ulm university universität
uulm

Ulm University | 89069 Ulm | Germany

Faculty of Engineering, Computer Science and Psychology
Institute of Databases and Information Systems

Recognition of Scholarly Publication Trends based on Social Data Stream Processing

Bachelor Thesis at Ulm University

Submitted by:
Lukas Jesche
lukas.jesche@uni-ulm.de

Reviewers:
Prof. Dr. Manfred Reichert

Advisor:
Klaus Kammerer

2021

Revision November 10, 2021

Satz: PDF-LaTeX 2$_\varepsilon$

Abstract

A large number of scientific publications are published continuously. It is difficult to distinguish essential publications from others. Bibliometric analysis allows such key publications to be identified typically months or years after publication, provided they have been cited sufficiently often. However, for current and high-frequency research areas, such as COVID-19, it is crucial to make this distinction shortly after publications are published. On social media, new publications are often shared and discussed within days of their release.

In the context of this thesis, an event-based system is presented that collects, processes, scores, and aggregates Twitter events that reference or discuss scientific publications. Twitter events are automatically linked to referenced publications, bibliometric and social media data is further processed to generate a trend score to rank publications over time. The developed proof-of-concept prototype stores data in a relational and a time-series database to provide trend data via a REST API. Moreover, the data is visualized with a modern web interface publicizing the current trends.

Contents

1

Introduction

During the last two years of the COVID-19 pandemic, thousands of people have discussed related new scientific publications on social media. These Discussions are starting imminently after publishing, and up to thousands of people can participate within hours. These Discussions, however, are not directly reflected by traditional bibliometric analysis methods. In order to help recognize these emerging trending publications, Altmetrics collect and analyze social media data to score publications. Currently, to the best of our knowledge, no system exists that publicizes trends based on sophisticated trend detection methods of new and well-discussed scientific publications on social media. Most related systems such as CrossRef [1], PlumX [2] or Altmetric [3] only collect the data and do not further analyze it to evaluate data content and impact. Since the publicity about topics like COVID-19 is enormous, not just researchers and people with scientific background are part of the discussion, which should be considered.

The main objective of this thesis is the development of a holistic web-based system that ultimately presents currently trending publications ranked by their social media impact in a web application.

Therefore, a wide range of problems must be considered. Data retrieval, processing and scoring of social media data and further linking with referenced scientific publications require a number of rule-based approaches. Extensive concepts, such as data aggregation and special database systems, shall ensure fast retrieval of this data even for numerous simultaneous users. Lastly, missing data needs to be considered.

In this thesis, a proof-of-concept prototype publication data is implemented based on a distributed, event-based microservice architecture. The presented prototype can continuously collect, process, aggregate, analyze and store social media events from

Twitter. Particular attention was paid to a stable and fault-tolerant execution environment to be able to process the majority of data continuously retrieved from a Twitter event stream, ensuring confident results. The system is built on a modular basis and can be extended with additional sources, processors and other components. Processed and analyzed data is provided by a REST API that, in turn, is used by a web frontend visualizing the data and results in a modern and appealing way.

1.1 Structure of Work

Following, the structure of this thesis is described: Chapter 2 introduce necessary background information as well as explaining important concepts. Chapter 3 defines the system requirements, setting the objective as well as presenting the (non)-functional requirements. Chapter 4 details the realization of the conception and present the used technologies. A view over the results of the system, as well as an evaluation, is given in Chapter 5. Finally, Chapter 6 concludes concept, development and findings of this thesis.

2

Fundamentals

In this chapter, principles necessary for understanding the work are explained: Bibliometrics and the fundamentals of stream processing are introduced. Then, the relationship between Twitter tweets and scientific publications is explained. In addition, the basics of trend calculation are introduced. Finally, related work is presented.

2.1 Bibliometrics

Scientific research is often based on existing research results to improve and develop new technologies or knowledge. In order to be able to use existing research results, there must first be aware of their existence, and they must therefore also be found and made available. Since not all research is equal in its research fields impact, specific bibliometric metrics and bibliometric indexes, databases of scientific works, have been introduced. Some of the most relevant indexes according to Hicks et al. are *Elsevier Scopus*[1], Google Scholar [2] and *Web of Science*[3] [4]. Such indexes allow the retrieval of ranked publications with additional metadata such as publication titles, authors, fields of study, citation count, date of publishing, and a DOI. A publication can be uniquely identified by its Digital Object Identifier, short DOI. Some indexes show abstracts depending on the licenses; however, most are referencing the original publisher website. Different techniques can be used to rank publications. One of the simplest is the raw citation count, the number of scientific papers that directly cite this work. However, studies have

[1] `https://www.elsevier.com/de-de/solutions/scopus`; accessed 7-November-2021
[2] `https://scholar.google.com`; accessed 7-November-2021
[3] `https://clarivate.com/webofsciencegroup/solutions/web-of-science/`; accessed 7-November-2021

shown shortcomings of this value alone. One of them is that publications are only be cited because of a high count, not because it remains relevant anymore [5] A different technique to rank publications is the so-called journal impact factor, which uses the number of citations from articles published in a year articles published in prior two years divided by the number of articles published in these prior years. Further, the h-index is used to describe the impact of an author based on the publication count, and citation count [6].

A common issue of these techniques is based on the citation count. Since the research process takes time, new publications may take weeks before their first citations are published. Before this, no data is available, and the publication will be ranked low by the indexes.

Altmetrics, short for “Alternative Metrics”, has been introduced to extend traditional bibliometrics adding more data to rank publications. One of the most known is the metric and the company Altmetric, established in 2011 [4]. Commonly, social media data is used for the Altmetric score, but news or blogs are included in some systems too. This data allows ranking the publications even if no citations have occurred yet. A study using a small dataset in a specific research field showed that metrics generated by Twitter cannot be replaced by traditional metrics and are therefore be considered as a supplement [7]. While generating and displaying data much faster than traditional techniques, the impact of Altmetrics depends on the platform, and the significant impact on social media, for example, does not mean the same as a high citation count.

2.2 Stream Processing / Event-Driven Architecture

Stream Processing is usually compared to *Batch Processing*, with the difference being that while Batch Processing requires all data to be available and stored, stream processing is using a stream of data only storing partially needed data for processing. This allows processing data in real-time, allowing it to be used in a wide variety of applications. It is used in Financial Trading systems, fraud detection as well as social media trend generation. Popular software to create such systems is *Apache Storm* or *Apache Spark*, that

allow receiving data streams, running processing tasks, storing events, and providing the calculated data over an API. *Event-driven Architectures* are system architectures based on the collection, communication, processing, and storing of events. This difference leads to a system running its calculations as soon as events are received, compared to traditional systems waiting to receive a request for processing. Apache Kafka is a popular software system to implement Event-Driven architecture. Unlike Apache Spark or Apache Storm, it allows building a flexible, robust System using external processors. The latter allows the development of processors with all the flexibility compared to the strict API usage of Apache Spark or Storm. Further, to use programming languages other than Java, supported APIs need to be available. Kafka's solution using a more message broker design allows using existing sources as well as distributed systems.

2.3 Publications and Twitter

Twitter is a microblogging social media service allowing people to post and retrieve messages. It is used by hundreds of millions of users and suits discussion very well compared to other social media platforms. The latter are called *tweets*. A tweet is restricted to 280 characters in length. A tweet, if not explicitly chosen not, is publicly accessible to see for everyone even without a Twitter account. A Twitter user will usually follow people and see what these people tweet about in a messaging timeline. Tweets can be liked, shared, retweeted, quoted, and responded to. Further, Twitter has a publicly available and free API allowing to retrieve, for example, tweets and author data. With the API, a filtered Stream endpoint allows continuously retrieving real-time tweet streams. Customizable rules can filter these streams. To help with research, Twitter allows an academic use of its API, which increases the limit set by Twitter.

The impact of social media on research and how scientists discuss publications and share ideas has been studied and is suggesting that Twitter can be a valuable contribution [8] but are also emphasizing that quality and therefore impact between individual tweets and discussions vary. Further, a study particularly analyzing Chinese publications showed results indicating that tweeted papers, although published in the same year and journal,

could retrieve 15% more citations compared to non tweeted papers [9]. A different study examining attitude and practices related to Citations on Twitter showed that citations, although different from traditionally, still impact the scholarly research [10]. Additionally, different methods have been proposed to measure the impact of such tweets. One such study introduced the Twitter Impact Factor (TIF), which is based on the number of retweets per original relevant tweet [11]. Similarly, Bornmann et al. introduced a *Twitter Index* to normalize twitter count in 2016 [12].

The difficulty of such metrics is founded in the extensive variability of impact in a tweet. The author, content, or type of tweet can massively influence how many people see and engage in the topic, resulting in different impacts. Researchers tweeting to their big like-minded following, starting discussions, impact magnitudes more than a bot with no followers retweeting simple URLs with no context. Therefore, a score is needed to quantify the impact of a single tweet. Fang et al. showed in a study from 2020 [13] that about 14.3% of tweets became unavailable within two years. However, this unavailability was higher for some publications and shows the importance of applying a score to recognize such impact-less tweets.

2.4 Trend Detection

Detecting, recognizing, and evaluating trends is an important field of research often used in market analysis and strategy assessment in the financial context. In the market context, trend detection is used to evaluate potential trades to find profitable ones. In social media, trends are rankings of the content generated by popularity, engagement, or other factors. Forums or News sites use similar rankings to ensure important new posts are shown. Different issues have to be accounted for to analyze data for trends. A time series allows *observations* of variables changing over time and is time-oriented. Time series data can be selected by a time range, allowing processing of the value progress. The data can be seen as 2-dimensional points (x, y) where x is the value at time y. *Outliers* are points in data that do not comply with an expected pattern, resulting in the need to be adjusted. This ensures it will not affect the data analysis negatively.

Anomalies, however, are patterns of time series that are distinguishable different from the expected curve. These Anomalies can be positive or negative, and if positive, they are often referred to as trends. Multiple techniques are used to rank or detect trending objects in time series data, each with different properties about *robustness* (not being affected by outliers) and *variability* (how fast a measurement reflects a trend or efficiency in the calculation). *Smoothing* can be used to help with the identification of trends by separating the signal from noise [14]. A signal, in this case, represents a pattern, while noise is all the outliers obstructing the recognition of these patterns. Smoothing techniques can usually be used for forecasting using the recognized pattern too. Smoothing methods can vary in complexity and can even work with seasonal data. Following, a few popular techniques are presented:

2.4.1 Exponential Moving Average (EMA)

The Exponential Moving Average is a technique used in data analysis and signal processing, allowing smooth data. It is a moving average, meaning only a subset of the newest n elements is used to evaluate newer data. It is defined as follows:

$$EMA = (1-\alpha)^{k-1} x_{n-N+1} + \alpha \sum_{s=0}^{N-2} (1-\alpha)^s x_{n-s}$$

where α is a weight, x is an infinite real sequence and N is a fixed number [15].

2.4.2 Kaufman's Efficiency Ratio (KER)

Kaufman introduced a metric to measure the efficiency of market movement, resulting in Kaufman's Efficiency Ratio [16]. This value is between 0 and 1. A linear movement and, therefore, the shortest path is required to achieve a value of 1. Any inefficiencies leading to a longer path will be reflected in values smaller than 1. Trending phases can be recognized by introducing a threshold value. A similar metric has been introduced by Tusher Chande [17]. The Chande Momentum Oscillator (CMO) can detect the same

phases; in fact, the absolute CMO multiplied by 100 equals the KER value. InfluxDB uses this circumstance to calculate KER [18].

2.4.3 Kaufman's Adaptive Moving Average (KAMA)

Kaufman's Adaptive Moving Average is based on an Exponential Moving Average, adjusting the weight α for every period introduced by Kaufman [16]. This weight is based on the Efficiency Ratio of the period, ranking more efficient periods higher. These adjustments account for market noise and volatility and can be used as an indicator for an overall trend.

2.4.4 Standard Deviation (SD)

Standard Deviation measures how closely the individual points are around the mean [19]. It is defined as follows:

$$SD = \sqrt{\frac{\sum (X_i - M)^2}{N - 1}}$$

where M is the mean, N is the count of values and X_i is representing a point at index i.

2.4.5 Exponential smoothing

Exponential smoothing is a simple yet effective technique to smooth and forecast time series data is. Compared to other techniques, there is no necessity of fitting a parametric model before creating, showing a good performance [20].
Holt winter's technique expands on exponential smoothing to account for trended and seasonal data, which is often called "double exponential smoothing".

2.4.6 Theil-Sen Slope Estimator

The Theil-Sen Slope Estimator can be used to estimate the magnitude of a monotonic trend, and was proposed first by Theil [21] in 1950 and further enhanced by Sen [22] in

1968. It is a popular non-parametric technique used to estimate linear trends, returning a slope value describing the trend. The idea of this technique is using the median of the slopes $(y_j - y_i)/(x_j - x_i)$ which are defined by all the pairs of the two-dimensional points (x_i, y_i) in the given set. Sen's extension removes pairs having the same x coordinate. This technique is robust against outliers and efficient to calculate [23].

2.4.7 Example: Hacker News Ranking Algorithm

Hacker News is a social news website that allows users to post stories, usually referencing external content but discussing internally. The ranking algorithm of these stories is available and discussed online [24]. For each story, a score is calculated, which is then used to rank the individual stories. The Score calculation Function looks essentially like this:

$$Score = \frac{(P-1)}{(T+2)^G}$$

where p is the voted points of a story, T is the time (in hours) since the submission, and G is the gravity, a constant. This Implementation will ensure that newer stories are ranked higher than old ones. The time frame can be adjusted using the constant G. The bigger G is, the shorter the period an average story will be ranked high.

Some discussed metrics will perform worse if not enough data is available. If only little data is available, outliers will have much more significant effects even on robust techniques. Before thresholds can be defined to determine a trending phase, data needs to be collected. To select and dial the options on different technique's data analysis of collected data is required.

2.5 Related Work

Collecting Twitter data is a common data source for Altmetrics due to its unique API. Some services only collect the Twitter data, while others use the collected data to score and ranking publications. The CrossRef Event Platform [25], which is open-source and accessible by an open API, collects event data about publications by using different

sources, e.g., social media platforms like Twitter. This data, however, is only collected and made available; no scoring or ranking is done. Similarly, PlumX Metrics is collecting social media data, including Twitter data. These are based on the occurrence count of events, tweets, for example. Differently than CrossRef, a web interface makes the data available, and rankings can be seen. Altmetric collects social media data and scores and ranks publications using collected data, including Twitter. While, however, Twitter data is analyzed, showing, for example, the author locations, there is no obvious scoring of individual tweets. However, one common shortcoming of these Services is that no trending publications are available as an open, accessible web page showing trends over a period of time.

