MANCHESTER 1824
The University of Manchester

Customisable Social Media Algorithms

A dissertation submitted to The University of Manchester for the degree of

Master of Science in Artificial Intelligence

in the Faculty of Computer Science

Year of submission

2025

Student ID

10833318

School of Engineering

Contents

Word count: 8032

Abstract

Social media algorithms for content recommendation are designed to maximise user engagement, where the user usually has little knowledge of or control over how their algorithm functions. Integrated into the Aether Social platform, this project creates a simple to use system for the creation of algorithms with open-ended customisation. Written using Node, React, Express, and MySQL, this system allows each user to choose from multiple different saved algorithms, and to apply each algorithm to different content feeds. Whenever a user is scrolling through posts, they have access to a ‘choose algorithm’ button that opens an interface to either choose an existing algorithm or create a new one. New algorithms can be chosen from templates, described using natural language, and adjusted with a range of specific options. The result is a system where users can quickly and transparently view, edit, switch between, assign and create algorithms while being easy to use and understand for a non-technical audience. Functioning of this system, as well as its subjective quality, was tested anonymously with real users by using real and artificial text-based social media posts. Based on successful development and user feedback, ideas for future work are also explored.

Declaration of originality

I hereby confirm that no portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright statement

i The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made *only* in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see `http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420`), in any relevant Thesis restriction declarations deposited in the University Library, The University Library's regulations (see `http://www.library.manchester.ac.uk/about/regulations/`) and in The University's policy on Presentation of Theses.

Acknowledgments

Thanks to Stian Soiland-Reyes for his advice and supervision throughout this project.

1 Introduction

1.1 Background and motivation

Social media feeds are to the human mind (biological neural network) what training data is to an artificial neural network [1]. Both are streams of information that gradually shape the behaviour of the resulting neural network. It is therefore important for stream of data to be of high quality and to optimally meet user needs.

However, current social media feeds optimise for attention maximisation for the viewing of advertisements [2], which is not aligned with providing data that maximises human knowledge and well-being. This therefore motivates the creation of a system that more effectively provides content that is well suited to human needs. Since all people are different, such a system needs to be customisable to individual users.

Aether Social is a social media platform that I have previously built based upon an undergraduate project to build a community social media platform [3]. It allows for the sharing of interactive content, like apps and websites, as well as text, images, and videos, in the form of social media posts. Users can follow each other, as well as send connect requests to allow direct messaging (similar to friend requests). However, apart from some primitive optional filtering, posts on this platform were sorted chronologically, meaning that there was no way for users to receive the most relevant and useful content for them. Also, most major social media platforms do not provide users with full control and transparency over how they are shown algorithmic content [4]. This provides the possibility of a project to build and test a system for giving users customisation options for social media recommendation algorithms.

This project was chosen to build upon previous work, rather than building an entirely new system or building an extension to another platform, because it allows full control over and access to a familiar codebase, as well as enabling full focus upon building the customisation system, rather than having to build out supporting technical infrastructure (databases, server hosting, user accounts, post upload etc.) from scratch.

1.2 Why customise algorithms?

Before exploring the methods for social media algorithm customisation, how users currently interact with algorithmic personalisation should first be surveyed. The paper “A Scoping Review of Personalized User Experiences on Social Media” [5] provides a broad overview of 30 existing research pa-

pers, including reviews and original research, relating to social media algorithms from the past 10 years. Recurring topics across these papers include user awareness of algorithmic personalisation, setting customisation, filter bubbles, and echo chambers. User awareness is particularly important to understand, since users must know the extent to which algorithms can be personalised to usefully utilise a customisation system. For example, Swart, 2021 [6] found that a majority of surveyed participants could recognise personalised content selections, although most were still unable to explain or understand how the underlying algorithms worked.

Perez Vallejos et al. (2021) [7] found that many young people feel disempowered by a lack of transparency and privacy controls, and that there was a roughly even split between those who prefer a personalised or a more general internet experience (52.8% vs 47.2% respectively), showing that a customisation system will need to incorporate a broad range of preferences. 72.4% believed that it is important to know how algorithms rank information, demonstrating the importance of making customisable algorithms easily explainable, since it would be harder to trust and control a system that is obscure and difficult to understand.

Swart, (2021) also discovered that, when the true workings of an algorithm are obscured, user develop their own mental models of how they think the algorithm is working and use these to try and 'game' the system. For example, a user believed that posts from certain people were being hidden, such as those from a friend who posts infrequently, requiring deliberate effort to let the algorithm know that the user was interested in this friend's posts. Others would like a certain type of post once, then see that content much more often. Some even believed that the microphones in their phones were listening to them, due to how unclear the workings of their recommendations were. These findings underscore the importance of transparency for algorithmic customisation, since users with false assumptions will not be able to correctly control their recommendations.

1.3 How should customisation be presented?

A good technical implementation of customisable algorithms has little purpose if it is hard for the user to understand and interact with. "User Control in Recommender Systems: Overview and Interaction Challenges" [8] conducts a user study on control over product recommendations on Amazon. They found that when users are given control features, they are often underutilised even when users are aware of the features. This was because features such as sliders or checkboxes were potentially ambiguous in the extent that they affected the recommendations, as well as requiring manual updates. Users also were unlikely to regularly engage with content specific adjustments, such as clicking boxes to indicate how they felt about individual recommendations.

The three key principles that were most important to users were clarity, low-effort interaction, and undoability. These would suggest that having preselected algorithm templates for users to choose from, that can be applied and removing with one click, would suit user needs. However, this paper is limited by its focus on adjusting recommendations once they have already been made. It does not explore how users can create a baseline recommendation system for themselves.

1.4 Preventing Filter Bubbles

Another crucial reason for giving users control over their recommendation algorithms is to lessen the effects of filter bubbles. These are algorithmic recommendations that filter out content that a user is likely to dislike or disagree with, narrowing the worldviews and news that the user is exposed to [9]. The resulting isolation from moderate or opposing views results in more extreme and radical opinions, causing more polarised politics in the real world. UCRS (User-Controlled Recommender Systems) can be used to warn of and help users escape from these bubbles [10]. In this model, users can invoke 4 control types: User-Feature Control and Item-Feature Control, each Fine-grained and Coarse-Grained, meaning that they refer either to specific content items or content groups. User-Features refer to avoiding bubbles caused by the characteristics of user groups, such as users of a similar age. Item-Features relate to the categories of the posts themselves, such as topics or sentiment. These controls are accessed through a User-Controllable Interface (UCI), which contains strength sliders that update recommendations in real time. These instant updates create clarity for the user and will be an important feature for a customisable system. A Bubble Alert Panel is used to warn users on a 1-5 scale about the severity of a bubble they may be in, as well as a trend indicator to help decide if intervention is needed to prevent the bubble from becoming more severe. The UCI was found to effectively mitigate filter bubbles (reducing post isolation by 30%) and boost post diversity without compromising, and often improving, accuracy in recommending relevant items. This can be applied to social media posts, and it will therefore be essential for a customisable algorithm system to provide an option to boost post diversity to help users avoid these dangerous bubbles. Although the bubble alert system is useful for increasing user awareness, it would be preferable for a customisable algorithm to enable preventing such bubbles from ever forming, such as an option to increase content variety, therefore not needing any explicit monitoring.

1.5 Use of Large Language Models (LLMs)

LLMs are a useful tool for implementing customisable social media algorithms, since they are both simple to interact with as well as offering customisation beyond a predefined set of options. This is because users can ‘talk’ with the interface to specify exactly what form they wish the algorithm

to take. InstructAgent, which is an LLM agent that can be addressed in natural language, helps to explore more about how this could work [11]. InstructAgent works by first using a parser, which extracts internal knowledge (information already contained within the LLMs weights) and external knowledge (information that will need retrieval from the internet) from the text. Top-k recommendations are then found, allowing a reranker to be used that adjust the content rankings based on the parsed user instructions. To reduce hallucinations, the agent assesses its new ranking against the instructions to ensure they are a reasonable fit. Although small-scale, fine-tuned LLMs are fastest for repeated direct analysis of posts, if the LLM was only used once to output a specific algorithm, more powerful remote models could be used, as well as having less data privacy concerns due to the API not seeing the posts and user interactions.

