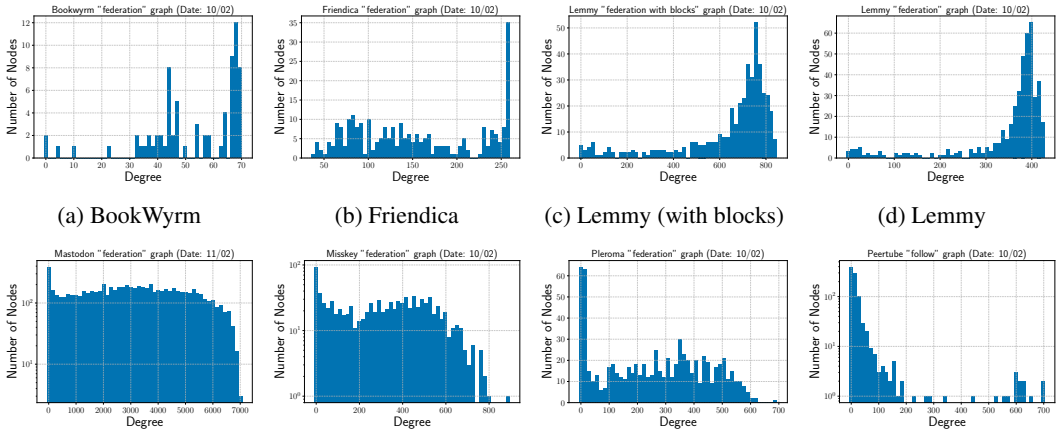
A Further analysis of Fedivertex

This appendix presents several additional plots complementary to the figures presented in the main text. In particular, we propose to further analyze the degree distribution of our graphs, and the language distribution within different Fediverse social media.

Degree distribution Fig. 5 and 6 details the degree distribution for each Fediverse software.

For federation graphs, we identify two distinct patterns. First, in BookWyrm, Lemmy, and Friendica, the degree distribution exhibits a peak at high degrees, and the proportion of nodes decreases with the degree. This behavior is particularly pronounced in Lemmy. In contrast, Mastodon, Misskey, and Pleroma exhibit a peak at low degrees (close to 0), followed by a relatively flat region for intermediate degrees, and a sharp drop for high degrees. This similarity can be attributed to the fact that these platforms are all centered on the same activity: micro-blogging.

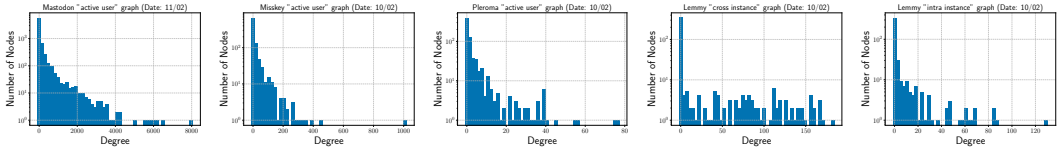
In the active user graphs, the degree distribution consistently follows a power-law-like pattern, with the number of nodes decreasing as the degree increases. The intra-instance graph of Lemmy exhibits a similar trend, though the pattern is less pronounced.



(a) BookWyrm (b) Friendica (c) Lemmy (with blocks) (d) Lemmy

(e) Mastodon (f) Misskey (g) Pleroma (h) PeerTube

Figure 5: Degree-histogram distributions for federation graphs. Version with and without block for Lemmy, and Follow graph for Peertube



(a) Mastodon (active users) (b) Misskey (active users) (c) Pleroma (active users) (d) Lemmy (cross-instance) (e) Lemmy (intra-instance)

Figure 6: Degree-histogram distributions for the three active-user graphs and Lemmy's cross- vs. intra-instance graphs.

Fig. 7 reproduces fig. 2 from the main text, but with a normalized degree: the degree is divided by the number of nodes. This normalization incorporates graph density: the closer a curve is to the top-right corner, the denser the graph. From fig. 7, Fedivertex graphs appear denser than the SNAP social graphs. While the two groups overlap in fig. 2, they are more distinctly separated in the normalized plot. This further illustrates major differences between Fedivertex and widely-used datasets.