3

Conception

This thesis aims to create a system that can monitor, analyze, process, and aggregate data in real-time to generate trends that can be viewed with a web-based interface. This results in different requirements, which are introduced in this Chapter.

The system must run 24/7 with as little downtime as possible to generate current, correct, and timely trends. The system should be able to automatically retrieve needed publication metadata and connect a tweet to a publication (switch). Further, it needs to allow the data to be queried by an API and visualized by a web interface. Challenges are that a constantly running system analyzing thousands of tweets every few hours generates loads of data that can slow down calculations and use considerable amounts.

Furthermore, the number of tweets in any given period cannot be controlled, which means there might be spikes that the system must handle. A constantly running system also needs ways to ensure updates are possible with little to no data loss. While the architecture is being designed to handle multiple potential event sources, this work is focused on using discussion data by Twitter. More precisely, a transformation of events to values is necessary to enable aggregation for a given publication. This value, called score, does qualify how important an event is. The score should consider the content of an event, the publicity it creates, and the sentiment of the content. Further, the score should be timing dependent on the publication age. This is an important aspect since there is a correlation between high tweet counts within a short time of publicity compared to later [26]. These scores can then be aggregated to create data for trend analysis. An accumulated sum of scores weighted by an exponential function is proposed to rank publications over time while shifting importance to new data. Such an accumulated sum reduces a score depending on its age and allows a higher ranking of publications

discussed more recently. This system should be able to run with low maintenance required and without using high-end server systems. This allows the system to run for an extended period of time by the ambalytics research project creating and analyzing trend data [27].

3.1 System Landscape Model

The System Landscape Model shows the software system concerning external users and systems (see Figure 3.1). The streaming platform runs dependably on Twitter and their users to create tweets; the latter is supplied to the processing system. The Frontend shows data of the stream processing platform API. Since the API is designed openly, it can be used by any other software project. However, the primary purpose of the API is to supply the web interface of "trends.ambalytics.com" to allow users to check trending publications.

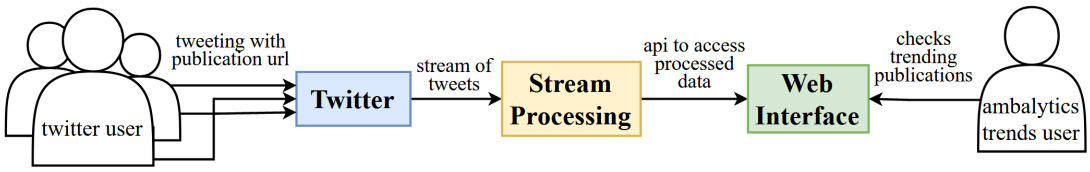

Figure 3.1: System Landscape Model

3.2 Software Architecture

The software architecture describes a platform to process, store, and visualize data and data trends from Twitter related to scientific publications. External data sources deliver events, especially tweets from Twitter, and external APIs retrieve data for publications and more. The processing platform needs to store its data and results in database systems. Therefore, a time series and a relational database are used to store the needed data efficiently. An API can then access these databases, which provides data access to a

front-end visualizing and presenting the data. The stream processing platform needs to be running nonstop and must be updated with little downtime and as little data loss as possible. Further, the system should be able to run on regular hardware. An overview of the architecture can be seen in Figure 3.2.

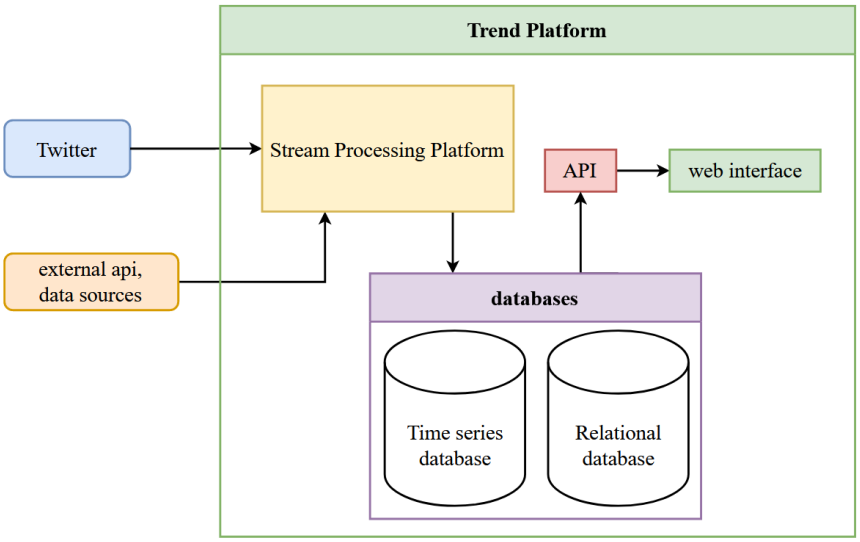

Figure 3.2: Software Architecture

3.3 Database Concept

During 24 hours, thousands of tweets are typically collected. Natively storing these directly in a MongoDB, or similar NoSQL Database, for example, would lead to massive storage needs decreasing performance drastically. Possible optimization steps include running the system on high-performance servers or optimizing data storage and processing steps. In general, the system should process data fast and run calculations for an extended period. Therefore, data is split into two categories.

The first category of data is *relational data*. It does not contain time information, is in a fixed format, and will store data selectable by publication, not time. This category of data can be stored in a relational database for efficient storage and quick query times.

The second category of data is *time-series data*. Each data point contains a timestamp and other values. Time-series databases are perfectly fit for storing and accessing this type of data efficiently. Certain run-time optimizations of time-series database systems allow quick access and quick calculations to data tracked over a period of time. Further, this kind of database allows running continuously or timed queries/tasks, optimizing the data for extended periods by down-sampling. The latter allows reducing the storage and performance needs a lot.

3.4 Requirements Analysis

After a thorough analysis of technical and business requirements, functional and non-functional requirements emerge as follows.

3.4.1 Functional Requirements

R-01 Retrieve Tweets using filtered streams
Get tweets as stream from Twitter by using a URL filter.
R-02 Migrate Tweet Schema
Map Twitter data schema to a generic data schema.

PP-01 Extract DOI from Tweet Data
Map Tweets to publications based on prioritized rules:
1) Try to extract DOI from Publisher URLs
2) Try to extract DOI from Meta Tags
3) Try to search for URL in CrossRef Events API
4) Try to search publication full text for DOI
PP-02 Link verified DOIs
Verify that the extracted DOI is existing, then add the DOI data to the Tweet Data.
PP-03 Store Publication Data
Add publication data to local databases by using the DOI as identifier as publication data

is needed for processing.

PP-04 Retrieve new Publication Data

If no Publication Data can be found in the local database, use external APIs to retrieve new Publication Data.

D-01 Tweet Data Model

Parse and handle raw Tweet JSON Data and extract important information from it.

D-02 Event Data Model

Provide a generic event data model that able to add further sources/systems, to map necessary information for processing, and contains publication data.

D-03 Aggregated Data Model

Provide a data model for aggregated data of publications, field of studies or authors.

S-01 Store Publication Data (Relational Data)

Provide methods, and interfaces to store and access publication data.

S-02 Store Processed Tweet Data (Relational Data)

Provide methods, and interfaces to store and access processed tweet data.

S-03 Store Processed Tweet Data (Time-series Data)

Provide methods, and interfaces to store and access descriptive statistics of timed event data.

S-04 Store Aggregated Data

Provide methods, and interfaces to store and access trend calculations with time-series data.

P-01 Extract Text Features

Extract relevant features from tweets, like “Twitter Entities”, “Hashtags”, “Author Location”, “Author Name” among others.

P-02 Text Analysis Methods

Provide methods to analyze text, i.e., to sanitize text, check for abstract similarity, and to calculate text sentiment and top words.

P-03 Score Features and Tweet

Score selected features, e.g., from text analysis, to generate a total “trend score” for an event.

A-01 Rank Publications for Trending
Publication should be ranked in different periods of time using a to be developed ranking function.
A-02 Calculate multiple different metrics
Compare and develop solutions to measure publication trends or publication impact on Twitter.
A-03 Tweet Top Publications automatically
Share a list of new trending publications on Twitter.
A-04 Setup down sampling of data
Reduce system resource requirements by optimizing the resolution of time series data.

V-01 Web-based User Interface
Provide a web-based user interface with charts to visualize trends and statistics.
V-02 Auto-Update
Update components in time to immediately show changes in data.
V-03 Statistics
Show statistics, for example, counts and scores for publications, authors and fields of studies.
V-04 Top Trending Publications
Show top trending publications in a table with a search function and the ability to sort different metrics.
V-05 Top Trending Fields of Study/Authors
Show the top trending authors/fields of studies in a table with a search function and the ability to sort different metrics.
V-06 Field of Study/Author Data
Provide a web page showing a specific author or field of study and visualize data aggregated for the respective entity.

C-01 Infrastructure Configuration

Provide configuration methods for hosts, service ports, message broker topics, service connections, service containers, databases, credentials.

3.4.2 Non-Functional Requirements

NF-01 Documentation

All components need documentation to ensure maintainability as well as further development.

NF-02 Reliable

The system must be running reliably to ensure continuous operation.

NF-03 Monitoring

The system must ensure to monitor itself and send an alert in case of non-function.

NF-04 Self correcting Container

A service container must detect errors on running and handle these to revert to a running state without losing data.

NF-05 Real-Time Processing

The system must be able to process events in real-time, i.e., within minutes of the origin time of an event.

NF-06 Significant Data Extraction

The System must process a significant percentage of events retrieved to ensure that calculated trends are correct.

NF-07 Reasonable System Requirements

The System should meet commodity hardware requirements, i.e., run on 16 GB of RAM and an average modern multicore CPU.

NF-08 Mobile-Friendly Frontend

The Frontend should be usable on a PC with a screen, mouse, and keyboard and be optimized for phones and other touch devices.

4

Implementation

This chapter presents the architecture, data models and implementations details of the trend stream processing system. First architecture and data state model will be presented following detail to used software and finally explaining the individual container used.

4.1 System Architecture

An overview of the architecture is shown in Figure 4.1. The developed proof-of-concept implementation is based on a Kafka broker connecting service containers. Each container is responsible for a set of tasks, e.g., linking Twitter tweets with publications. The proof-of-concept system is configured using a docker-compose file defining the Apache Kafka topics, settings, and tokens. Importantly, the container used in the system are configured and can be modified depending on environment variables, which runs the system locally for testing differently than the production system.

The container connected to the Kafka Broker responsible for processing events are called worker. Each worker consumes an event in a given state and processes it. A source, i.e., the amba-connector-twitter container, retrieves new tweeter from the Twitter API and creates new events on Kafka. These events are then consumed by the amba-worker-percolator linking the tweet events to a publication. After successful linking, the event state is updated from unlined to linked before being sent to Kafka. Some events may contain new publications not known by the system. These events are unknown events consumed by the amba-worker-pubfinder which is retrieving missing publication

data and storing it in the PostgreSQL. Linked events are consumed by the amba-worker-discussion processing and scoring the individual tweets. Finally, the amba-aggregator is storing the data in the InfluxDB and running the trend calculations.

Each container executes one particular task, resulting in more data added to the event. This is either the result of processing or retrieving data from external sources. Each container runs and maintains requests and database connections independently. Kafka allows to configure the worker in such way that all events are consumed once through all consumer using the same ID independently if they're just different threads or entirely different containers. Additionally, it keeps an index of consumed events, allowing a restart or update without loosing events.

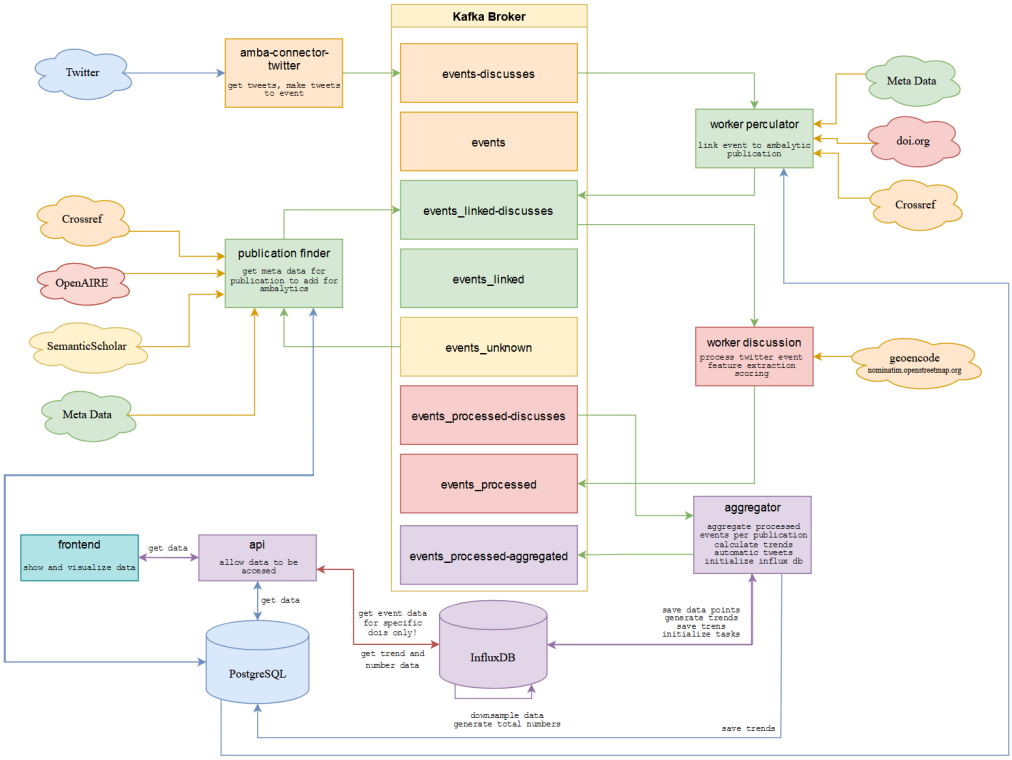

Figure 4.1: System Architecture Overview

This architecture is very robust and is designed to be running with as little downtime as possible. The pipeline itself allows for updates in production without losing data.

