Will Admins Cope? Decentralized Moderation in the Fediverse

Ishaku Hassan Anaobi
Queen Mary University of London

Aravindh Raman
Telefonica Research

Ignacio Castro
Queen Mary University of London

Haris Bin Zia
Queen Mary University of London

Damilola Ibosiola
Queen Mary University of London

Gareth Tyson
Hong Kong University of Science and Technology (GZ)

ABSTRACT

As an alternative to Twitter and other centralized social networks, the Fediverse is growing in popularity. The recent, and polemical, takeover of Twitter by Elon Musk has exacerbated this trend. The Fediverse includes a growing number of decentralized social networks, such as Pleroma or Mastodon, that share the same subscription protocol (ActivityPub). Each of these decentralized social networks is composed of independent instances that are run by different administrators. Users, however, can interact with other users across the Fediverse regardless of the instance they are signed up to. The growing user base of the Fediverse creates key challenges for the administrators, who may experience a growing burden. In this paper, we explore how large that overhead is, and whether there are solutions to alleviate the burden. We study the overhead of moderation on the administrators. We observe a diversity of administrator strategies, with evidence that administrators on larger instances struggle to find sufficient resources. We then propose a tool, WatchGen, to semi-automate the process.

CCS CONCEPTS

• **Networks** → **Network measurement**; • **Information systems** → **World Wide Web**.

KEYWORDS

Fediverse, Content Moderation, Federation policies, Decentralized Web, Decentralized Social Networks

ACM Reference Format:
Ishaku Hassan Anaobi, Aravindh Raman, Ignacio Castro, Haris Bin Zia, Damilola Ibosiola, and Gareth Tyson. 2023. Will Admins Cope? Decentralized Moderation in the Fediverse. In *Proceedings of the ACM Web Conference 2023 (WWW '23), May 1–5, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3543507.3583487

1 INTRODUCTION

The Fediverse encompasses a group of increasingly popular platforms and technologies that seek to provide greater transparency and openness on the web. [13, 18, 30, 34]. Well known Fediverse platforms include microblogging services (*e.g.* Pleroma [38], Mastodon [33]) and video sharing platforms (*e.g.* PeerTube [37]). The acquisition of Twitter by Elon Musk [11] has exacerbated this popularity with a large migration of Twitter users to the Fediverse [8].

In Fediverse social networks, individuals or organisations can install, own, and manage their own independent servers, also known as **instances** [15, 54]. For these instances to interact, they rely on **federation** [41], whereby instances interconnect in a peer-to-peer fashion to exchange posts. Note that this allows for users to exchange content across platforms. This results in a physically decentralized model that is logically interconnected where users can interact globally. Unfortunately, this creates challenges for instance **administrators**, as activities on one instance impact others via federation. For example, recent work has shown that hateful material generated on one instance can rapidly spread to others [53].

To overcome this, most Fediverse social network implementations have in-built **federation policies**. These policies enable administrators to create rules to ban or modify content from instances that matches certain rules, *e.g.* banning content from a particular instance or associating it with warning tags. Although a powerful tool, this imposes an additional overhead on administrators [6, 14, 26]. Thus, we argue it is vital to better understand this process, and propose ways to improve it.

This paper examines administrator activities in the Fediverse. We focus on Pleroma, a federated microblogging platform with similar functionality to Twitter. We collect a large-scale dataset covering 10 months: this includes 1,740 instances, 133.8k users, 29.9m posts, associated metadata, and importantly, the policies setup by the administrators. We find that instances are often "understaffed", with the majority of instances only having a single administrator, and recruiting no other moderators to assist, despite many having over 100K posts. This leads us to conjecture that some administrators may be overwhelmed. Indeed, we find that instance administrators often take many months before applying policies against other instances, even in cases where they exhibit clearly controversial traits (*e.g.* posting a large number of hate words).

We therefore turn our attention to the policy configurations employed. We observe a growing number of instances enacting a wide range of policy types. Common are 'maintenance' policies, such as those which automatically delete older posts (`ObjectAgePolicy`), as well as those aimed at preventing the spread of certain content (*e.g.* `HashtagPolicy`, which flags up posts with certain hashtags). We further observe a range of bespoke policies created by administrators, via the `SimplePolicy`, which can be configured to trigger a range of actions based on certain rules (*e.g.* blocking all connections

WWW '23, May 1–5, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04…$15.00
https://doi.org/10.1145/3543507.3583487

from certain instances). The laborious nature of this moderation work leads us to explore automated techniques to assist administrators. We build a set of models to predict administrator actions. We embed them in WatchGen, a tool that can propose a set of instances for administrators to focus their moderation efforts on. To the best of our knowledge, this is the first study of Fediverse administrators. We make the following observations:

(1) We find a diverse range of 49 policies used by administrators, capable of performing various management and moderation tasks. Despite this, we see that 66.9% of instances are still running, exclusively, on the default policies alone (Section 4).
(2) The number of administrators does not grow proportionately with the number of posts (Section 5). This seems to impact moderation. For example, it takes an average of 82.3 days for an administrator to impose a policy against an instance after it first encounters it, even for well-known and highly controversial ones (*e.g.* `gab.com` [5]).
(3) Intuitive features, such as the number of mentions and frequent use of hate words, are good indicators that an instance will later have a policy applied against it (Section 6). This suggests that there are key traits that garner more attention by moderators.
(4) We show that it is possible to predict (F1=0.77) which instances will have policies applied against them (Section 6) and design *WatchGen*, a tool that flags particular instances for administrators to pay special attention to.

2 PLEROMA: OVERVIEW

Pleroma is a lightweight decentralized microblogging server implementation with user-facing functionality similar to that of Twitter. In contrast to a centralized social network, Pleroma is a federation of multiple independently operated servers (aka instances). Users can register accounts on these instances and share posts with other users on the same instance, or on different instances. Through these instances, users are able to register accounts and share posts (called statuses) to other users on the same instance, other Pleroma instances, or instances from other Fediverse platforms, most notably Mastodon.

Federation. We refer to users registered on the same instance as **local**, and users on different instances as **remote**. A user on one instance can follow another user on a separate instance. Note that a user registered on their local instance does not need to register with the remote instance to follow the remote user. When the user wants to follow a user on a remote instance, the local instance subscribes to the remote user on behalf of the local user using an underlying subscription protocol (ActivityPub [2]). This process of peering between instances in the Fediverse is referred to as **federation**.

The federated network includes instances from Pleroma and other platforms (*e.g.* Mastodon) that support the same subscription protocol (ActivityPub). Accordingly, Pleroma instances can federate and target their policies at non-Pleroma instances. The resulting network of federated instances is referred to as the *Fediverse* (with over 23k servers [16]).

Policies. Policies affect how instances federate with each other through different rule-action pairs. These allow certain actions to be executed when a post, user, or instance matches pre-specified criteria. For example, the `SimplePolicy` can perform a range of actions when a remote instance matches certain criteria such as `rejecting` connections. Note, there are numerous in-built policies, but tech-savvy administrators can also write their own bespoke policies.

Administrators. Instances are hosted and managed by specialized users called administrators. By default, the creator of an instance will take on the role of the administrator, however, it is also possible to delegate such responsibilities to multiple others. Instance administrators are responsible for carrying out the day-to-day administrative tasks on the instances. These include managing the front-end, users, uploads, database, emoji packs and carrying out administrative email tasks. The instance administrator is also responsible for accepting new user registrations and removing users where necessary. The administrator updates and backs-up the instance, set the terms of service and retains the ability to shutdown the instance. One essential responsibility of the instance administrator is the moderation of content (although they can also assign the role to other users called *moderators*). This can make instance administration a cumbersome task, and administrators a very important part of the Fediverse.