1.6 Existing options for customisation

To inspire what specific features a customisable system should have, existing options need to be analysed. “Beyond Explicit and Implicit: How Users Provide Feedback to Shape Recommendations” explores how users engage with personalised recommendation systems (PRS) by interviewing 34 regular social media users [12]. Explicit interactions include marking as not interested, subscribing to a topic, blocking a user, or reporting a post, and are the most clear and direct method that users have to control their recommendations, usually used for immediate corrections to clearly unwanted content, although these interactions are specific to individual posts, requiring repeated use to help build the feed. Implicit interactions can be intentional, where the user indirectly acts to influence the algorithm, such as searching for topics, skipping posts, time spent looking at a post. These were most important for increasing content diversity and relevance. Interactions can also be implicit unintentional, such as liking and sharing posts, and browsing profiles. Understanding user intention is a crucial aspect of recommender systems, since users are active agents in shaping recommendations, rather than passive recipients. Giving users an explicit choice over which interactions are used, such as boosting or supressing certain words, will therefore be an important component of building a customisable algorithm system, since it will provide users with more granular control. With no way for the users to comprehensively know everything that influences their algorithms, it will be useful for a customisable system to collate all influencing aspects in one place.

1.7 Benefits of algorithm personalisation

Although current social media algorithms may be focused on engagement, their personalisation can also offer genuine utility. It is therefore important that custom algorithms offer similar benefits, so as to be preferable. Personalised recommender systems are core components of Very Large

Online Platforms (VLOP's), but are not the same in all parts of the world. For example, the European Union's Digital Services Act (DSA) requires that these platforms give users the option for non-personalised recommendations [13]. This therefore provides an opportunity to compare both approaches and see which is preferred by users, providing guidance for how to best create customisable algorithms. "Contesting Personalized Recommender Systems: A Cross-Country Analysis of User Preferences" surveyed 6,217 across 6 countries to determine the extent that users challenge the default personalised recommendations [14]. They were asked to split themselves between using collaborative filtering (based on similar users), content filtering (based on past content), and random posts. It was found that users largely preferred personalised recommendations over non-personalised (74.3% vs 24.7%). However, it should be noted that this survey focused only on user intent to challenge default recommendations, as well as forcing a dichotomy between personalisation and non-personalisation with no options for how the personalisation functioned.

1.8 Aims and objectives

The first objective is to create a user-friendly frontend interface for viewing, creating, editing, and assigning social media algorithms. This interface should be easily accessible wherever a user is viewing posts, and simple to use and understand for users who are not familiar with algorithms. All features of the algorithm should be transparently shown, so that the user is always fully aware and in control of how their algorithm works. Enough hardcoded options need to be present to cover most changes that an average user would wish to make, as well as having the flexibility to make customisations that are not present by default.

The second objective is to build a backend system that effectively understands and implements the user's customisations. This system must be implemented wherever posts are sent to the frontend. It needs to be possible to send natural language customisation instructions that can be converted into specific algorithmic adjustments. Although only analysis of text content needs to be focused on, some customisations for other content types should be included too. Algorithmic manipulation of posts needs to aim not just to filter out or include certain features but also perform analysis of less precise qualities such as sentiment and similarity. The final aim should be to assign a single score to each post, to be used in the context of ranking posts within an infinitely scrolling feed where new content is continuously fetched.

The third objective is to test the system against real world user needs. Such testing must assess the suitability of given customisation options by gathering feedback on what features needs to be added and removed, as well as the ergonomic design of the interface. Although a technical implementation may be successful, testing should also aim to get opinions on the quality of recommen-

dations provided by the custom algorithm.

2 Methodology part 1: Customisation interface

2.1 Platform

On Aether Social, posts are made to feeds. Feeds are subdivided into channels for different topics, and it is these feeds that custom algorithms are applied to. Group feeds are where many different users can post content, and each user has a user feed where only they can create posts. Feeds can be dragged and dropped together to form 'combined feeds' that display all the posts from the feeds it contains. These combined feeds can also have custom algorithms applied to them. To discover new content, there is an explore page which can have its own custom algorithm applied to it too.

The base platform is written fully in JavaScript, using the Node, Express, React JavaScript frameworks, as well as MySQL as a database. Interactions with the database are done using the Sequelize library, which avoids the need to write raw SQL queries, and frontend HTTP requests handled using Axios. These same technologies are used for this project. Code for this project is kept in its own `custom_algorithms` folders in the front and backend and is commented with `'/Project code'` within existing files that had to be modified.

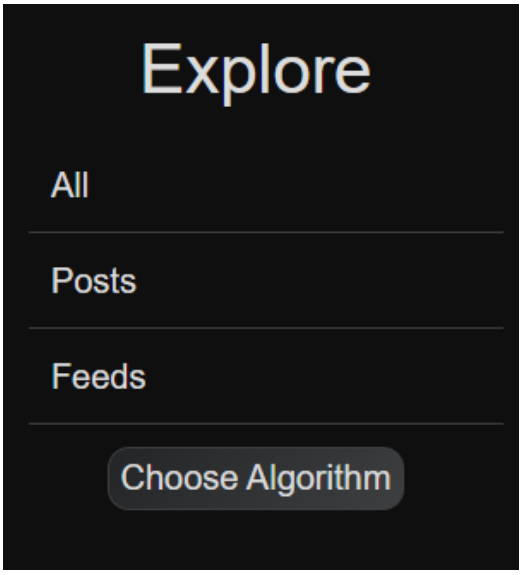

Fig. 1. Button that opens the algorithm customisation interface, located in the right aside of the explore feed

2.2 Algorithm components

The algorithm selection and creation interface consist of two components: AlgorithmSelector and, within it, AddAlgorithm. These separate out the handling of creating algorithms and assigning them to channels, although visually they appear to be one interface. Wherever custom algorithms are needed, the AlgorithmSelector component is imported, and the locationId passed to it as a prop, as well as a refresh posts handler to communicate with the parent component. Explore and Following feeds and the search results page are treated as special cases where ‘explore’, ‘following’ and ‘search’ are passed as the location ids.

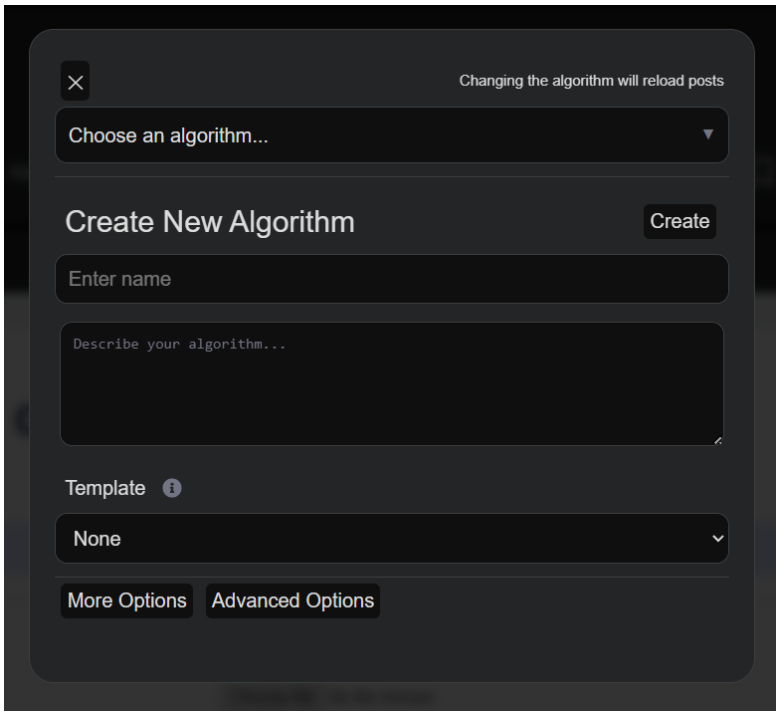

Fig. 2. Base interface opened after ’Create Algorithm’ button has been clicked

2.3 User interface

When viewing a channel feed, there is a ’Choose Algorithm’ button on the right sidebar. This opens an interface over the channel content where the user can select from a dropdown of previously created algorithms. This dropdown contains options for assigning, editing, or deleting each algorithm. When editing an existing algorithm, the customisations are filled in and ready to edit in the same way as creating a new algorithm. When the selected algorithm is changed, either through switching, editing or creating a new algorithm, the posts in the channel are immediately refreshed to show the newly applied algorithm, as indicated by text at the top of the interface, so that the user can view immediate feedback.