Additional containers can be set up to either integrate in the current processing pipeline or just use the event data as needed. Further, more sources ingesting events can be added. The developed API allows retrieving data stored in both the InfluxDB and PostgreSQL databases. The API is used by a web frontend visualizing the data.

4.2 Data State Model

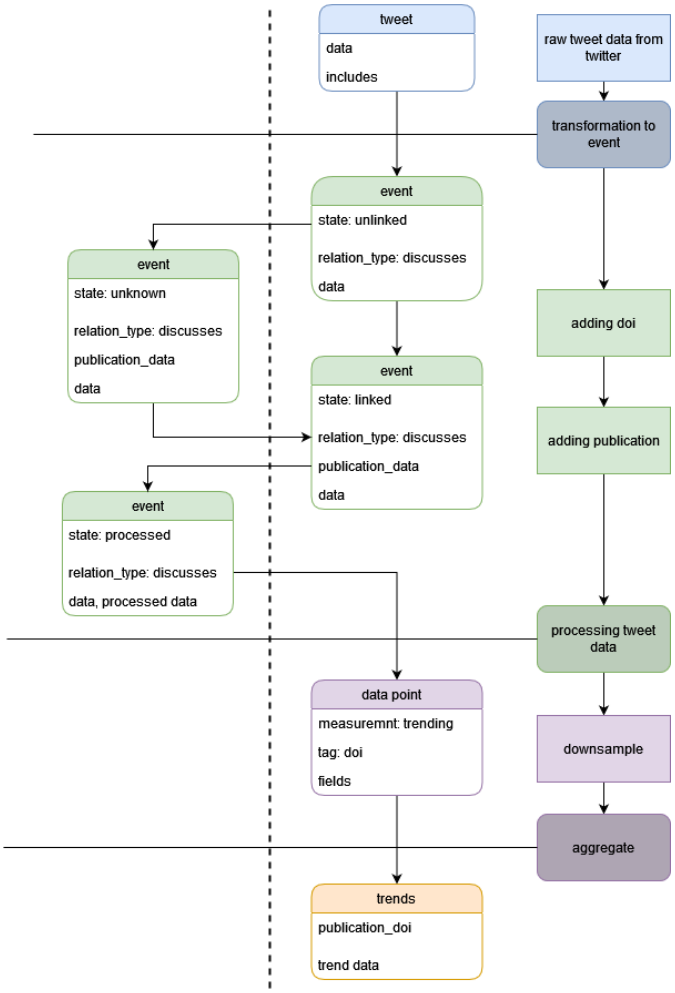

Figure 4.2: Data State Model

The Data State Model shows different data stages and highlights data transformations applied during processing of the developed streaming platform. First, tweet data obtained by Twitter is stored in JSON. The Twitter connector decodes, transforms and extracts necessary data such as creation time, tweet ID and author, into an own event model. The extracted data is used to set the subject attributes. The initial state as well as the type and the source are set by constants.

The event model is valid within the processing pipeline. For the processing of the events several states are necessary: the initial *unlinked* state denotes, that publication data for a tweet is missing/unlinked, while the *linked* event includes publication data. Internally, an *unlinked* tweet can become an *unknown* event that contains a known DOI but misses publication data. A *processed* event will add data to the subject data. Both events are separated in the Figure 4.2 since they maintain their data model since only existing, empty, field are filled.

Once a tweet event is linked, another data transformation is performed by adding scores and other statistical data about the event as a data point in a time-series database. The selected data is down sampled over time in the time-series database, while maintaining its data structure. Down sampling is a process of aggregating data to reduce its resolution while decreasing storage and computational resources.

The final transformation step aggregates events using linked DOIs. This changes the data to a publication-based type. Moreover, analysis results for fields of studies and authors can group multiple DOIs, respectively events together. The data is still not transformed, since its only further aggregated by the group function.

4.2.1 Docker

The implementation uses Docker as a platform for developing and running the pipeline. Using Dockerfiles, each container is its own virtual system running in a configured way optimized for its task. These individual container can then be run individually or setup with a docker compose file to run in a configured way to describe the whole

system infrastructure. Additionally, docker applications can be scaled, managed and automatically deployed by Kubernetes.

To set up keys as well as environment defaults for testing or production deployment, individual environment files can be used. Environment files define one or more environment variables and are automatically detected while running the docker compose command, and ensure the separation of secrets from code and version control systems.

Since all programs run in their own virtual systems, a crash of one program does not crash other containers. Additionally, Docker Hub allows hosting docker systems that run applications already configured. This allows the usage of databases, message brokers and any kind of software provided by Docker Hub in a conventient way.

Usually, the setup can be done by either setting environment variables or mounting a file. This allows high reusability of software and abstracts the infrastructure level. These systems can be run either on a single machine or be executed distributed over many systems, allowing for high performance and higher availability.

4.2.2 Apache Kafka

Apache Kafka is a streaming platform to throughput and process events at high sample rates. Kafka is currently one of the most popular frameworks and mainly used for ingesting data in processing systems [28]. In the current setup, Kafka uses ZooKeeper, which is a system coordinating task for Kafka.

A usual infrastructure pattern is to use Kafka as a central event processing system, and to build processing services that are coordinated by events sent though Kafka. Kafka allows a high throughput while being reliable, available and keeping storage. A Kafka service can be restarted without losing any events that have been collected by Kafka, since it will keep an index on which events have been consumed. This allows for the creation of a stable platform which can be scaled to any needed size to process any amount of data.

Although Kafka allows processing to be done directly, it is used only to connect the services and reliably distribute events. This is done by creating so-called *consumers* and *producers*.

A consumer is a service that connects a topic setup in Kafka, and receives new event. A producer, however, ingests new events into a Kafka topic. Kafka allows for the setup of topics, which can be used to group events and allow a consumer to only read the specific data. To be more precise, different topics are created corresponding to the state of events that will be configured. In order to improve efficiency of the system, additional topics are created especially reserved for discussion type events. Since the main load of events will be of type discussion, using an own topic only containing relevant types of events ensures only relevant data is consumed by the processors.

4.2.3 PostgreSQL

In order to implement the relational database, a PostgreSQL container is used. PostgreSQL is a widely used open-source database that can be run in a docker container.

The data schema, as shown in Figure 4.3, is designed to be compatible to the schema used by ambalytics to store publications while at the same time extending it to store information about discussion events as well as trending data.

Table “Publication” is the central table storing publication data. The table is referenced in other tables by a unique “DOI” field. In order to extend the information about the publications, there are tables to store “fields of study”, “authors” and “metadata sources”. Further, known citations and references of a publication are stored in their tables, “publication_citation“ and “publication_reference“, using a combined key based on the DOI’s of the publications. Information about unresolvable DOI’s are stored in an extra table, “publication_not_found“, to analyze the success rate of the pub finder.

Discussion data stored in the PostgreSQL is aggregated for each DOI. Thereby, each data point has a “type”, “value” and “count” field. The generic definition of the data schema allows adding or removing types without changing the database schema. The

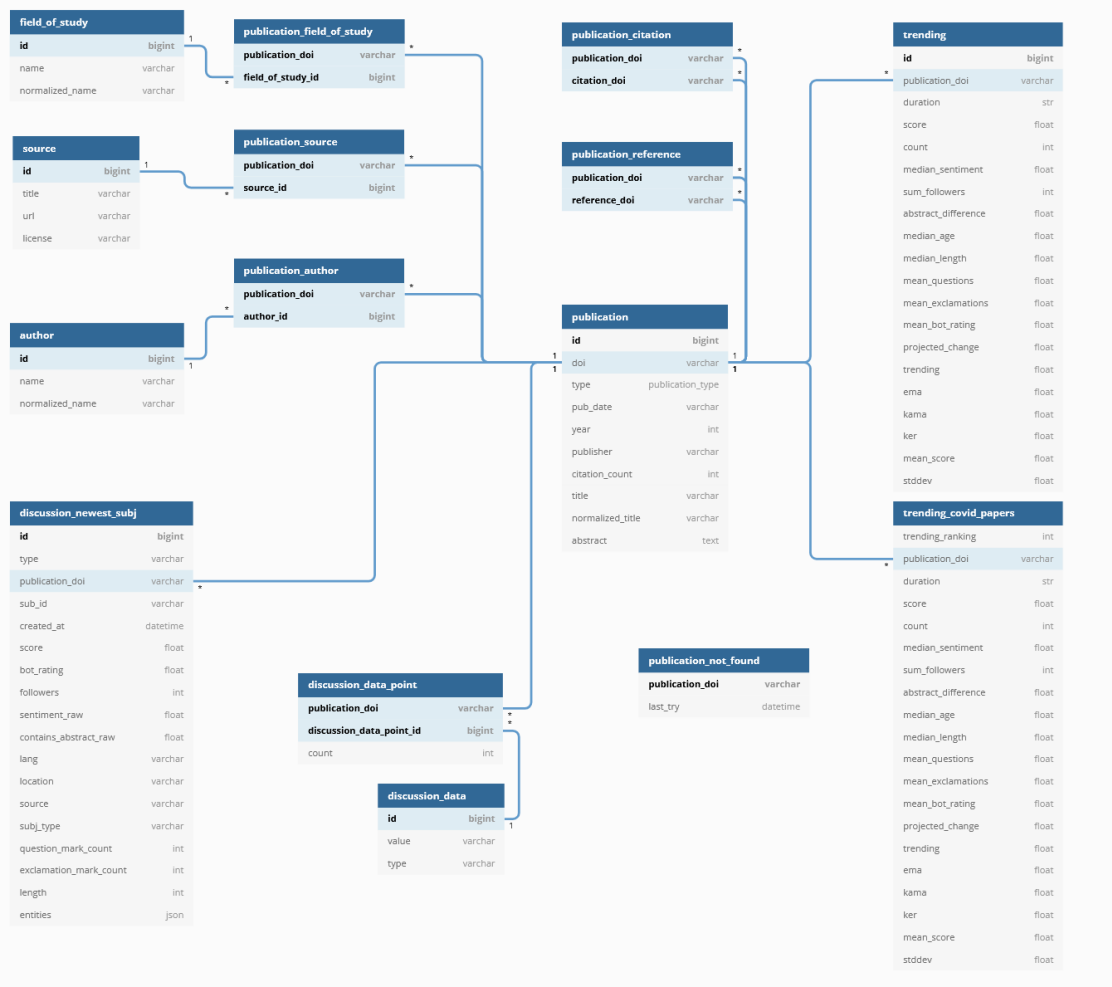

Figure 4.3: PostgreSQL Data Schema

Table named 'discussion_newest_subj' is used to store the latest tweet with processed information related to a publication.

Lastly, trending publications are stored in a separate table, saving the newest trending values as well as the duration of the trend calculation. COVID publications are determined by checking the top 3 entities selected by twitter to match the term "COVID". Since such a query is costly and slow, a materialized view is created, which is refreshed every time the trending table updates. The refresh is configured to return the old table while a

recalculation is done to ensure no performance issues created by a request. Fields of study and authors tables can be joined fast enough in real time by using indexes.

4.2.4 InfluxDB

To implement the time-series database, an InfluxDB container is used. InfluxDB is one of the most popular time-series database to store and query time-series data efficiently. A new Kafka event creates a InfluxDB data point with the following data about the event: “bot rating”, “abstract similarity”, “count”, “exclamations”, “followers”, “length”, “questions”, “score”, “sentiment”, “content score”, “time score”, “type factor”, and “user score”.

The data schema persists as it is aggregated while being down sampled, except for the last 4 values used to calculate the event score. These values are stored to evaluate the weights used by the discussion worker and are not permanently relevant.

More important, API queries should always be limited by using DOI or by selecting buckets storing prepared data, to ensure little cardinality in queries that result in less API response time. Since the number of publications is high (consider 100000 publications after seven days of Twitter data processing), the resolution and event count of the events is reduced by aggregating them, resulting in reduced storage and cardinality. This helps with running trend calculations, statistics and general data query tasks.

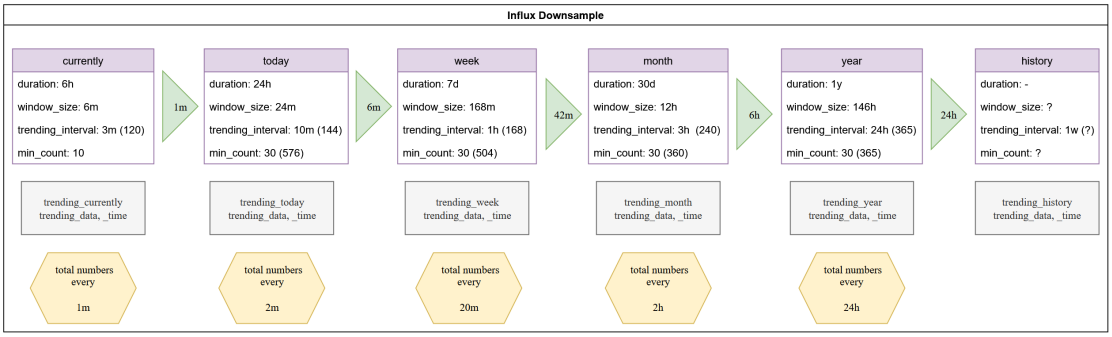

Figure 4.4: InfluxDB Downsampling and Total Number Generation Schema

Tasks are set up to automatically run in defined time intervals (see Figure 4.4), and read a set interval of time, aggregate it and store it in a new bucket. The task will read multiple

intervals of time, ensuring that an event is added even if a processing delay, for example publication data retrieval, added the event with a bigger lag than a single down sampling interval would cover.

Performance in InfluxDB is largely dependent on the cardinality of stored data, i.e., the number of unique measurement, tag set, and field key combinations in an InfluxDB bucket [1]. In order to access data optimized for specific publications in an InfluxDB, a DOI is set as a tag, allowing influx to retrieve the selected data directly without reading and filtering all the data. Note that for tags indices are applied that are similar to the ones in relational databases.

Furthermore, it must be ensured that queries are not exceeding a certain cardinality to further ensure fast API responses within read timeouts as well as ensuring a responsive system still able to run other tasks. This is important since not just the API may query data, but additional tasks are set up to handle data, optimize storage and querying as well as to calculate metrics to detect trends.