3 DATA COLLECTION

Instance & Administrator Dataset. Our measurement campaign covers 16th Dec 2020 – 19th Oct 2021. We first compile a list of Pleroma instances by crawling the directory of instances from distsn.org and the-federation.info. We then capture the list of instances that each Pleroma instance has ever federated with using each instance's Peers API.[1] Note, this includes both Pleroma and non-Pleroma instances. In total, we identify 9,981 instances, out of which 2,407 are Pleroma and the remainder are non-Pleroma (*e.g.* Mastodon).

We then collect metadata for each Pleroma instance every 4 hours via their public API.[2] We record the list of administrators and any delegated moderators. We also obtain the number of users on the instance, the number of posts, the enabled policies, the applied policies as well as the instances targeted by these policies, and other meta information.

From the 2,407 Pleroma instances, we are able to gather data from a total of 1,740 instances (72.28%). For the remaining 667 instances: 65.1% have non existent domains, 17.9% are not found (404 status code), 6.4% instances has private timelines (403), 4.5% result in Bad Gateway (502), 1.3% in Service Unavailable (503), and under 1% return Gone (410).

User Timelines. Users in Pleroma have three timelines: (*i*) a *home* timeline, with posts published by the accounts that the user follows (local and remote); (*ii*) a *public* timeline, with all the posts generated within the local instance; and (*iii*) the *whole known network*, with *all* posts that have been retrieved from remote instances that the local users follow. Note, the *whole known network* is not limited to remote posts that a particular user follows: it is the union of remote posts retrieved by all users on the instance. We use the public Timeline

[1] <instance.uri>/api/v1/instance/peers

[2] <instance.uri>/api/v1/instance/

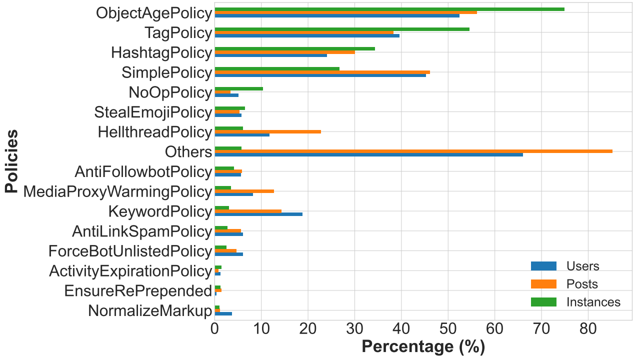


Figure 1: The top 15 policies and percentage of instances that use each policy (sorted by the percentage of instances).

API[3] to gather posts data from 819 instances (the remaining 912 instances have either no posts or unreachable public timelines).

Ethics. Our dataset covers Pleroma instances and their administrators. We exclusively focus on the policies that these administrators set, and do not investigate other aspects of administrator behavior (*e.g.* the posts they share). All data is available via public APIs. We emphasize that administrators, themselves, are the ones who control access to these APIs. Hence, the administrators covered in this paper consent for others to use this data. Further, the policies studied do not work on a per-user granularity and, thus, we cannot infer anything about individual users. All data is anonymized before usage, and it is stored within a secure silo.

4 EXPLORING POLICY CONFIGURATIONS

Policy Footprint. We first quantify the presence of policies across instances. In total, we observe 49 unique policy types. From our 1.74k Pleroma instances, we retrieve policy information from 93.2% of instances (the remainder do not expose their policies). These cover 94.2% of the total users and 94.5% of all posts. Figure 1 shows the distribution of the top 15 policy types enabled by the administrators across instances and the percentage of users signed up within those instances as well as the posts on the instances. We see a wide range of policies with diverse functionalities and varying coverage based on which metric is considered. For instance, whereas the `ObjectAgePolicy` (which performs an action on a post once it reaches a certain age) is installed on 74.8% of instances, this only covers 52.4% of the users. In contrast, the `KeyWordPolicy` (which performs an action on any posts containing a given keyword) covers 18.8% of users, but just 3% instances. Critically, there is a highly uneven distribution of policies, with the the top-5 covering 92.3% of all instances, 73.6% of users and 88.8% of the posts.

Default Policies. Default policies come auto-enabled with new installations. Prior to version 2.3.0 in March, 2021, only the `ObjectAgepolicy and NoOpPolicy` are enabled by default. Since version 2.3.0, the `TagPolicy` and `HashtagPolicy` are also enabled with a new installation (or upgrade). 66.9% of instances only have these default policies running. Relying solely on default policies may indicate several things. For example, administrators maybe unaware of management and moderation functionalities, unable to use them or simply not have sufficient time. Alternatively, they may actively choose not to use them.

Note, while the `TagPolicy` allows tagging user posts as sensitive (default: nsfw), the `Hashtagpolicy` allows the tagging of hashtags (*e.g.* nsfw sensitive). We find 54.6% and 34.3% of instances enabling these policies respectively. The other Pleroma default policy is the `NoOpPolicy`. This allows any content to be imported. This describes the default state of a new instance. Interestingly, we see administrators paying more attention to this policy: 89.7% of the instances have actively disabled it.[4] This suggests that administrators are aware and concerned about importing undesirable content.

Non-Default Policies. Non-default policies are those that instance administrators have to actively enable. Instances with these policies may indicate a more proactive administrator. We find 45 non-default policies during our data collection period.

The most powerful policy available is the `SimplePolicy`, enabled on 28.8% of instances. This policy allows administrators to apply a wide range of actions against specific instances (*e.g.* `gab.com`). The most impactful and common is the `reject` action.[5] 56.9% of instance that enable the `SimplePolicy` employ the *reject* action. Interestingly, although we see only 28.8% of instances with the `SimplePolicy` enabled, its application affects 85.4% of users and 90.3% of the posts on the Pleroma platform. We see noteworthy instances being amongst the top targets of this policy (*e.g. kiwifarms.cc and anime.website*), which are all commonly understood to share controversial material. Interestingly, only 18.5% of instances with the `SimplePolicy` applied against them are from the Pleroma platform (the most are from Mastodon [39]). This means that 81.5% of the recipients are from federated instances outside of Pleroma.

Policy Growth. We next look at how the use of policies has changed over time. We conjecture that the longer administrators run their instances, the more experienced they become. As such, we expect to see greater application of policies. Here we focus on the 5 most popular policies as they account for 92.3% of the instances, 73.6% of users and 88.8% of the posts. For completeness, we include the sum of the other less popular policies too. Figure 2 presents the percentage of instances that activate each policy over time. Across our measurement period, we observe a growth of 40% in the total number of policies used. This suggests that the use of policies is becoming more common. 28.5% of these policies are introduced by new instances coming online, with newly installed default policies, *e.g.* `ObjectAgepolicy`, `TagPolicy` and `HashtagPolicy`. The remainder are instantiated by pre-existing instance administrators that update their policies, suggesting a relatively active subset of administrators.

We also inspect the growth on individual instances. Overall, 42% of instances add policies during our measurement period. Of these instances, 52.3% enable only one extra policy and we see only a small minority (1.9%) enabling in excess of 5 new policies (*e.g.* `chaos.is` enables 13 and `poa.st` 12). A closer look at these instances show they mostly add common policies. However, we also see a wide range of other less common policies (*e.g.* `KeywordPolicy`).

[3] <instance.uri>/api/v1/timelines/public?local=true

[4] Note, this is overridden if a user enabled any other policy.

