WIRTSCHAFTS
UNIVERSITÄT
WIEN VIENNA
UNIVERSITY OF
ECONOMICS
AND BUSINESS

Bachelor Thesis

Open Ranking Algorithms for Decentralized Social Media

Felix Rötzer

Date of Birth: 28.04.1999
Student ID: H11934985

Subject Area: Information Business

Studienkennzahl: 11934985

Supervisor: Univ. Prof. Dr. Axel Polleres

Department of Information Systems & Operations Management, Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria

DEPARTMENT FÜR WIRTSCHAFTSINFORMATIK
UND OPERATIONS MANAGEMENT DEPARTMENT
OF INFORMATION SYSTEMS AND OPERATIONS
MANAGEMENT

Contents

Abstract

In recent years, social media has shaped public discourse, raising major concerns about data-misuse of centralized platforms like Facebook or Twitter to improve their ranking algorithms for profit maximization. Mastodon, a new decentralized social media platform, has shown to be a leading alternative to people who value their privacy and support the decentralization of power. However, the advantages come with the cost of decreased capability to recommend fitting content. In this paper, we are going to evaluate the ranking techniques used from social media platforms, by analyzing the open-source Twitter algorithm to extract meaningful insights that can be useful for implementation in decentralized protocols. The results will be directly implemented in an associated Mastodon server for WU campus.

1 Introduction

As social media becomes increasingly central to public discourse, the power dynamics of ranking algorithms in centralized platforms — such as Twitter (now X) — have raised significant concerns. These concerns include polarization through algorithmically driven echo chambers [3] and the erosion of user privacy through misuse of personal data [11]. These large institutions generally create ranking algorithms that focus on generating the most profit and maximizing screen time of their users [28].

In contrast to these big institutions, decentralized online social networks (DOSNs), like Mastodon [17] or Bluesky [5] are not owned or managed by a single organization, but through a distributed network of servers that are managed by various users. This structure makes them more robust and secure, by eliminating a single point of failure [14]. These platforms mostly rely on open standards or protocols, with Mastodon, for instance, utilizing ActivityPub - a decentralized protocol that allows for federated communication between different platforms [34]. Bluesky, another prominent example, which was founded by Jack Dorsey, the former CEO and co-founder of Twitter, has developed its own protocol, called the AT protocol [4]. Since Elon Musk´s takeover of Twitter, Mastodon has gained significant traction, with many users migrating from Twitter to the decentralized platform [1]. Mastodon uses a chronological post order rather than a ranking algorithm. However, this approach presents challenges, particularly regarding content discovery and user engagement, which impact the platform´s broader adoption. This thesis seeks to examine the potential for an open and possibly community-driven ranking algorithm in decentralized social media as an alternative to traditional centralized models. By exploring the principles underlying ranking algorithms on centralized platforms such as Twitter, it also investigates the feasibility of creating a simple, adaptable ranking algorithm on Mastodon that incorporates these principles while ensuring transparency, user privacy, and potential community governance.

2 Purpose

The main objective of this study is to identify the key variables and priorities in ranking algorithms used by centralized institutions like Twitter and demonstrate how these insights can be used to improve user experience in decentralized social media platforms like Mastodon. By analysing these ranking mechanisms, this study aims to provide valuable insights that contribute to the existing body of literature.

With the implementation of a new Mastodon server for the WU Wien campus, this study will also explore the technical dynamics of building and maintaining a decentralized social media server. Moreover, the study aims to implement various open ranking algorithms by modifying the Mastodon source code. This infrastructure will provide a foundation for future research, enabling user studies to analyze engagement and preferences in decentralized environments compared to centralized platforms.

Lastly, this research aspires to raise awareness of decentralized platforms as viable alternatives to traditional social media platforms, demonstrating how improved open ranking algorithms can enhance their potential for broader adoption.

3 Research question

The central question guiding this research is: "How do ranking algorithms on centralized platforms like X work in detail, and how can selected aspects be leveraged to enhance user experience on decentralized platforms like Mastodon while preserving the inherent advantages of decentralized architectures? "

4 Methodology

The research question will be addressed through a combination of literature analysis and exploratory investigation. To establish a baseline understanding of ranking algorithms, we will first conduct a comprehensive literature review to identify general ranking methodologies in various domains. This foundational exploration will include recommender systems (RSs) and search engines to establish a broader understanding of ranking techniques and their applications. Additionally, we will examine key algorithms used in RSs and search engines.

Furthermore, we will conduct an extensive source code analysis, integrating the insights gathered from the literature review to identify key metrics and priorities that ranking algorithms can be divided into. This analysis will help us understand how centralized platforms adjust their algorithms. For this purpose, the open-source ranking algorithm from X (formerly Twitter) will be a primary reference [30].

A review of academic and industry literature will be used to outline the principles and mechanisms of DOSNs, emphasizing how their architecture influences the challenges these protocols face. Specifically, we will examine different protocols such as ActivityPub, WebSub and the AT Protocol in detail. Building on these insights, the study will investigate how ranking approaches used in conventional ranking algorithms might improve the design of ranking systems for decentralized social media platforms such as Mastodon.

Additionally, the study will involve the implementation of a custom Mastodon server for the WU Wien campus, serving as a preliminary framework for exploring and experimenting with alternative ranking mechanisms. This implementation will be carried out on a Linux-based virtual machine (VM) designed for testing purposes, hosting the Mastodon server and provide a scalable environment. The implementation process, including server integration and the development of various ranking algorithms, will be comprehensively documented in a GitHub repository. Through this documentation, we aim to provide valuable insights into the development and experimentation of decentralized ranking systems in a real-world context.

5 Background

To develop an understanding of social media ranking algorithms, this section provides the necessary background by introducing the concept of recommender systems (RSs). To achieve this, we will define RSs, briefly outline their history and provide different information filtering techniques of RSs.

This will be followed by an introduction into decentralized protocols, such as the ActivityPub and the AT-Protocol. These protocols are utilized by DOSNs such as Mastodon or Bluesky to establish the required architecture for platforms, to function on a decentralized basis.

This solid grasp of basic ranking concepts, along with thorough understanding of the underlying technology powering decentralized social media platforms, will provide the essential groundwork for advancing the objectives of this thesis going forward.

5.1 Recommender systems

In this thesis, we define a recommender system as a "functional software system that applies at least one implementation to make recommendations" [19]. In 1979, Elaine Rich developed one of the earliest RSs known as Grundy. It was a simple book recommendation algorithm that primarily relied on the approach of stereotyping [19].

With the mainstream adoption of social media and the immense amount of available data, RSs became fundamental in today´s society. However, their applications go far beyond just social media, encompassing domains such as book recommendations, e-learning, music and movies as highlighted from Roy in 2022 [27].

Recommender systems employ different approaches, but their filtering methods can broadly be categorized into content-based filtering, collaborative filtering and hybrid filtering techniques [19]. As visualized in Figure 1, the approaches can be further subdivided. However, an in-depth discussion of all subcategories falls beyond the scope of this thesis.

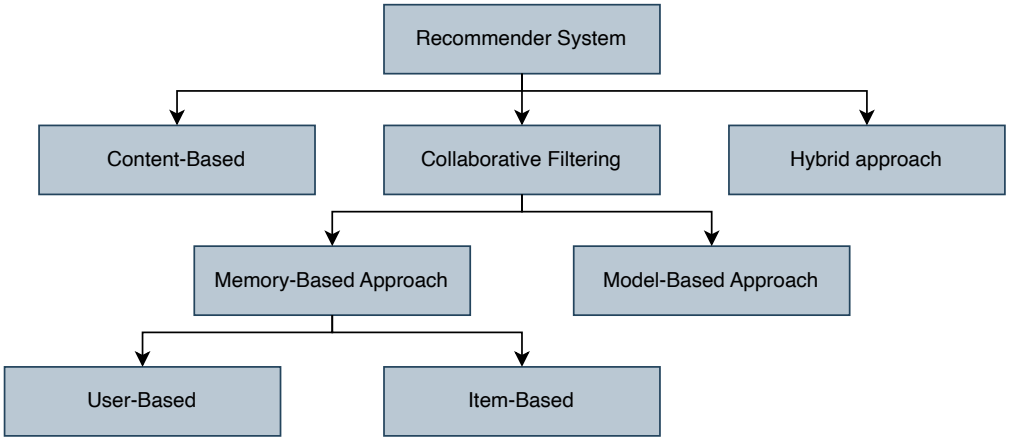

Figure 1: Overview Recommender Systems. Adapted from Roy (2022) [27]

5.1.1 Collaborative Filtering

Collaborative filtering (CF) is one of the most widely used methods in social media algorithms today. The term traces back to Goldberg´s Tapestry system from 1992 [10]. However, the definition of CF as it is understood today was introduced later, in 1994, through the research contributions of Resnick with the GroupLens system. This system computed the similarity between readers of Usenet newsgroups based on their ratings of different articles [24].

The core idea behind CF is to generate recommendations based on the ratings of other users with similar interests. A user's interests are determined based on based on historical rating data for various items. In this context, users with similar preferences are reffered to as neighbors [19]. After identifying similar users, a subclass of nearest neighbors is selected to generate well-fitting recommendations using a regression model [19].

One of the primary advantages of CF is that it does not require prior knowledge of item features to generate recommendations. This makes it particularly useful for discovering new interests by suggesting previously unseen items [27]. As illustrated in Figure 1, CF can be broadly categorized into model-based and memory-based techniques [2].

Model-based CF relies on pre-trained models to predict user preferences. These models are trained on the CF principles, specifically focusing on user similarity. In contrast, memory-based methods generate recommendations in real-time by directly analyzing relationships between users and items. However, memory-based approaches have certain drawbacks, such reduced

efficiency when handling large datasets and potential scalability challenges due to the high computational demands [2].

Figure 2 provides a visualization of the collaborative filtering workflow. It depicts how data is forwarded through the recommender to the web page, where the user interacts with the recommended content. The content is being rated from the user, which creates a user rating profile reflecting the preferences of the user. These ratings are then compared against the profiles of similar users, and the ratings of highly similar users are leveraged to generate new content recommendations for the web page.