Language distribution Fig. 8 presents histograms of the language distribution in Lemmy, Peertube, Mastodon, and Misskey. This figure shows diverse language distributions among Fediverse social media. First, Mastodon and Lemmy have a large majority of English-speaking instances. Second,

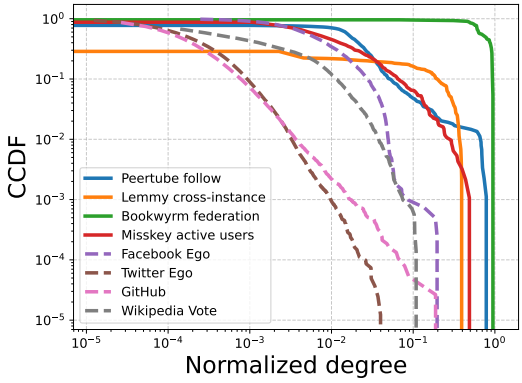
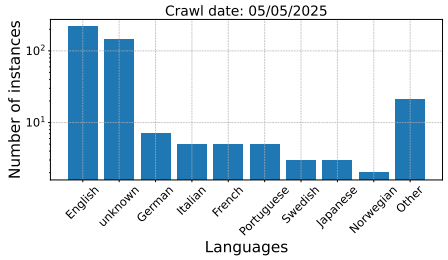
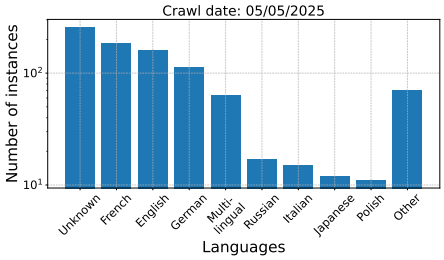


Figure 7: Complementary Cumulative Distribution Function (CCDF) of the degree for several Fedivertex graphs and other widely-used graphs [34] based on social networks after normalization by the total number of nodes

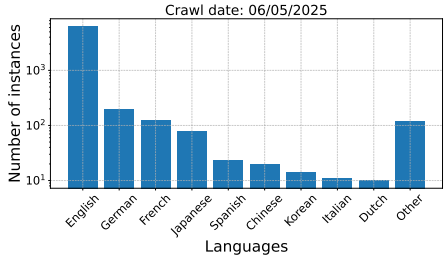
Peertube is dominated by European languages. We observe more diversity than in Mastodon because French and German have a number of nodes similar to English. Third, Misskey shows a unique language distribution within the Fediverse as the Asian languages (especially Japanese) are the most spoken languages. This phenomenon is explained by the fact that Misskey (contrary to other Fediverse software) has been developed by a Japanese developer.



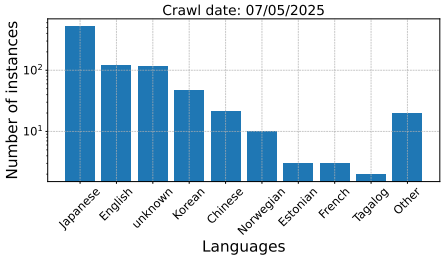
(a) Lemmy



(b) Peertube



(c) Mastodon



(d) Misskey

Figure 8: Histograms of the language distribution among four different Fediverse software, logarithmic scale

For a thorough exploratory data analysis we refer to our 5 Kaggle notebooks that cover these aspects, but also other dimensions such as weight or user distribution: `https://www.kaggle.com/datasets/marcdamie/fediverse-graph-dataset/code`.

B Python API examples

Our Python package `Fedivertex` produces a straightforward interface to interact with our dataset. This package seamlessly downloads and extracts graphs from our CSV files using the NetworkX

```python
from fedivertex import GraphLoader

loader = GraphLoader()
G = loader.get_graph(software = "misskey", graph_type = "active_user")

nb_nodes = G.number_of_nodes()
nb_edges = G.number_of_edges()
degree = G.degree()
```

Listing 3: Code example to extract statistics about a Fedivertex graph

```python
from fedivertex import GraphLoader
loader = GraphLoader()

# Extract the latest graph
G_latest = loader.get_graph(software = "misskey", graph_type = "active_user")

# Extract the i-th graph in the dataset (sorted chronologically)
i = 3
G_i = loader.get_graph(software = "misskey", graph_type = "active_user", index=i)

# Extract the graph of a specific data
G_2403 = loader.get_graph(software = "misskey", graph_type = "active_user", date="20250324")
```

Listing 4: Code example of the graph selection in Fedivertex

format, a popular Python package to manipulate graphs. This appendix demonstrates several simple operations that can be performed using our package.