[5] This blocks all connections from a given instance

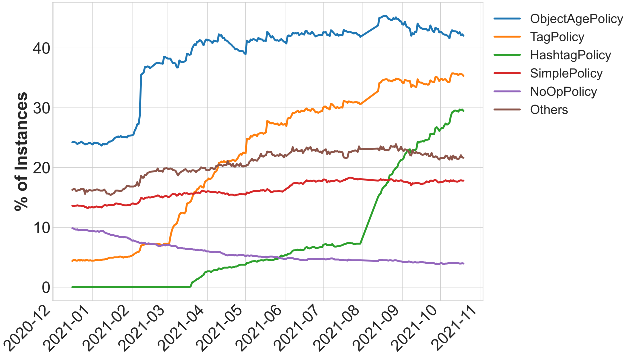


Figure 2: Time series showing the percentage of instances (Y-axis) that use the 5 most popular Pleroma policies. We include the sum of all the remaining policies as "Others".

In contrast, the use of the SimplePolicy, with the most flexible range of moderation actions, has remained relatively stable. Actions under the SimplePolicy have instance-wide effect and can effectively control instance federation. Overall, we only see 28.8% of instances enabling this policy, without much growth across the measurement period (as seen in Figure 2). This could imply that administrators are unaware of this policy, do not have time to moderate their instances at this level or maybe find this policy too blunt (not fine-grained enough). The latter could lead to other issues, which administrators seek to avoid (*e.g.* collateral damage [22]). It is also worth noting that the SimplePolicy is one of the most complex, and administrators potentially shy away from these more labour-intensive policies. We argue that the diversity of policies could potentially overwhelm (volunteer) instance administrators (see Section 5). This suggest that they require further support to automate this process (see Section 6).

5 CHARACTERISING ADMINISTRATORS

5.1 Distribution of Administrators

Number of Administrators Per-Instance. We observe a total of 2,111 unique administrators from 1,633 instances (93.8% of 1.74k).[6] Figure 3 presents the distribution of the number of administrators per instance. Although a majority of instances (71.6%) are managed by a single administrator, we also see some instances with a larger number of administrators (*e.g.* rakket.app: 16 and poa.st: 13).

Administrator Workload. We next test if the number of administrators increases proportionately to the number of posts. We treat this as a rudimentary proxy for how much moderation must take place on an instance. Figure 4 presents the distribution of posts on instances *vs.* the number of administrators. Generally, we find that instances with more posts do have more administrators on average, *e.g.* instances with multiple administrators have more posts, with a ratio of 6:1. However, this is driven by a few instances (*e.g.* poa.st).

Table 3 summarizes the top 10 instances that see the largest growth in administrators. Many of them are small instances with under 1000 users, and a proportionately small number of posts. This suggests that administrator growth does not necessarily occur on the instances that need it the most. To test if the number of administrators grow proportionately to the number of posts, Figure 5 plots the *growth* of administrators *vs.* the growth of posts on each individual instance during our data collection period. We see that a growth in posts on a given instance does not necessarily correspond to the recruitment of new administrators. In fact, only 6.9% of instances record a growth in administrators during this period. Overall, there is a weak correlation (Spearman coefficient of 0.19 for the number of posts *vs.* number of administrators). In total, we see a 60.3% increase in the number of posts, but just a 35.6% growth in administrators. Unsurprisingly, instances that grow their administrator pool *do* become more active. On average, instances with a growing number of administrators have 1.5x more policies than other instances. Specifically, looking at the policy with the most impact (reject), these instances apply it 1.8x more than others. Interestingly, instances with an increasing number of administrators also have 4x more policies applied against them.

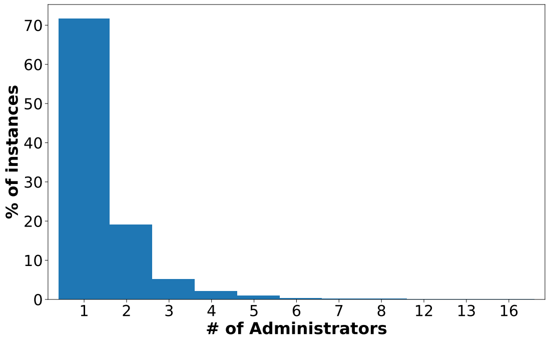


Figure 3: Instances (%) by number of administrators.

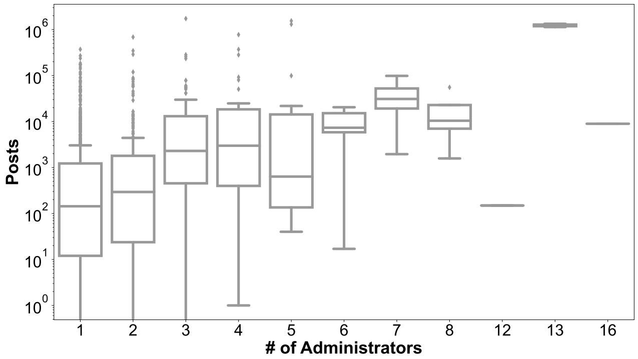


Figure 4: Box plot of the number of posts per instances with different number of administrators.

5.2 Administrators' Response Lag

The previous section has shown that administrators face a growing moderation workload. To study this workload, we now look at how long it takes administrators to apply polices against particular

[6] The remaining instances do not publish their administrator(s) information.

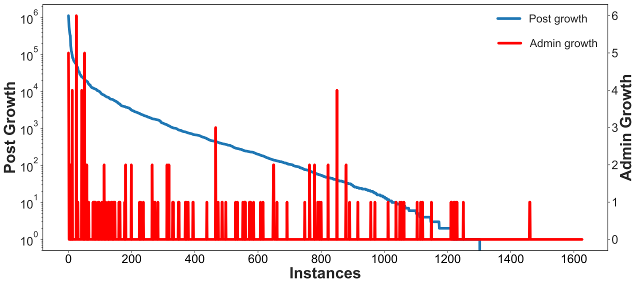


Figure 5: Per instance growth in the number of administrators (Y2-axis) and posts (Y1-axis). Individual instances are on the X-axis, sorted by the number of posts.

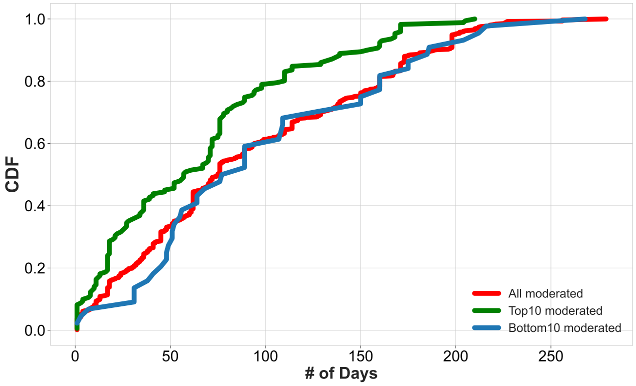


Figure 6: CDF showing the distribution of days from federation to moderation for all moderated instances. We also show results for the top 10 and bottom 10 instances, based on the number of policies applied against them.

instances. We focus on the SimplePolicy as this is clearly geared towards moderation, has instance-wide targeting, and lists the target instances. For each SimplePolicy against a given instance, we compute the lag between the date of the implementation of the policy and the date when the targeted instance was first federated with. This is a rudimentary proxy for how long it took an administrator to identify the problem. We temper our analysis with the fact that there could be many reasons for this delay, which we have limited vantage on.

Policy Creation Delay. Figure 6 presents the distribution of delays (as defined above). Note, we exclude the 55% of federations that occurred before the beginning of our data collection (as we cannot know their timestamp). We plot the delay distributions for applying policies against: (*i*) All instances; (*ii*) "Controversial" instances with the most policies applied against them (top 10); and (*iii*) "Benign" instances with the fewest policies against them (bottom 10).