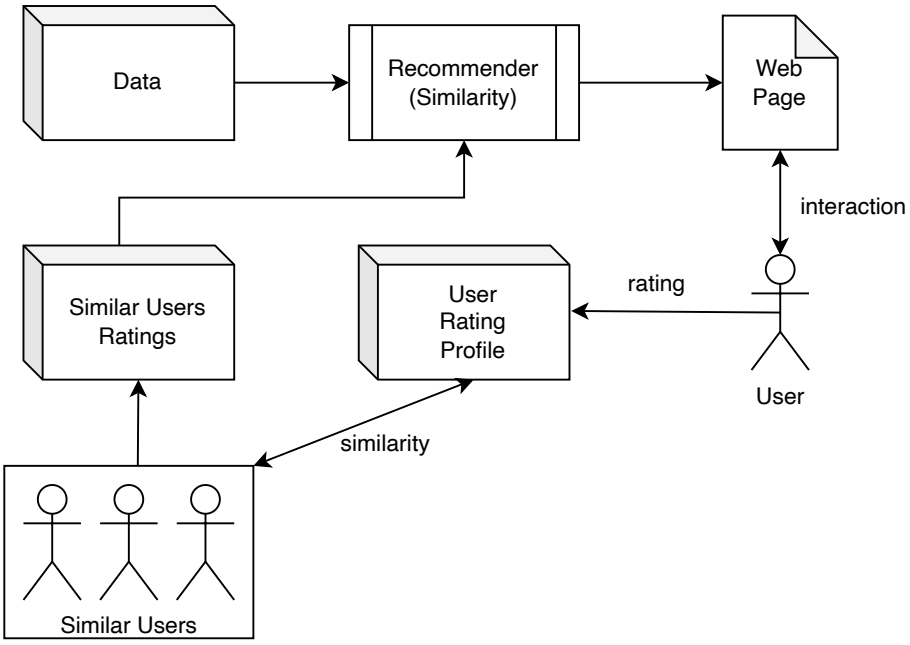

Figure 2: Workflow Collaborative Filtering. Inspired by Roy (2022) [27]

5.1.2 Content-Based Filtering

Content-based filtering (CBF) is a recommendation technique that relies solely on a user´s item rating history. Unlike collaborative filtering, which evaluates correlations between different users, CBF focuses on the correlation between a user´s preferences and the characteristics of individual items [19]. Therefore, obtaining detailed information about various features of items is

essential. However, this requirement can be a limitation when data availability is scarce [27].

For instance, in the case of a music streaming platform, the features of the underlying song might include the artist, genre, or release year. When a user positively interacts with a particular song, the associated features are aggregated to construct a user profile that reflects their feature preferences. This profile is then used to compare against the features of other songs to generate recommendations. Since this technique prioritizes items that share strong similarities with previously liked content, CBF usually ignores different content that the user eventually might like, but has not yet interacted with.

One of the earliest implementations of CBF was Letizia, developed by Lieberman in 1995. The system had the goal of assisting the user in web browsing by recommending interesting items based on the users browsing history [19]. Lieberman's approach used the web history to form the user's profile, represented as a list of weighted keywords including all the words found on the previously visited web pages. This weighted keyword list was then used to compute the relevance of alternative webpages, with the most promising ones being suggested based on the presence and frequency of these specific terms [19].

Figure 3 highlights the workflow of the content-based filtering approach. It shows how the user interacts with the web page and rates the content. Based on the user ratings, a user rating profile is generated. This user rating profile is then used to compare for similarity with the new incoming content data. The most relevant content is then recommended and displayed on the web page.

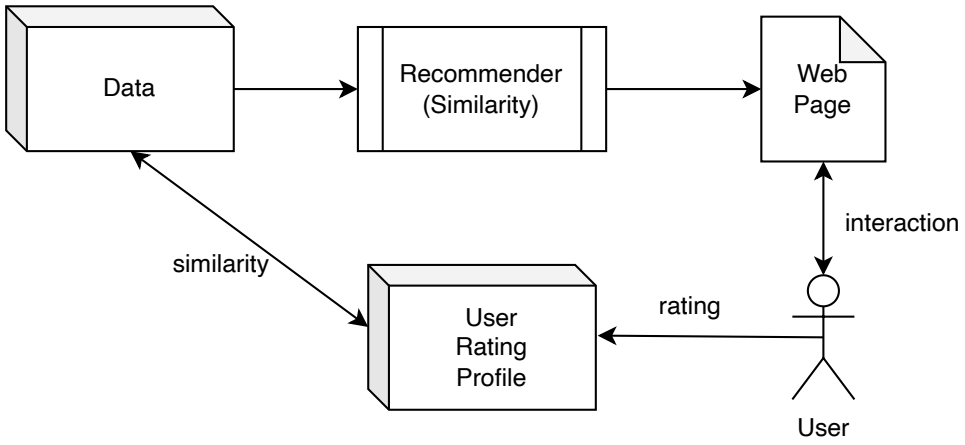

Figure 3: Workflow Content Based Filtering. Inspired by Roy (2022) [27]

5.1.3 Hybrid Approach

A hybrid filtering approach combines collaborative filtering (CF) and content-based filtering (CBF) to address the limitations of each method when applied independently [19]. Extensive reseach indicates that hybrid systems consistenly outperform standalone CF or CBF techniques [27]. In modern recommender systems, integrating multiple approaches within ranking algorithms is considered the state of the art. By leveraging the complementary strengths of CF and CBF, hybrid models enhancfe recommendation accuracy and effectiveness, making them a preferred choice in many real-world applications.

5.2 Decentralized Protocols

Decentralized Open Social Networks (DOSNs) operate on a foundation of innovative protocols that enable a seamless decentralized server-architecture, eliminating dependency on any central authority. In this model, users connect to a single server, however, the connected server can communicate with all the other servers connected to the protocol, ensuring seamless interoperability across the decentralized ecosystem [37].

Several protocols facilitate DOSNs, including OStatus, Diaspora, Matrix, Nostr, Bluesky's AT or the ActivityPub. This research focuses on analyzing three protocols: ActivityPub [34], WebSub [33], and the AT-protocol [4]. However, a detailed technical examination will be conducted only on the ActivityPub protocol. The reason for this choice is that Mastodon, the platform we will use for further investigation in later chapters, is based on this protocol.

Figure 4 provides a simplified visual representation of a decentralized server architecture. The three interconnected nodes in the middle represent servers, while the surrounding elements connected via dashed lines depict user devices interacting within the network.

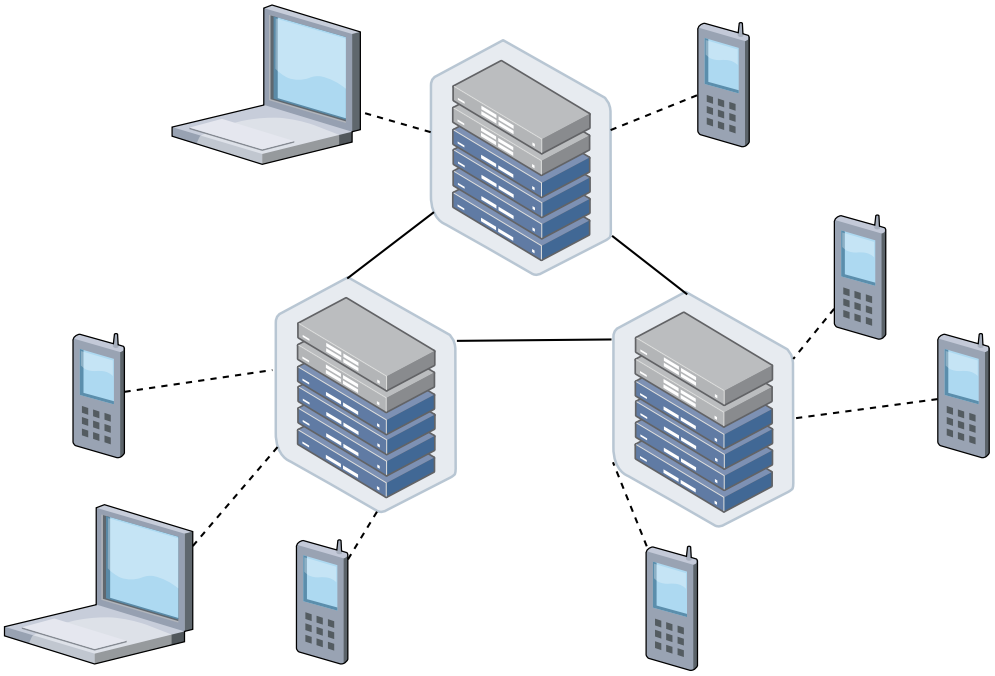

Figure 4: Representation of a decentralized protocol architecture

5.2.1 About ActivityPub

ActivityPub is a decentralized social networking protocol that provides social media platforms the necessary infrastructure to operate a decentralized social network. ActivityPub was pubished in 2018 by the World Wide Web Consortium (W3C) [35]. Due to its open and decentralized nature, it is accessible to anyone interested in building upon the protocol. Mastodon, currently represents the biggest and most popular project using the ActivityPub [36], followed by projects like PeerTube. Mastodon was founded by the German software engineer Eugen Rochko in 2016 as a decentralized alternative to mainstream social media platforms.

The protocol offers a client-to-server Application Programming Interface (API), that can execute a variety of different actions. These activities include creating, updating, deleting or following to name a few. Besides that it also includes a federated server to server API for delivering content between different servers. This means, in general, we can distinguish between the server-to-server layer made up of all interconnected servers, and the social network layer, formed by the relationships between users of a given instance [37]. The client-to-server API is reffered to as the "Social API", whereas the server-to-server API is reffered to as the "Federation Protocol" [34]. The social media platform Mastodon is only using the server-to-server federation protocol [32]. From the open infrastructure of the ActivityPub protocol it follows that different servers can communicate with each other. For example, it is possible for a user from Mastodon, to follow another user from PeerTube and vice versa as it is visualized in Figure 5.