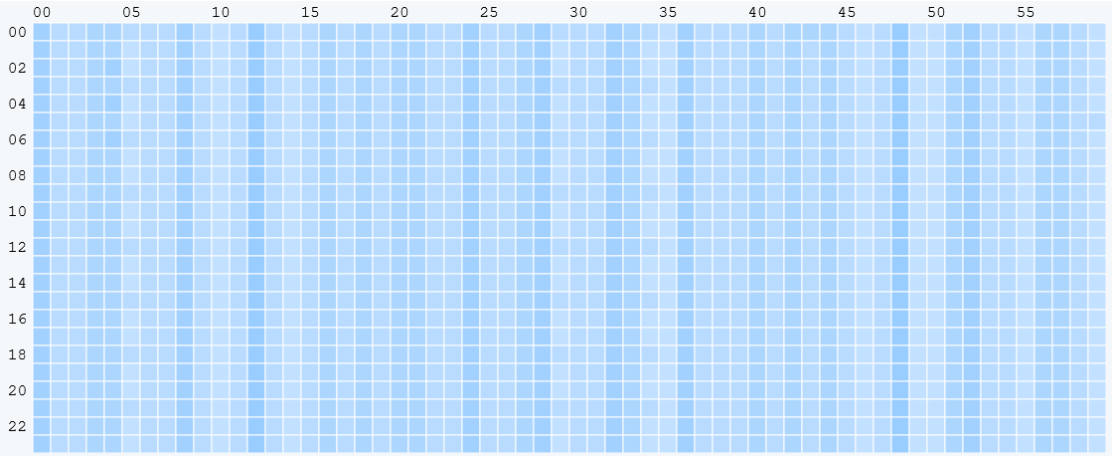

Figure 4.5: Heatmap of CRON Jobs Collisions (darker color denote more running jobs)

In order to manage these task efficiently, CRON jobs are used to start these tasks at a defined time of day. The individual tasks are spaced out over the day, running at their defined starting times based on system time shared by all containers. A visualization of

[1] https://docs.influxdata.com/influxdb/v2.0/reference/glossary/#series-cardinality; accessed 5-November-2021

tasks can be seen in Figure 4.5. Every minute of every hour is displayed as cell, a darker color indicates more tasks running simultaneously. At a maximum, 5 Tasks are running in the same minute. However, these tasks are varying in cardinality and therefore time duration needed to finish. Table 4.1 shows an overview of all configured CRON jobs. In short, regular task are set to start at the beginning of a minute while longer running task are started with an offset allowing for fast tasks to be finished before longer running queries are executed to ensure maximum performance to each individual task.

CRON Time	Job Name	Time Offset	Duration
`*/3 * * * *`	trend currently	`15s`	`>5s`
`* * * * *`	numbers currently	`4s`	`1s`
`3-59/5 * * * *`	mail alert	`1s`	`<1s`
`2-59/5 * * * *`	mail error	-	`<1s`
`1-59/10 * * * *`	trend today	`15s`	`>5s`
`*/4 * * * *`	numbers today	`10s`	`3s`
`*/4 * * * *`	task today	`6s`	`1s`
`25 * * * *`	trend week	`15s`	`>5s`
`26 * * * *`	numbers week	`15s`	`>5s`
`19 * * * *`	task week	`15s`	`>5s`
`5 */3 * * *`	trend month	`15s`	`>5s`
`29 */3 * * *`	numbers month	`15s`	`>5s`
`34 */3 * * *`	task month	`15s`	`>5s`
`4 2 * * *`	trend year	`15s`	`>5s`
`4 4 * * *`	numbers year	`15s`	`>5s`
`4 6 * * *`	task year	`15s`	`>5s`
`43 */72 * * *`	task history	`15s`	`>5s`
`59 5 * * *`	covid papers	`15s`	`>5s`

Table 4.1: Scheduled InfluxDB CRON Jobs

Time intervals for down sampled data ensure a maximum number of elements to run queries against, which is needed since the publication count is growing with the interval time. However, new, down sampled data can only be as recent as the down sample tasks has run, resulting in a time lag. Compared to the total duration, the lag can be dismissed. Most fields of an event are aggregated by using a *mean* function. The latter is a stream aggregator in flux and does not require loading all individual data points into memory, for example required by a median function. Using the mean further allows consecutive down sampling compared to the median. Follower counts, tweet scores and

counts are aggregated using the *sum* function in flux. Furthermore, InfluxDB tasks are created that update numbers regularly to allow fast statistical calculations over all data. Metrics data generated by the tasks are stored in an own bucket to foster fast retrieval.

In order to monitor API response speeds, another bucket stores all API requests with their respective URL and response time. Two tasks are set up to monitor the system that check the number of new events. Based on a defined threshold, an email is sent notifying about issues or a system failure, for example, if there are no new events over a specific period of time. An overview of buckets and their retention policy can be found in Table 4.2. Thereby, additional retention time was added (compared to the actual used duration of the buckets) which is a safety margin in order to ensure no data is lost while data may be processed.

Bucket Name	Retention Policy
currently	7h
today	25h
week	7d 1h
month	30d 1h
year	365d 1h
numbers	7d
api_monitor	7d
history	-

Table 4.2: InfluxDB Buckets and Bucket Retention Policy

4.2.5 Amba Analysis Streams Package

The Amba Analysis Streams package is used as a Kafka connection wrapper to abstract from infrastructure implementation details by providing functions to connect to Kafka and PostgreSQL. It defines the event model used in the streaming platform and provides base consumer and producer classes. The package is implemented as a python package that is hosted on pypi.org, and documented with mkdocs.

The consumer and producer are capable of running in multiple processes to allow for parallel processing to better utilize modern CPUs. Both have built in monitoring capabilities: a counter shared by all processes is updated for each processed event. A

thread running a function every few seconds is checking the counter and resetting it. If no data is processed over a defined period of time (meaning multiple consecutive check function runs), the container is restarted automatically by closing all python processed. This heart beat function ensures that even unforeseeable errors, such as container crashes or blockings are resolved by restarting the container and providing a clean system state.

4.3 Components

4.3.1 Twitter Connector

The twitter connector is the main source of event data. It connects the Twitter API with the Amba Analysis Stream platform. Therefore, the connector uses a combination of keys, and access to a *filtered stream* with a defined set of attributes. The filtered stream, a Twitter API feature, is configured per account prior to accessing the stream. Filters may contain a list of rules, in our case a list of domains that contain the most popular publisher as well as plain doi.org URLs, see Table A.1. If tweets match these rules, they are added to the filtered stream and are accessible by the twitter connector. The Twitter connector is implemented in python and uses the amba-event-streams package.

For a successful connection and initiation of a tweet stream, a so-called *Bearer Token* that identifies the account is necessary. The token further ensures that the correct rules are applied. The Twitter API account used is an academic account provided by Twitter, that allows retrieving up to 10 million tweets a month. The data retrieved from Twitter can be adjusted by setting “keys” in the request. These keys allow to extend the retrieved Twitter data in JSON format and may add data about the tweet author, referenced tweets and additional tweet data.

Once established, the connection persists and every received tweet is decoded and transformed into an “amba-event” object. At this point, the subject of the amba-event, i.e., the tweet, is filled with data extracted from the JSON. The state of this event is set to *unlinked* since the connection to the object, the publication, has not been made yet.

Once completed setting the subject and general attributes, the event is published to Kafka.

In case of connection issues or any sorts of error, the connector is able to recognize such errors and restarts the connection. In order to ensure reliability, a processing thread periodically checks how many tweets have been received. If there are no tweets for multiple consecutive checks, the twitter connector container is restarted to reset to working state.

4.3.2 Percolator

The percolator component connects a discussion event, i.e., a tweet, to a publication or at least a DOI. Without the connection, the event can not be further processed by the analytical components. The percolator is based on ideas developed by CrossRef, that use a percolator for linking events as well [29]. The percolator is developed in python using the amba-event-stream package. It runs as a docker container.

In order to improve linking throughput, three processes run simultaneously in the percolator to reduce the time overhead generated by waiting on web responses. Since all processes will connect to Kafka by using the same consumer ID, identifying them to be the same type, Kafka automatically shares the events equally between the 3 processes. A DOI resolver class is used, defining static functions to retrieve the DOI from event data.

One way of linking data is using Meta Tags, these are HTML tags embedded in the source code of a webpage. These tags <meta> will not be displayed by a browser and are used to specify information to automatic systems processing the page. Traditionally, these have been used by Search Engines and are nowadays additionally used to provide data for custom titles, descriptions and all kinds of crawler information. Each meta tag has a name attribute as well as a content attribute. The name is used to identify what metadata is stored, the content will contain the actual value. This allows to filter the tags for only relevant data which can easily be identified without the need of analyzing the content. This ensures correct data compared to a full-text analysis, which may result in

wrong results since no context analysis is done. For Example, a citation may be found using the full-text analysis but wrongly be used to link the event.

Multiple methods of extraction are used to find the DOI for given discussion events. An overview of the process can be seen in Figure 4.6. First, the tweet data is checked for URLs. If that fails or these URLs do not contain a DOI, additionally, all referenced tweets will be checked. A response may not contain an URL itself but reference the original tweet the response is responding to. Thus, the URL needs to be linked. Note that multiple URLs are available from the Twitter API, that differentiate in their characteristics: a short URL, an expanded URL as well as an unwound URL. Since the DOI ideally can be extracted from the URL itself, the expanded and unwound URL are both checked. Sometimes only the expanded URL contains the DOI, while otherwhile the unwound URL does.

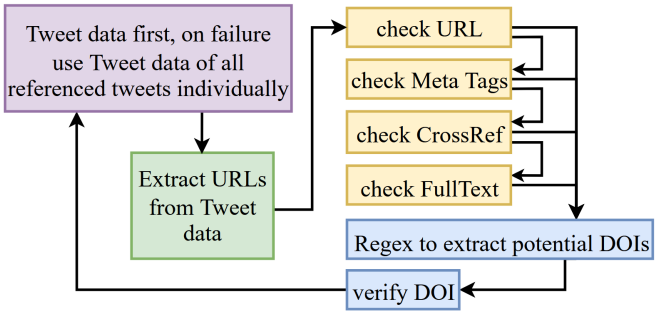

Figure 4.6: Processing Schema of Linking a Tweet with a Publication DOI

The function of linking an URL with a DOI is cached up to 10000 URLs by a Least recently used (LRU) cache. Caching this function allows for very little storage needed to cache since function parameter and result are both small. The cache generally allows faster processing and less need for requests in general. The LRU caching strategy ensures the most used are staying in the cache. The data is static and likely to not change ensuring time is not relevant for cache to expire.

A URL to be linked is first checked with multiple regex to extract potential DOIs. These regex are based on CrossRef but extended to suit the publisher URLs registered in the

system. Since the DOI specification is not extremely strict, there needs to be verification of a potential DOI.

To confirmation that a DOI exists, a check is done by sending a request to doi.org. Due to DOI specification, an existing DOI is linking to the correct publication. If no DOI can be extracted using the regex on the URL, a request is sent to the CrossRef event API. The event rest API of CrossRef allows checking if they already linked a URL to a publication. If again no DOI can be retrieved, a request using the URL is sent. The response is subsequently checked for a set list of meta tags. If one or more such tags are found, their values are extracted and a DOI linking started on each of them. The last function if all other fail is to check the fulltext HTML for a DOI.

All publisher URLs are checked to ensure their working in the system. However their articles and pages on their sites which are not registered with a DOI. This means the percolator will not be able to link tweets linking to these URLs.

Once a DOI for a tweet is found, the database is queried for the publication data. If the data can be retrieved it will be added to the event, the event is then set to linked and published. Otherwise, it is set to unknown before sending it to Kafka. If no DOI can be found for a tweet, it's not further processed in the pipeline and the data will be lost.

4.3.3 Pubfinder

The Pubfinder component retrieves publication metadata and, if possible, abstracts for publications that are linked to an event. The Pubfinder component is implemented in python by using the amba-event-streams package.

The process of retrieving publication data can be seen in Figure 4.7. The Pubfinder is built around its main worker utilizing individual sources which are integrated similar. This architecture allows adding, remove or modify sources since their only difference is in the source retrieval and transformation. This means the Kafka and database connections are maintained by the main worker, while the sources are separated to only retrieve information for a given DOI.

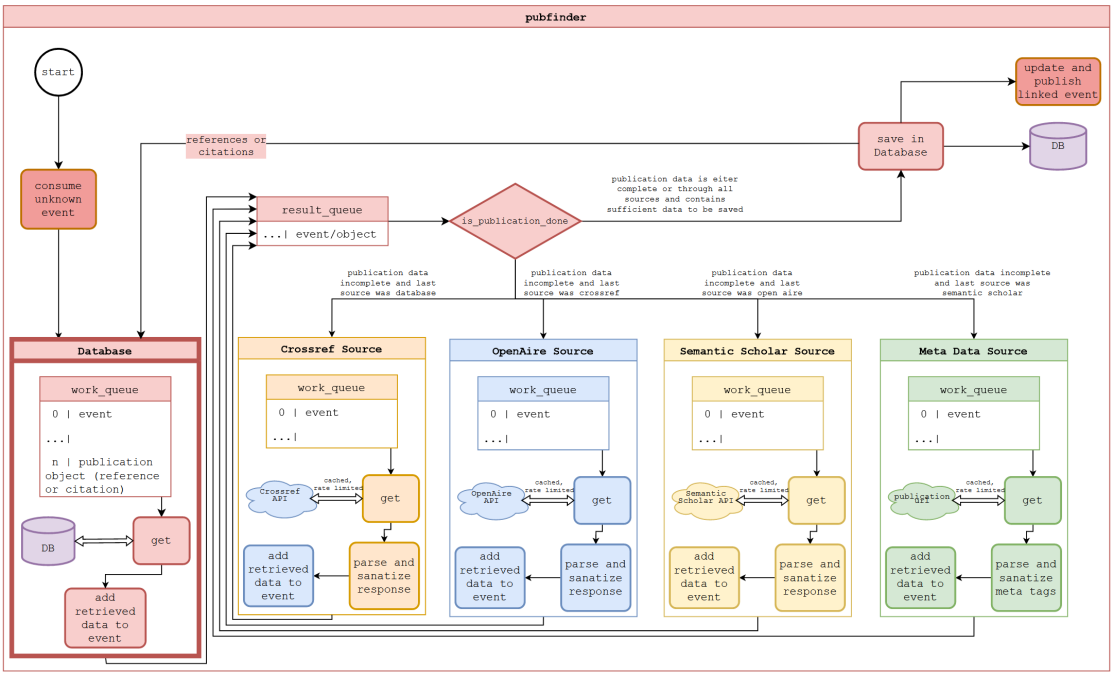

Figure 4.7: Process Overview of Retrieving Publication Data

All sources have their own retrieval mechanisms and data transformations, as well as API limits that must be regarded. The following sources are used: ambalytics [27], CrossRef [1], OpenAIRE [30], Semantic Scholar [31] and Meta Tags. CrossRef, OpenAIRE and Semantic Scholar are service that offer APIs, while Meta Tags are Dublin Core HTML header meta tags that may be embedded in the respective publication web page of the publisher. Table 4.3 shows the individual fields that can be retrieved by a source, as well as their rate limit and URL. Ambalytics is the internal database (see 4.7), is technically not a real source and therefore not listed in the table. Even though CrossRef and Meta Tags are not rate limited, the number of requests per time unit is limited to comply with the recommended usage of the respective APIs.

