID text SECONDARY KEY,
                    account text NOT NULL,
                    created_at text NOT NULL,
                    in_reply_to_id text,
                    in_reply_to_account_id text,
                    sensitive boolean NOT NULL,
                    spoiler_text text,
                    spoiler_clean_text text,
                    visibility text NOT NULL,
                    language text,
                    uri text NOT NULL,
                    url text NOT NULL,
                    replies_count integer,
                    reblogs_count integer,
                    favourites_count integer,
                    edited_at text,
                    content text,
                    reblog boolean,
                    rebloggedbyuser text,
                    media_attachments text,
                    mentions text,
                    tags text,
                    emojis text,
                    card text,
                    poll text,
                    instance_name text NOT NULL,
                    toottext text,
                    muted boolean,
                    reblogged boolean,
                    favourited boolean,
                    UNIQUE(globalID, accountglobalID)
                ); """
dbconnection = connectTo_weekly_toots_db(database)
execute_create_sql(dbconnection, sql_create_toots_table)

apibaseurl = "https://mastodon.social/api/v1/timelines/public"
```

```
res = collect_timeline_apidirect(dbconnection=dbconnection,
              url=apibaseurl, max_id=max_id_snowflake, since_id=min_id_snowflake,
              savedir=DIR_out,
              instance_name=server,
              auth_dict=auth_dict,
              cutoff_date=lower.strftime("%Y-%m-%d"))
logging.FileHandler("{}/logging/logging_{}.txt".format(DIR_out,
               datetime.now().astimezone(pytz.utc).strftime("%Y-%m-%d"))).close()
```

Quality control
The requirements for installing and running the package and testing suite are included in auxiliary files pyproject.toml, requirements.txt, setup.cfg and tox.ini. The tests run each of the main functions separately and they can be performed by running the command:

pytest

from the root directory (full tests last $\sim$ 10 minutes). Optionally, the flag '-s' may be used to print test outputs (quite long) to the standard output. The output of the command will be a coverage report in the *htmlcov* directory. To view the report the user may execute the following command:

cd htmlcov python -m http.server,

and visit the localhost link (`http://localhost:8000/`) in a browser. For testing in a clean environment for several python versions (3.8-3.10), the user may employ Tox by running

tox

in the root directory. Note that this takes significantly longer to run, so it is best to perform it as a last check. Tox is a more general testing tool that allows the user (and the developer) to check that the package builds and installs correctly under different environments (e.g. Python implementations, versions or installation dependencies), to run tests in each of the environments with the test tool of choice, and also to act as a frontend to continuous integration servers. For example, using tox, whenever the code is pushed to the remote repository, the test suite is automatically run using GitHub Actions (see repository file *.github/workflows/tests.yml* for further details on this process). The output of the automated tox testing report may be found in the GitHub Actions tab on GitHub.

(2) Availability

Operating system
This package can be run on any operating system where Python can be run (GNU/Linux, Mac OSX, Windows).

Programming language
`Python 3.8`

Additional system requirements
None.

Dependencies
The full dependencies list (packages and versions) is included in the requirements.txt file in the GitHub repository.

1 Software location:

Archive

Name: PyMstdnCollect
Persistent identifier: 10.5281/zenodo.13119144
Licence: MIT
Publisher: Ioannis Chalkiadakis
Version published: v0.0.7
Date published: 28/07/24

Code repository GitHub

Name: PyMstdnCollect
Persistent identifier: https://github.com/ichalkiad/PyMstdnCollect
Licence: MIT
Date published: 28/07/24

Language
English.