5.2.2 How to interact with ActivityPub?

Both types, the client-to-server and the server-to-server API, are using the HTTP methods GET and POST for server communication [34]. The client-to-server GET request queries the latest messages in the network stream of the underlying server, whereas the POST request can be used to send messages to the world. Under "messages to the world" we understand creating, liking, deleting and other typical social network functionalities, in this context. On the other hand, the server-to-server requests are used on a federated level, such as the POST request is used to send someone a message [34].

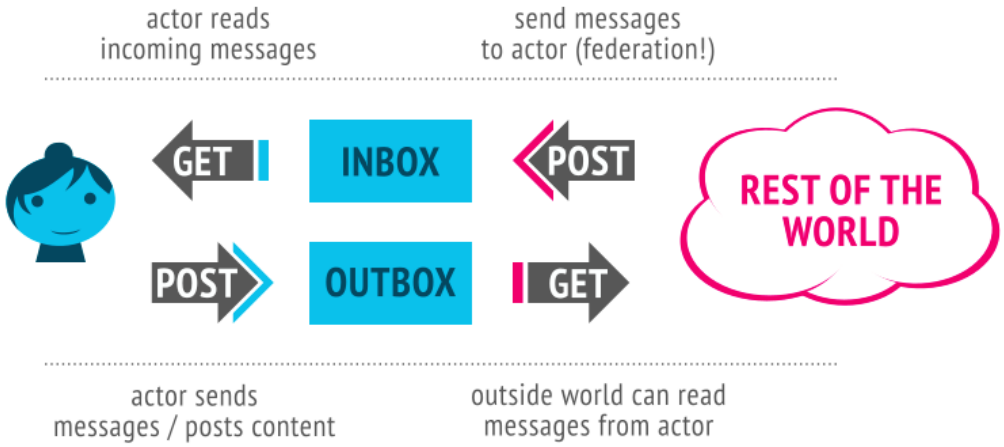

Figure 5: Interaction with the ActivityPub Protocol, Source: `https://www.w3.org/TR/activitypub`

The requests, both GET and POST, use ActivityStreams for its vocabulary in the ActivityPub protocol. ActivityStreams is a model for representing potential activities, meaning different kinds of actions used in the DOSN (e.g. like, create or delete content). These HTTP requests are formatted in a JSON-based syntax and contain all necessary information to successfully execute the request [32].

```
{"@context": "https://www.w3.org/ns/activitystreams",
 "type": "Like",
 "id": "https://social.example/alyssa/posts/5312e10e-5110-42e5-a09b-934882b3ecec",
 "to": ["https://chatty.example/ben/"],
 "actor": "https://social.example/alyssa/",
 "object": "https://chatty.example/ben/p/51086"}
```

Figure 6: Example of a JSON document for a POST request, Source: `https://www.w3.org/TR/activitypub`

In Figure 6 we can see an example of a JSON document for a POST request on a server-to-server level. In this sample we see a number of different parameters, namely "context", "type", "id", "to", "actor" and "object". Each of the parameters is required to form a useful request to the server. The POST request is sent to the server and acts as an instruction that the server then acts upon. In our case the "type" parameter specifies the action,

which is a "Like". The "object" variable specifies the underlying post that should be liked and the "actor" parameter tells the server who is the user that the like is executed by.

5.2.3 The AT-Protocol

The AT protocol is another decentralized federated protocol, which was invented for Bluesky, which is a new Twitter-style social network [12]. Developed by previous Twitter founders, including Jack Dorsey, this protocol embraces a decentralized nature and is designed to support multiple apps and the same user identity, social graph and user data storage servers can be shared between all its applications [12]. Since Elon Musk took over Twitter and implemented changes, many of the platform's users have been migrating to Bluesky. As of late 2024, Bluesky has surpassed 20 million registered users, with its user base steadily growing [8].

One major reason why Bluesky decided to invent their own protocol instead of building upon the ActivityPub protocol, was account portability [6]. Account portability in this context means that a user can use different servers and platforms inside the AT-protocol with the same account. This can prevent sudden bans, server shutdowns, and policy disagreements [6].

6 Conventional Ranking Approaches: Example Twitter

Traditional social media platforms such as Twitter or Facebook have leveraged algorithmic post-ordering mechanisms for a long time. Usually, these ranking algorithms lack in transparency and remain closed-source [15]. In 2023, Twitter made parts of its recommendation system and source code public, making it a valuable candidate for understanding the algorithm's functionality at a high level [30]. It should also be mentioned that Twitter's algorithm takes a particularly extreme and capitalistic approach in regards to engagement maximization. This includes limiting content filtering mechanisms, which can promote polarizing and divisive content [18]. The reason for this is that sensationalist content can trigger emotional reactions, which often lead to a strong psychological urge for users to share their feelings with others through engagement [25].

In this section of the thesis, we aim to explore the structure and functionality of Twitter's algorithm, with a particular focus on the data it uses and the processes involved in its pipeline. Specifically, we will examine the mechanics behind Twitters's algorithmically driven "For you" timeline and compare it to the "Following" timeline. This analysis will provide valuable insights into the potential for implementing open ranking algorithms within Mastodon, which will be explored in the next chapter. Given that Mastodon's default post ordering follows a stricly reverse chronological order, understanding alternative ranking mechanisms will offer a foundation for optimizing content discovery and user engagement.

6.1 What ranking approaches does Twitter offer?

In general, Twitter offers two different timelines with distinct ranking approaches. The algorithmically driven "For you" timeline serves as a globally curated timeline and, in contrast, acts as an alternative to Mastodon's default local timeline. Additionally, Twitter also features a "Following" timeline, which presents posts exclusively from accounts a user follows, ranked in strict reverse-chronological order. This provides a simpler and less curated experience compared to the algorithm-driven "For you" timeline [31]. Therefore, the "Following" timeline can be compared to Mastodon's default timeline, which also ranks the content in reverse-chronological order. Since a policy change by Twitter on March 17, 2016, the "For you" timeline has been serving as the platform's default option [31]. This means that users actively have to switch to the "Following" timeline if they prefer it. This policy change caused almost all of the Twitter users to exclusively interact with the "For you" timeline, significantly increasing engagement on the platform [21].

Understanding the design and operation of these timelines offers valuable insights for how decentralized platforms could adopt lightweight algorithmic solutions to enhance user engagement without compromising the principles of openness and user autonomy.

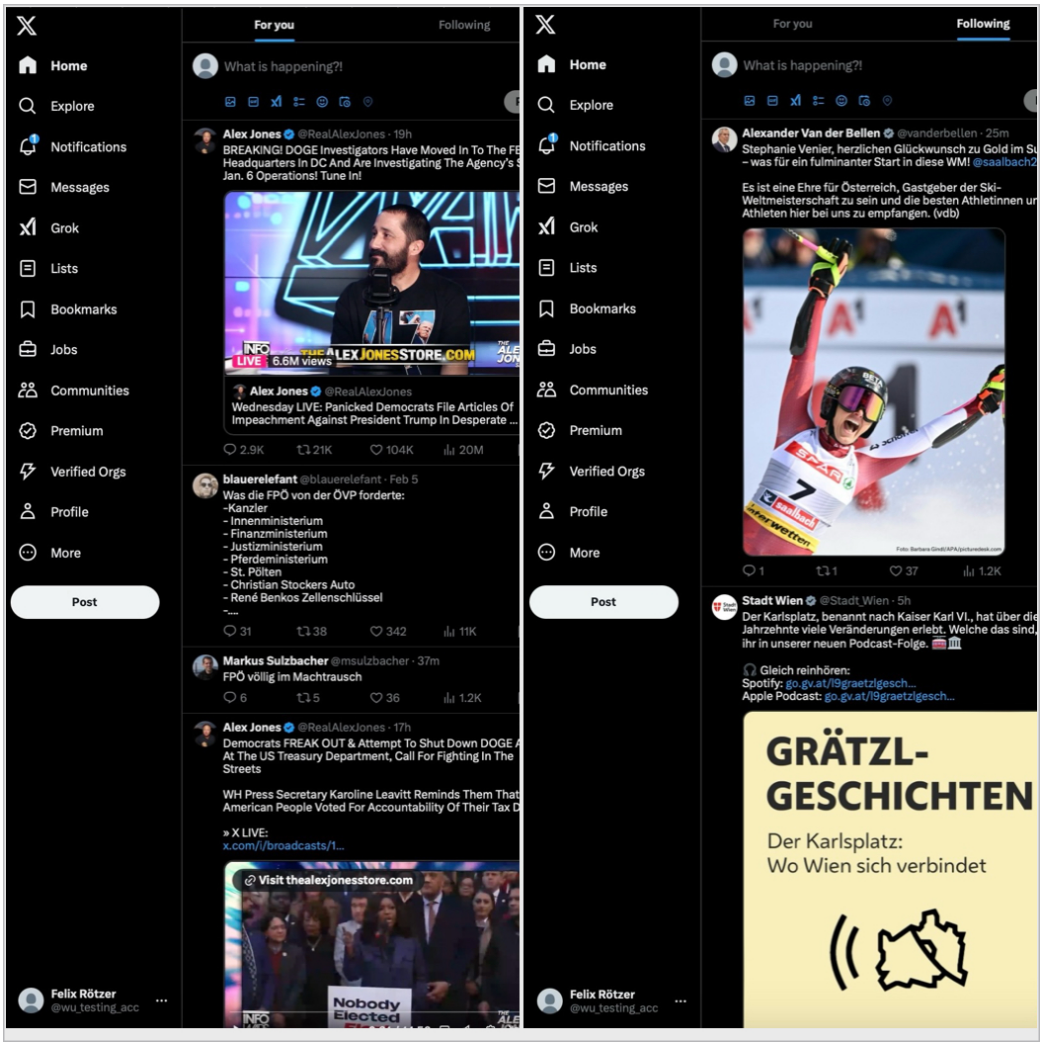

Figure 7: Examples of "For you" and "Following" timelines for a new Twitter account, Source: `https://www.twitter.com`

In Figure 7 we can see a visual comparison of the two timelines offered by Twitter. The algorithmically curated "For you" timeline is shown on the left, while the "Following" timeline appears on the right. To highlight the observable differences between these timelines, we created a dummy account named 'wu_testing_acc'. Moreover, we chose to follow only three profiles - 'Alexander Van der Bellen', 'Stadt Wien' and 'Armin Wolf' - in order to simplify the setup, making it easier to isolate and analyze the distinct behaviours of each timeline.