It takes administrators an average of 82.3 days to apply any form of policy against other instances. Although, on average, it takes more time for a policy to be applied on the "bottom 10" instances than the "top 10" instances (74.7 and 59.5 days respectively), we see that there is a noticeable lag (almost 3 months) between federation occurring and policies being imposed. This may suggest that administrators find it difficult to keep-up with the need to rapidly identify instances that justify policy imposition.

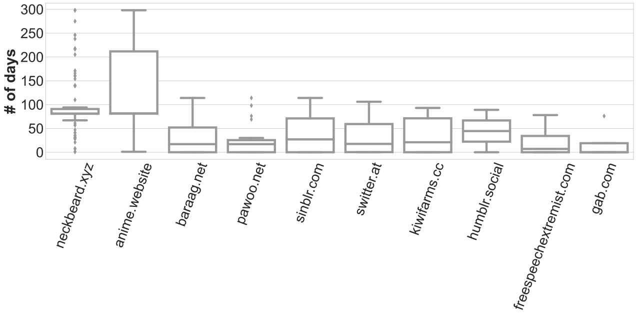


Figure 7: Box plot showing the distribution of the number of days from federation to the imposition of policies for the top 10 instances with the most policies applied against them.

Delay for Controversial Instances. We next extract the top 10 instances that receive the most policies targeted against them. For each one, Figure 7 plots the distribution of delays (*i.e.* how long it takes other instances to impose a policy against them). In-line with expectations, we see that administrators take less time to apply policies against instances like gab.com, known for its right-wing stance (average of 19 days). However, we see much longer delays for other controversial instances that are less well-known (*e.g.* neckbeard.xyz), averaging up to 98.4 days. These instances are quite active, with significant growth in posts during our measurement period (*e.g.* neckbeard.xyz: 789.4k and kiwifarms.cc: 469.2k). With other instances such as anime.website posting "lolicon" (suggestive art depicting prepubescent females), it is expected that policies would be swift, however, we see a very wide breadth of delays. The diverse nature of these administrator reactions indicates that any future automated moderation tools should be specialized to the preferences of individual administrators.

5.3 Administrators & Moderators

Moderation Delegation. As administrators are responsible for a wide range of activities, they can delegate the task of content moderation to select individuals. These accounts are referred to as **moderators**. Of our 1.74k instances, 47% of them (819) expose information in our dataset. From these, only 12% (98) of instances have assigned the role of moderator to any other accounts. Of these, 73.5% (72) of the instances have the administrator also doubling as a moderator, while 29.6% (29) of the instances assign the entire moderator role to an account that is not the administrator. This implies that only 3.5% of instances have dedicated account(s) assigned the role of moderator.

Are moderators helpful? We conjecture that instances with dedicated moderators outside of their administrator team might be swifter in the application of policies. Figure 8 shows the percentage of instances that enable the 15 most popular policies (Figure 1). We

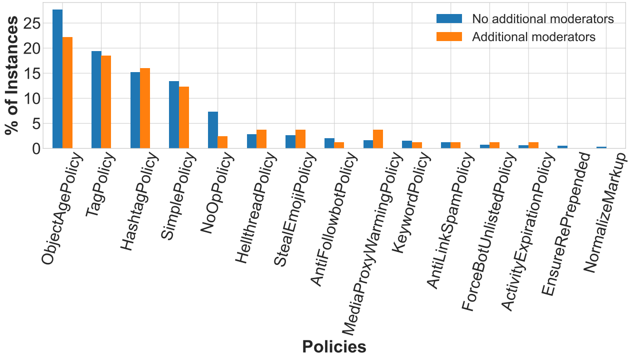


Figure 8: The percentage of instances that enable the top 15 most popular policies. We separate instances into two groups: (*i*) Instances without additional moderators; and (*ii*) instances with additional moderators outside of the administrator set.

present two bars for each policy: (*i*) Instances with additional moderators (who are not an administrator); and (*ii*) Instances without additional moderators. There is a broadly similar distribution across these two groups. However, we notice that instances without additional moderators have approximately 3x more of the `NoOpPolicy` configured. Recall, this is the default state of an instance and allows any content to be imported. This begins to suggest that instances with additional moderators do pay greater attention to policies.

We expand this analysis in Figure 9, where we show the number of `SimplePolicy` actions and the delay to apply a policy after federation (in days) for instances in the two groups. We use the `SimplePolicy` for this analysis as it is the only moderation policy with instance-wide targeting and a list of targeted instance domains. The plot shows that instances with moderators take less time (average 103 days) to impose a `SimplePolicy` after federation, compared to instances without dedicated moderators (average 111 days). The figure also shows a marked difference in the number of instances that apply the `SimplePolicy`. Only 38% of the instances with dedicated moderators apply no `SimplePolicy` actions, compared to 70% for those without. This confirms that instances with additional moderators are more proactively moderated.

6 WATCHGEN: AUTOMATING MODERATION

Our results indicate that moderation is labor-intensive. We now explore techniques to assist administrators. We propose *WatchGen*,[7] a tool that recommends to administrators a "watchlist" of instances that may require federated moderation. This watchlist must be on a per-instance basis, as different administrators may have varying views on what is considered appropriate for the instance they manage. WatchGen, helps administrators to more proactively identify instances requiring attention with regards to content moderation. We build WatchGen by compiling a large feature set for each instance, and experimenting with a number of classification models to flag instances that are more likely to require attention.

Feature Selection. We first extract features for each instance. These features include information about user (*e.g.* number of users) and administrator activities with respect to moderation (*e.g.* number of rejected instances). We also extract features from post content (*e.g.* number of hate words in posts). We experiment with a total of 38 features (see Table 5). Through extensive manual experimentation, we distil this down to the 16 most determinant features (highlighted in Table 5).

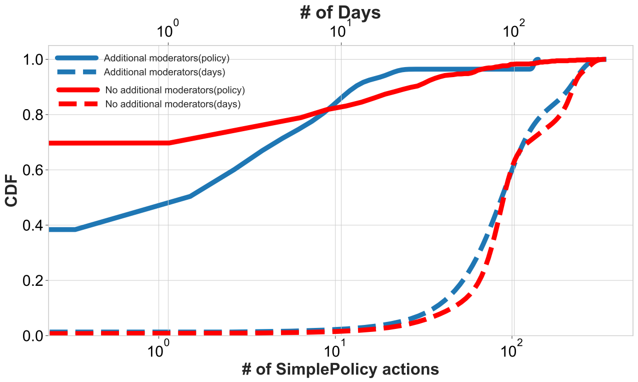


Figure 9: CDF of the number of `SimplePolicy` actions per instance (X1-axis) and the lag (in days) for instances to impose a policy after federation (X2-axis). We separate instances into (*i*) those with dedicated moderators; and (*ii*) those without dedicated moderators.

Model Training. Next, we train multiple machine learning models using the `sklearn` library, and GridSearchCV within 5-fold cross-validation to find the optimal hyper-parameter settings. We detail below the hyperparameters for each model.

Logistic Regression (LR). We only tune the C hyperparameter. This regularization parameter controls how closely the model fits to the training data. We test for best value of "C" using the values {0.001, 0.01, 0.1, 1, 10, 100, 1000}.

Multilayer Perceptron (MLP). We tune three hyperparameters: (*i*) hidden layer-size: dictates the number of hidden layers and nodes in each layer. We use a single hidden layer with varying hidden layer-sizes {10, 50, 100}; (*ii*) activation function: determines the type of non-linearity introduced into the model. We employ 3 activation functions {relu, tanh, logistic}; and (*iii*) learning rate: we tune how the initial learning rate parameter changes in finding the optimal model using {constant, invscaling, adaptive}.