Since all sources are needed for a full check, the maximum throughput is limited by the slowest source or in this case, the source with the lowest rate limit. However, using this rate limit as base for the worst-case calculation 100 publications per 5 minutes will amount to 28800 per 24h period. Each source is integrated similarly to the main python program, but runs it own threads independently. In order to allow fast and

Source	CrossRef	OpenAIRE	Semantic Scholar	Meta Tags
URL	api.crossref.org	develop.openaire.eu	api.semanticscholar.org	-
Rate Limit	-	3600/h	100/5m	-
Title	x	x	x	x
Authors	x	x	x	x
Year	x	x	x	x
Type	x	-	-	-
Citation Count	x	-	x	x
Pub Date	x	x	-	x
Abstract	x	x	x	x
References	x	-	-	-
Citations	x	-	x	x
Publisher	x	x	x	x
Field Of Study	x	x	x	x
License	x	-	-	-

Table 4.3: Publication Meta Data Sources

parallel workloads, the process can either be implemented asynchronously allowing to use the await syntax to avoid blocking the thread or using queues and multi-threading. Unfortunately, the support of asynchronous python libraries is still limited and running blocking functions in asynchronous environment would defeat its workings. Following, the Pubfinder is developed based on the *queue architecture pattern*.

Queues allow each source to run indecently and multi-threaded while reading and writing events into and from queues. A worker queue for each source contains all events, and all sources share a result queue. The queue is implemented as *deque* allowing for thread-safe operation and adding references and citations. Using a deque allows to add new DOI's to the front while adding references and citations to the end, resulting in earlier retrieval for the more important event publications. The management of the queues is done in the Pubfinder, checking for finished publications and adding an element to the correct source if more data is required.

Since a publication resolution runs through each source until it is complete and implements different limits and speeds, the order is critical for overall speed. The API with the lowest rate limit should be last. This order of sources reduces the total request count affecting the limit since publications are more likely to be complete before and therefore reducing the count.

Further, it will not slow down the processing. Each source uses the same base way of finding data. The given DOI will request the data in a defined way while ensuring API limits are kept. Once data is retrieved, the needed data is extracted and mapped into the internal publication structure. If a source adds data, it will also add its source. Finally, the publication data is added to the event, and the event is added to the result queue.

Once a publication is complete or run through all sources and contains enough data, it is stored in PostgreSQL. Then publication data is added to the event, the event state is changed to *linked*, and finally, the event is sent to Kafka.

4.3.4 Discussion Worker

The discussion worker, more accurately the Twitter worker, is a component that transforms event data to processed data, especially the score allowing to rank and qualify the tweet itself. Therefore, tweet features that allow scoring need to be extracted from the tweet, and a total score has to be calculated based on all features.

The developed scoring mechanism allows representing the impact of an event on the discussion of a publication and, therefore, the publication itself. It is implemented in python using the amba-event-streams package.

Hashtags, Entities, Author Name and Location, and “top used tweet words” are extracted from a tweet but are not used to generate a score (since this data is interesting to be collected but does not qualify the tweet at all). Author Location extracts the location data from the Twitter supplied author data. This location data is a user-defined string that is geo-encoded using a free service[2].

The total tweet score is a weighted sum of part scores multiplied by a factor generated by its type. While not changing the content of a tweet, the tweet type is a significant indicator of its impact. While a retweet is the fastest way to tweet, it does not add personal options and is much less likely to be seen by people in their timeline, thus resulting in the lowest possible factor. On the other hand, a quoted tweet and a response have a bigger chance to be seen and add value to the discussion and therefore resulting in a higher factor.

[2] `https://nominatim.openstreetmap.org`; accessed 30-October-2021

Type Factor		Score Abstract Similarity		Score Sentiment		Followers
quoted	0.6	>0.9	3	>0.6	10	$\log_2 followers$
replied_to	0.7	>0.8	5	>0.33	9	
retweet	0.1	>0.5	10	>0.1	7	
tweet	1	>0.2	3	<-0.1	2	
		else	1	<-0.33	1	
				<-0.6	0	
				else	5	

Table 4.4: Meta Score Calculation

Finally, an original tweet is the highest factor since it is likely to start a discussion. The different types and their respective factors can be seen in Table 4.4.

The second-biggest factor in the impact of a tweet, and therefore the highest weighted, is the tweet author itself. The number of followers that can see the tweet, whether they are verified or not, and the bot detection is significant for the author's scoring. Details to each of the individual scoring can be seen in Table 4.5.

The tweet content is essential for scoring as well. In order to generate a content score, the length, sentiment, and percentage of abstract matching of the tweet are considered. Sentiment and Abstract similarity are calculated using the *Spacy framework* with eight language packages (de, es, en, fr, ja, it, ru, pl) [32]. In the case of unknown languages, neutral values are returned. Further text preprocessing is done to improve results and performance. Therefore, all stop words, short words with less than three letters, URLs, and words neither a Noun, Propn (proper Noun), or Verb are removed. The sentiment varying between 1 (positive) and -1 (negative) is linearly in buckets over proportional, favoring a positive sentiment. The abstract similarity is a bit more complicated to score. While a high value is bad since it is not adding anything, a low value indicates that the tweet content is likely not about the publication content. The length scoring bucketing is in three main buckets; one is just a link or a few words, the second is a sentence max, and the last requires a few words. Exact values can be seen in Table 4.4.

Furthermore, a score is calculated based on the time that has passed since the publication was published. The score is based on studies showing the importance of early sharing increasing the citation count in the future. The most crucial time range is a week.

Score Length		Score Time	Score Bot		Score Verified	
<50	3	$\log(X))/\log(1/7)+3)*10$	no Bot	10	verified	10
<100	6	score <= 30	Bot	1	not verified	5
else 100	10	score >= 1				

Table 4.5: Content Score Calculation

The formula used can be seen in Table 4.5. The values are limited in both directions and will always be between 1 and 30. Additionally, it is helping to highlight new research. Its weight to the overall score is relatively low.

Finally, the following weighted sum with the following weights is used to calculate the total score of the tweet.

$$score = type_{factor} * (3 * score_{time} + 6 * score_{user} + 5 * score_{content})$$

The score, the extracted data, and features are then stored in the event, which after a status update to processed will be sent to Kafka.

4.3.5 Aggregator

The Aggregator is implemented in python using Faust, a python stream processing library. The Aggregator is based on the ideas of Kafka streams. Even though Faust is built for stream processing, it is designed for more manageable and smaller tasks. However, Faust is well integrated into Kafka and allows an easy setup for asynchronous multiprocessing, event-driven processing as well as time-driven processing. Processing is achieved through agents which can run defined by CRON Syntax, allowing the scheduling to be in line with the InfluxDB.

This Component is the primary ingesting source to the InfluxDB and is responsible for the initial setup of the InfluxDB. All needed buckets, as well as the needed tasks, need to be created. The Aggregator uses a key supplied by the environment and used by the InfluxDB to allow access. The Kafka connection consumes new processed discussion events and writes their relevant data into the InfluxDB. Since the writing rate

is relatively low, the data is written synchronously as soon as possible. To ensure the writing succeeds even if their network issues or the InfluxDB is temporarily unavailable, retries are set up in case of failure.

The main task of the Aggregator is the trend calculation. Its setup is together with the InfluxDB needed to ensure correct trending calculations and schedules. All trend calculations are run as CRON agents, meaning they will always run at a specific time. The times are coordinated with the InfluxDB tasks. An overview of all tasks and their running time can be seen in Figure 4.5. Compared to timer agents, this allows restarts without resetting the timer and keeping the calculation at regular intervals. Running multiple trending queries on the InfluxDB increases the system requirements to ensure timely calculations before running out of time, resulting in read time-outs. Further, it allows optimization at which time what trend is calculated to avoid collisions of the calculations. A queue is used to avoid such collisions and ensure the calculations are not run directly. Instead, all calculations run sequential, reducing the load on the database. Trends are always started with an offset similar to the long-running task setup in InfluxDB, ensuring better resource management.

Before the trends are calculated, the influx is queried to retrieve only publications that have enough events. A minimum of events needs to exist to be considered helpful to the trend calculation. The minimum significantly reduces the series cardinality and ensures that prediction and trend algorithms have enough valid data to produce sensible results. Using these DOIs first, a query run in the influx calculates most of the trending values. After that, it retrieves needed data to calculate the Theil-sen's Slope Estimator value using *pyMannKendall* [33]. To reduce the load on the influx and reduce single query times, the DOIs are split in a list of 200, ensuring the cardinality stays low and the system load relatively low. The added overhead timewise due to multiple requests is not relevant.

The InfluxDB does most of our calculations, reducing the need to transfer data and allowing an optimal and efficient calculation. These calculations are split into two categories. One uses windowed data meaning aggregating multiple events within a time window, the durations of which can be seen in Table 4.6. If no events are available, a 0 window is created to ensure evenly spaced data needed for Holt Winter's prediction [34]

Duration	Gravity	Window Duration
6 Hours	-0.00036	6 Minutes
24 Hours	-0.000025	24 Minutes
7 Days	-0.00000357142857	168 Minutes
30 Days	-0.000000833333333	12 Hours
365 Days	-0.000000833333333	146 Hours

Table 4.6: Trending Score Configuration Values

and moving averages. The other categories used the event data directly to calculate the mean values and the trending score, the sum of all exponentially weighted scores of a publication, with each weighted score defined as:

$$score_{weighted} = score_{event} * \exp{(gravity * \frac{time_{now} - time_{event}}{10^9})}$$

Where time is a timestamp in nanoseconds and gravity is a negative value selected for a duration of trend calculation. The negative gravities used ensure a that newer events, with less time differences, will be exponentially higher weighted than older and are defined as seen in Table 4.6. Depending on the gravity, the time duration most relevant can be adjusted. However, all events will always be used to calculate the score, since the weight is always bigger than 0.

Once all trends are calculated, all old trending rows are removed from the PostgreSQL. Then the new ones are stored. Additionally, calculation results are stored with the time the calculation occurred that allows analyzing the development of trends over a period of time. Finally, new trends wilt trigger an update to refresh the materialized view of trending COVID-19 papers. Lastly, the Aggregator uses an agent to retrieve the top three relevant publications every morning, with COVID-19 as a top entity. The tweet content is generated by mixing fixed strings and the publication data. The title of the publications is shortened to follow tweet restrictions. A URL to the corresponding web page is added too. The tweet is then sent to the Twitter API using the bot credentials read from the environment.

4.3.6 API

The implementation of the API is based on *FastAPI*[3], a python high-performance framework that is running in a docker container. There are four primary REST endpoints: publications, authors, field of study, and statistics. Further, a system check is available to check if the API and the system are working as expected. The API is automatically generating interactive documentation with open API JSON. All responses are timed; these times are added to the result and stored in the InfluxDB. The calculated times allow monitoring and identifying slow queries. Further, all results over 1000 Bytes are being compressed with GZIP, reducing the network data. The PostgreSQL access is implemented using SQLAlchemy[4] and InfluxDB using their respective clients.

The “publications” endpoint allows querying publications, trending publications, and trending publications for a specific author or field of study and retrieving a single publication. Trending publications can be retrieved with offset, order, limit, search, and for a specific duration to allow them to be ideally queried for a table loading the data as needed.

Fields of study are similar to authors in aggregating publications. Their endpoints allow retrieving single elements as well as data suited for tables. Further access to trending fields of studies and authors are possible.

The “stats” endpoint allows querying data extending the trending table data to analyze publications further. It allows access to data from PostgreSQL and data from the influx specifically for a given duration. Nearly all stats endpoints offer an option to query data for either a publication, an author, or a field of study.

The API is adjusted to the database scheme and will not query the InfluxDB for slow high cardinality queries. Instead, it uses PostgreSQL with optimized InfluxDB queries and specific buckets to keep the response times low and responsibility high.

[3] `https://fastapi.tiangolo.com/`; accessed 10-November-2021
[4] `https://www.sqlalchemy.org/`; accessed 10-November-2021

4.3.7 Frontend

The Frontend is implemented in JavaScript based on Vue3 and the *PrimeVue*[5] *UI component library*. The Frontend is a single page application, i.e., everything is contained and hosted with one HTML file, which can be statically hosted. Additionally, a *progressive web app (PWA)* is provided using the “vue-pwa” feature, allowing for a cached application and app-like behavior on supported systems.

The Frontend is optimized to work with different devices and screens, reordering the layout in responsive ways. It allows its usage on phones, tablets, and screens of different resolutions and sizes. Other meta tags generate a pretty preview card with an image on sharing the URL on Twitter. The individual pages are loaded in junks to improve the initial loading speed of the site. The JS framework heavily influences the speed with its big initial load and runtime at the start, further slowed down by all the charts. However, once the JS is loaded or cached in the case of PWA, the website works very fast, and even in regular browsers, the service worker running the PWA will be caching the site limiting speed by the API.

On top of the page, the user can choose for which duration the data should be displayed. The duration is added as a query to the URL to keep it persistent even when sharing the URL. Although not all data is used for querying data, most data depends on and will change. The Web Page is using special meta data tags to be used by Twitter to allow a better presentation if the URL is shared. This can be seen in Figure 4.8.

Figure 4.8: Screenshot of the Twitter Link Preview

[5] https://www.primefaces.org/primevue/; accessed 10-November-2021

Pages

“Home” Page

The homepage shows a quick overview of the top trending publications, authors, and fields of study on the left. The newest processed tweet is in the middle, displaying extracted data and adding the metadata for the publication referenced. This can be seen in Figure 4.9. On the right, the total stats for the currently selected duration is shown. The next row shows the world cloud and world map using data from all processed tweets. The following two charts display the Twitter activity and the trending changes over the selected time. At the bottom of the Page, top types, languages, hashtags, and entities are displayed.