We can observe that in the "Following" timeline, only posts from the

followed accounts are displayed - specifically from 'Alexander Van der Bellen' and 'Stadt Wien'. These posts are clearly ranked in reverse-chronological order: the most recent post appeared 25 minutes ago, followed by another published five hours earlier.

In contrast, the "For you" timeline for the same account does not feature posts from the followed profiles among the top results. Instead, it presents posts in a mixed chronological order, with no clear prioritization of recency or user-follow relationships.

Moreover, it is evident that the top results in the "For you" timeline tend to prioritize highly polarizing content. Notably, among the top results, two posts feature Alex Jones, a well-known American conspiracy theorist, while the other two posts display content related to the Austrian political party "FPÖ", which is generally associated with right-wing ideologies. While the observation aligns with broader patterns of polarization prioritization seen in Twitter's algorithm, it represents a single example and should not be used to draw generalized conclusions.

6.2 What data is necessary for the recommendation pipeline?

In order for the algorithm to function effectively, Twitter must have the necessary data available before initiating the recommendation pipeline. The data is represented in the form of different knowledge graphs (KGs), which can be divided into three primary components, which are visualized in Figure 8 [30].

The social graph represents the network of relationships between users, capturing all following connections. The tweet engagement graph focuses on user interactions with content - such as likes, retweets, replies and impressions - providing insights into how users engage with the platform. Finally, the user data graph stores detailed information about individual users, including their account details. Together, these knowledge graphs enable Twitter to gain a comprehensive understanding of the activity on the platform, which is essential for optimizing content recommendations and identifying the most relevant tweets [30].

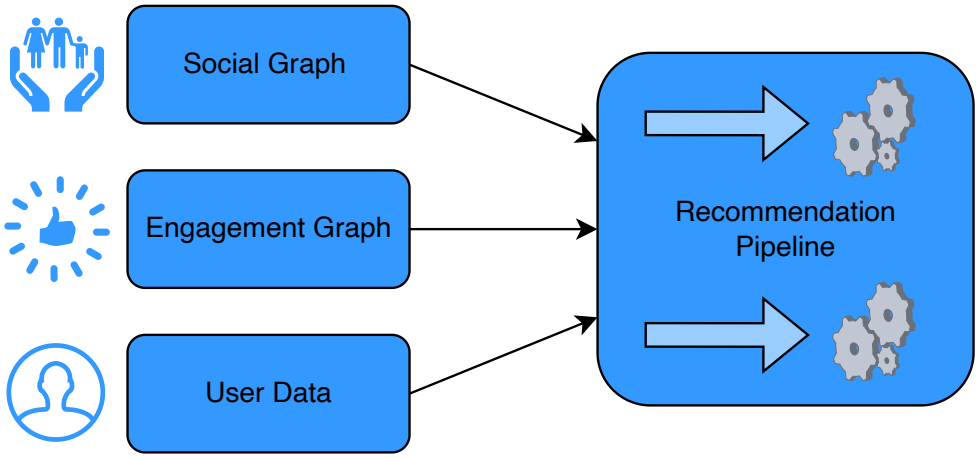

Figure 8: Categories of knowledge graphs powering Twitter's recommendation system, Source: `https://www.blog.x.com/engineering/en_us/topics/open-source/2023/twitter-recommendation-algorithm`

6.3 What metrics can the data be divided into?

Each type of data and its associated factors influece the ranking process in unique ways and carry a distinct weight in the ranking algorithm, significanlty influencing whether or not a post gains exposure.

```
private val RankingParams: scr.ThriftRankingParams = {
  scr.ThriftRankingParams(
    `type` = Some(scr.ThriftScoringFunctionType.TensorflowBased),
    selectedTensorflowModel = Some("timelines_rectweet_replica"),
    minScore = -1.0e100,
    retweetCountParams = Some(scr.ThriftLinearFeatureRankingParams(weight = 20.0)),
    replyCountParams = Some(scr.ThriftLinearFeatureRankingParams(weight = 1.0)),
    reputationParams = Some(scr.ThriftLinearFeatureRankingParams(weight = 0.2)),
    luceneScoreParams = Some(scr.ThriftLinearFeatureRankingParams(weight = 2.0)),
    textScoreParams = Some(scr.ThriftLinearFeatureRankingParams(weight = 0.18)),
    urlParams = Some(scr.ThriftLinearFeatureRankingParams(weight = 2.0)),
    isReplyParams = Some(scr.ThriftLinearFeatureRankingParams(weight = 1.0)),
    favCountParams = Some(scr.ThriftLinearFeatureRankingParams(weight = 30.0)),
    langEnglishUIBoost = 0.5,
    langEnglishTweetBoost = 0.2,
    langDefaultBoost = 0.02,
    unknownLanguageBoost = 0.05,
    offensiveBoost = 0.1,
    inTrustedCircleBoost = 3.0,
    multipleHashtagsOrTrendsBoost = 0.6,
    inDirectFollowBoost = 4.0,
    tweetHasTrendBoost = 1.1,
    selfTweetBoost = 2.0,
    tweetHasImageUrlBoost = 2.0,
    tweetHasVideoUrlBoost = 2.0,
    useUserLanguageInfo = true,
    ageDecayParams = Some(scr.ThriftAgeDecayRankingParams(slope = 0.005, base = 1.0)),
    selectedModels = Some(Map("home_mixer_unified_engagement_prod" -> 1.0)),
    applyBoosts = false,
  )
}
```

Figure 9: Function within Twitter's source code that determines the weighting of ranking parameters, Source: `https://www.github.com/twitter/the-algorithm`

Figure 9 highlights one of the most critical components of Twitter's open-source algorithm: the RankingParams function. While the RankingParams function plays a pivotal role in determining a Tweet's relevance based on specific engagement metrics, it is important to note that this represents only

one aspect of the overall ranking process. The RankingParams score serves more as a Tweet-specific metric rather than the determinant of a Tweet's position in a user's feed. This function encapsulates various engagement metrics or factors, some of them assigned a specific weight, which partially determine how content is ranked on the platform. The scores of different metrics are aggregated together by calculating a sum to get a combined score. The features in the figure that are associated with a weight are processed through a linear conversion function ('ThriftLinearFeatureRankingParams'), which generates a score based on whether the value is high or low. Twitter does not provide the exact conversion values inside their source-code. This score is then multiplied with the associated weight, meaning that a weighted sum is calculated. In contrast, the features without an associated weight are simply added on whether the parameter is true or false, as they do not require weighting. To provide some more understanding of the function parameters shown in Figure 9, we listed some of the most important engagement metrics below, along with their explanation. While different weights are provided

- favCountParams: obtains the number of likes for a given post.
- retweetCountParams: obtains the number of retweets for a given post.
- replyCountParams: obtains the number of comments for a given post.
- ageDecayParams: obtains the recency of a post.
- urlParams: checks if URLs are included in a post.
- tweetHasImageUrlBoost: checks if the post includes an image.
- langEnglishTweetBoost: checks if the post in composed in English language.

As shown in Figure 9 above, the different parameters vary significantly in both their function and impact on ranking. To better understand these differences, we have categorized them into distinct groups based on their characteristics.

Figure 10 below illustrates this categorization, dividing the metrics into two key parts: dynamic vs. static. Dynamic metrics, such as recency and engagement evolve over-time. In contrast, static metrics, such as filtering criteria, remain fixed and are determined by predefined rules. The figure also highlights which parameters from Twitter's 'RankingParams' function fall into each of these categories, providing a clearer comparison. This classification helps us to better understand how various factors influence ranking

and how similar mechanisms could be adapted for decentralized platforms like Mastodon.

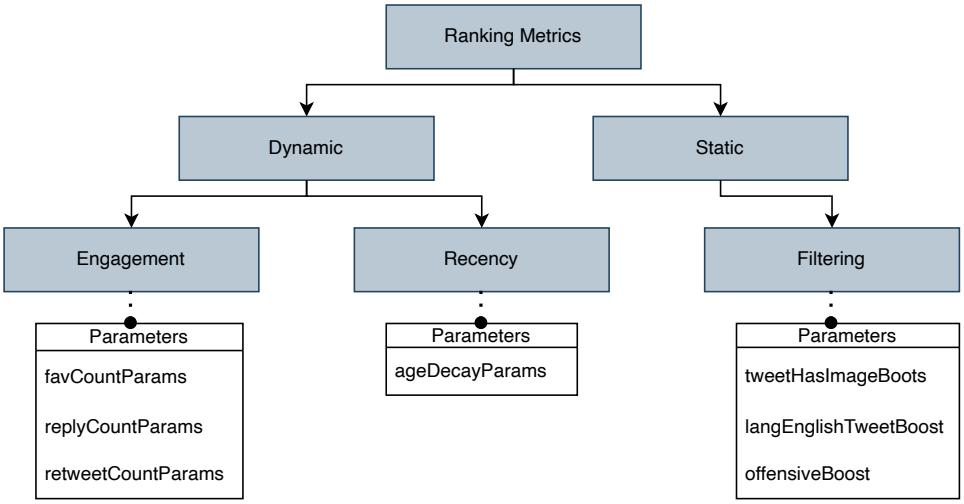

Figure 10: Classification of different ranking metrics.

