The Journal of Supercomputing (2023) 79:16664–16687
https://doi.org/10.1007/s11227-023-05312-1

An online and highly-scalable streaming platform for filtering trolls with transfer learning

Chun-Ming Lai[1] · Ting-Wei Chang[1] · Chao-Tung Yang[1,2]

Accepted: 14 April 2023 / Published online: 29 April 2023

Abstract
The internet has reached a mature stage of development, and Online Social Media (OSM) platforms such as Twitter and Facebook have become vital channels for public communication and discussion on matters of public interest. However, these platforms are often plagued by improper statements or content, propagated by anonymous users and trolls, which negatively impact both the platforms and their users. Existing methods for dealing with inappropriate information rely on (semi)-manual offline assessments, which do not fully account for the streaming nature of OSM feeds. In this paper, we implement a robust and decoupled system that considers social media data as streaming data. With a publisher and consumer model, our system can process more than 179 MB of data per second with only 166.3 ms latency using Apache Kafka. Accordingly, we deploy a well-trained transfer learning model to classify incoming data streams, with an accuracy of 0.836. Our proposed architecture has the potential to assist online communities in developing more constructive and flawless OSM platforms. We believe that our contribution will help address the challenges associated with improper content on OSM platforms and pave the way for the development of more effective and efficient solutions.

Keywords Trolls · Online social media · Apache Kafka · Kubernetes · Transfer learning

✉ Chao-Tung Yang
ctyang@thu.edu.tw

Chun-Ming Lai
cmlai@thu.edu.tw

Ting-Wei Chang
twwwww1015@gmail.com

[1] Department of Computer Science, Tunghai University, No. 1727, Sec.4, Taiwan Blvd., Taichung 407224, Taiwan R.O.C.

[2] Research Center for Smart Sustainable Circular Economy, Tunghai University, No. 1727, Sec.4, Taiwan Blvd., Taichung 407224, Taiwan R.O.C.

1 Introduction

Contemporary societies place great emphasis on the value of freedom of speech. With the ease and ubiquity of the internet, individuals are able to share their thoughts and opinions on Online Social Media (OSM) platforms with little constraint. Despite the advantages of free speech, however, the absence of explicit limits has led to a proliferation of reckless and inappropriate online behaviors. In recent years, incidents of cyberbullying, hate speech, and other forms of abusive online conduct have become increasingly prevalent. Hate speech, in particular, refers to the use of violent and discriminatory language, negative statements, and targeted slurs aimed at defaming or discriminating against individuals on the basis of race, nationality, religious belief, sexual orientation, and other personal characteristics. The problem has become a critical issue which affects OSM users mental health and well-being [1].

The National Human Rights Commission reports that more than one in five children, entertainers, and internet celebrities have experienced online bullying, with 62.5% of them exhibiting symptoms of online social anxiety and 26% expressing suicidal thoughts [2]. Hinduja et al. [3] conducted a survey of a representative sample of U.S. youth aged 12 to 17, and the results showed that those who experienced school-based or online bullying were significantly more prone to report suicidal thoughts. Furthermore, students who reported being bullied both at school and online were even more likely to not only report suicidal thoughts but also to make attempts. For example, a 24-year-old Taiwanese actress Yang Youing (Cindy) committed suicide after suffering from prolonged cyberbullying. Cindy expressed in her suicide note the hope that her death would bring attention to the seriousness of netizens' bullying.[1] OSM companies rarely delete comments, as it may negatively impact their profit. However, it would be advantageous for individuals and organizations to have a scalable real-time filter to detect obvious trolls and hate speech.

Although OSM platforms such as Twitter have been updating and enforcing their violent threat policies to ensure that users can safely engage with each other, attackers are able to create multiple accounts to express offensive content and harass others easily on most OSM platforms. Due to the bandwagon effect [4] on the OSM, the mainstream could become an abettor if inappropriate content is not removed in real time. Fortunately, with the rapid development of Artificial Intelligence (AI) and Natural Language Processing (NLP), troll content removal can be regarded as a binary classification problem. Utilizing heterogeneous features, including content, temporal behavior, and account behavior collected through relatively massive data, many researchers have shown convincing results in negative content detection [5, 6]. However, most previous research was based on offline data training and testing, without considering that OSM platforms are essentially an online data streaming platforms with large amounts of data ingress.

[1] https://www.taipeitimes.com/News/taiwan/archives/2015/04/23/2003616602.