Random Forest (RF). We tune 2 hyperparameters. (*i*) n_estimators: the number of independent trees (estimators). We test using 3 values {5, 50, 250}; and (*ii*) max_depth: the depth of the trees. We test for best result using 6 different depths {2, 4, 8, 16, 32, None}.

Gradient Boosted Trees (GB). We tune three hyperparameters. (*i*) n_estimators: The number of independent trees (estimators). We test with 4 value {5, 50, 250, 500}; (*ii*) max_depth: The depth of the trees. We test with 5 values {1, 3, 5, 7, 9}. (*iii*) Learning rate: This impacts the speed and granularity of the model training. We test 5 values {0.01, 0.1, 1, 10, 100}.

[7] https://github.com/anaobi/WatchGen.git

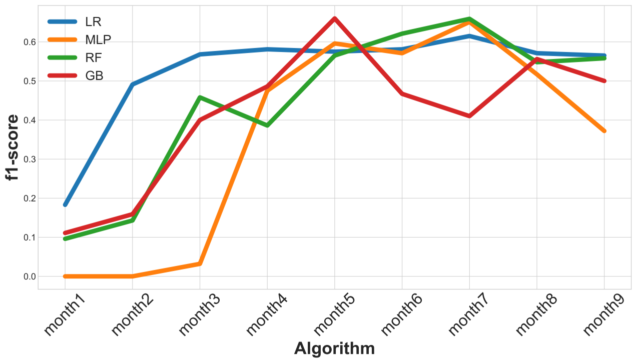


Figure 10: Time series of f1-scores for the Logistic Regression, Multi-Layer Perceptron, Random Forest and Gradient Boosted Trees models. Note that we exempt month 10 as this leaves insufficient test data.

6.1 Generating a Global Watchlist

Task. We first assume a WatchGen central broker that compiles a global pool of training data, collected from all instances through their public APIs (similar to us in Section 3). We use this global pool of training data, with an 80:20 split, to predict if a given instance will be subject to *any* policy (by any other instance). We then produce a 'watchlist' of instances that may be worthy of attention.

To investigate how long it would take to garner sufficient data to train WatchGen, we also train several models on datasets covering increasing time windows. We first train on one month of data and increase the training dataset by one month at a time (up to 9 months). For our test dataset, we use the data remaining after the training snapshot.

Results. Table 1 summarizes the result with the global pool of training data (80:20 split) with Random Forest being the best performing model (f1=0.77). Recall, that we also run experiments with a training set based on varying time windows. Figure 10 presents the f1 scores based on the size (duration) of the training set. We observe that it takes at least 5 months for a model to achieve its best score (*e.g.* Gradient Boosted Trees is month 5 and Random Forest in month 7). Note that the training sets are different from Table 1 and hence the scores differ.

Feature Importance. We next inspect which features are most important. This sheds insight into which characteristics are most related to triggering policies. We use the in-built functions for feature importance. Figure 11 presents the feature importance for the explainable models. We see that the top 3 features (transformed post, average number of mentions in a post, and number of posts on an instance) are all related to the number of posts on an instance. This suggests that the likelihood of an instance having a policy applied against it is closely related to the amount of content its users post. In other words, the more users and posts on an instance, the higher the probability of having a policy applied against it. This is expected as such instances are likely to attract more attention.

Features such as the number of mentions and hate words in the posts also play an important role. This is in-line with prior work that observed how mentions and quote retweets result in more attention [17]. To better understand the importance of these secondary metrics, we retrain the model without the two top features (number of posts and transformed posts). We show the results in Table 4. Confirming our prior assertion, we retain relatively good performance. For Random Forest, we attain an f1 of 0.62 (*vs.* 0.77 with the full feature set in Table 5). This confirms that these other factors play an important role in determining if an instance has a policy applied against it. In other words, in addition to the size of an instance, other features are required to obtain a fairly good prediction of instances being subject to any policy.

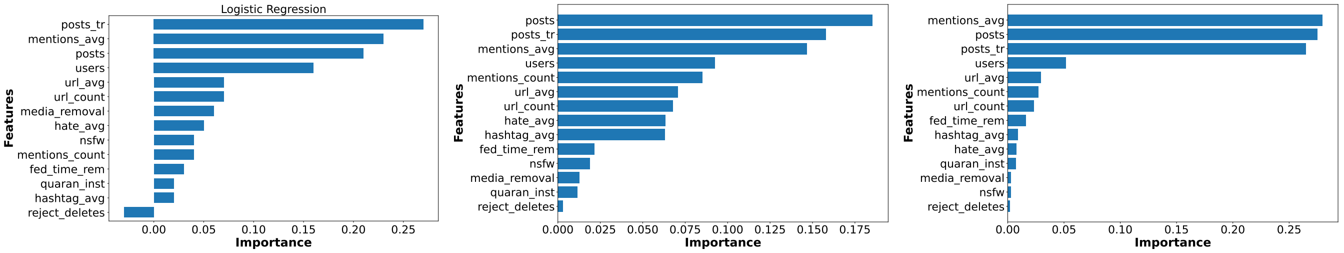
Algorithm	Acc.	Prec.	Recall	f1 score
Logistic Regression	0.86	0.85	0.34	0.49
Multi-Layer Perceptron	0.57	0.34	0.42	0.53
Random Forest	0.92	0.88	0.68	0.77
Gradient Boosted Trees	0.89	071	0.71	0.71

Table 1: WatchGen performance results when using global training pool and the full feature set.

6.2 Generating a Local Watchlist

Task. Our prior WatchGen models assume a central pool of training data, aggregated from all instances. This may be infeasible in practice due to the decentralized nature of the Fediverse. Hence, we next investigate how well our best model (Random Forest) performs when decentralizing the training process. For each instance, we extract its federated peers and exclusively build a local training set from their data (using the features highlighted in Table 5). For each pair of instances, we tag whether or not a directed policy is imposed, *i.e.* each instance only considers the policies it locally sees. Finally, each instance trains its own local model using the first 8 months of data (and tests on the last 2). This creates one independent model per-instance. Based on this, WatchGen predicts whether a policy will be applied against the instance.

Results. Figure 12 presents the distribution of performance metrics per-instance. As expected, we observe an overall performance drop compared to the prior task based on a global model. Instances attain an average f1 score of 0.55. This is largely due to the significant reduction in per-instance training data. That said, we observe a wide array of performances across the instances: 42.6% of instances achieve above 0.6 f1, with a tail of 8.3% attaining below 0.4 f1. We find that performance is impacted by the training set size. Instances that perform relatively well (>=0.6 f1), tend to be larger (*i.e.* more posts and users). For example, 65.4% of the best performing instances (>=0.6 f1) have a local post count of over 50k (*e.g.* `neckbeard.xyz` and `freespeechextremist.com`). In contrast, only 4.4% of instances that perform poorly (<0.6 f1) have over 50k posts (*e.g.* `princess.cat` and `sleepy.cafe`). This implies that as instances grow, their local performance will improve. The above experiments show that instances *can* use these locally trained models to generate a personalized watchlist of instances they peer with. Thus, we argue that these automatically compiled lists can helps administrator pay attention to these instances.



(a) Logistic Regression

(b) Random Forest

(c) Gradient Boosted Trees

Figure 11: Feature importance for our explainable models.