The installation is straightforward as our package is available on the Python Package Index (PyPI): `pip3 install fedivertex`. The dataset is downloaded automatically the first time a graph is loaded. The dataset is stored in cache thanks to the `CroissantML` package in the folder `.cache/croissant`.

Listing 3 shows how to extract basic statistics from the latest Misskey active user graph. Our method `get_graph` outputs a NetworkX object, so we can apply all functions and methods from NetworkX. This example simply calls `number_of_nodes`, `number_of_edges`, and `degree`, but one can use the entire NetworkX API. We refer to NetworkX documentation for further information about the capabilities of their API: `https://networkx.org/documentation/stable/index.html`

In Fedivertex, a graph is defined using three elements: the software (e.g., Misskey), the graph type (e.g., active user), and the date. For date selection, the package provides three different options, illustrated in listing 4. First, if no date is provided to `get_graph`, the latest graph is automatically selected. Second, if the user provides an (integer) index i, the i-th graph is selected. This index works like a Python list index, so -1 is accepted and the indexing starts at 0. Third, the user can provide a date in a string format `YYYYMMDD` (e.g., 20250324).

As our dataset contains a large range of possible software, graph types, and dates, we also provide utility functions to simplify the interactions. Listing 5 presents these functions. First, `list_all_software` lists all the available software. Second, `list_graph_types` lists the available graph types for a given software. Third, `list_available_dates` lists the available dates for a given software and graph type.

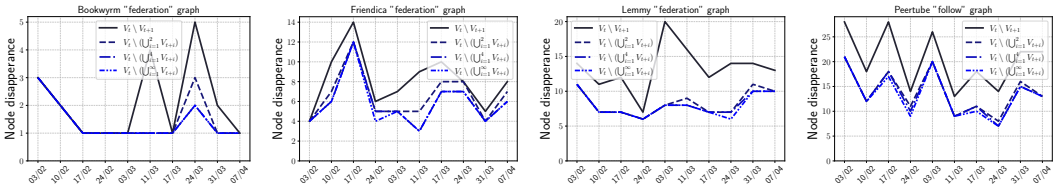
```python
from fedivertex import GraphLoader
loader = GraphLoader()

print(loader.list_all_software())
# ["bookwyrm", "friendica", "lemmy", "mastodon", "misskey", "peertube", "pleroma"]

print(loader.list_graph_types("peertube"))
# ["follow"]

print(loader.list_available_dates("peertube", "follow"))
# ["20250203", "20250210", "20250217", "20250224", "20250303", "20250311",    "20250317",
#    "20250324", "20250331", "20250407", "20250414", "20250421", "20250428", "20250505"]
```

Listing 5: Code example of the utility functions in Fedivertex

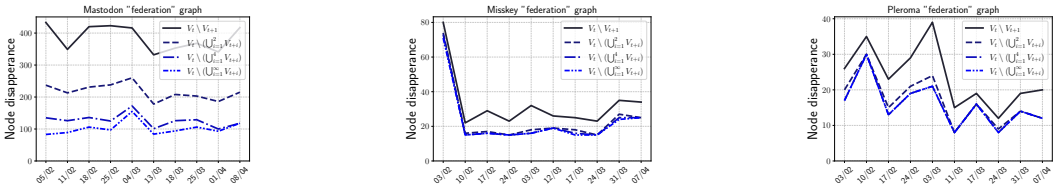
As seen in listing 1 in main text, it is possible to use graph extraction with the option "`only_largest_component = True`." When set to true, the method only returns the largest component of the graph. As many graph algorithms require connected graphs, this option eases benchmarks of such algorithms. Additionally, listing 2 shows how to extract the language information to use it as a ground-truth label (e.g., in community detection like in table 4)

For further advanced examples, we refer to our Kaggle notebooks that analyze the dataset using the Python package:
`https://www.kaggle.com/datasets/marcdamie/fediverse-graph-dataset/code`.