To address real-time troll negative comment detection, in this work, we utilize Kubernetes to instantly deploy the Kafka producer–consumer distributed system to achieve high throughput, low latency and high scalability. The producer collects data from the Twitter API and inputs them into the Kafka broker. Once the consumer is available, it pulls a topic from the broker and judges whether the pulled message is from a troll based on our pretrained transfer learning [7] model (Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM) [8] and Bidirectional Encoder Representations for Transformers (BERT) [9]). Our approach to detecting malicious behavior on OSMs integrates both content and deep learning, in contrast to other existing methods that rely on account-level features and traditional machine learning techniques like Decision Trees or SVMs [10, 11]. Our pretrained models are based on an open dataset published on Kaggle that includes 43136 training samples and 8587 test samples. For the 8587 test samples, our system exceeded 80% on all well-known metrics, such as accuracy, recall, precision and F1 score. Accordingly, we also compare several important hyperparameters (number of partitions, incoming data size per second, throughput and batch size) to evaluate how these attributes affect the performance of our proposed framework. Furthermore, we provide a comprehensive list of experimental details, allowing anyone to estimate the resources necessary to deploy a similar environment if they have a similar incoming streaming data size.

In contrast to other security issues on OSM platforms, such as fake news and disinformation, which still lack a clear definition, there is no gray area with respect to negative and offensive content toward specific targets, which can cause constant damage if not removed in real time. Essentially, we aim to construct a decentralized system for instant semantic analysis of trolls' offensive comments from Twitter posts. We use Kubernetes to quickly deploy and manage big data stream processing to achieve low-latency, high-throughput data transmission performance. To the best of our knowledge, our system is the first system dedicated to troll negative comment detection that considers OSM data as streamed ingress data. In addition, we apply transfer learning to avoid the collection of a large amount of labeled data. The main contributions of this paper are listed as follows:

1. We demonstrate the use of Docker, Kubernetes, Apache Kafka, and Zookeeper to construct a scalable and reliable online streaming system for the real-time filtering of trolls.
2. We evaluate the effects of different parameters on the performance in Kafka Cluster to tune the best standard parameters and improve the performance of the overall data flow.
3. We provide a detailed list of hardware and software configurations, enabling anyone to understand the system's performance and throughput and facilitating the implementation of a similar negative online content filtering system.
4. We exploit different state-of-the-art AI and NLP models in transfer learning for semantic recognition training [12], and use the training results of these models to compare the differences in the performance of different models according to the evaluation indicators.

The structure of this paper is as follows. Section 2 introduces relevant knowledge of high-performance computing components and AI tools. In Sect. 3, we describe the system architecture how the producer and consumer are combined and how they operate with respect to cluster management. The experimental results and further discussion are disclosed in Sect. 4. Finally, we present the conclusion in Sect. 5.

2 Background review and related work

This section briefly introduces the tools used in this paper, including Kubernetes, Apache Kafka, Zookeeper and neural networks in NLP. In addition, several related works will be summarized.

2.1 Kubernetes

Kubernetes is abbreviated as kube, and because there are eight letters between the first and last letters, it is also referred to as K8s. It is mainly divided into two parts, a master and workers. The master is mainly responsible for storage and management, while the workers are responsible for execution and monitoring. K8s is an open source system for automatically executing application deployment and microservice management. It can also be used to automate deployment, expansion and manage multiple Linux containers on multiple machines [13]. In this research, we set up a master node and three worker nodes, as suggested in [14]. The advantages of Kubernetes are as follows:

- It offers flexible deployment.
- It can be automatically restarted after a container fails.
- It provides a complete authentication mechanism to meet strict security concerns in the management of computing resources.
- It supports load balancing. When container traffic reaches its peak, K8s can distribute traffic on the network to maintain stability. In addition, self-scheduling is performed based on current resources.

2.2 Apache Kafka

Apache Kafka was originally developed by LinkedIn to solve the data pipeline problem. It is a popular event streaming platform developed in Scala and Java. The overall architecture consists of three roles: producer, broker and consumer. The producer is responsible for collecting information and sending it to the broker. The broker is responsible for receiving the information and persisting the information locally. The consumer digests the data pushed by the broker. Kafka is widely used to process a large amount of real-time data, such as message systems, storage systems, and stream processing [15], and it can easily handle tens of thousands of requests per second. Since the data are stored on a hard drive, they can be stored persistently.