6.3.1 Engagement

Different types of engagement are often the first category that comes to most people's minds when considering ranking algorithms, as they directly measure how users interact with content. As illustrated in Figure 10, engagement can come in many forms. Likes, comments, and retweets are among the most basic and widely used engagement metrics, offering users a quick and simple way to express approval. In Twitter's 'RankingParams' function, these are among others represented by the parameters 'favCountParams', 'replyCountParams', and 'retweetCountParams'. However, more sophisticated engagement metrics provide deeper insights into user engagement. For instance, these could include whether the person bookmarks or downloads a post, or how long they spent watching a given piece of content.

```
object FavParam
    extends FSBoundedParam[Double](
      name = "scored_tweets_model_weight_fav",
      default = 1.0,
      min = 0.0,
      max = 100.0
    )
```

Figure 11: Twitter source-code function for the like factor, Source: https://github.com/twitter/the-algorithm/blob/main/home-mixer/server/src/main/scala/com/twitter/home_mixer/product/scored_tweets/param/ScoredTweetsParam.scala

In Figure 11, we can see a function that gives a score based on the number of likes. Twitter applies a default score of 1 for each like and sets an upper bound of 100, meaning that all posts with 100 likes or more have the value of 100 for this parameter. Up to 100 likes there is a linear increase in this parameter. By introducing an upper bound for the like-based parameter, Twitter ensures that posts with relatively few likes still retain the potenial to surface in users' feeds. This choice also prevents the parameter from disproportionally dominating the ranking and allows other parameters to contribute to the visibility of a Tweet. In chapter 7, we ensure that our own algorithmic implementations account for this principle by similarly limiting the influence of high values.

6.3.2 Recency

Recency is a critical factor in ranking algorithms, especially in dynamic environments like social media platforms, as it can significantly impact user engagement. As mentioned above, the 'ageDecayParams' parameter shown in Figure 10 is the primary component in determining the influence of recency in Twitter's timeline ranking algorithm [30]. This parameter applies a time-based decay over time, meaning that a post's relevance diminishes the longer it has been published.

```
struct ThriftAgeDecayRankingParams {
  // the rate in which the score of older tweets decreases
  1: optional double slope = 0.003
  // the age, in minutes, where the age score of a tweet is half of the latest tweet
  2: optional double halflife = 360.0
  // the minimal age decay score a tweet will have
  3: optional double base = 0.6
}(persisted='true')
```

Figure 12: Twitter source-code function for the age decay factor, Source: `https://github.com/twitter/the-algorithm/blob/main/src/thrift/com/twitter/search/common/ranking/ranking.thrift`

In Figure 12 we see a function from Twitter's source-code for calculating the score of the age decay. The function uses a decay rate of 0.003 and a decay half-life of 360 minutes. The source-code does not show the exact formula to calculate the score with these parameters. However, the half-life parameter clearly indicates a non-linear decay given that the score of a Tweet halves every 360 minutes. Moreover, it includes the parameter that sets a minimal age decay score. This prevents old posts from getting completely irrelevant. We will use a similar approach in our own algorithmic implementations later in this paper.

An example where recency is the sole factor is reverse chronological post-ordering. This method is the default mechanism on Mastodon's local timeline [17], as well as on Twitter's "Following" page [30]. In this approach, posts are displayed strictly in the order they were published, with the most recent content appearing first [12].

6.3.3 Filtering

Filtering is an inherent property of a post - fixed and automatically applied based on predefined criteria that depend on the type of content. Due to the category's static nature, it represents a clear contrast to the other two categories, engagement and recency, which are dynamic factors that can be subject to change over time. Filtering for content types is a broad concept with numerous possible specifications that can vary significantly across different social media platforms and regions, depending on distinct content policies, user behaviour, and organizational objectives. In the case of Twitter's algorithm, we can certainly notice a variety of filters, as illustrated in Figure 10.

For instance, the 'tweetHasImageUrlBoost' parameter evaluates whether a post includes an image URL and adjusts its ranking accordingly. Similarly,

language-based filters such as the 'langEnglishTweetBoost' and 'unknownLanguageBoost' parameter modify a post's weight depending on whether it is written in English or an unrecognized language. In addition to the language of the post, the location of the user who published it can also serve as a ranking metric. Additionaly, parameters like 'offensiveBoost' - which accounts for potentially harmful content - and 'multipleHashtagsOrTrendBoost' - which considers the number of hashtags - demostrate the diverse and nuanced filtering mechanisms integrated into ranking algorithms.

6.4 What does the recommendation pipeline look like?

In Figure 13 below, we can see a simplified model of Twitter's recommendation pipeline. Within the codebase of Twitter's algorithm, this section is referred to as the 'HomeMixer', which is responsible for delivering the most relevant Tweets to a user's "For you" timeline [30]. As such, it represents the most critical component for understanding the dynamics of ranking algorithms in the context of our research question.

As illustrated in the figure, this process can be structured into three main stages - namely canditate sourcing, ranking and finally applying heuristics and filters - through which every candidate must pass before appearing in a user's feed. Every day, approximately 500 million posts are made on Twitter, forming a vast pool of candidate Tweets that enter the recommendation pipeline [31]. The Mixing stage, shown on the right side of the figure, is a final step where ads are integrated into the timeline. Since this process is not relevant to our research focus, we do not discuss it in detail.

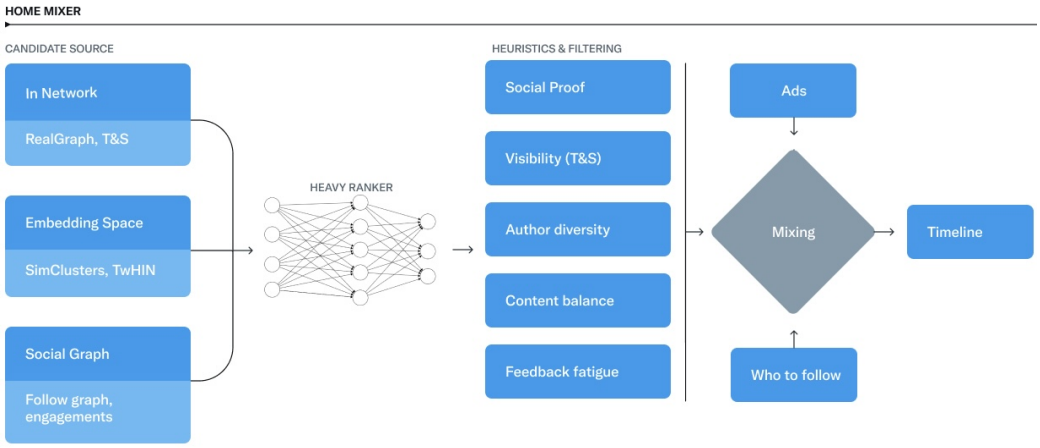

Figure 13: Recommendation pipeline of the Twitter algorithm, Source: `https://www.blog.x.com/engineering/en_us/topics/open-source/2023/twitter-recommendation-algorithm`

Candidate Sourcing is the process through which Twitter's algorithm gathers the most promising posts for a given user before applying any ranking. For each committed request, this process selects a pool of 1,500 Tweets [31]. This selection can be further categorized into two distinct parts: in-network sources, which consist of content from accounts the user follows, and out-of-network sources, which include Tweets from accounts outside

the user's direct follow connections. Each of the two categories contributes roughly 50 % to the final pool [31].

The in-network sources are selected using the so-called 'RealGraph', a model designed to predict the chance of interaction between two users [31]. This approach is deeply rooted in collaborative filtering principles, which also highlight the similarity between users, as previously mentioned in the background section about recommender systems.

On the other hand, the out-of-network sources presents a more complex challenge given that is requires identification of relevant content from users with whom there is no direct follow connection. Therefore, Twitter uses two approaches for addressing this: the 'Social Graph' and 'Embedding Spaces'. The Social Graph approach tries to estimate relevant content based on the analysis of engagement patterns among a user's network, focusing on interactions with followed accounts. This method also has a strong connection to the recommender systems approach of collaborative filtering, as it relies on the engagement of similar users - where similarity is assumed based on one user following another. On the other hand, the approach of Embedding Spaces generates numerical representations of users interests and Tweets. Therefore, Twitter implemented a tool called 'SimClusters', which clusters related types of content. 'Embedding Spaces' is therefore very closely related to content-based filtering, as it computes the similarity between items. As we can see, Candidate Sourcing employs a hybrid approach to recommender systems, combining both collaborative-filtering and content-based filtering techniques. This aligns with established knowledge that hybrid models are among the most effective strategies for personalized recommendations [27].

After the first selection process with 'Candidate Sourcing', the 1500 candidates are then being ranked to serve the most relevant Tweets first. Ranking is mostly done with what Twitter calls the 'Heavy Ranker', which is a large, neural network-based algorithm with 48 million parameters [31]. The neural network is constantly trained on user interactions and leverages engagement metrics - such as those outlined in the previous chapter - to optimize for positive engagement.

'Heuristics and Filtering' represent the final stage of the recommendation pipeline, which helps to adjust the output of the 'Heavy Ranker'. Examples include filters that remove Tweets from blocked users, filters that assure consecutive Tweets are not from a single author or filters that assure the in-network and out-of-network Tweets are fairly balanced.

7 Mastodon for Ranking Algorithms

This section provides insights into the technology and functionality of the decentralized social platform Mastodon. Therefore, we will provide comprehensive documentation about the process of integrating a new Mastodon server, along with the various implementation possibilities for its setup and configuration. Additionally, we will use insights from our analysis of conventional algorithmic approaches, such as Twitter's, to make informed assumptions about useful configurations for the Mastodon server. Using these assumptions, we will develop and test various ranking algorithms for the platforms, which will be publicly available through a dedicated GitHub repository.

Furthermore, we aim to provide the necessary documentation to enable other instance owners and creators to implement open ranking algorithms on their own servers, fostering greater transparency and customization within decentrlaized social media.

7.1 Tech Stack

In the following paragraphs, we describe in detail which hardware setup was used for the implementation of the Mastodon server, and what the server environment looked like as well as the software that was necessary to successfully integrate the server. Throughout this process, we closely followed the official Mastodon documentation to ensure a reliable and properly configured deployment [17].

7.1.1 Hardware and Infrastructure

Given the fact that a Mastodon server should always run and thus be connected to the internet, it was necessary to provide a fitting server infrastructure. In our case, we were provided a dedicated server container from Vienna University of Business and Economics just for the testing purposes of this research project.