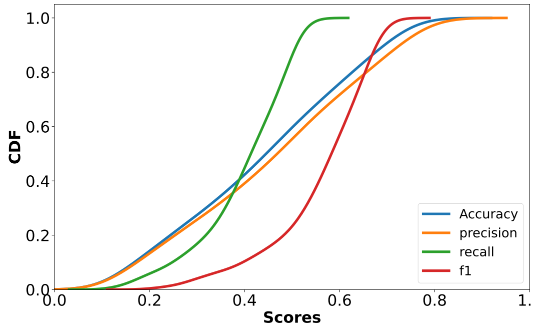


Figure 12: CDF of per-instance performance for Random Forest trained on data from local and federated instances.

7 RELATED WORK

Social Network Studies. Extensive work has been carried out in the area of online social networks. However, most of these are on centralized social networks (*e.g.* Facebook and Twitter) [3, 19, 28, 29, 31, 36]. A number of these look at the anatomy of social graphs [23] and moderation challenges [20]. Others look into areas ranging from the evolution of user activities to demographics [32, 45]. In contrast to Pleroma, these social networking platforms tend to rely on central (commercial) administrators and moderators [48].

Fediverse and Decentralized Web. Only a small set of studies have focused on the Fediverse or Decentralized Web applications. Raman *et al.* looked at the challenges in the Fediverse, with a particular focus on the infrastructure and resilience of Mastodon [39]. Trautwein *et al.* studied the Inter Planetary File System (IPFS), a decentralized storage solution [44]. Guidi *et al.* and Datta *et al.* studied the structure, data management, and privacy aspects of decentralized social networks [1, 7]. Recent works have examined the standardization of related protocols [27, 35]. Bielenberg *et al.* analyzed the growth, topology and server reliability of Diaspora (a decentralized social network) [4]. Similarly, Zignani *et al.* studied the evolution of the Mastodon social graph [55]. Our work differs in that we focus on exploring *administrator* actions within the Fediverse.

Online Moderation. Prior work has investigated the roles that volunteer moderators play in platforms like Twitch [51]. Text-based content classification and filtering has been extensively studied too. These include computational techniques to detect cyberbullying [10, 12, 49], anti-social posting [25, 42, 43, 52], and hate speech [9, 21, 24, 40, 46, 47, 50]. These models have proven effective in reducing the workload of human moderators. For example, Cheng et. al. [25] use random forest and logistic regression classifiers to predict whether a user will be banned, reducing the manual load on moderators. Similarly, Zia *et al.* [53] look at detecting the spread of toxic posts specifically in Pleroma (although not administrator reactions). In our prior work, we also studied the use of federation policies [22]. Here, we build on this, with a focus on the actions undertaken by administrators. We further propose WatchGen to assist administrators. To the best of our knowledge, this is the first large-scale study of administrator activities in the Fediverse. We hope that this can further contribute to the wider understanding of moderation in other platforms.

8 CONCLUSION AND DISCUSSION

We have studied instance administrators in a popular Fediverse platform, Pleroma. Although 66.9% of instances are still running on default policies, we observe an uptake of more sophisticated management functions. We find evidence that some administrators may become overwhelmed with the growing number of posts and users they must manage. For instance, it takes an average of 82.3 days for administrators to apply any policy against a newly federated instance. Another sign of the overhead is that just 3.5% of instances share the load across multiple moderators. This lack of moderators may come with challenges: instances with fewer moderators tend to employ less sophisticated policy strategies (*e.g.* 70% of them apply no `SimplePolicy` actions). To alleviate this, we have proposed WatchGen, a tool that identifies instances in need of closer attention. We show that WatchGen can predict which instances will later have a policy imposed (f1 = 0.77).

Our study opens up a number of lines of future work. First, we wish to expand our work to cover other Fediverse platforms, *e.g.* Mastodon or PeerTube. Second, we plan to experiment with alternate feature sets that can better identify instances that will later require policy attention. Through this we hope to improve WatchGen and pilot its deployment. Last, we want to perform a qualitative study to better understand the subjective opinions of administrators that underlie these trends. We conjecture that such qualitative insights might be invaluable for improving WatchGen.

ACKNOWLEDGEMENTS

This research was supported by EPSRC grants EP/S033564/1, EP/W032473/1, UKRI DSNmod (REPHRAIN EP/V011189/1), and EU Horizon Framework grant agreement 101093006 (TaRDIS).

REFERENCES

[1] Datta A, Buchegger S, Vu L-H, Strufe T, and Rzadca K. 2010. Decentralized online social networks. In: Furht B (ed) Handbook of social network technologies and applications. In *Springer*. 349–378.

[2] ActivityPub. 2018. https://www.w3.org/TR/activitypub/.

[3] Yong-Yeol Ahn, Seungyeop Han, Haewoon Kwak, Sue Moon, and Hawoong Jeong. 2007. Analysis of topological characteristics of huge online social networking services. In *In Proceedings of the 16th international conference on World Wide Web*. 835–844.

[4] Bielenberg Ames, Helm Lara, Gentilucci Anthony, Stefanescu Dan, and Zhang Honggang. 2012. The growth of Diaspora – A decentralized online social network in the wild. In *INFOCOM Workshops*.

[5] Naomi A Arnold, Benjamin Steer, Imane Hafnaoui, Hugo A Parada G, Raul J Mondragón, Félix Cuadrado, and Richard G Clegg. 2021. Moving with the Times: Investigating the Alt-Right Network Gab with Temporal Interaction Graphs. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (2021), 1–17.

[6] Rajadesingan Ashwin, Resnick Paul, and Budak Ceren. 2020. Quick, community-specific learning: How distinctive toxicity norms are maintained in political subreddits. In *Proceedings of the 14th International AAAI Conference on Web and Social Media, ICWSM 2020*. 557–568.

[7] Guidi B, Conti M, Passarella A, and Ricci L. 2018. Managing social contents in decentralized online social networks: a survey. In *Online Social Networks and Media*, Vol. 7. 12–29.

[8] Haris Bin Zia, Jiahui HE, Aravindh Raman, Ignacio Castro, Nishanth Sastry, and Gareth Tyson. 2023. Flocking to Mastodon: Tracking the Great Twitter Migration. In *Arxiv*.

[9] Peter Burnap and Matthew Leighton Williams. 2014. Hate speech, machine classification and statistical modelling of information flows on Twitter: Interpretation and communication for policy decision making. In *In Internet, Policy and Politics Conference, Oxford, United Kingdom*.

[10] Ziems C, Vigfusson Y, and Morstatter F. 2020. Aggressive, repetitive, intentional, visible, and imbalanced: Refining representations for cyberbullying classification. In *In Proceedings of the 14th International AAAI Conference on Web and Social Media, ICWSM 2020*. 808–819.

[11] Joseph Cox. 2022. 30,000 New Users Signed Up for Mastodon After Elon Musk Bought Twitter. https://www.vice.com/en/article/n7npd7/30000-new-users-signed-up-for-mastodon-after-elon-musk-bought-twitter.

[12] Karthik Dinakar, Roi Reichart, and Henry Lieberman. 2011. Modeling the detection of Textual Cyberbullying. In *In The Social Mobile Web*. 11–17.

[13] Trinh Viet Doan, Tat Dat Pham, Markus Oberprieler, and Vaibhav Bajpai. 2020. Measuring Decentralized Video Streaming: A Case Study of DTube. In *IFIP Networking 2020*. 118–126.

[14] Chandrasekharan Eshwar, Samory Mattia, Srinivasan Anirudh, and Gilbert Eric. 2017. The Bag of Communities. In *Advances in Neural Information Processing Systems*. 3175–3187.

[15] Megan Farokhmanesh. 2017. A beginner's guide to Mastodon, the hot new open-source Twitter clone. https://www.theverge.com/2017/4/7/15183128/mastodon-open-source-twitter-clone-how-to-use.

[16] The Federation. 2019. https://the-federation.info/.