Below this is an area for editing an existing or creating a new algorithm. To avoid the need to al-

ways specify every detail of an algorithm, there is the option to select from a set of 12 predefined templates, each consisting of hardcoded JavaScript Object Notation (JSON). Each option can be hovered over to display an information tooltip that provides more detail about how that option works. To avoid visual clutter and to make the interface less intimidating to unfamiliar users, there is a 'more options' menu for simple customisations that most users may wish to make, as well as an 'advanced options' for more technically inclined users. Below the name entry section is a text box for describing how they wish the algorithm to work, without having to use any options. This lets the user avoid having to use many different individual options, as well as providing open-ended customisation beyond the fixed options provided.

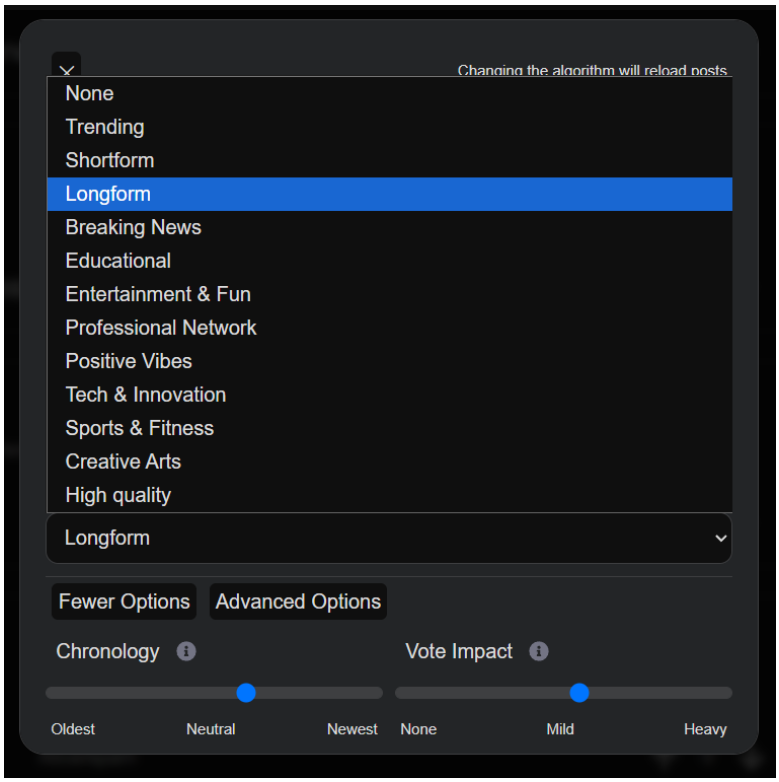

Fig. 3. List of predefined algorithm templates that can be applied with a click

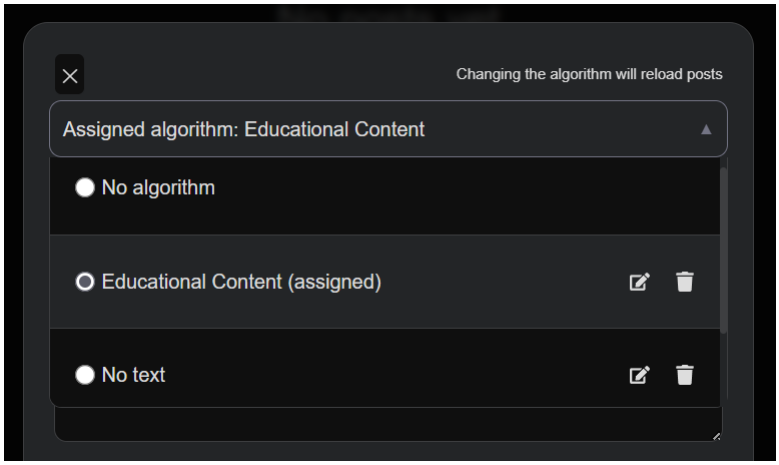

Fig. 4. List of a user's algorithms, with options for selecting, editing, and deleting

2.4 Vote impact

Each post can be up or downvoted by users. Traditional social media algorithms use upvotes/likes to decide what content to show [15]. How many votes a post has could affect its ranking, since less upvoted and more downvoted posts could be an indicator of lower quality. However, a user may not wish to have their algorithm be affected this way, especially when viewing controversial posts that may be unfairly downvoted. There is therefore a 'vote impact' slider, measured with a multiplier ranging from 0 to 1, that ranges from votes not being considered, to 'heavy', where the vote score is applied. The default value is set to 0 on user feeds, and 1 elsewhere, so that the parameter does not unexpectedly adjust recommendation behaviour unless specified.

2.5 Chronology

The chronology of what posts are shown can also be controlled through a slider from older to newer. offering an alternative to traditional algorithms which prioritise new and recent content [16]. Users can still see the most recent posts if they wish, or prioritise older posts, or have no preference, which mixes the two. This was implemented because some users may not want any algorithmic control at all and just see posts in chronological order, whereas others may find value in old posts that would otherwise be hidden by algorithms that value newness. By default, chronology is set to 0.8 for new algorithms, since 1 means full chronological order without any other considerations.

2.6 Sentiment

Sentiment can be adjusted from –1 (negative), to +1 (positive, with 0 meaning no sentiment adjustment, which is the default). Although it may seem unusual to provide an option for negative sentiment, this was included to maintain balance and neutrality, since the purpose of the system is for the user to decide for themselves what they wish to see. For example, a user may wish to compare post results from two otherwise identical algorithms, one positive and the other negative. If there was only an option for neutral to positive sentiment, this would appear biased and limit functionality.

2.7 Variety

Due to the problem of filter bubbles arising from traditional social media algorithms, it is important for a customisable system to give the option to increase the diversity of content viewed. A less homogenous feed may also be more interesting. Therefore, a slider for content variety ranges between diverse, mixed, and similar, where posts that the algorithm will serve are semantically com-

pared with previously upvoted posts. If variety is low, similar posts to previous upvotes are seen, and more different posts for higher varieties. This creates the possibility of viewing posts on the same by creating two separate algorithms with opposite varieties, so the user can view both sides of a story.

2.8 Content lengths

Minimum and maximum durations for videos can be set, to accommodate users who wish to view long or shortform content separately. A slider ranges from no lower limit to no upper limit, with a logarithmic increase in durations. The same can be done for text content, which is measured by word count. Specific limits for videos range from 5 seconds to 60 minutes, and 1 word up to 5000 for text.

2.9 Content types

Four main content types are available: text, images, videos, and interactive content. All are included by default, and can be excluded by clicking a checkbox. Since posts can mix these content types using blocks, a post is filtered out if it includes any of the forbidden content types. The code for interactive content may contain images, videos, and text as part of an app, so the filtering is done by block type, not only HTML tag detection.