The container is configured with 128GB of storage, 16GB of RAM and a 4-core CPU, providing the necessary computational resources for our needs. It should be noted that if greater scalability and adoption on WU campus is required, these resources would need to be expanded. However, for testing purposes, this configuration is sufficient. The server was accessible via SSH login exclusively from the university's internal network, which included access through the WU-Wifi network, the WU-Lan, and the WU-VPN service, which ensured a secure and controlled environment.

Additionally, to meet Mastodon's system requirements, the server runs Ubuntu 24.04 LTS, a long-term support release that ensures compatibility with the Mastodon software stack. Ubuntu is a Linux-based operating system known for its stability, security and strong community support. A dedicated web domain on which the Mastodon server could publicly be accessed was also necessary. Therefore, the university provided the domain 'social.ai.wu.ac.at', which is part of the official domain structure of WU Vienna. To manage incoming web traffic efficiently, an Nginx instance was set up as a reverse proxy. The domain was configured to point to the correct PORT of our Nginx instance running on our server container, ensuring proper request handling and secure access.

7.1.2 Software

After setting up the server environment correctly and ensuring that both the hardware and domain were properly configured, the next step was to install the necessary software on our virtual machine (VM). Firstly, we had to

prepare the system environment. Therefore, we configured two system repositories, while also creating a dedicated Mastodon user and installing necessary software packages. The system repositories were configured for Node.js and PostgreSQL. The Mastodon user inside the environment was created as a repository to conduct the actual Mastodon installation. A detailed list of the required system packages can be found in the figure below. Among these, Redis holds particular significance [23]. With its in-memory data storage, Redis is essential for our algorithmic implemenations and code explanations. This is because our algorithms retrieve posts from its short-term memory to construct the correct feed, and allow an efficient feed generation process.

```
apt update
apt install -y \
  imagemagick ffmpeg libvips-tools libpq-dev libxml2-dev
  ↪ libxslt1-dev file git-core g++ libprotobuf-dev
  ↪ protobuf-compiler pkg-config gcc autoconf bison
  ↪ build-essential libssl-dev libyaml-dev libreadline6-dev
  ↪ zlib1g-dev libncurses5-dev libffi-dev libgdbm-dev nginx
  ↪ nodejs redis-server redis-tools postgresql
  ↪ postgresql-contrib certbot python3-certbot-nginx
  ↪ libidn11-dev libicu-dev libjemalloc-dev
```

The next crucial part after preparing the system environment was to create a PostgreSQL database, which stores our instance's essential data. Lastly, we installed the latest version of Mastodon through their official Github repository inside the Mastodon user we created, to ensure all Mastodon-related files and processes were isolated from the rest of the system [17].

7.2 Algorithmic Implementations: Enabling Personalized Social Media Feeds

In this section, we explore the algorithmic implementations designed to provide personalized social media feeds within decentralized platforms. As part of this project, we created an open-source GitHub repository that enables other Mastodon instance owners to integrate these algorithms within their own servers. To achieve this, modifications were made to the Mastodon source-code, ensuring seamless integration of the algorithms into the platform's content delivery pipeline. Identifying the exact sections of the code that required modification proved to be a challenge, given that the Mastodon codebase consists of hundreds of files and an extensive number of lines of code. However, various algorithms have been developed, allowing server administrators to select the approach that fits their community's needs.

The GitHub repository is named 'Mastodon Algorithms', which can be found in the references [26]. The repository also includes implementation options that might not be pratical to use, however, we added them to showcase a variety of examples. This includes changing the default home timeline to show all the server posts, which can be found in the 'default_local' directory. Also, we implemented a method that offers a home timeline that is strictly ranked based on the number of likes of the underlying posts, which is also highly unpratical. This method can be found in the 'favourite_simple' directory.

However, for practial reasons, we selected two primary algorithms, which we will showcase in detail in this section. The two approaches are the 'N-Degree Feed Algorithm' and the 'Engagement Algorithm with Time Decay Factor'. The corresponding implementations can be found in the project's GitHub repository under the directories 'second-degree' and 'mixed_algo', respectively. The code snippets provided to analyze the logic behind the algorithm are written in Python, a language chosen for its readability and widespead use, ensuring accessibility to a broad audience. However, the actual implementation and source code modifications within Mastodon are carried out in Ruby, given that the Mastodon source-code is mainly done in Ruby. As mentioned, the actual Ruby implementations can be found in GitHub and can be modified if needed [26]. Furthermore, since the various components of the codebase are highly interdependent, it was more practical to explain the algorithms using Python rather than representing exact snippets from the actual Ruby implementation. Understanding the algorithm solely from a Ruby section from the actual code would require extensive background knowledge of the entire Mastodon code.

7.2.1 N-Degree Feed Algorithm

The N-Degree Feed Algorithm enables users to view posts from their direct connections (first degree) and extended networks (up to degree N of seperation). This approach fosters content discovery without overwhelming users with distant and less relevant posts. By default, the algorithm is set to two degrees, striking a balance between computational performance and content diversity. However, server administrators can adjust the degree parameter to suit their platform's requirements.

In the code snippet presented below, we can observe the underlying logic and step-by-step execution of the algorithm, illustrating its functionality and key processes.

```
1  def get_n_degree_feed(user_id, posts, following_matrix,
   ↪  limit=10, degree=2):
2
3      all_ids = {user_id}
4      current_degree_ids = {user_id}
5
6      for _ in range(degree):
7          next_degree_ids = set()
8          for uid in current_degree_ids:
9              next_degree_ids.update(following_matrix.get(uid,
               ↪  []))
10         next_degree_ids -= all_ids
11         all_ids.update(next_degree_ids)
12         current_degree_ids = next_degree_ids
13
14     feed = [post for post in posts if post["user_id"] in
       ↪  all_ids]
15
16     feed = sorted(feed, key=lambda x: x["timestamp"],
       ↪  reverse=True)[:limit]
17
18     return feed
```

The function 'get_n_degree_feed' requires several key parameters to generate a personalized feed based on a user's social connections, which can be found in line 1. The parameter 'user_id' is an integer that represents the unique identifier of the user for whom the feed is generated. The 'posts'

parameter is a list of dictionaries, where each dictionary represents a post containing attributes such as 'id', 'user_id', 'content' and 'timestamp'. The 'following_matrix' is a dictionary that maps each user's ID to a list of user IDs they follow, allowing the algorithm to traverse social connections up to a specified degree. The 'limit' parameter is an integer that restricts the number of posts returned in the final feed, with a default value of 10 in our case. Finally, the 'degree' parameter is an integer that determines the number of connection layers included in the feed, with a default value of 2 to limit computational effort.

Firstly, we initialize 'all_ids' and 'current_degree_ids', which can be observed in lines 3 and 4. At the beginning, they both only contain the user id parameter value. In line 6, we start to iterate through the number of degrees that was specified in the parameters. Each iteration is initialized with a set called 'next_degree_ids' that will be used to obtain the user ids of the current iteration. Then, we loop through each of the user ids and use the following matrix to observe the followed users of each user id, adding it to the next_degree_ids. In line 10, we remove the user ids from 'all_ids' from 'next_degree_ids' to make sure that all duplicates are removed. The 'all_ids' variable is finally updated in each iteration, to make sure we found all the user ids. Using a list comprehension, we can finally get all the posts from the user ids we observed in the previous step. Finally, we sort the posts by reverse chronological post ordering, which we can inspect in line 16.

To implement this algorithm into the Mastodon source-code, major changes were made inside a Ruby file named 'home_feed.rb'. Apart from this file, no other adaptations were necessary. In specific, we changed the 'from_redis' function, renamed it to 'get_second_degree_feed' and made the required changes.

```
1  def from_redis(limit, max_id, since_id, min_id)
2    max_id = '+inf' if max_id.blank?
3    if min_id.blank?
4      since_id   = '-inf' if since_id.blank?
5      unhydrated = redis.zrevrangebyscore(key, "(#{max_id}",
       ↪  "(#{since_id}", limit: [0, limit], with_scores:
       ↪  true).map { |id| id.first.to_i }
6    else
7      unhydrated = redis.zrangebyscore(key, "(#{min_id}",
       ↪  "(#{max_id}", limit: [0, limit], with_scores: true).map
       ↪  { |id| id.first.to_i }
8    end
```

```
9
10    Status.where(id: unhydrated)
11  end
```

In the code from above, we can see how the data from Redis is sorted by its score inside the 'from_redis' function. In the general case, this score is just based on the age of the Redis entry. Basically, the 'unhydrated' variable stores the ids in chronological order and then the 'Status' function is called from a different section of the code, to retrieve the underlying posts of the ids we observed. We included this section to highlight the difference in syntax between the actual Mastodon code and a more general algorithmic representation, as well as to showcase a section of the actual Mastodon source-code. However, the explanatory value of this specific snippet is relatively low, as it represents only a small part of the entire logic. For further exploration, visit the Mastodon source-code [17], as well as the code adaptions on our GitHub [26].

7.2.2 Engagement Algorithm with Time Decay Factor

To create a more realistic simulation of modern social media platforms like Twitter, we developed a more sophisticated algorithm that blends the local reverse chronological ordering with algorithmic ranking. The design of this algorithm is inspired by the insights we got in the previous chapter about Twitter's 'For you' timeline. We aimed to design the algorithm in a way that approximately one-third of the content is ranked based on global algorithmic scoring, while the remaining two-thirds follow the default chronological timeline of Mastodon displaying posts from followed users. As discussed in previous chapters, Twitter, by contrast, balances its feed with an equal split between global and local content. This balance seeks to maintain the relevance and engagement benefits of algorithmic curation, while still having a strong focus on a person's following network. It is important to emphasize that this implementation reflects just one possible configuration of the algorithm, which can be adapted or extended to suit different objectives. With the help of the code snippet below, we will provide a detailed explanation of the algorithm that was implemented in the Mastodon source-code.

```
import math
from datetime import datetime, timedelta

def get_algorithmic_feed(global_posts, local_feed):

    decay_rate = 3600 * 24 * 10
    current_time_unix = int(datetime.now().timestamp())

    algo_feed = []
    for post in global_posts:
        post_id = post["id"]
        likes = post["likes"]
        created_at_unix = int(post["created_at"].timestamp())
        time_diff = current_time_unix - created_at_unix

        time_decay_factor = math.exp(-time_diff / decay_rate)
        like_factor = math.sqrt(max(likes, 1))

        combined_score = like_factor * time_decay_factor

        algo_feed.append({
            "post_id": post_id,
```

```
23              "combined_score": combined_score
24          })
25
26      algo_feed.sort(key=lambda x: x["combined_score"],
        ↪  reverse=True)
27
28      return algo_feed
```

As a first step, we needed to import the necessary packages, in our case this was the 'math' package and the 'datetime' package. The function for this algorithm only required us to include two parameters. The first argument is 'global_posts', which is expected to be a dictionary containing keys such as 'id', 'likes', and 'created_at' of all the posts from the global timeline. The second argument is 'local_feed', which contains all the posts from Mastodon's default local timeline. On line 6, the 'decay_rate' is defined as the number of seconds in ten days. This value is determines how quickly a post's score decays over time and can be adapted accordingly, to give the age a stronger or weaker weight in comparison to the number of likes. Longer periods result in slower decay. Then, we set the 'current_time_unix' to the current time in seconds since unix epoch, to later be able to calculate the age in seconds since the post was created. The next step was to initialize the 'algo_feed' list, before starting the iteration over all the posts using a for loop on line 10. For each iteration we calculated a 'combined_score', based on the like count and the age of a post.