[17] Kiran Garimella, Ingmar Weber, and Munmun De Choudhury. 2016. Quote RTs on Twitter: Usage of the new feature for political discourse. In *Proceedings of the 8th ACM Conference on Web Science*. 200–204.

[18] Barbara Guidi, Marco Conti, Andrea Passarella, and Laura Ricci. 2018. Managing social contents in Decentralized Online Social Networks: A survey. *Online Social Networks and Media* (2018).

[19] Kwak H., C. Lee, H. Park, , and Moon S. 2010. What is twitter, a social network or a news media?. In *In Proceedings of the 19th International Conference on World wide web, WWW '10*. 591– 600.

[20] Damilola Ibosiola, Ignacio Castro, Gianluca Stringhini, Steve Uhlig, and Gareth Tyson. 2019. Who watches the watchmen: Exploring complaints on the web. In *The World Wide Web Conference*. 729–738.

[21] Waleed Iqbal, Muhammad Haseeb Arshad, Gareth Tyson, and Ignacio Castro. 2022. Exploring Crowdsourced Content Moderation Through Lens of Reddit during COVID-19. In *Proceedings of the 17th Asian Internet Engineering Conference*. 26–35.

[22] Hassan Anaobi Ishaku, Raman Aravindh, Castro Ignacio, Zia Haris Bin, De Cristofaro Emiliano, Sastry Nishanth, and Tyson Gareth. 2021. Exploring content moderation in the decentralised web: The pleroma case. In *CoNEXT 2021 - Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*. 328–335.

[23] Ugander J., Karrer B., Backstrom L., and C. Marlow. 2011. The anatomy of the facebook social graph. In *arXiv preprint arXiv:1111.4503*.

[24] R Tallal Javed, Mirza Elaaf Shuja, Muhammad Usama, Junaid Qadir, Waleed Iqbal, Gareth Tyson, Ignacio Castro, and Kiran Garimella. 2020. A first look at COVID-19 messages on WhatsApp in Pakistan. In *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 118–125.

[25] Cheng Justin, Danescu-Niculescu-Mizil Cristian, and Leskovec Jure. 2015. Antisocial behavior in online discussion communities. In *Proceedings of the 9th International Conference on Web and Social Media, ICWSM 2015*. 61–70.

[26] Lou Jing Kai, Chen Kuan Ta, and Lei Chin Laung. 2016. A collusion-resistant automation scheme for social moderation systems. In *Conference on Human Factors in Computing Systems-Proceedings*. 1157–1162.

[27] Prashant Khare, Mladen Karan, Stephen McQuistin, Colin Perkins, Gareth Tyson, Matthew Purver, Patrick Healey, and Ignacio Castro. 2022. The Web We Weave: Untangling the Social Graph of the IETF. In *Proceedings of the International AAAI Conference on Web and Social Media*, Vol. 16. 500–511.

[28] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. 2010. Structure and evolution of online social networks. In *In Link mining: models, algorithms, and applications. Springer*. 337–357.

[29] Traud A. L., Mucha P. J., and Porter M. A. 2012. Social structure of facebook networks. In *Physica A: Statistical Mechanics and its Applications*. 4165–4180.

[30] La Cava Lucio, Greco Sergio, and Tagarelli Andrea. 2021. Understanding the growth of the Fediverse through the lens of Mastodon. In *Applied Network Science*, Vol. 6. Issue 1.

[31] Cha M., Haddadi H., Benevenuto F., and Gummadi P. K. 2010. Measuring user influence in twitter: The million follower fallacy. In *In Proceedings of the 5th International Conference on Web and Social Media, ICWSM '10*.

[32] Lydia Manikonda, Yuheng Hu, and Subbarao Kambhampati. 2014. Analyzing user activities, demographics, social network structure and user-generated content on Instagram. In *arXiv preprint arXiv:1410.8099 (2014)*.

[33] Mastodon. 2016. https://joinmastodon.org.

[34] Zignani Matteo, Quadri Christian, Galdeman Alessia, Gaito Sabrina, and Rossi Gian Paolo. 2019. Mastodon content warnings: Inappropriate contents in a microblogging platform. In *Proceedings of the 13th International Conference on Web and Social Media, ICWSM 2019*. 639–645. Issue Icwsm.

[35] Stephen McQuistin, Mladen Karan, Prashant Khare, Colin Perkins, Gareth Tyson, Matthew Purver, Patrick Healey, Waleed Iqbal, Junaid Qadir, and Ignacio Castro. 2021. Characterising the IETF through the lens of RFC deployment. In *Proceedings of the 21st ACM Internet Measurement Conference*. 137–149.

[36] Seth A Myers, Aneesh Sharma, Pankaj Gupta, and Jimmy Lin. 2014. Information network or social network? The structure of the Twitter follow graph. In *In Proceedings of the 23rd International Conference on World Wide Web*. 493–498.

[37] PeerTube. 2018. https://joinpeertube.org.

[38] Pleroma. 2018. https://pleroma.social/.

[39] Aravindh Raman, Sagar Joglekar, Emiliano De Cristofaro, Nishanth Sastry, and Gareth Tyson. 2019. Challenges in the Decentralised Web: The Mastodon Case. In *ACM IMC*. 217–229.

[40] Amir H Razavi, Diana Inkpen, Sasha Uritsky, and Stan Matwin. 2010. Offensive language detection using multi-level classification. In *In Canadian Conference on Artificial Intelligence. Springer*. 16–27.

[41] Lorenz Schwittmann, Christopher Boelmann, Matthaus Wander, and Torben Weis. 2013. SoNet–Privacy and Replication in Federated Online Social Networks. In *Distributed Computing Systems Workshops*.

[42] Sara Owsley Sood, Elizabeth F Churchill, and Judd Antin. 2012. Automatic identification of personal insults on social news sites. In *Journal of the American Society for Information Science and Technology*. 270–285.

[43] Chancellor Stevie, Lin Zhiyuan, and De Choudhury Munmun. 2009. This post will just get taken down": Characterizing removed pro-eating disorder Social media content. In *2009 6th IEEE Consumer Communications and Networking Conference, CCNC 2009*. 1157–1162.

[44] Dennis Trautwein, Aravindh Raman, Gareth Tyson, Ignacio Castro, Will Scott, Moritz Schubotz, Bela Gipp, and Yiannis Psaras. 2022. Design and evaluation of IPFS: a storage layer for the decentralized web. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 739–752.

[45] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P Gummadi. 2010. On the evolution of user interaction in facebook. In *In Proceedings of the 2nd ACM workshop on Online social networks*. 37–42.

[46] William Warner and Julia Hirschberg. 2014. Detecting hate speech on the world wide web. In *In Proceedings of the Second Workshop on Language in Social Media. Association for Computational Linguistics*. 19–26.

[47] Zeerak Waseem and Dirk Hovy. 2016. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *In Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*. 88–93.

[48] Ellen Wauters, Verónica Donoso, and Eva Lievens. 2014. Optimizing transparency for users in social networking sites. *info* (2014).

[49] Jun-Ming Xu, Benjamin Burchfiel, Xiaojin Zhu, and Amy Bellmore. 2011. An Examination of Regret in Bullying Tweets. In *In Proceedings of the North American*

Chapter ofthe Association for Computational Linguistics (NAACL-HLT). 697–702.

[50] Zhi Xu and Sencun Zhu. 2010. Filtering offensive language in online communities using grammatical relations. In *In Proceedings of the Seventh Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference*. 1–10.

[51] Wohn Donghee Yvette. 2013. Volunteer Moderators in Twitch Micro Communities. 1–13.