Fig. 5. More specific options for algorithm customisation

2.10 Advanced options

All features placed in the separate advanced options menu followed responses from second round user testing that suggested such customisations, although still useful, would be used less often that other options. Chronology can be more closely controlled through a drop-down menu of options, where they can specify which dates to or from they want to see posts. Active days of the week can be chosen too. These time controls enable algorithms to align to the daily and weekly habits of users. For example, they may wish to view different content on weekends, or on certain holidays, as well as in the mornings before work compared to when they get home. Others may wish to view posts from a certain time period, such as within a few days surrounding a major event, meaning that they can use the algorithm as an adjustable, temporary filter. Posts from other social media sites can be included in posts, as well as embeddings of other websites, so sliders for priorotising these additional content types are provided.

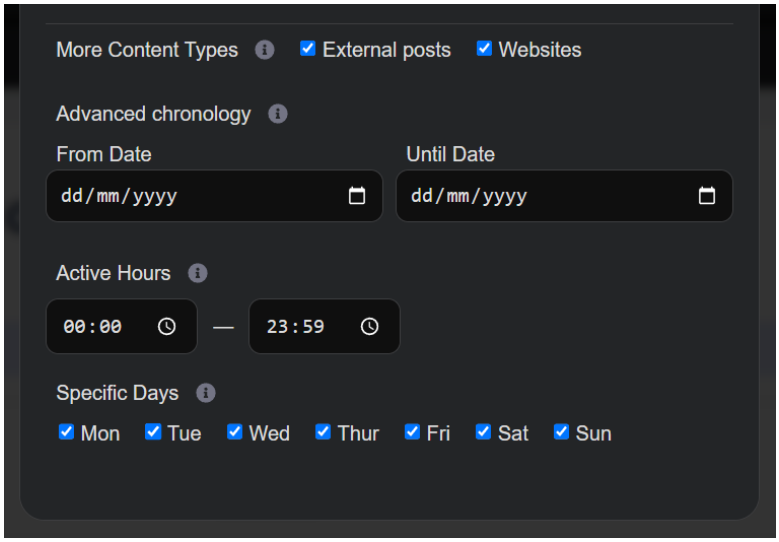

Fig. 6. Advanced options for less common customisation use cases

3 Methodology part 2: Customisation system

3.1 Creating the algorithm

Once all the parameters are set by the user and the create algorithm or save changes button is pressed, all the data is passed in the body of the request sent to `/api/create_algorithm`. This route is also used for edits, since if matching table entries are found they are updated instead of being created new. Two separate tables are used: Algorithms and AlgorithmLocations. The Algorithms table stores all data specific to the algorithm itself, such as the algorithm name and code. AlgorithmLocations is used as a join table to connect which algorithms have been applied to specific

channels by which users. All algorithms are referred to by a 32-digit Version 4 Universally Unique Identifier (UUIDv4), which acts as a randomly generated primary key. This was chosen due to the negligible chance of random duplication, as well as avoiding concurrency and security risks associated with sequential keys [17].

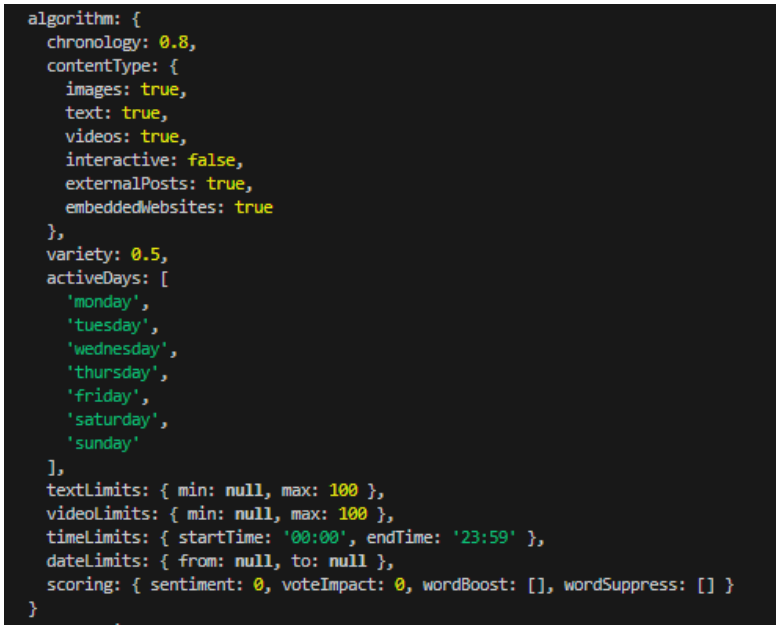

Fig. 7. JSON output of an algorithm that excludes interactive content

3.2 Customisation using an LLM

Not all features that a user may wish their algorithm to have can be provided, especially since more options would make the interface more complicated to use and understand. It is much easier for users to specify exactly what they want by using natural language. Large Language Models (LLMs) can be used to translate natural language instructions into JSON code. If custom instructions are provided, the whole JSON structure is generated. This is because the user may not have used any of the provided controls and instead written their entire customisation in the algorithm description. If any controls have been used, their parameters are passed to the model to be included in the generated code. If no custom instructions are provided, the LLM is skipped entirely and the JSON constructed manually.

When using LLMs, maintaining consistency is essential, since even advanced models can make mistakes or interpret instructions differently. As a result, the precise JSON structure had to be given in the prompt, as well as specific instructions about how data should be inserted into the JSON. The model is not limited to providing only the existing instructions but also is also prompted to provide lists for customFilters and customScoring. This allows for instructions such as “Only show posts from users with more than 1000 followers” or “Always include John456’s latest posts”.

Initially, GPT-4.1-mini was used to create the JSON. However, GPT-5-mini was released on 7 August and used in the final implementation due to its improved accuracy and reduced cost [18]. Although the API is not free, usage was taken out of preexisting credits, so for this project it had no actual cost. Since the LLM is used only once upon uploaded, it is appropriate to use a larger scale model, unlike a smaller, local model which would only be needed for continuous use.

```
const systemPrompt = `
    You are an expert algorithm creation assistant.
    Output a single, valid JSON object strictly following this schema:
    {
        "chronology": number (-1 to 1),
        "variety": number (0 to 1),
        "contentType": { "images": boolean, "videos": boolean, "text": boolean, "interactive": boolean, "externalPosts": boolean, "embeddedWebsites": boolean },
        "activeDays": string[] (each must be a lowercase full weekday name, e.g. "monday", "tuesday"),
        "textLimits": { "min": number | null, "max": number | null },
        "videoLimits": { "min": number | null, "max": number | null },
        "timeLimits": { "startTime": string | null, "endTime": string | null },
        "dateLimits": { "from": string | null, "to": string | null },
        "scoring": {
            "sentiment": number (0 to 1),
            "voteImpact": number (0 to 1),
            "wordBoost": [],
            "wordSuppress": []
        },
        "customFilters": [],
        "customScoring": []
    }
    Posts are made to feeds. Each poster_id references a user feed who made the post.
    Database schemas for custom filters/scoring:
        Post fields: post_id, parent_id, feed_id, channel_id, poster_id, title, content, text_body, replies, views, upvotes, downvotes, text_length, word_count,
        Feed fields: feed_id, feed_name, decription, follower_count, is_group
        Channel fields: channel_id, channel_name
    Merge the form settings with the user's custom instruction. If contradiction, prioritise following custom instruction.
    No text outside the JSON.
`;
const userContent = `
    Form Settings:
    ${JSON.stringify(algorithmJson, null, 2)}
    Custom Instruction:
    "${customInstruction}"
`;
```

Fig. 8. Prompt sent to the OpenAI API to generate algorithm JSON, including the full structure and specified options

3.3 Algorithm handling

Upon opening the interface, a request is immediately sent to the backend route for fetching user algorithms. This finds all entries in the Algorithms table where the stored viewer id is the same as that of the logged in user. These algorithms, including their names and JSON code, are returned to the frontend.