(3) Reuse potential

`PyMstdnCollect` was developed to support research outputs of the NODES project and a number of networks and social science publications using data collected with the present library are currently in the process of submission. The library is intended for social science, network theory, political science and natural language processing researchers who want to have access to social media data, particularly those who aim to study newly developed social networks. Decentralised networks such as Mastodon offer a promising venue for online discourse where phenomena such as polarisation, toxicity, mis- and disinformation are limited or absent, due to the lack of a centralised managing authority and profit-driven recommendation algorithms. By assisting with the collection of digital trace data from Mastodon, `PyMstdnCollect` aims to support researchers in conducting research that will empower the users of such networks as well as instance moderators and policy-makers to develop observatories of social phenomena and the user communities that form on this new online debate space. Note, however, that any data collected with `PyMstdnCollect` must be used according to the rules of each Mastodon instance they originate from and the ethics of the broader Mastodon and Fediverse communities. It remains the sole responsibility of the user of the present software to ensure that their research and use of data is fully compliant with Mastodon instance rules and user privacy choices. We refer the library users to recent research that provides recommendations on how to best approach research with Mastodon (Roscam Abbing & Gehl 2024) in a way that is beneficial to both the researchers and the Mastodon user community.
`PyMstdnCollect` is released under the MIT license and any contributions are welcome as well as any recommendations for improvements - these may be submitted as Issues on the GutHub repository, or by emailing the authors.

List of contributors
Ioannis Chalkiadakis developed and tested the software, and wrote the paper. David Chavalarias provided guidance on the choice of Mastodon as the platform for research and gave advice on its decentralised operation.

Funding statement
The development of `PyMstdnCollect` was supported with funding from the NODES project of the European Commission under grant agreement CNECT/2022/5162608.

Competing interests
The authors declare that they have no competing interests.

References
Diener, L. & contributors (2016), 'Mastodon.py GitHub repository'.
URL: *https://github.com/halcy/Mastodon.py/*

Eden, T. (2022*a*), 'Getting Started with Mastodon's Conversations API'.
URL: *https://shkspr.mobi/blog/2022/11/getting-started-with-mastodons-conversations-api/*

Eden, T. (2022*b*), 'Snowflake IDs in Mastodon (and Unique IDs in the Fediverse more generally)'.
URL: *https://shkspr.mobi/blog/2022/12/snowflake-ids-in-mastodon-and-unique-ids-in-the-fediverse-more-generally/*

Gehl, R. W. & Zulli, D. (2023), 'The digital covenant: non-centralized platform governance on the mastodon social network', *Information, Communication & Society* **26**(16), 3275–3291.
URL: *https://doi.org/10.1080/1369118X.2022.2147400*

Gruzd, A. & Mai, P. (2020), 'Communalytic: A computational social science research tool for studying online communities and discourse'.
URL: *https://communalytic.org/*

Hipp, R. D. (2020), 'SQLite'.
URL: *https://www.sqlite.org/index.html*

Issues - Mastodon GitHub repository (2022*a*), 'Can't see toots on people's profiles or replies to many toots from self-hosted instance'.
URL: *https://github.com/mastodon/mastodon/issues/14017*

Issues - Mastodon GitHub repository (2022*b*), 'When opening a post, show all replies'.
URL: *https://github.com/mastodon/mastodon/issues/22674*

La Cava, L., Greco, S. & Tagarelli, A. (2021), 'Understanding the growth of the fediverse through the lens of mastodon', *Applied network science* **6**, 1–35.

La Cava, L., Greco, S. & Tagarelli, A. (2022), 'Information consumption and boundary spanning in decentralized online social networks: the case of mastodon users', *Online Social Networks and Media* **30**, 100220.

Roscam Abbing, R. & Gehl, R. W. (2024), 'Shifting your research from x to mastodon? here's what you need to know', *Patterns* **5**(1), 100914.
URL: *https://www.sciencedirect.com/science/article/pii/S2666389923003239*

Schatto-Eckrodt, T. (2023), 'mastodon-toolbox GitHub repository'.
URL: *https://github.com/Kudusch/mastodon_toolbox*

Schoch, D. & hong Chan, C. (2023), 'Software presentation: Rtoot: Collecting and analyzing mastodon data', *Mobile Media & Communication* **11**(3), 575–578.
URL: *https://doi.org/10.1177/20501579231176678*