[52] Haris Bin Zia, Ignacio Castro, and Gareth Tyson. 2021. Racist or sexist meme? classifying memes beyond hateful. In *Proceedings of the 5th Workshop on Online Abuse and Harms (WOAH 2021)*. 215–219.

[53] Haris Bin Zia, Aravindh Raman, Ignacio Castro, Ishaku Hassan Anaobi, Emiliano De Cristofaro, Nishanth Sastry, and Gareth Tyson. 2022. Toxicity in the Decentralized Web and the Potential for Model Sharing. *ACM SIGMETRICS* (2022).

[54] Matteo Zignani, Sabrina Gaito, and Gian Paolo Rossi. 2018. Follow the "Mastodon": Structure and Evolution of a Decentralized Online Social Network. In *ICWSM*.

[55] Matteo Zignani, Sabrina Galto, and Gian Paolo Rossi. 2018. Follow the "Mastodon": Structure and evolution of a decentralized online social media. In *ICWSM*. 541–550.

A APPENDIX

Policy	Description	% Instances	% Users	% Posts	Growth in Inst.	% Growth in Inst
ObjectAgePolicy	Applies action based on post age	74.80	57.00	65.30	352	73.50%
TagPolicy	Applies policies to individual users based on tags	58.50	39.40	31.30	509	707.40%
HashtagPolicy	List of hashtags to apply actions against	36.40	16.20	21.20	479	15,833.00%
SimplePolicy	Wide range of actions applied against instances	28.80	39.70	36.30	83	30.70%
NoOpPolicy	Default state of an instance	11.50	5.90	3.70	-98	-63.70%
StealEmojiPolicy	List of hosts to steal emojis from	7.00	6.10	5.40	29	80.50%
HellthreadPolicy	Performs action when a threshold of mentions is reached	6.50	10.90	19.80	21	42.80%
AntiFollowbotPolicy	Stops bots from following users on the instance	4.50	6.20	6.90	13	40.60%
MediaProxyWarmingPolicy	Crawls attachments using their MediaProxy URLs	3.60	7.00	8.30	16	72.70%
KeywordPolicy	Matches a pattern in a post for an action to be taken	23.00	19.40	10.00	9	36.00%
ForceBotUnlistedPolicy	Makes all bot posts to disappear from public timelines	2.70	7.00	5.50	27	675.00%
AntiLinkSpamPolicy	Rejects posts from likely spambots by rejecting posts from new users that contain links	2.70	6.70	6.80	12	85.70%
ActivityExpirationPolicy	Sets a default expiration on all posts made by users of the local instance.	1.30	1.20	0.73	11	366.60%
EnsureRePrepended	Rewrites posts to ensure that replies to posts with subjects do not have an identical subject	1.30	0.40	1.80	6	66.60%
NormalizeMarkup	processes messages through an alternate pipeline	0.9	4.2	1.4	6	150%

Table 2: The top 15 policies applied by administrators with the percentage of instances applying the policies. It shows the percentage of users and posts on the instances applying them, and their growth during our measurement period.

Instances	Admin growth	# Admins	Users	User Growth	Posts	Post Growth	Hate count	URL count	Mentions Count	Hashtag Count	nsfw	Media Removal	Federated Timeline Removal	Reject	Quaran -tined
disqordia.space	6	8	53	33	55.5k	5.1k	NA	NA	NA	NA	3	2	15	71	17
poa.st	5	13	9.7k	9.46k	1.14m	453.3k	78.2k	19.5k	60.8k	20.1k	4	3	2	1	0
pleroma.nobodyhasthe.biz	5	6	128	79	20.5k	1.12k	42.8k	2.6k	42.2k	2.2k	0	1	0	2	0
pleroma.pt	4	7	450	448	24.9k	1.8k	449	75	246	29	8	5	0	1	
pleroma.foxarmy.ml	4	5	8	7	40	7	NA	NA	NA	NA	0	0	0	0	0
varishangout.net	4	7	924	856	98.5k	2.6k	4	1	0.0	3	0	3	9	6	0
mindset.rage.lol	3	5	10	8	635	444	NA	NA	NA	NA	0	0	0	0	0
neckbeard.xyz	3	13	2k	1.22k	1.34m	789.4k	883	136	607	177	0	0	0	2	0
fedi.absturztau.be	2	4	900	463	775.8k	327k	12.9k	1.9k	9.5k	2.4k	3	0	0	14	12
childpawn.shop	2	3	183	176	3.8k	441	NA	NA	NA	NA	0	0	0	0	0

Table 3: Top 10 Instances with the largest increase in number of administrators during our measurement period.

Algorithm	Acc.	Prec.	Recall	F1 score
Logistic Regression	0.73	0.24	0.20	0.21
Multi-Layer Perceptron	0.81	0.00	0.05	0.10
Random Forest	0.87	0.69	0.57	0.62
Gradient Boosted Trees	0.87	0.73	0.54	0.62

Table 4: WatchGen performance results using global training pool and excluding post features (number of posts and transformed posts).

Feature	#Description	#Representation	Number
Users	Number of users registered on an instance	Count	133.8k
posts	Number of posts by users on an instance	Count	29.9m
hate_count	Number of hate words on an instance from hatebase.org	Count	36m
url_count	Number of URLs in user posts on an instance	Count	4.8m
reject	Number of instances to completely reject any flow of material from	Count	8.7k
nsfw	Number of instances to tag all user posts as "Not Safe For Work"	Count	934
media removal	Number of instances to remove media from "	Count	630
federated timeline removal	Number of instances to un-list all user posts from the federated timeline	Count	2.4k
posts_tr	Transformed number of posts using Box Cox transformation	Count	2.8k
reject_deletes	Number of instances to remove all banners from	Count	158
quaran_inst	Number of instances where private (DMs, followers-only) activities will not be sent	Count	1k
mentions_count	Number of mentions in user posts on an instance	Count	24m
hate_avg	Average number of hate words on an instance from hatebase.org	Count	1.5
url_avg	Average number of URLs in user posts on an instance	Count	0.2
hashtags_avg	Average number of hashtags in user posts on an instance	Count	0.3
mentions_avg	Average number of mentions in user posts on an instance	Count	0.8
hashtags_count	Number of hashtags in user posts on an instance	Count	7m
hate_percent	Average percentage of hate words in a post from hatebase.org	Percentage	2.2%
url_percent	Average percentage of URLs in user posts on an instance	Percentage	8.4%
hashtags_percent	Average percentage of hashtags in user posts on an instance	Percentage	6.6%
mentions_percent	Average percentage of mentions in user posts on an instance	Percentage	2.5%
followers	Number of followers of users on an instance	Count	169k
following	Number of remote users that users on an instance follow	Count	8.9k
reblogs_count	Number of reblogs by users on an instance	Count	7.2k
replies_count	Number of posts replied by users on an instance	Count	24.5k
users_tr	Transformed number of users using Box Cox transformation	Count	1.3k
hate_tr	Transformed number of hate_count using Box Cox transformation	Count	4.3k
url_tr	Transformed number of url_count using Box Cox transformation	Count	3k
accept	Number of instances to accept all material from	Count	635
report removal	Number of instances to remove all reports from	Count	91
avatar removal	Number of instances to remove all avatars from	Count	266
banner removal	Number of instances to remove all banners from	Count	291
followers_only	Number of instances that user posts are only seen by their followers	Count	99
active_halfyear	Number of active users in half a year	Count	9
active_month	Number of active users in a month	Count	7
hash_ftr	Number of hashtags to remove activities from the federated timeline	Count	7
hash_rej	Number of of hashtags to reject activities from	Count	6
hash_sen	Number of hashtags to mark activities as sensitive	Count	365

Table 5: Summary of all extracted features used for model training.