Once a success message has been received from the backend upon creation of the algorithm, a separate route for assigning algorithms is called by the AlgorithmSelector component. This results in the AlgorithmLocations table being updated to include the viewer id and channel id of the newly created algorithm, as well as a unique id. This is done through a separate route, rather than when creating the algorithm, because the same route is also used when switching between algorithms on a channel or choosing which algorithm to newly assign to a channel, so it is simpler to have all algorithm assignment handled in one place.

When editing, the adjusted specifications are sent to the backend through the same route as creating an algorithm. This is to avoid code repetition, since the only difference between the two op-

erations is if an existing entry is being updated or if a new entry is being created. The Algorithms table is checked to see if an entry for the specified algorithm name and user id already exists (algorithm names must be unique), and if so, the entry is updated with the new JSON code. If nothing is found, a new entry is created.

When the backend receives a delete request, transactions are used to only delete entries from the Algorithms and AlgorithmLocations table when every operation has been confirmed as successful, to avoid errors where deletions are made to one database but not the other. Deleting an algorithm is handled separately to removing an algorithm, since the latter only removes an entry in the AlgorithmLocations table while preserving the entry in the Algorithms table. To minimise database operations, the route for assigning algorithms updates the existing location entry with the new algorithm id, rather than removing the old entry and creating a new one. The route for algorithm removal is therefore only called when 'No algorithm' is selected.

```
router.post('/assign_algorithm', authenticateCheck, async (req, res) => {
    let transaction;
    try {
        transaction = await sequelize.transaction();
        const { algorithmId, locationId } = req.body;
        const viewerId = req.session.viewer_id;
        const [record, created] = await AlgorithmLocations.findOrCreate({
            where: { location_id: locationId, viewer_id: viewerId },
            defaults: { id: v4(), algorithm_id: algorithmId },
            transaction
        });
        if (!created) {
            await record.update({ algorithm_id: algorithmId }, { transaction });
        }
        await transaction.commit();
        res.status(200).json({ success: true });
    } catch (error) {
        if (transaction) await transaction.rollback();
        console.log("error assigning algorithm:", error);
        res.status(500).json({ success: false, message: 'Failed to assign algorithm.' });
    }
});
```

Fig. 9. Route for applying an existing algorithm to a new location

```
router.delete('/delete_algorithm', authenticateCheck, async (req, res) => {
    let transaction
    try {
        transaction = await sequelize.transaction();
        const { algorithmId, locationId } = req.body;
        const viewerId = req.session.viewer_id;
        const existing = await Algorithms.findOne({
            where: { algorithm_id: algorithmId }
        });
        if (!existing) {
            return res.status(400).json({ success: false, message: 'Algorithm not found.' });
        }
        await AlgorithmLocations.destroy({ where: { algorithm_id: algorithmId, location_id: locationId, viewer_id: viewerId }, transaction });
        await Algorithms.destroy({ where: { algorithm_id: algorithmId }, transaction });
        await transaction.commit();
        res.status(200).json({ success: true });
    } catch (error) {
        if (transaction) await transaction.rollback();
        res.status(500).json({ success: false, message: 'Failed to delete algorithm.' });
    }
});
```

Fig. 10. Route for deleting an algorithm, using a transaction to ensure nothing is deleted unless all operations are successful

3.4 Post parsing

To allow algorithms to be applied in different express routes, all algorithmic manipulation occurs within an ApplyAlgorithm function, that is then imported for use in the channel posts, search results, explore page, and combined feed routes. All posts on Aether Social consist of single files of HTML, combining multiple different content types, each contained in its own content block. Text, images, videos, and interactive content therefore must be parsed from these HTML files by reading the content type of the blocks contained within the post, and detecting <img> and <video> tags in media blocks. Upon post upload, the text body is extracted for analysis, performed using functions in a class named ContentAnalyser. The results from applying this class are stored in the post database entry, with columns for text and videos lengths, word and sentence counts, content types, sentiment scores, and embeddings.

3.5 Fetching posts

ApplyAlgorithm accepts parameters including channelId. Search queries are sent to the posts database, which is indexed by `channel_id` to minimise the rows to search, while being filtered for any chronological constraints. The returned posts are then further filtered to remove any violated constraints, such as content type or words, and further search queries are sent to the database until a full set of 20 satisfying the algorithm are ready to be sent. If in explore or following, the users own posts are excluded, and if the Main feed channel is being viewed, all posts from all channels in the feed are fetched. These final posts, along with their scores, are what ApplyAlgorithm returns, ready to be used by whatever route uses the function.

3.6 Pagination

When scrolling through a channel in a feed, posts are fetched in batches using `/api/channel_posts`. IDs of posts already viewed are sent to the route, so that repeat posts are not fetched. When approaching the bottom of the loaded posts, detected using an intersection observer at the bottom of the page, the route is called again and fetches posts in batches of 20. The same happens wherever else posts are fetched.

3.7 Basic adjustments

Algorithmic adjustments consist either of filtering or scoring. Filters entirely exclude any posts that violate their specified conditions and is performed first, followed by valid posts being scored by ap-

plying weights that boost or penalise the final value used to rank the post. If no explicit chronological considerations are made, newer posts will be favoured by default, meaning that if a feed is viewed a month later with the same algorithm, different posts will be seen due to the previous posts now being a month older. This is done to prevent being stuck forever viewing the same posts, while also allowing the option to leave a channel and revisit it soon after to see the same posts, rather than previously viewed posts being filtered out completely.

User feed channels have different methods of displaying posts by default, if no algorithm is applied. User feeds have their posts shown in chronological order without any algorithmic manipulation since there are likely to be far fewer posts, meaning that viewers would not want to risk missing out on any posts due to algorithmic manipulation. However, chronological ordering by default for posts in group feed channels and other environments would not be appropriate, since large numbers of high- and low-quality posts could be mixed. Instead, the vote score is calculated as usual, with this being the only scoring applied to the post. This results in highly viewed and upvoted posts, with fewer downvotes, are prioritised. The algorithm for ranking 'Hot' posts on Reddit was the inspiration for this method [19], since subreddits are similar environments to group feed channels.

3.8 Sentiment analysis

Sentiment is calculated using winkNLP Node Package Manager (NPM) module [20]. This uses a pretrained, lightweight (1MB) language model to output a sentiment score between –1 and 1. It was chosen for its high accuracy (>90%) and high speed, processing over 650,000 tokens per second. The distance from the user's specified sentiment score is then compared. For example, if the sentiment is 0.3, then posts close to a score of 0.3 will be seen more often. Not all posts have sentiment that can be analysed by the system, such as images or videos, meaning that by default they would score zero and very high or low sentiment settings would result in these posts being unreasonably penalised (e.g. if sentiment was set to 0.8, the user would never see any images). Adjustments based on sentiment are therefore applied only to posts containing text, with other content not containing any text treated as if it has the intended sentiment. This sentiment score is calculated upon post upload and stored in the database.