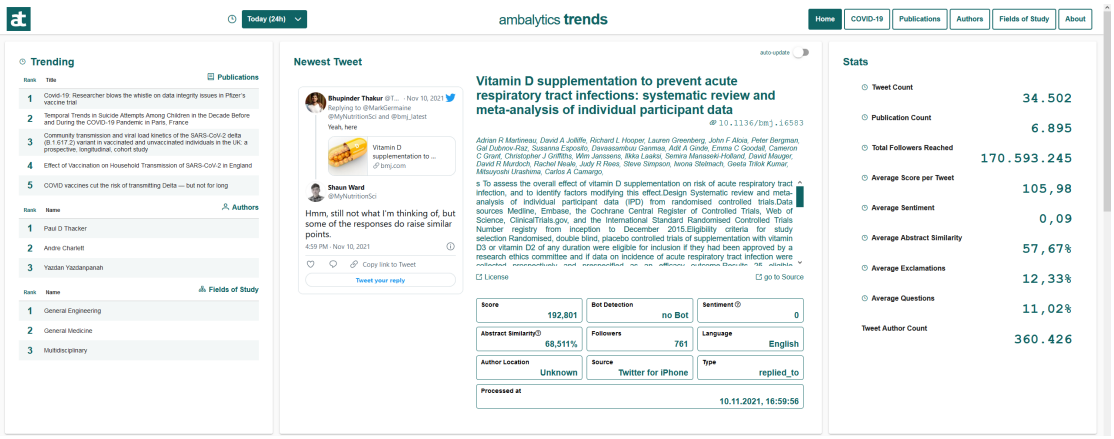

Figure 4.9: Screenshot of the Home Page

“COVID-19” Page

The COVID-19 page shows only trending publications about COVID-19. A chart displays the changes in trending data, updating automatically. A Table showing further information is located beneath. This looks similar to the Publications page that can be seen in Figure 4.10 Changes in the table will trigger changes in the chart, thus ensuring the publications in the chart and table are the same. The search can be used to filter publications based on their title.

“Publications” Page

Trending publications show all trending publications together with a chart showing the publication’s trend values over the duration. The Table and chart work like on the COVID-19 page. A Screenshot can be seen in Figure 4.10.

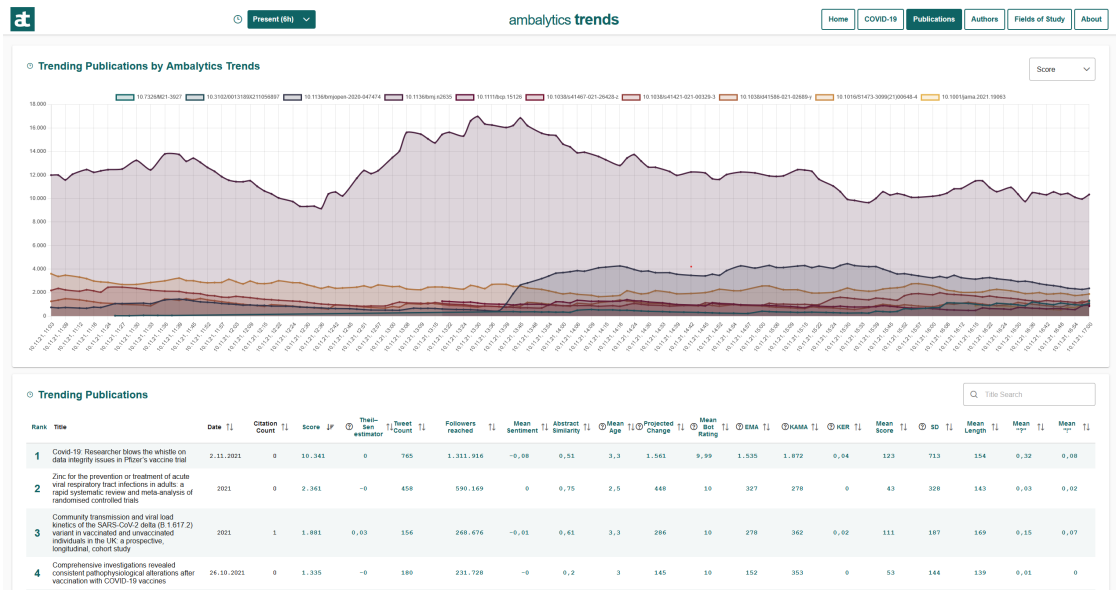

Figure 4.10: Screenshot of the Publications Page

“Publication” Page

The publication site is accessed by its DOI in the URL. If available and a public license is recognized, the abstract is shown. Otherwise, only metadata is displayed. Additionally, the sources are shown and linked. Only sources used to retrieve the publication metadata are displayed. Authors and fields of study are also displayed. A border is used around the box to highlight the publication information. An external link icon in the top right of the box allows a user fast access to the publication source. A screenshot can be seen in Figure 4.11. A profile chart shows how the publication ranks in different categories against other publications. A stats container displays actual numbers collected in the system. World map and Wordcloud are shown filtered based on the publication. Tweet and Twitter activity are displayed before tweet type, language, hashtags and top entity charts.

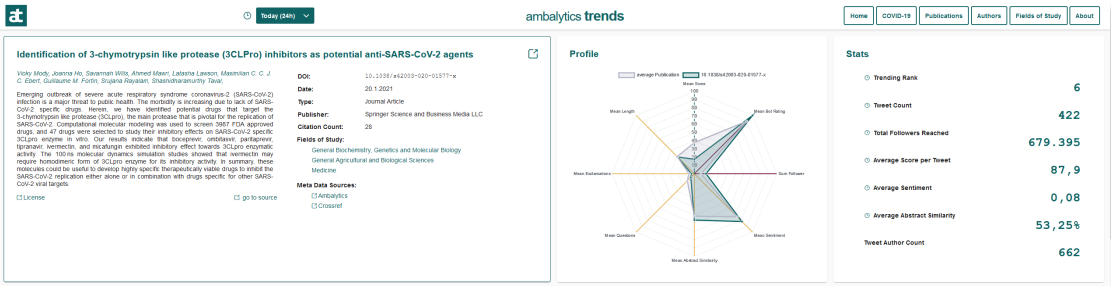

Figure 4.11: Screenshot of a Publication Page

“Fields of Study” Page

The Fields of Study Page shows a table with currently trending fields of study and their respective trend data. This can be seen in Figure 4.12

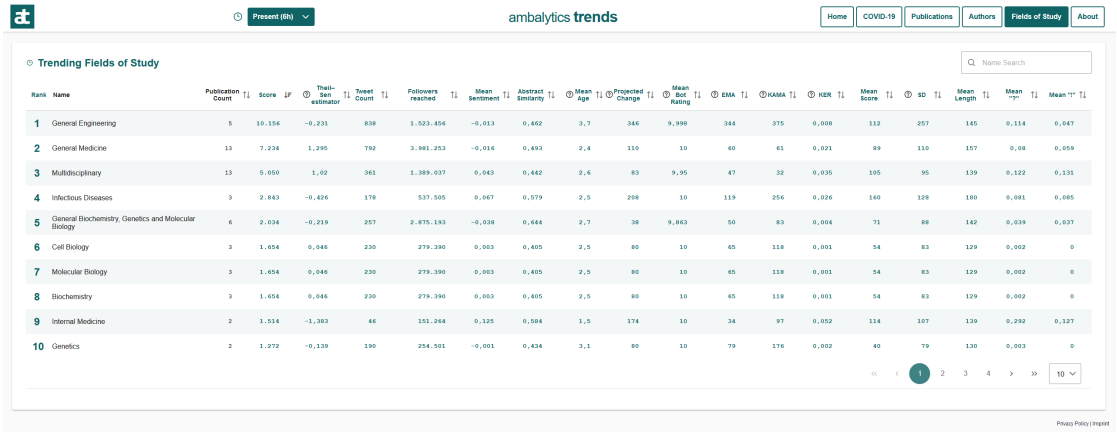

Figure 4.12: Screenshot of the Fields of Study Page

“Field of Study” Page

The Field of Study Page shows an overview of a field of study. On top, a table with trending publications in this field of study is shown. This can be seen in Figure 4.12. Beneath the publications, trend history is displayed next to a profile chart and a stat container. World map and Wordcloud follow before the newest Tweet and Twitter activity is shown.

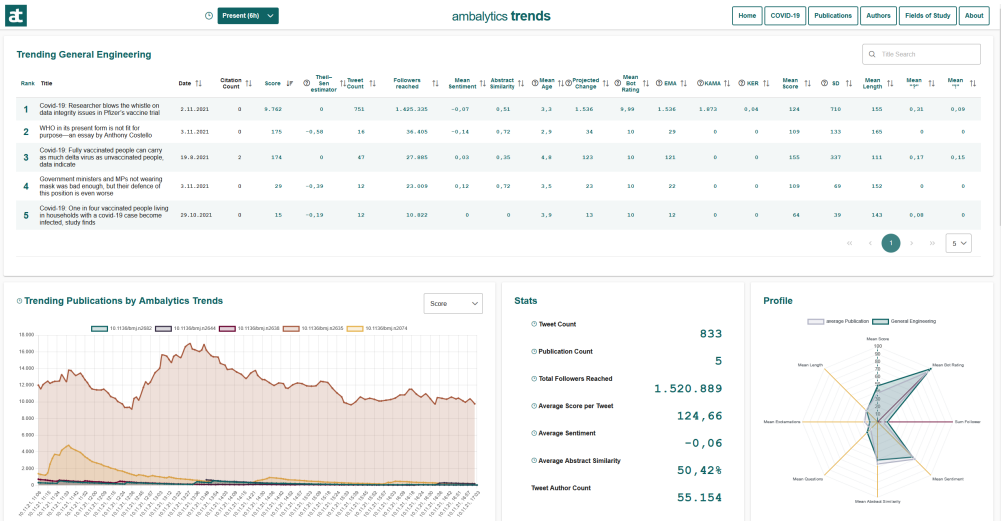

Figure 4.13: Screenshot of a Field of Study Page

"Authors" Page

The Authors Page displays a table with currently trending authors and their respective trend data. This is similar to the Fields of Study Page shown in Figure 4.12.

"Author" Page

The Author Page shows an overview of a specific author, similiar to the Field of Study Page shown in Figure 4.13.

"About" Page

The About Page shows an explanation of how the system works, links to Publication Sources, Source Code, Ambalytics and an Imprint. This can be seen in Figure 4.14.

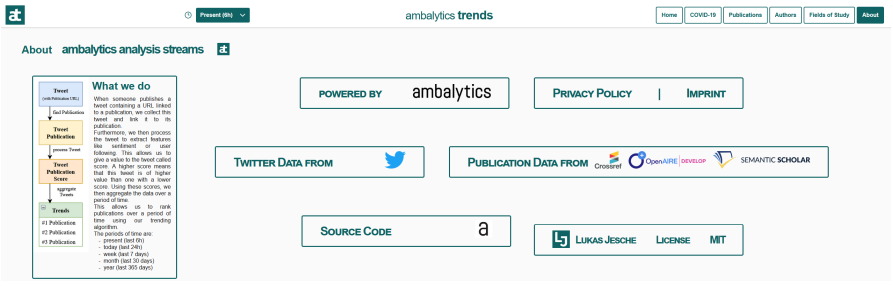

Figure 4.14: Screenshot of the About Page

Visual Components

Word Cloud

The word cloud component displays the top words used in all tweets collected. The size of the word corresponds to its count. An Example from the Homepage can be seen in Figure 4.15. The bigger the word, the bigger the count. The Font Size mapper scales the words to maximize a difference in sizes while still being readable linearly. With a click on a word, a corresponding Twitter page is opened. This may be the search or the profile page of a Twitter user. This data for this component is only saved as total data accumulated and will not change with the duration selection. The map is available for all publications, single publications, authors, or fields of study.

Figure 4.15: Word Cloud Component

Map

The Map Component displays the number of Tweet Authors displayed on a world map. A logarithmic color scale is used to visualize differences. An Example from the Homepage can be seen in Figure 4.16.

The scale allows apparent differences compared to a linear color scale. This data for this component is only saved as total data accumulated and will not change with the duration selection. The map is available for all publications, single publications, authors, or fields of study.

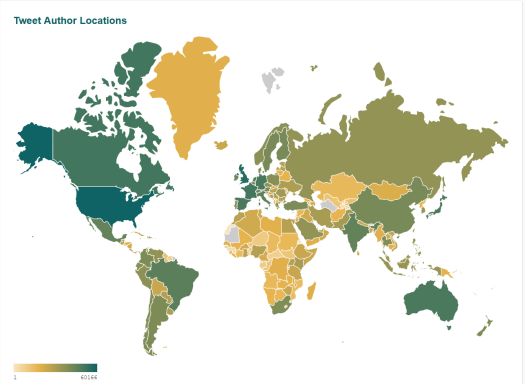

Figure 4.16: Map Component

Line Charts

There are two kinds of line charts displayed. One shows the event data over

some time, while the other shows calculated trend data.. An Example from the Homepage can be seen in Figure 4.17

The top trending publications will be shown on the home page, while on the COVID-19 Page, Field of Study pages, and Author pages, filtered publications are shown. By default, the score is selected, but the user can choose between all stored metrics to view and compare the publications over the selected duration. By clicking on a Publication in the legend, it will be highlighted, and a Link with the title is shown, which can be used to open the corresponding publication page.