The recency was factored in with the 'time_decay_factor', observable in line 16, which gradually declines with an increase in the age. The variable is incorporated in a non-linear manner. In particular, the factor decreases exponentially over time. This implies that more recent posts receive a proportionally higher 'time_decay_factor' rating. The non-linear approach for the recency factor is inspired by the insights from chapter 6 about the Twitter algorithm. Other work such as the contributions from Filipovic also suggests a non-linear approach [9].

The engagement was factored in with the 'like_factor' variable, that increases with the amount of likes. This variable is also incorporated in a non-linear manner. Specifically, likes are included using a root function, such that the marginal contribution of each additional like decreases as the total number of likes increases. This approach ensures that Tweets with fewer likes are not disproportionally excluded from the ranking, allowing for a more balanced consideration of engagement. The approach is inspired by the insights from the Twitter algorithm in chapter 6. In the Twitter algorithm a

hard upper bound is applied to the like factor. While we adapt this idea by using a root function instead of a fixed cap, both methods have the purpose of limiting the dominance of highly liked content. Also, the 'like_factor' was given a lower bound of 1, to make sure posts with 0 likes are also factored into the feed calculation. To finally get the algorithmic timeline, we just added all the results to the 'algo_feed' list and sorted the list based on the 'combined_score' in descending order.

It is important to emphasize

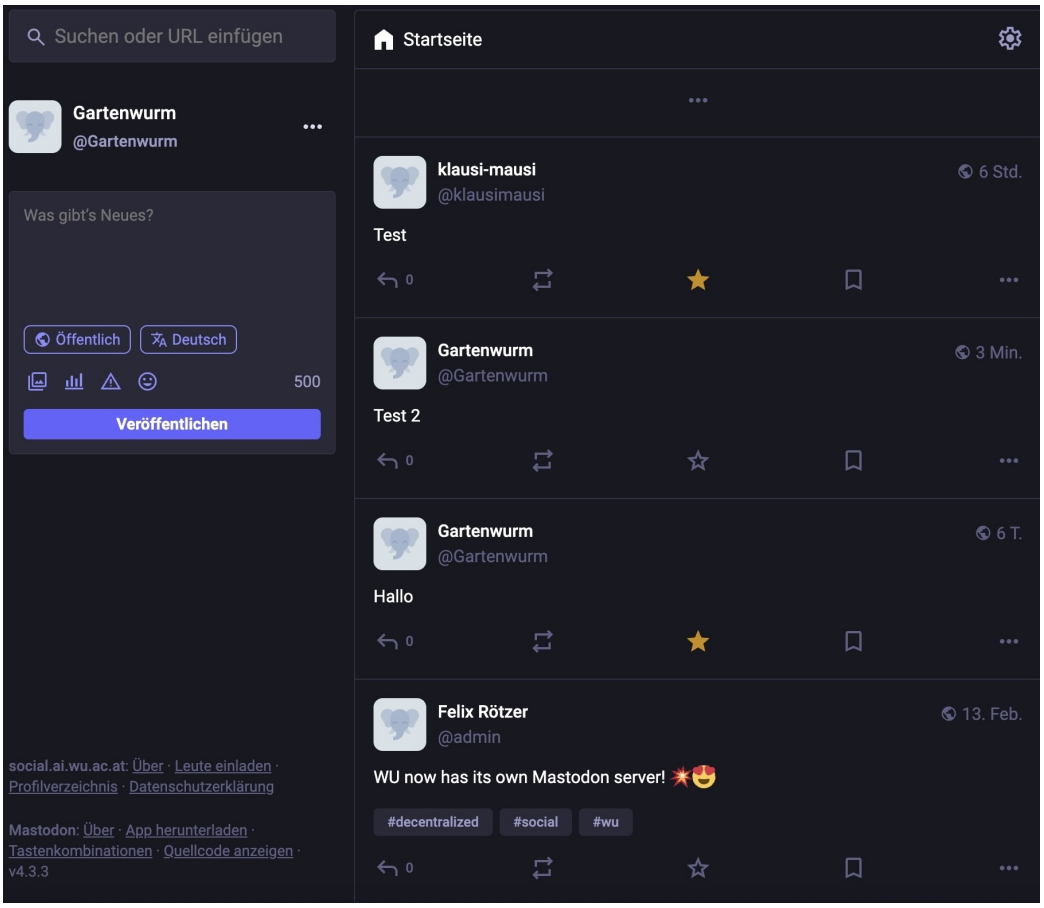

Figure 14: Sample Timeline of the Engagement Algorithm with Decay Factor, Source: `https://social.ai.wu.ac.at/home`

In Figure 14, we can observe an example of a timeline that utilizes the Engagement Algorithm with Decay Factor. Since engagement, measured in likes, is combined with an age-based decay factor, the posts are not strictly

sorted by age. This is clearly visible in the example, given that the post 'Test' is ranked above 'Test 2', eventhough it was posted 6 hours ago compared to 3 minutes ago. The specific scores used for ordering these two posts, which are derived from the algorithm presented above, are shown in Figure 15 below. The figure shows output from the Rails console inside our server environment. The 'Test' post has received two likes, as indicated by the 'favourites' parameter, whereas the 'Test 2' post has zero likes. On the other hand, 'Test 2' is higher in the 'created_at' parameter, indicating a later unix time. Finally, the 'combined_score' parameter shows 1.375 for the 'Test' post in comparison to 0.998 for the 'Test 2' post and is therefore ranked first.

Figure 15: Score Outputs for Engagement Algorithm

8 Related Work

The research on the technical architecture of ranking algorithms in centralized social media platforms has historically been relatively limited due to the closed-source nature of most platforms. The recent open-sourcing of Twitter´s algorithm [30] has led to more research in the field, however much of the recent work has focused more on the social and ethical implications of Twitter´s algorithm such as the promotion of low-credibility content [7] or the polarizing architecture the platform has [18]. Despite that, significant work has been done in related domains such as recommender systems, which social media ranking algorithms are partly based upon. One research paper that stand out in providing an overview of the state of research on recommender systems (RS) is 'A systematic review and research perspective on recommender systems' [27]. Research on decentralized social media, has seen a strong push since Mastodon was founded back in 2016. Most of the research that has been conducted focused on their architecture [38] and what challenges this form of social media usually comes with [22]. Even though there have been a reasonable number of contributions, studies exploring how open ranking algorithms could increase performance and reach of decentralized platforms have not yet been carried out yet. However, there have been some efforts on constructing alternative content-ranking algorithms to mitigate some of the problematic influential mechanisms centralized platforms come with [13], which could potentially benefit decentralized social media platforms. This thesis builds on the foundations of previous research, by investigating the potential of open ranking algorithms for platforms like Mastodon by analyzing common ranking methods used on centralized platforms.

9 Limitations and Future Work

The server-based algorithm design may not be the optimal solution for all users, as some individuals might prefer more personalized, user-specific ranking implementations tailored to their unique preferences. Moreover, the approaches developed in this paper can only be applied by instance owners, limiting accessibility for individual users. These aspects represent key limitations of the approach presented in this paper.

Building on the algorithmic implementations and insights presented in this thesis, future work could explore the development of community-driven and user-based algorithmic approaches. This could, for example, be achieved relatively simply through profile metadata, which each user can specify in the frontend of their profile section in Mastodon. The text fields for the profile metadata could be used to provide algorithmic instructions or reflect individual preferences for personalizing the timeline, tailoring the content to each user's unique interests and interactions. To make the implementation user-friendly, it would make sense to offer a selection of predefined instructions, each representing different ranking approaches, such as 'strict chronological', 'algorithmic', which users can enter and customize, allowing them to quickly choose their preferred timeline. To implement this, the data within the profile metadata would need to be fetched from each user and integrated into the ranking logic of Mastodon's codebase.

Such an approach would also enable scalable quantitative analysis of algorithmic preferences of users in future research, as the metadata entered by all users could be analyzed to identify patterns and trends in user preferences. This would provide valuable insights into the effectiveness of custom ranking algorithms, fostering continuous refinement and optimization based on real-world user preferences. Building upon the implementations from this thesis, the extension of algorithms to provide greater flexibility in customization would be an important area for future work.

In the following list we summarized the most promising topics for future work based on limitations of this paper:

- Integration of personalized algorithmic customization through profile metadata: Implementing profile metadata that allows users to input preferences or select ranking approaches to personalize their timeline.
- Analysis of user preferences to refine and optimize ranking approaches: Collecting and analyzing metadata from users to identify common preferences and enhance ranking algorithms accordingly.
- Expansion of algorithm selection options: Offering a broader range of

ranking approaches that can be easily customized by users.

- Scalable and community-driven algortihmic solutions: Exploring alternative ways to integrate community feedback and preferences, promoting collective decision-making in ranking methods.
- Testing and validating custom ranking models on larger datasets: Conducting empirical research to test effectiveness of custom ranking algorithms on a large Mastodon instance.