C Experiments on defederation

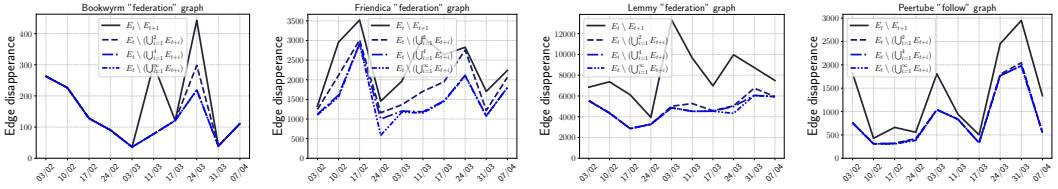


(a) BookWyrm (b) Friendica (c) Lemmy (d) PeerTube

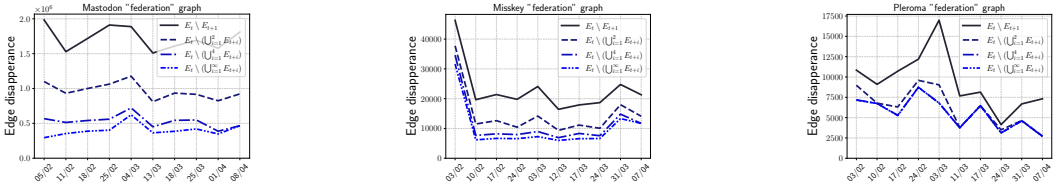


(e) Mastodon (f) Misskey (g) Pleroma

Figure 9: Node deletion at different time horizons for federation graphs, and the follow graph (for PeerTube).



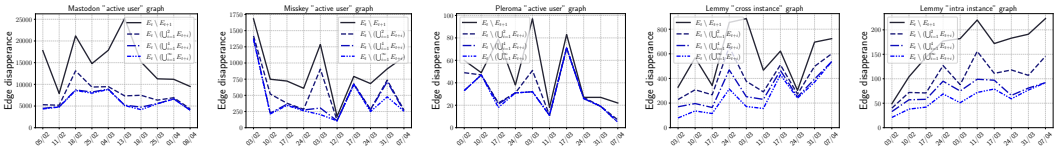
(a) BookWyrm (b) Friendica (c) Lemmy (d) PeerTube



(e) Mastodon (f) Misskey (g) Pleroma

Figure 10: Edge deletion at different time horizons for federation graphs, and the follow graph (for PeerTube).

In section 5.2, we focused on edge deletion prediction by using the same method than for edge prediction, with limited results. A natural question is thus: is the poor performance due to noise in the deletion process? To answer this question, we must assess whether edge deletion is a random process (e.g., caused by network instability) or whether it persists over time. In particular, fig. 4 does not exhibit clear trends and tends to suggest that deletions could be accidental. We test this hypothesis by comparing the edges and nodes deleted between two consecutive weeks (as reported in fig. 4) to



(a) Mastodon (active users) (b) Misskey (active users) (c) Pleroma (active users) (d) Lemmy (cross-instance) (e) Lemmy (intra-instance)

Figure 11: Edge deletion at different time horizons for the three active-user graphs and Lemmy's cross- vs. intra-instance graphs.

Graph	Betweenness	Eigenvector centrality	Pagerank	Random
Misskey AU	6	16	16	7 ± 2.4
Misskey fed.	7	12	11	7 ± 2.3
Friendica fed.	11	13	13	7 ± 2.4
Pleroma fed	13	3	12	8 ± 2.5

Table 5: Comparison of Betweenness, eigenvector centrality and Pagerank scores on different graphs by reporting the number of correct predictions in Top-50 scores (the higher the better)

those that remain deleted over longer time horizons: two weeks, four weeks, and until the end of the measurement period.

We report the results for node deletion in fig. 9, and for edge deletion in fig. 10 and fig. 11, depending on the type of graph. The results show that most variations are permanent, at least within the time scale of our study, as soon as they persist beyond two weeks.

The majority of deletions appear to be permanent, except in BookWyrm, which can be explained by the small size of the network, and in Mastodon, where node and edge deletion durations follow a continuum. The only other case showing a progressive increase in the proportion of affected edges with respect to duration is Lemmy cross- and intra-instance, which can be explained by the construction of these graphs: they depend on recent message exchanges between instances and may naturally include servers whose users interact only occasionally.

In these cases, edge deletion does not appear to be an artifact of the crawling process, but rather reflects actual communication dynamics in the graph.