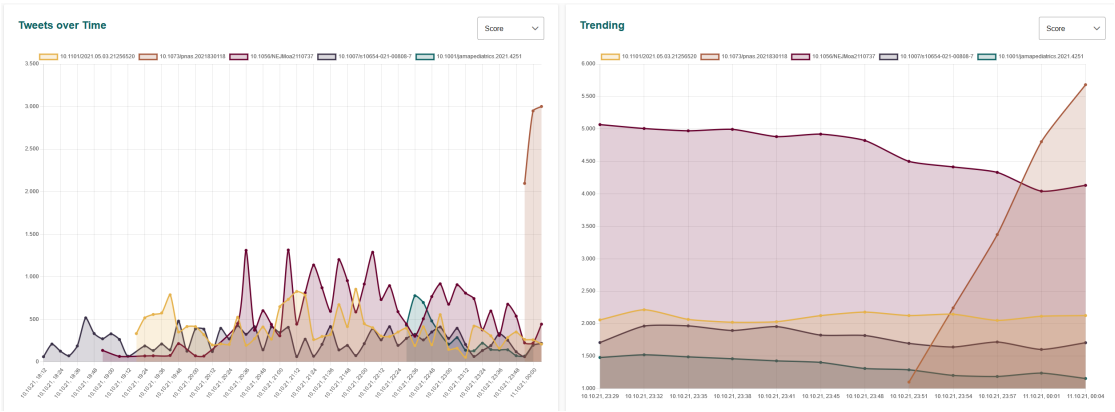

Figure 4.17: Line Charts Component

Donut Charts

Twitter-specific data shows the top values of Languages, Hashtags, Tweet Types, and Twitter Entities as Donut charts. An example can be seen in Figure 4.18.

Stats

The stats component shows some interesting numbers for the current page. It contains numbers only calculated for the current duration like tweet count, publication count, total followers reached, average score per tweet, average sentiment, or average abstract similarity.

Ambalytics Tweet

The Tweet component shows the newest tweet next to the linked publication details and

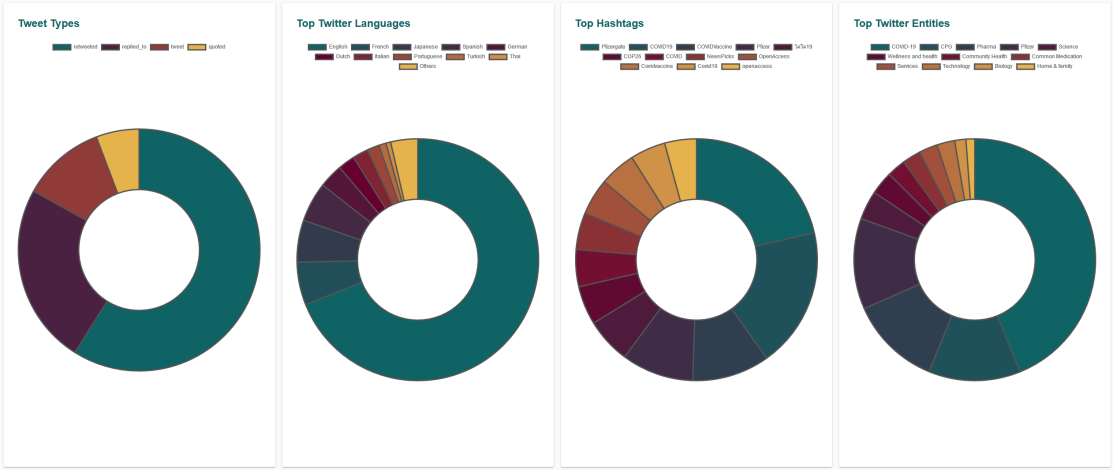

Figure 4.18: Donut Charts visualizing Twitter-specific Data

some processed data. It will ask before loading external twitter content. Additionally, it can be set to update the tweet automatically. Shown are the score, the Bot Rating, Sentiment, Abstract Similarity, Followers, Language, Author Location, Source, Type, Entities, and time of processing.

Profile Chart

The Profile component displays a series of stats and compares it on a scale from 0-100 to the average trending publication in the selected duration. The component shows the mean score, Mean Bot Rating, Sum Follower, Mean Sentiment, Mean Abstract Difference, Mean Questions, Mean Exclamations, and Mean Length.

5

Evaluation

This chapter presents various evaluations that show, for example, how well the linking of tweets with publications works and the performance of the trend metrics used (e.g., KAMA).

5.1 DOI Ratio (Percolator)

The Percolator can add a DOI to 94% of all collected events. This number is retrieved by analyzing the event data over a period of 12 hours and a total of 14155 events. Of these, 13311 could be linked to a DOI. 844 events could not be linked. The reason for these unlinked events is twofold. First, not every URL on the publisher domains is used to retrieve events links to a publication registered with a DOI. Since the platform registers only publications with a DOI, these tweets are not used for processing. Second, failures to resolve the DOI from the URL may occur. Since all publishers are tested and use a confirmed method covered by the Percolator, this will only occur if the requests result in errors. Figure 5.1 displays the ratio between events linked (green) and events where no DOI is found (orange). Table 5.1 displays the absolute event count as well as the relative percentage.

Further, the ratio between events that are linked directly is evaluated, i.e., the ratio between a known publication versus unknown events which need publication data retrieval. In the analyzed data, over a period of 12 hours, a total of 13311 events have been counted, of which 8065 or 61% could be linked to an existing publication (see Table 5.2). 39% or 5246 events needed to be sent to the Pubfinder with an unknown

	# Events	Percentage
Linked Events	13311	94%
Missing Publication	844	6%
Total Events	14155	100%

Table 5.1: Missing DOI Ratio (12 hours)

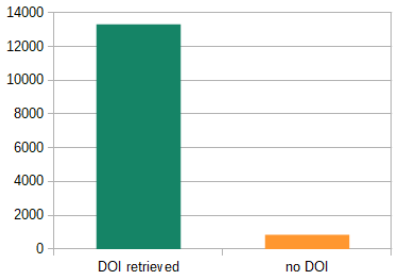

Figure 5.1: Missing DOI Ratio

state. Figure 5.2 shows a chart visualizing and comparing the data. The evaluated percentage ratio is variable and would rapidly decrease if the Pubfinder component were not included as a processing step. Following this circumstance shows the importance of retrieving needed publication data as soon as possible.

	# Events	Percentage
Linked Events	8065	61%
Unknown Events	5246	39%
Total Events	13311	100%

Table 5.2: Unknown DOI Ratio (12 hours)

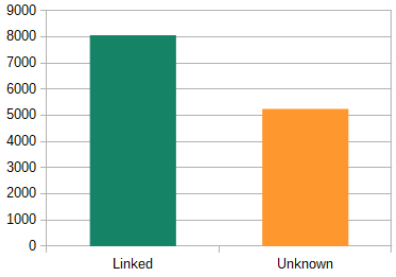

Figure 5.2: Unknown DOI Ratio

5.2 Information Retrieval Ratio (Pubfinder)

The Pubfinder component can find data of 90% of publications that are linked or discussed in tweets. For 69% of publications, license information can be retrieved. The total numbers are displayed in table 5.3. For this evaluation, a total of 10012 DOI's have been retrieved by the Pubfinder over 24 hours. After that, the database was analyzed. A Chart visualizing and comparing the evaluation data can be found in Figure 5.3. Note that publication data may not be found at all by the Pubfinder, or just not enough relevant data to be considered. One reason is that a publication is too new or just not registered with any of our API services.

	# DOIs	Percentage
Publication data found	9019	90%
Missing Publication data	993	10%
Total DOIs	10012	100%

Table 5.3: Missing Publication Information Ratio (24 hours)

Figure 5.3: Missing Publication Information Ratio

Further, the retrieval success rate of publication license information was evaluated. Therefore, data was accumulated over 24 hours, containing a total of 8026 retrieved publications. As a result, the system can find license information for 86% of publications (see Table 5.4). A Chart is shown in Figure 5.4 comparing and visualizing missing versus found licenses.

	# Publications	Percentage
License found	6905	86%
Missing License	1121	14%
Total Publications	8026	100%

Table 5.4: Missing License Information Ratio (24 hours)

Figure 5.4: Missing License Information Ratio

5.3 Total Yield of Tweets

The System can add publication data needed for processing in 77% of events. This number is retrieved by analyzing the event data over a period of 6 hours and a total of 9476 tweets. Of these, 7296 could be linked to a publication with a DOI, and publication data could be retrieved. 2180 events could not be linked, or no publication data was available. Figure 5.5 illustrates the ratio between events with publication data (green) and events where no DOI or publication data is found (orange). Table 5.5 contains the absolute event count as well as the relative percentage. This percentage is subject to

change, however, since a popular linked publication being widely discussed is increasing that percentage, a discussion about a non-DOI article would decrease it.

	# Events	Percentage
Linked Events	7296	77%
Missing Publication	2180	23%
Total Events	9476	100%

Table 5.5: Yield of Tweets (6 hours)

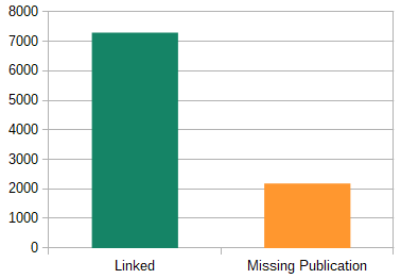

Figure 5.5: Yield of Tweets

5.4 Language Processing

Language Detection is done by Twitter and supplied in their tweet data stream. In 94% of tweets, a language could be identified by Twitter using data aggregated over days with a total of 417827 scored tweets. Of these, 86% of tweets can be processed by the language processing component. Since not all Tweets contain actual text, some may only be a URL, numbers of less than 100% are expected. The results are detailed in Table 5.6 and displayed in Figure 5.6.

	# Events	Percentage
Processable	360392	86%
Unprocessable	57435	14%
Language known	394545	94%
Unkown	23282	6%
Total Events	417827	100%

Table 5.6: Language Processing Effectiveness (several days)

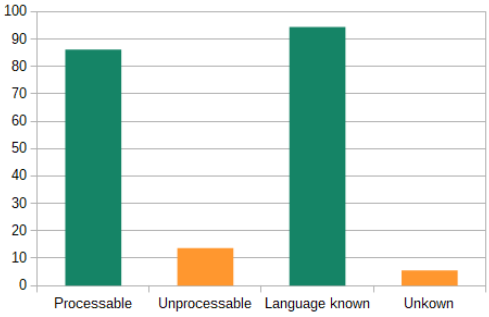

Figure 5.6: Language Processing Effectiveness

5.5 Comparison of Trend Value Results

5.5.1 Comparison of Score, KAMA, and EMA

An example showing data based on one publication is selected to compare trending value results. This selection allows better comparison since it is based on the same data compared to just using the mean of all publications. The selected publication is the most discussed publication, ensuring that the best case for the algorithms is data availability.

Following the findings, EMA and KAMA are generally following the same trend (see Figure 5.7; KAMA, EMA are linearly scaled by factor 5 to allow a better visual comparison). The EMA is the most stable of the displayed metrics, reacting more slowly to changes but delivering a stable line over time. The KAMA, instead, is very unstable and changes irregularly, reflecting the market adjusting behavior. It is, however, the fastest in reflecting changes of the three displayed. By adjusting variables used to calculate the KAMA, it may produce smoother results. Importantly, this data shown is collected over a period of days while the smoothing data set is limited to a few hours. This results in the impression of a not very smooth KAMA even tough, relative to its defined duration it produces smooth results.

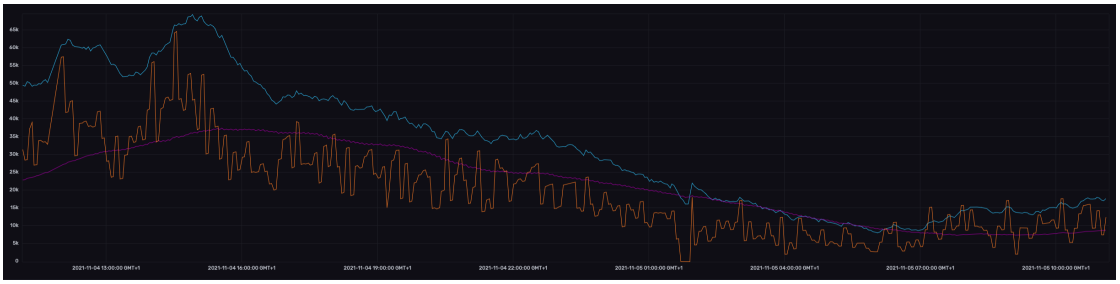

Figure 5.7: Comparison of Trending Score (blue), EMA(red), and KAMA(orange)

5.5.2 Comparison of Score and KER

As displayed in Figure 5.8, the KER value has a delay compared to the score calculation. Note that in the Figure, KER is linearly scaled by factor $2 * 10^5$ to allow a better visual comparison. As an explanation, the score calculation uses an exponential weight depending on how recent the events are. In contrast, the KER uses no such weights. However, it shows that introducing an applicable threshold to extract the different phases is possible and can further improve trend recognition.

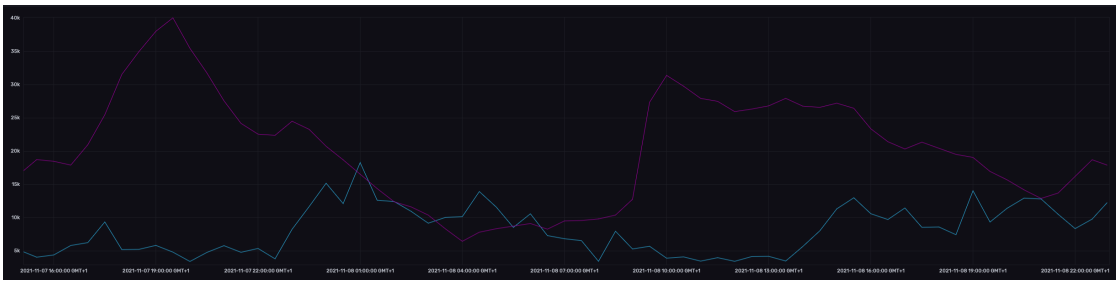

Figure 5.8: Comparison of Trending Score (blue) and KER(red)

5.5.3 Comparison of Score and Projected Score

The projected change and score are following similar curves (see Figure 5.9; projected change values are linearly scaled by factor 5 to allow a better visual comparison). However, it can be seen how a threshold for trending evaluation would work clearly indication phases.