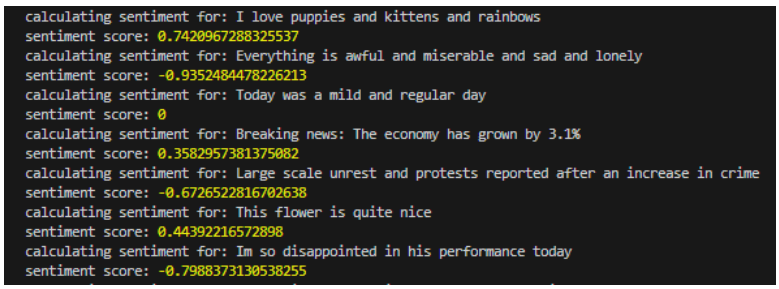

Fig. 11. A range of different texts with varying positive, neutral, or negative sentiment scores

3.9 Calculating variety

Variety is calculated by comparing the similarity of a given posts embeddings to those of the users 100 most recently upvoted posts. Embeddings are constructed using the pretrained all-MiniLM-L6-v2 sentence transformer, which converts text bodies into 384 dimensional vectors [21]. These vectors are calculated by tokenising the text input and passing these token through multiple transformer layers, with these hidden states pooled and normalised to produce a single unit vector that captures the semantic meaning of the text. Cosine similarity is calculated between a post embedding and those of the recently upvoted posts. If variety is set to high, posts with low similarity will be prioritised.

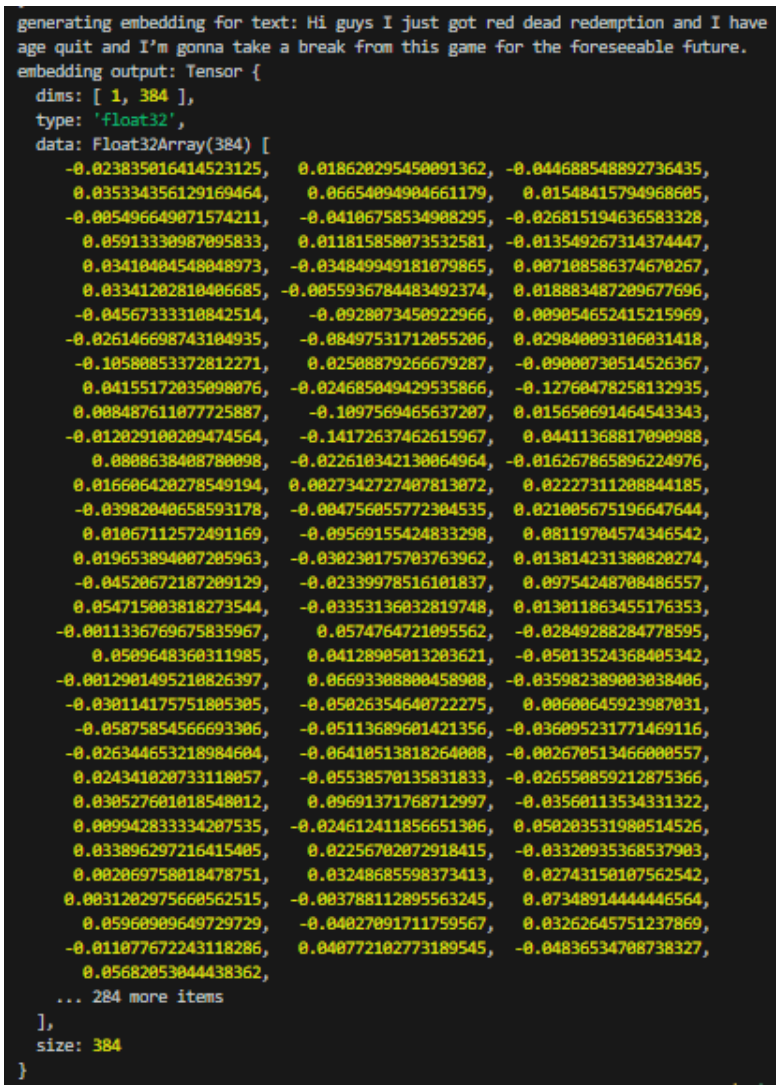

Fig. 12. Log of a post's semantic embedding vector with 384 dimensions

3.10 View handling

Views are only incremented when a user interacts with the post (that is not their own) through voting, opening the replies, or clicking directly on the post content. This is because simply scrolling past a post without interaction may indicate lack of user interest, so the view count only represents those users who decided to engage with the post rather than scroll past. However, to prevent repeatedly being shown posts that the user does not wish to engage with, returned posts are still tracked and stored as excludedPosts, so that future post searches do not return them.

3.11 Vote score

Vote scores are applied after all other scoring has been completed. Vote quality (upvotes divided by downvotes) and engagement (total votes divided by views) are multiplied together, with each pa-

rameter given a value of at least one to avoid division by zero, to produce a vote score. The vote impact is then applied to this total to determine how much the vote score should affect the final score. After all algorithmic adjustments are complete, this results in each post being assigned a final score, with higher scoring posts being listed first within the returned batch of 20 posts.

3.12 Application locations

Custom algorithms must apply in several different environments. Feed channels are the most general-purpose use case, but there are also combined feeds which collate all posts from their contained feeds (of which the user's 'following' feed is a special case, since this combines all their followed feeds). These too can have custom algorithms applied. There is also the explore page, which recommends new content from across the site. Since the purpose is explicitly to recommend new content, previously viewed posts must be treated differently. The search results page can also have custom algorithm applied too, although previously viewed posts are not considered.

4 Results and discussion

4.1 Test dataset creation

To test the objective technical function and subjective recommendation quality of customisable algorithms, a large and diverse set of test posts needs to be created. This test set needs to include sufficient variety of posts that each customisation option can be individually tested, as well as in combination with other options.

For initial algorithm testing during development, individual posts were created to get immediate feedback. For example, posts with the string 'apple' were created, including with different capitalisations and whitespaces, and then different algorithms applied to see if those posts were boosted, penalised, or supressed completely. The same was done for sentiment, with 10 different posts of varying emotional intensity. Which posts were boosted to the top and bottom could then be seen to ensure that the top posts were those perceived as closest to the intended algorithmic sentiment. This also provided the opportunity to use console logs to analyse the specific computed sentiment for each post.

Once the implementation had been technically validated, it was tested for recommendation quality using a much larger set of posts. Only text posts were used, since image and video analysis is beyond the scope of this project. Two datasets were used. The first dataset was generated artificially

using the faker.js NPM library [22]. This allowed for the creation of thousands of fake but realistic posts from fake users, with varied names, dates, text contents, topics, posts lengths, votes, views, reply counts, and timestamps. 750 profiles were created, each with 100 posts. Profile pictures were sourced from an open-source profile picture Kaggle dataset [23]. Posts are made from one of 10 topics, and each generated user has channels on their feed for each topic. The second consisted of over 700,000 publicly downloadable posts from Twitter, accessed via Kaggle [2], with the purpose of providing more realistic looking posts so that the recommendations do not entirely rely on the quality of post generation. These two datasets are then randomly spliced together with an even amount from each, with the posts made to their relevant topic channels.

4.2 Round 1 testing: Interface design

The first round of user testing focused on the interface itself, to explore what features and visuals worked well and needed improvement in the initial prototype. This was done anonymously, without video recording, and consisted of answering 3 predefined questions while using the interface, as well as open ended discussion to gain further ideas. They were asked “What is most useful to you about the interface”, “Is there anything you’d like to remove or adjust about the interface?”, and “Is there anything you’d like to add to the interface?”. Four individuals were interviewed, enough to get an initial impression about the interface without taking too much time to conduct numerous and extensive interviews. These individuals were all university students who were asked individually to voluntarily participate. Two users had a computer science background, and two did not, to ensure that testing reflected technical and non-technical users.

Easily creating and switching between algorithms was received well, for example because “It’s annoying to have politics and tech posts mixed together”. Another user said that “I like how everything that influences the algorithm is transparently shown”, as well as “the info icons are useful for helping me understand”, showing that knowledge of how an algorithm works is an important part of the system, and that the implementation is intuitive to use.

Most feedback for improvement was about what more options and capabilities the users wished to have. These included support for multiple languages, options to control content length (e.g. short vs long form videos), and the ability of the algorithm to learn your preferences over time (similar to traditional, machine learning social media algorithms) while still being manually adjustable. These changes are beyond the scope of this project but show that there are abundant possibilities for future work.