We also present a baseline for node deletion. Similarly to the case of edges, we report the number of actual deletions among the top-50 nodes with the highest scores. We propose to predict deletion for nodes that are the least central in the graph, as this may indicate lower activity. Other metrics, and incorporating additional information about the nodes, could certainly improve these predictions. We report the results in table 5.

D Additional remarks about the crawler

We discuss in this section the robustness of the crawling, and legal procedure prior the crawler deployment. We refer to section 3.5 for the ethical concerns related to the crawling procedure.

Robustness Fig. 4c represents the variations of the number of nodes in different Fediverse software. For all these social networks, the variations looks erratic. Such patterns could raise concerns regarding the robustness of the crawling procedure. However, these erratic variations are a natural phenomenon of the Fediverse and are not an artifact created by the crawler.

To put these variations into perspective, fig. 12 represents the overall evolution instead of the node variation between two snapshots. We observe that the number of nodes is relatively stable. The patterns from fig. 4c are barely distinguishable on fig. 12 because they concern a minority of nodes.

Even though these patterns involve only a minority of nodes, they are worth examining. Most instances on the Fediverse are maintained by volunteers who cover the operational and maintenance costs themselves. Many are operated by a single individual who may lack the time, financial resources,

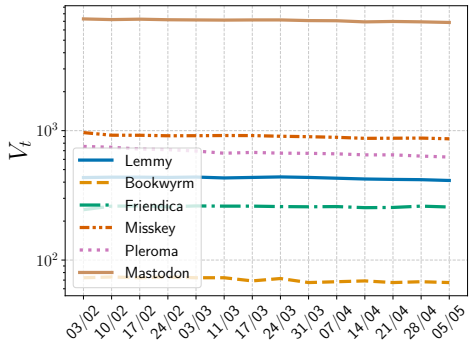


Figure 12: Evolution of the number of nodes over time for the six software of Fedivertex

or motivation to ensure long-term stability. While the biggest instances benefit from a high stability due to their teams of experienced engineers, smaller nodes can quickly appear, disappear, and eventually reappear.

A natural question that follows from this observation is whether our crawler adequately captures the dynamics of unstable instances. Our approach relies on a curated list maintained by Fediverse Observer, which continuously crawls the Fediverse to discover new instances and assess the availability of existing ones. Notably, instances that are temporarily unavailable remain listed, meaning our crawler does not ignore unstable nodes. Thanks to the frequent updates and broad coverage provided by Fediverse Observer, the resulting instance list is both extensive and up-to-date. We thus argue that our crawler offers a realistic and consistent view of the Fediverse.

Our source code is publicly available: `https://github.com/MarcTOK/Franck`. The crawler implementation has remained consistent since the first crawl, with only minor changes introduced to improve logging completeness.

Legal compliance Before crawler deployment, we have been involved in active discussions with the legal departments of our institutions to cover any potential legal issue. We worked in particular on the compliance with data privacy regulations, especially GDPR. As stressed in section 3.5, our graphs only contain aggregated information (i.e., server-level information), and we only query public APIs, and thus are less privacy sensitive than existing works (e.g., the Webis Mastodon corpus [55] that compiled 700 million Mastodon posts). The crawling logs are kept only for the time necessary to confirm the crawl success and deleted afterwards.

Our work is among the first research projects on the Fediverse, and the guidelines for research in this field still need to be refined. Existing guidelines tend to focus on centralized mainstream social media, and are thus not well adapted to scenarios where instances are run by distinct legal entities, possibly subject to different regulations and norms. The discussions with the legal departments allowed us to clarify the legal requirements in this context. We shared our experience with the Network of Alternative Social Media Researchers, a transdisciplinary international collective of academics studying the cultures, technologies, and practices of non-corporate social media[5], to help formalize a dedicated ethical framework for research on alternative social media.

In addition to the discussions with the legal department, we have also coordinated our efforts with the IT department to avoid any disturbance caused by our crawler. Notably, our crawler is formally declared to the security operations center of our university, is assigned specific IP addresses, and is under the surveillance of the IT department. These measures are complementary to the preventive strategy of a *slow* crawl, aiming to minimize disturbances for the instances, as detailed in section 3.5. After four months of experiments, we have received no complaints, neither from our university nor from Fediverse instance administrators.

[5]Testimony at `https://www.socialmediaalternatives.org/2025/05/07/asm-research-ethics.html`