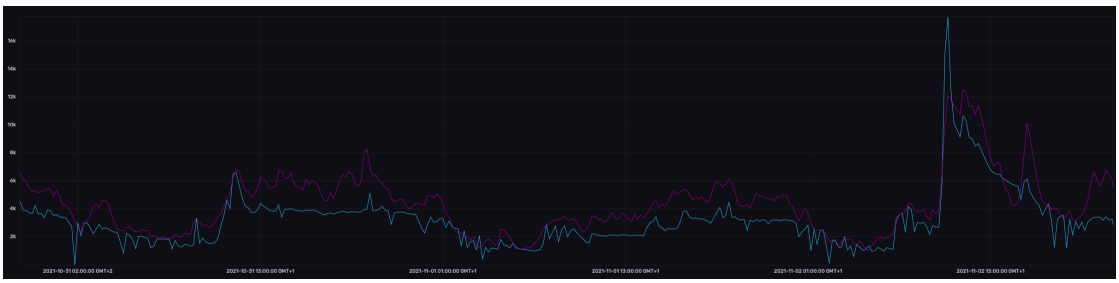

Figure 5.9: Comparison of Trending Score (blue) and Projected Change (red)

5.5.4 Comparison of Score and Theil-Sen Slope Estimator

A Score value increase is corresponding to positive trending values (see Figure 5.10; Theil-Sen values are linearly scaled by factor 10^7 to allow a better visual comparison). In this comparison, more important than the absolute value is the sign of the Theil-Sen Slope Estimator. The absolute value can be seen changing much more as the total data is reducing at the end of the selected time period.

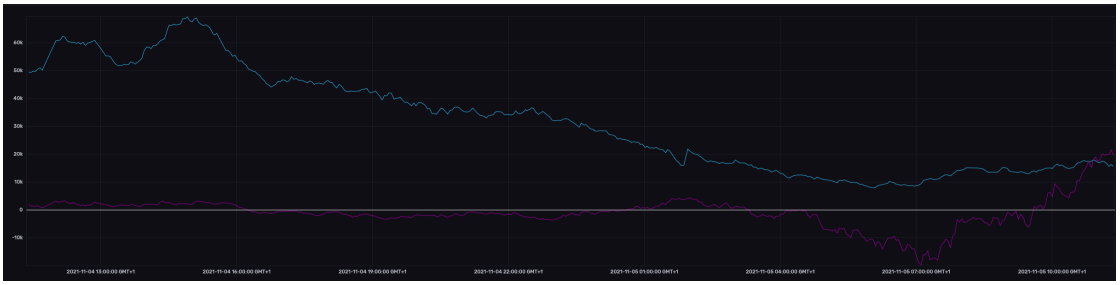

Figure 5.10: Comparison of Trending Score (blue), Theil-sen Slope Estimator (red)

5.5.5 Comparison of Score and Mean Age

The mean age impacts the score calculation and follows a reversed trend compared to the score (see Figure 5.11; mean age is linearly scaled by factor 4 to allow a better visual comparison). However, please note that the mean age decreases linearly while the score is adjusting exponentially.

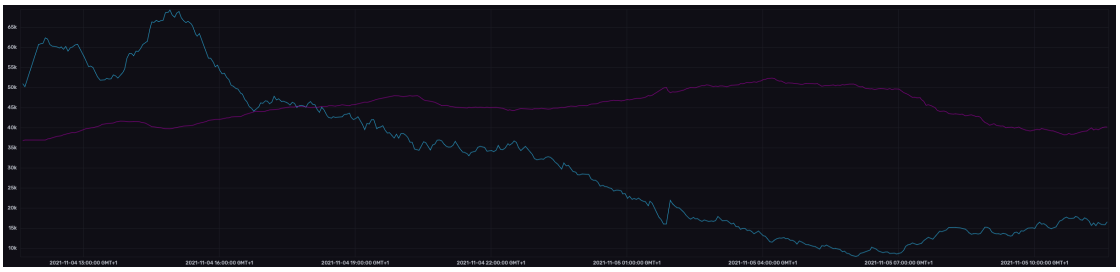

Figure 5.11: Comparison of Trending Score (blue) and Mean Age in Seconds (red)

5.5.6 Comparison of Score and Count

The count is impacting the total score. However, the apparent differences reflect the effect of using a score (see Figure 5.12; tweet count values are linearly scaled by factor 10 to allow a better visual comparison). Further, the exponential weights reducing the impact of old events can be observed.

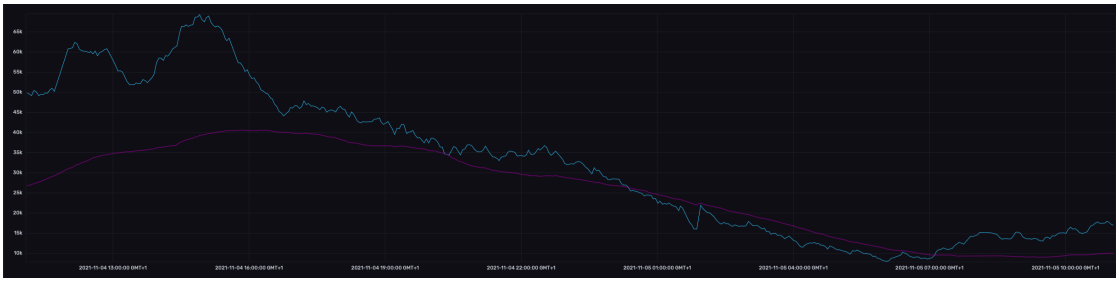

Figure 5.12: Comparison of Trending Score (blue), Tweet Count (red)

6

Conclusion

Scientific publications are published in large numbers and at short intervals, especially in research fields that are in focus, such as COVID-19. It is increasingly difficult for people working in science and those interested in science to keep track of publications.

In this thesis, a system was developed that analyzes data from Twitter and uses insights about scientific publications shared and discussed on Twitter to derive statements about the importance of scientific publications. The developed system can recognize trends and show relevant publications over a period of time. It can be used to identify potentially interesting publications, even just a day after publication. The system can retrieve the majority of events and process them within seconds.

The developed system meets all the requirements surveyed and can, for example, be expanded and scaled without significant effort. Processed Twitter and publication data are presented attractively in a web-based application. The system is able to link, process and aggregate hundreds of thousands of events referencing tens of thousands of publications per day. A final evaluation of various aspects, such as the data provided by external APIs and the performance of the implemented text analysis methods, showed promising results.

However, the developed approach and proof-of-concept implementation can be further improved. Due to the downsampling process running less often for bigger durations, there is a lag in data, meaning data is not considered for a period of time. This issue can be addressed by running the tasks more often; however, it would require more processing. Further, the time-series database system may become slow if accessed while calculating trends or running other heavy tasks. Similarly, the PostgreSQL database not always returns data fast enough, i.e., slowing down the user experience and performance scores.

Possible workarounds are running both databases in more fitting systems, adding a cache on webserver level, and using materialized tables to ensure the API and the Frontend stay fast–even if multiple users visit the site while trends calculations are running.

In order to further enhance publication trend detection, the scoring of the tweets itself could be improved, e.g., additional features could be extracted and ranked. The Trend Calculation respectively the ranking algorithms could be improved by further analyzing the evaluation results. Finally, the system is developed generically and allows for covering more research data sources, such as additional social media platforms.

Bibliography

[1] Hendricks, G., Tkaczyk, D., Lin, J., Feeney, P.: Crossref: The Sustainable Source of Community-owned Scholarly Metadata. Quantitative Science Studies **1** (2020) 414–427

[2] PlumX: PlumX Website. `https://plumanalytics.com/learn/about-metrics/` (2021) [Online; accessed 7-November-2021].

[3] Altmetric: Altmetric Website. `https://www.altmetric.com` (2021) [Online; accessed 7-November-2021].

[4] Hicks, D., Wouters, P., Waltman, L., de Rijcke, S., Rafols, I., Langhanke, G.: Bibliometrie: Das Leidener Manifest zu Forschungsmetriken. Nature (2015) 429–431

[5] Thelwall, M.: Bibliometrics to Webometrics. Journal of Information Science **34** (2008) 605–621

[6] Hirsch, J.E.: An Index to Quantify an Individual's Scientific Research Output. Proceedings of the National academy of Sciences **102** (2005) 16569–16572

[7] Haustein, S., Bowman, T.D., Holmberg, K., Peters, I., Larivière, V.: Astrophysicists on Twitter: An in-depth analysis of tweeting and scientific publication behavior. Aslib Journal of Information Management (2014)

[8] Darling, E.S., Shiffman, D., Côté, I.M., Drew, J.A.: The role of Twitter in the life cycle of a scientific publication. Technical report, PeerJ PrePrints (2013)

[9] Shu, F., Lou, W., Haustein, S.: Can Twitter increase the Visibility of Chinese Publications? Scientometrics **116** (2018) 505–519

[10] Priem, J., Costello, K.L.: How and Why Scholars Cite on Twitter. Proceedings of the American Society for Information Science and Technology **47** (2010) 1–4

[11] Cardona-Grau, D., Sorokin, I., Leinwand, G., Welliver, C.: Introducing the Twitter Impact Factor: An Objective Measure of Urology's Academic Impact on Twitter. European urology focus **2** (2016) 412–417

[12] Bornmann, L., Haunschild, R.: How to Normalize Twitter Counts? A First Attempt based on Journals in the Twitter Index. Scientometrics **107** (2016) 1405–1422

[13] Fang, Z., Dudek, J., Costas, R.: The Stability of Twitter Metrics: A Study on Unavailable Twitter Mentions of Scientific Publications. Journal of the Association for Information Science and Technology **71** (2020) 1455–1469

[14] Montgomery, D.C., Johnson, L.A., Gardiner, J.S.: Forecasting and Time Series Analysis. McGraw-Hill (1990)

[15] Klinker, F.: Exponential Moving Average versus Moving Exponential Average. Mathematische Semesterberichte **58** (2011) 97–107

[16] Kaufman, P.J.: Trading Systems and Methods. Volume 591. Wiley (2013)

[17] Paesler, O.: Technische Indikatoren - simplified. FinanzBuch (2006)

[18] influxdata: kaufmansER() function. `https://docs.influxdata.com/flux/v0.x/stdlib/universe/kaufmanser/` (2021) [Online; accessed 7-November-2021].

[19] Streiner, D.L.: Maintaining Standards: Differences between the Standard Deviation and Standard Error, and when to Use Each. The Canadian Journal of Psychiatry **41** (1996) 498–502

[20] Gelper, S., Fried, R., Croux, C.: Robust Forecasting with Exponential and Holt–Winters Smoothing. Journal of Forecasting **29** (2010) 285–300

[21] Theil, H.: A Rank-invariant Method of Linear and Polynomial Regression Analysis. Indagationes mathematicae **12** (1950) 173

[22] Sen, P.K.: Estimates of the Regression Coefficient based on Kendall's Tau. Journal of the American Statistical Association **63** (1968) 1379–1389

[23] Peng, H., Wang, S., Wang, X.: Consistency and Asymptotic Distribution of the Theil–Sen Estimator. Journal of Statistical Planning and Inference **138** (2008) 1836–1850

[24] Salihefendic, A.: How Hacker News Ranking Algorithm Works. https://medium.com/hacking-and-gonzo/how-hacker-news-ranking-algorithm-works-1d9b0cf2c08d (2015) [Online; accessed 7-November-2021].

[25] CrossRef: CrossRef Event Data. https://www.crossref.org/services/event-data/ (2021) [Online; accessed 7-November-2021].

[26] Eysenbach, G.: Can Tweets Predict Citations? Metrics of Social Impact based on Twitter and Correlation with Traditional Metrics of Scientific Impact. Journal of Medical Internet Research **13** (2011) e123

[27] Kammerer, K., Reichert, M., Pryss, R.: Ambalytics: A Scalable and Distributed System Architecture Concept for Bibliometric Network Analyses. Future Internet **13** (2021) 203

[28] Le Noac'h, P., Costan, A., Bougé, L.: A Performance Evaluation of Apache Kafka in Support of Big Data Streaming Applications. In: IEEE International Conference on Big Data (Big Data). (2017) 4803–4806

[29] CrossRef: CrossRef Event Data Percolator. https://github.com/CrossRef/event-data-percolator (2021) [Online; accessed 7-November-2021].

[30] Rettberg, N., Schmidt, B.: OpenAIRE–Supporting a European Open Access Mandate. College & Research Libraries News **76** (2015) 306–310

[31] Ammar, W., Groeneveld, D., Bhagavatula, C., Beltagy, I., Crawford, M., Downey, D., Dunkelberger, J., Elgohary, A., Feldman, S., Ha, V., et al.: Construction of the Literature Graph in Semantic Scholar. arXiv preprint arXiv:1805.02262 (2018)

[32] Srinivasa-Desikan, B.: Natural Language Processing and Computational Linguistics: A practical guide to text analysis with Python, Gensim, spaCy, and Keras. Packt (2018)

[33] Hussain, M.M., Mahmud, I.: pyMannKendall: A Python Package for Non Parametric Mann Kendall Family of Trend Tests. Journal of Open Source Software **4** (2019) 1556

[34] Kalekar, P.S., et al.: Time Series Forecasting Using Holt-Winters Exponential Smoothing. Technical Report 13, Kanwal Rekhi School of Information Technology (2004)

A

Tables

Domains	Scientific Category
doi.org	General
dx.doi.org	General
emerald.com/insight/content/	General
sciencedirect.com/science/article	Natural sciences
degruyter.com/document/doi/	Natural sciences
link.springer.com/article/	Natural sciences
onlinelibrary.wiley.com	Natural sciences
nature.com/articles	Natural sciences
sciencemag.org	Natural sciences
journals.sagepub.com	Natural sciences
journals.plos.org	Natural sciences
frontiersin.org/journals	Natural sciences
tandfonline.com/doi/	Natural sciences
mdpi.com/journal	Natural sciences
iop.org	Natural sciences
cochranelibrary.com/cdsr	Medicine
nejm.org/doi	Medicine
thelancet.com/journals	Medicine
bmj.com/content	Medicine
pnas.org/content	Medicine
jamanetwork.com/journals	Medicine
acpjournals.org/doi	Medicine
n.neurology.org/content	Medicine
doi.apa.org/record	Psychology
ieeexplore.ieee.org/document	Computer Science
dl.acm.org/doi	Computer Science
jmir.org	Computer Science
journals.aps.org	Physics
biorxiv.org/content	Repositories
arxiv.org/	Repositories
academic.oup.com	Repositories
jmcc-online.com/article	Repositories
journals.elsevier.com	General

Table A.1: Tweet Selector URLs and Domains

List of Figures

List of Tables

Name: Lukas Jesche Student ID: 964097

Declaration

I hereby declare that I have developed and written the enclosed Bachelor Thesis by myself and have not used sources or means without declaration in the text. This Bachelor Thesis has never been used in the same or in a similar version to achieve an academic grading or was published elsewhere.

Ulm, 10.11.2021 ..... Lukas Jesche .....

Lukas Jesche