All users commented that it was hard to decide what to focus on first when viewing the initial pro-

totype and that, although the diverse array of options was good, it seemed cluttered and crowded. This feedback resulted in the separation of customisations into menus for more and advanced options. Since more numerous and in-depth user interviews would likely result in many more ideas for options, this inspired the addition of natural language descriptions, processed by an LLM, to accommodate open-ended customisations.

4.3 Round 2 testing: Recommendation quality

Interviews for round 2 were conducted similarly to round 1 but also focused on how well the recommendations met user intentions and needs, as well if there were any further considerations they would like the algorithms to make. 5 users, all different from round 1, participated and were each asked individually to take part. The same questions as in round one were asked (to gather further feedback on interface changes) as well as “How well do the recommendations satisfy your intentions when making the algorithm?”, “What ways would you use the customisations in your everyday social media browsing?”, and “Are there any ways the recommendations could be improved?”

Initial quality of sentiment analysis was good but could be improved. One user noted that “A lot of the posts match the sentiment I want, but not all”. This was because sentiment analysis worked by using the ‘Sentiment’ NPM module, which breaks the text into word tokens, then assigning a positive or negative score to each word that may convey emotion, based on how emotionally strong the word is, then averaging these scores together. Although this worked well for many posts, it considered each word individually without context, resulting in not all posts receiving a score that matched their actual meaning. To improve accuracy, winkNLP was used instead.

Since 75,000 posts were included in the test database, there were noticeable lags in loading time that users commented on. This was caused by the initial code using many database join operations per post, as well as performing all computations each time posts were fetched. To solve this, the test dataset was recreated with each post having relevant algorithmic information, such as tokens and sentiment, calculated and stored upon upload. Also, the original post similarity approach required comparing the similarity of bigrams and trigrams (pairs and triplets of phrases), as well as keywords and tokens, all using cosine similarity. These computations combined were very expensive, especially for longer texts, as well as struggling with texts that were semantically similar but lexically different. To improve computational performance and subjective accuracy, similarity was redeveloped to compare embedding vectors instead.

Advanced options initially included a slider for image sizes to be controlled in the same way as videos and text lengths, so that users could prioritise high-definition images. However, users suggested

that they wouldn't need such an option. Instead, they would prefer for there to be sliders to prioritise certain content types, rather than just tick boxes to include/exclude them.

A common request was for post authors to be considered too, not just the content of the posts themselves. Although specific customisations relating to users would require new frontend features, such as input boxes for feed names, filters for follower numbers, and user ages, the custom instruction LLM prompt was updated to provide the database schemas for tables relating to user feeds and channels.

5 Conclusions and future work

5.1 Project conclusions

All three project objectives were successfully met. The frontend interface is easy to access, use, and understand for new and non-technical users. Extensive customisation options are presented, included open-ended natural language input. The backend system successfully encodes and applies these customisations to returned text posts, as well as facilitating adjustments for other content types. Filter bubbles, which are a major flaw of traditional social media algorithms, are mitigated well by analysing post similarity using language models. Meeting these objectives were confirmed by user testing, which showed that there is a genuine need for customisable social media algorithms that the developed system helps to meet.

The implemented system is multipurpose, suited for use in search results, explore page recommendations, followings, combined feeds and on feed channels. These varied content environments help to thoroughly test algorithmic customisation for a variety of social media use cases and justifies the need for a diverse set of customisations that users can easily piece together and suit their needs.

Testing showed that there are multiple different ways to implement algorithmic customisations with varying speeds and accuracies. The lesson from these tests showed that, although traditional natural language processing approaches are still effective, machine learning implementations provide results that most closely match user intentions. Although over 2000 lines of code were written, the current implementation is still relatively small compared to the many more ways that the scope and quality of customisations could be improved. As also suggested by user feedback, there are many ways the system could be extended through future work.

5.2 Multimodality

To make the project achievable within the given timeframe, only text content is considered. In reality, many social media content includes images and videos, and Aether Social also include interactive content that isn't found elsewhere. Integrating algorithmic analysis of these content types is therefore a priority for future development. Analysis of multimodal content is already well established, such as video recommender systems that use speech recognition, visual computing techniques like Optical Character Recognition (OCR), and object detection [25]. Deep learning can also be used, such as for recommending YouTube videos [26]. To minimise computational requirements, each post would be scanned once upon upload and relevant algorithmic algorithm stored.

5.3 Social graphs

Connecting with other users is a core feature of social networks. However, the current implementation only considers each user in isolation. X's open source recommendation algorithm [27] provides valuable insight into how user networks can be used to improve recommendations. An improved customisable system would have the option for using data about other user's votes, views, and comments, especially those from connections or follows, can be used to generate more relevant recommendations. This can be used both to provide posts that other users found useful, as well as providing posts to be shared with each other. How exactly this is implemented would itself be a customisation, as well as a privacy control, since a user may not wish other users to know what posts they are viewing. There could be options for which individual pieces of data should be considered, as well as if analysis should be restricted only to connections, or to similar user profiles from across the site. This data would also be included when calculating the similarity parameter, further helping to prevent filter bubbles if similarity is set low (meaning that posts are different from what is seen by other users).

5.4 Adaptable algorithms

Once a custom algorithm is set, it will only change if edited manually by the user. Based on feedback, an option for algorithms to automatically learn and adjust preferences over time would better meet user needs. Learning could be done using votes and views, as well as watch history and interactions such as post viewing duration. A time weight decay would mean that recent interactions have the most impact. Personal information such as age and location could be considered too, as well as life situation (e.g. students, relationship status etc.), potentially filled in through a 'Tell me about yourself' style survey. To use each piece of data would be an option chosen by the user, with

encryption for storing sensitive information. This gives both the advantages of increased personalisation, while still having explicit control over privacy. Implementing adaptability could be done similarly to the Facebook ecosystem [28], where users and posts are embedded as vectors into a global high-dimensional model and continuously updated with new information. Candidates for post recommendations are then found by finding the nearest neighbours to a user. It would be essential for information that a vector has learnt about a user to be parsed into specific options and displayed transparently, both so that the user knows exactly how their algorithm works, as well as giving them the option to change the adaptations if needed.

5.5 Languages

Support for posts in languages other than English would benefit both non-English speakers, as well as those who are multilingual or learning a foreign language. Both the sentiment analysis and the text embedding models are English-only, so separate models for different languages would need to be included. Using Byte Pair Encodings (BPEs), other languages can be translated for use in English analysis models [29]. But specific multilingual features, as indicated by Round 1 of user testing, are needed. This could include both having all recommendations in one language, meaning that switching between algorithms changes the language of posts, as well as mixing posts of different languages together. More fine-grained CEFR language levels could be included, such as A2 or B1, so that posts are understandable to speakers of different proficiencies. These could be implemented using Automatic Readability Assessments (ARAs) conducting using word banks associated with each level [30], or the BERT language model for increased accuracy [31]. Such language controls would be of most use in search results and the explore page, since individual channels and feeds are likely to be monolingual.

6 References

[1] Korte, M. (2020). The impact of the digital revolution on human brain and behavior: where do we stand? Dialogues in Clinical Neuroscience, [online] 22(2), pp.101–111. doi:https://doi.org/10.31887/DCNS.2020.22.2/mkorte.

[2] Bhargava, V.R. and Velasquez, M. (2020). Ethics of the Attention Economy: The Problem of Social Media Addiction. Business Ethics Quarterly, [online] 31(3), pp.321–359. doi:https://doi.org/10.1017/beq.2020.32.

[3] Hiseman, S. (n.d.). Community Social Media Website. [online],

Available at: https://pats.cs.cf.ac.uk/@archive_file?...

[4] Laing, L. (2021). Algorithmic Transparency and Content Amplification - VOX - Pol. [online] VOX - Pol. Available at: https://voxpol.eu/algorithmic-transparency-and-content-amplification/.