10 Conclusion

In this paper, we have demonstrated how open ranking algorithms can be useful for decentralized social media platforms, specifically through our implementation and analysis within the Mastodon ecosystem. The openness of decentralized protocols, particularly ActivityPub - the foundation of Mastodon - made it an ideal platform for this research, providing the flexibility and transparency needed to implement and evaluate these ranking algorithms effectively. By analyzing the source-code of the Twitter algorithm, we gained insights into how ranking mechanisms in major social media platforms generally work and explored possible ways to customize them. The Twitter algorithm reveals a complex ranking pipeline, where metrics are divided into multiple layers such as recency, engagement, and other contextual factors. This stands in stark contrast to Mastodon's default reverse chronological feed, which prioritizes content based solely on recency. The developed algorithms, implemented through direct modifications in the Mastodon source code, provide instance owners with a straightforward plugin solution to tailor content ranking, allowing for easy customization and integration. This study highlights how such implementations could pave the way for more personalized algorithmic experiences on the Mastodon platform, while still keeping the principles of user data privacy, decentralization, and transparency that define the ecosystem. This work represents a pratical step toward making platforms like Mastodon more accessible and appealing to mainstream users.

References

[1] R. R. Abbing and R. W. Gehl. Shifting your research from X to Mastodon? Here´s what you need to know. *Patterns*, 5(1):100914, 2024. DOI: 10.1016/j.patter.2023.100914.

[2] H. Al-bashiri, M. Abdulhak, A. Romli, and F. Hujainah. Collaborative Filtering Recommender System: Overview and Challenges. *Journal of Computational and Theoretical Nanoscience*, 23(9):9045–9049, 2017. DOI: 10.1166/asl.2017.10020.

[3] R. A. Arguedas, C. Robertson, R. Fletcher, and R. Nielsen. Echo chambers, filter bubbles, and polarization: a literature review. Research report, Reuters Institute for the study of journalism, University of Oxford, 2022. DOI: 10.60625/risj-etxj-7k60.

[4] Bluesky. Source code for the AT-protocol, 2022. GitHub Repository, URL: `https://github.com/bluesky-social/atproto`, Accessed Jan 30, 2025.

[5] Bluesky. Bluesky Homepage, 2024. URL: `https://www.bsky.social`, Accessed on Feb 23, 2024.

[6] Bluesky. AT protocol, 2025. URL: `https://atproto.com`, Accessed Feb 27, 2025.

[7] G. Corsi. Evaluating Twitter´s algorithmic amplification of low-credibility: an observational study. *EPJ Data Science*, 13(18), 2024. DOI: 10.1140/epjds/s13688-024-00456-3.

[8] D. Di Placido. The X (Twitter) Exodus To Bluesky, Explained, 2024. Forbes, URL: `https://www.forbes.com/sites/danidiplacido/2024/11/19/the-x-twitter-exodus-to-bluesky-explained`, Accessed Mar 8, 2025.

[9] M. Filipovic, B. Mitrevski, D. Antognini, E. Glaude, D. Faltings, and C. Musat. Modeling Online Behavior in Recommender Systems: The Importance of Temporal Context, 2021. DOI: 10.48550/arXiv.2009.08978.

[10] D. Goldberg, D. Nichols, B. Oki, and D. Terry. Using Collaborative Filtering to Weave an Information Tapestry. *Commun. ACM*, 35(12):61–70, 1992. DOI: 10.1145/138859.138867.

[11] W. Hong and J. Y. Thong. Internet Privacy Concerns: An Integrated Conceptualization and Four Empirical Studies. *MIS Quarterly*, 37(1):275–298, 2013. DOI: 10.25300/MISQ/2013/37.1.12.

[12] M. Kleppmann, P. Frazee, J. Gold, J. Graber, D. Holmgren, J. Johnson, B. Newbold, and J. Volpert. Bluesky and the AT Protocol: Usable Decentralized Social Media. In *Proceedings of the ACM Conext-2024 Workshop on the Decentralization of the Internet*, CoNEXT 2024, pages 1–7. ACM, 2024. DOI: 10.1145/3694809.3700740.

[13] T. Koree. Good Social Media: Constructing an Alternative Content-Ranking Algorithm for Social Network Sites, 2023. Bachelor Thesis, URL: `http://essay.utwente.nl/94370`, Accessed Mar 21, 2025.

[14] D. Laseur. What is decentralized social media? Pros and Cons, 2022. Flatline Agency, URL: `https://www.flatlineagency.com/blog/what-is-decentralized-social-media`, Accessed on Nov 23, 2024.

[15] Paddy Leerssen. The Soap Box as a Black Box: Regulating Transparency in Social Media Recommender Systems. *European Journal of Law and Technology*, 11(2), 2020. DOI: 10.2139/ssrn.3544009.

[16] M. Masnick. Protocols, Not Platforms: A Technological Approach to Free Speech. Research report, URL: `https://perma.cc/MBR2-BDNE`, accessed dec 2, 2024, Knight First Amendment Institute, Columbia University, 2019.

[17] Mastodon. Mastodon homepage, 2024. URL: `https://www.joinmastodon.org`, Accessed on Feb 23, 2024.

[18] S. Milli, M. Caroll, S. Pandey, Y. Wang, and A. Dragan. Twitter´s Algorithm: Amplifying Anger, Animosity, and Affective Polarization. *ArXiv*, 2023. DOI: 10.48550/arXiv.2305.16941.

[19] M. Montaner, B. Lopez, and P. Esteva. A Taxonomy of Recommender Agents on the Internet. *Artificial Intelligence Review*, 19(4):285–330, 2003. DOI: 10.1023/A:1022850703159.

[20] A. Narayanan. Understanding Social Media Recommendation Algorithms. Research report, URL: `https://perma.cc/F3NP-FEQX`, accessed jan 15, 2025, Knight First Amendment Institute, Columbia University, 2023.

[21] S. Perez. Twitter says few users have opted out of its new, algorithmic timeline, 2016. TechCrunch, URL: https://techcrunch.com/2016/03/18/twitter-says-few-users-have-opted-out-of-its-new-algorith\mic-timeline, Accessed Feb 5, 2025.

[22] A. Raman, S. Joglekar, E. D. Cristofaro, N. Sastry, and G. Tyson. Challenges in the Decentralised Web: The Mastodon Case. In *Proceedings of the Internet Measurement*, IMC 2019, pages 217–229. Association for Computing Machinery, 2019. DOI: 10.1145/3355369.3355572.

[23] Redis. Redis Homepage, 2025. URL: https://redis.io, Accessed Mar 03, 2025.

[24] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, pages 175–186, 1994. DOI: 10.1145/192844.192905.

[25] B. Rimé. Emotion Elicits the Social Sharing of Emotion: Theory and Empirical Review. *Emotion Review*, 1(1):60–85, 2009. DOI: 10.1177/1754073908097189.

[26] F. Roetzer. Mastodon Algorithms, 2025. GitHub Repository, URL: https://github.com/felixroetzer/mastodon_algorithms, Accessed Feb 24, 2025.

[27] D. Roy and M. Dutta. A systematic review and research perspective on recommender systems. *Journal of Big Data*, 9(59), 2022. DOI: 10.1186/s40537-022-00592-5.

[28] M. Ruwe. The difference between centralized social media platforms and upcoming decentralized social media platforms: a comparative study, 2023. Bachelor Thesis, URL: https://essay.utwente.nl/95357, Accessed Dec 30, 2024.

[29] G. Shani, A. Meisles, Y. Gleyzer, L. Rokach, and D. Ben-Shimon. A Stereotypes-Based Hybrid Recommender System for Media Items. In *Intelligent Techniques for Web Personalization and Recommender Systems in E-Commerce - Papers from the 2007 AAAI Joint Workshop, Technical Report*, pages 76–83, 2007. URL: https://aaai.org/papers/ws07-08-009-a-stereotypes-based-hybrid-recommender-system\-for-media-items, Accessed Jan 12, 2025.

[30] Twitter. Source code for Twitter´s recommendation algorithm, 2023. GitHub Repository, URL: https://github.com/twitter/the-algorithm, Accessed Jan 21, 2025.

[31] Twitter. Twitter's Recommendation Algorithm, 2023. URL: https://blog.x.com/engineering/en_us/topics/open-source/2023/twitter-recommendation-algorithm, Accessed Jan 4, 2025.

[32] W3C. Activity streams: A format for representing activities, 2017. URL: https://www.w3.org/TR/activitystreams-core, Accessed Jan 12, 2025.

[33] W3C. Websub: A protocol for real-time notifications on the web, 2017. URL: https://www.w3.org/TR/websub, Accessed Jan 12, 2025.

[34] W3C. ActivityPub: A protocol for decentralized social networking, 2018. URL: https://www.w3.org/TR/activitypub, Accessed Jan 7, 2025.

[35] World Wide Web Consortium, 2024. URL: https://www.w3.org, Accessed Dec 12, 2024.

[36] Z. Zhang, J. Zhao, G. Wang, S. Johnston, G. Chalhoub, T. Ross, D. Liu, C. Tinsman, R. Zhao, M. Van Kleek, and N. Shadbolt. Trouble in Paradise? Understanding Mastodon Admin's Motivations, Experiences, and Challenges Running Decentralised Social Media. *Proceedings of the ACM on Human Computer Interaction*, 8:1–24, 2024. DOI: 10.1145/3687059.

[37] M. Zignani, S. Gaito, and G. P. Rossi. Follow the "Mastodon": Structures and Evolution of a Decentralized Online Social Network. *Proceedings of the International AAAI Conference on Web and Social Media*, 12(1):541–550, 2018. DOI: 10.1609/icwsm.v12i1.14988.

[38] M. Zignani, C. Quadri, S. Gaito, H. Cherifi, and G. P. Rossi. The Footprints of a "Mastodon": How a Decentralized Architecture Influences Online Social Relationships. In *IEEE International Conference on Communications Workshops*, pages 472–477, 2019. DOI: 10.1109/INFCOMW.2019.8845221.