[5] Eg, R., Tønnesen, Ö.D. and Tennfjord, M.K. (2023). A scoping review of personalized user experiences on social media: The interplay between algorithms and human factors. Computers in Human Behavior Reports, [online] 9(1). doi:https://doi.org/10.1016/j.chbr.2022.100253.

[6] Swart, J. (2021). Experiencing algorithms: How young people understand, feel about, and engage with algorithmic news selection on social media. Social Media + Society, [online] 7(2), pp.1–11. doi:https://doi.org/10.1177/20563051211008828.

[7] Perez Vallejos, E., Dowthwaite, L., Creswich, H., Portillo, V., Koene, A., Jirotka, M., McCarthy, A. and McAuley, D. (2021). The impact of algorithmic decision-making processes on young people's well-being. Health Informatics Journal, 27(1), doi:https://doi.org/10.1177/1460458220972750.

[8] Jannach, D., Naveed, S. and Jugovac, M. (2017). User Control in Recommender Systems: Overview and Interaction Challenges. Lecture Notes in Business Information Processing, pp.21–33.doi:https://doi.org/10.1007/978-3-319-53676-7_2.

[9] Fletcher, R. (2020). The truth behind filter bubbles: Bursting some myths. [online] Reuters Institute for the Study of Journalism. Available at: https://reutersinstitute.politics.ox.ac.uk/news/truth-behind-filter-bubbles-bursting-some-myths.

[10] Wang, W., Feng, F., Nie, L. and Chua, T.-S. (2022). User-controllable Recommendation Against Filter Bubbles. Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval. doi:https://doi.org/10.1145/3477495.3532075.

[11] Arxiv.org. (2018). InstructAgent: Building User Controllable Recommender via LLM Agent. [online] Available at: https://arxiv.org/html/2502.14662v1.

[12] Arxiv.org. (2025). Beyond Explicit and Implicit: How Users Provide Feedback to Shape Personalized Recommendation Content. [online] Available at: https://arxiv.org/html/2502.09869v1.

[13] European Commission (2022). The digital services act. [online] European Commission. Available at: https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/europe-fit-digital-age/digital-services-act$_en$.

[14] Starke, C., Ljubiša Metikoš, Natali Helberger and Claes de Vreese (2024). Contesting personalized recommender systems: a cross-country analysis of user preferences. Information Communi-

cation Society, pp.1–20. doi:https://doi.org/10.1080/1369118x.2024.2363926.

[15] Milli, S., Carroll, M., Wang, Y., Pandey, S., Zhao, S. and Dragan, A.D. (2025). Engagement, user satisfaction, and the amplification of divisive content on social media. PNAS Nexus, [online] 4(3). doi:https://doi.org/10.1093/pnasnexus/pgaf062.

[16] Sprinklr.com. (2025). Social Media Algorithm and How They Work in 2025 | Sprinklr. [online] Available at: https://www.sprinklr.com/blog/social-media-algorithm.

[17] pazvanti (2021). Exposing sequential IDs is bad! Here is how to avoid it. [online] DEV Community. Available at: https://dev.to/pazvanti/exposing-sequential-ids-is-bad-here-is-how-to-avoid-it-1mjp.

[18] Openai.com. (2015). GPT-5 is here. [online] Available at: https://openai.com/gpt-5/.

[19] Winkjs.org. (2017). winkNLP - NLP in Node.js. [online] Available at: https://winkjs.org/wink-nlp/.

[20] Hugging Face (n.d.). sentence-transformers/all-MiniLM-L6-v2 · Hugging Face. [online] huggingface.co. Available at: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.

[21] Medium. (2025). Medium. [online] Available at: https://medium.com/hacking-and-gonzo/how-reddit-ranking-algorithms-work-ef111e33d0d.

[22] fakerjs.dev. (n.d.). Faker | Faker. [online] Available at: https://fakerjs.dev/.

[23] passionate-nlp (2021). Twitter Sentiment Analysis. [online] Kaggle.com. Available at: https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis?resource=download.

[24] Osman Kagan Kurnaz (2024). Human Profile Photos Dataset. [online] Kaggle.com. Available at: https://www.kaggle.com/datasets/osmankagankurnaz/human-profile-photos-dataset?resource=download.

[25] Lubos, S., Felfernig, A. and Tautschnig, M. (2023). An overview of video recommender systems: state-of-the-art and research issues. Frontiers in big data, [online] 6. doi:https://doi.org/10.3389/fdata.2023.1281614.

[26] research.google. (n.d.). Deep Neural Networks for YouTube Recommendations. [online] Available at: https://research.google/pubs/deep-neural-networks-for-youtube-recommendations/.

[27] GitHub. (2023). Twitter’s Recommendation Algorithm. [online] Available at: https://github.com/twitter/the-algorithm.

[28] Naumov, M., Mudigere, D., Shi, H.-J.M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C.-J., Azzolini, A.G., Dzhulgakov, D., Mallevich, A., Cherniavskii, I., Lu, Y., Krishnamoorthi, R., Yu, A., Kondratenko, V., Pereira, S., Chen, X. and Chen, W. (2019). Deep Learning Recommendation Model for Personalization and Recommendation Systems. arxiv.org. [online] doi:https://doi.org/10.48550/arXiv.1906.00091.

[29] Mengjie Zhao and Hinrich Schütze. 2019. A Multilingual BPE Embedding Space for Universal Sentiment Lexicon Induction. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 3506–3517, Florence, Italy. Association for Computational Linguistics.

[30] Vajjala, Sowmya. (2021). Trends, Limitations and Open Challenges in Automatic Readability Assessment Research. 10.48550/arXiv.2105.00973.

[31] Imperial, J. (n.d.). BERT Embeddings for Automatic Readability Assessment. [online] Available at: https://arxiv.org/pdf/2106.07935.

Appendices

A Project outline

Background

This project will build upon my previous work to develop a social media website called Aether. This website consists of interactive content posted to feeds, which can be followed by users. Users can organise their feeds into 'deepfeeds', like folders on a computer. For example a user's deepfeed for 'Manchester Uni' could contain other deepfeeds for 'Societies', 'Lecturers', 'Friends' and so on. Then these deepfeeds contain the individual feeds that the user has put within them. Clicking on a feed or deepfeed lets the user view all posts within. This approach allows the user to carefully curate their content by topic. However, posts within feeds and deepfeeds are currently displayed chronologically, without any form of sorting.

Description

The aim of the project will be to develop algorithms for sorting and recommending content that can be highly customised by the user. This in contrast to algorithms on most other social media platforms, which are usually an unknown 'black box' designed to maximise attention and time spent on a platform. Algorithms will be specific for each feed/deepfeed, and can be saved and applied across multiple feeds. They will have fine-grained adjustments, such as for up/downvotes, time, views, keywords etc. Machine learning could be incorporated too, again with user controllable parameters. Algorithms will be customised using an interface that is easy to understand for non-technical users, as well as being able to understand requests using natural language e.g. a user could type 'On Sunday's, don't show me posts about Formula 1 until after 5pm'. Researching further ideas about how to customise algorithms will be a key focus of the project, such as how posts from other platforms could be included in recommendations via Fediverse protocols. To test the product, synthetic user and post data will be generated. Previous work and work done as part of this project will be clearly distinguished.

Deliverables

A systematic literature review on currently existing recommendation algorithms

An interface and backend system for creating and adjusting content algorithms for feeds

B Risk assessment

Risk of copyright infringement is mitigated by using only using datasets and code modules with an open-source license. To avoid self-plagiarism, all original code relating to the project is kept separate from preexisting code, or clearly commented, to ensure original work is clearly defined. Due to the personal data risks of interviews conducted with real users, all responses are anonymous, without video or audio recording, and involve no collection of personally identifiable information (PII).