Kafka has the following characteristics:

- It has high throughput and low latency. Kafka can achieve a throughput of millions of messages, and the latency can be as low as milliseconds, which can be used for real-time and batch data processing.
- The data are persisted and stored on a disk, providing offset reading and writing to maintain performance and improving data reliability through Kafka's replication mechanism to prevent data loss.
- It can store multiple types of data formats and supports data compression during data transmission in gzip, snappy, lz4 and other compression formats.
- Data are stored in different brokers in a decentralized manner, using the topic as the unit and partitioning for decentralized storage, improving the overall scalability.
- It has high fault tolerance. When any broker in a cluster goes down, Kafka can still rely on the remaining machines to provide external services [16, 17]. Kafka has the capability to support thousands of clients simultaneously reading and writing data.

2.3 Zookeeper

Zookeeper is an open source distributed application coordination service, an important component of Apache Hadoop and Apache HBase, providing configuration maintenance, naming, distributed architecture synchronization, cluster management and other services to solve data management problems.

Zookeeper is used to serve a distributed system [18], and it is also a distributed system. A Zookeeper cluster requires at least 3 machines because a leader will be chosen from the cluster, and voting for a leader requires the approval of a majority of members in a strict sense. As long as more than half of the nodes survive, Zookeeper can serve normally, meaning that the number of Zookeeper clusters is generally an odd number, but this is not a mandatory requirement. When there are more machines, the better the loading performance of Zookeeper is, the more clients can be served at the same time. However, the insertion performance will decrease because Zookeeper must ensure that all available services are successfully written to guarantee strict consistency. Therefore, the number of machines cannot be increased blindly.

2.4 Transfer learning and BERT

Word and content embedding is currently a common starting point when performing NLP tasks after necessary preprocessing. On the other hand, transfer learning is a method used in machine learning to transfer knowledge from an original domain to a new domain so that the domain can achieve good learning outcomes. According to the techniques used in transfer learning methods, transfer learning methods can be roughly divided as follows: (1) Transfer learning based on feature selection. (2) Transfer learning based on feature mapping. (3) Weight-based transfer learning. Here, we introduce two required steps in NLP to conduct the experiment.

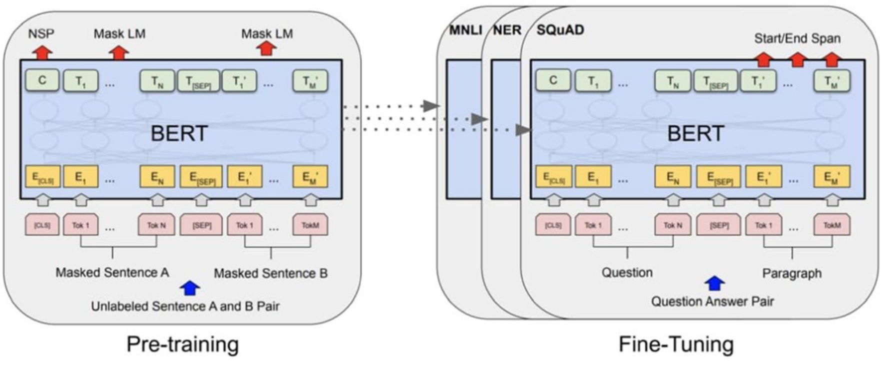


Fig. 1 BERT training method

BERT is an encoder of a two-way transformer. The BERT model was launched by the Google AI Research Institute in 2018 and is often used in semantic sentiment analysis [19]. It is composed of 2.5 billion words from Wikipedia and 800 million from BookCorpus. BERT is a model trained as input from a large amount of text data without labels and supports more than 70 languages, such as English, Chinese, Korean, Portuguese, and Vietnamese. It is a model for NLP pretraining technology, and it already has many relevant features [20] and can be directly applied to tasks in other languages.

The training method of BERT is mainly divided into two steps: pretraining and fine-tuning (as shown in Fig. 1). The following two methods will be introduced:

- Pretraining: Pretraining consists of two steps, "Cloze" training, and "Next Sentence Prediction" training, so that the model can learn to predict the correct sentence.
- Fine-tuning: During fine-tuning, most neural network architectures of the pretrained model are frozen, and the remaining few neural network architectures are trained.

2.5 Related works

With the increasing attention given to security concerns on OSM platforms, there are two main directions, content and accounts. For example, Lai et al. [12] and Saha et al. [21] demonstrate that content level features can be successfully applied in detecting fake news based on an offline Twitter dataset. They also demonstrated that several linguistic and semantic features, such as TF-IDF, behave differently between normal news and fake news. In terms of hate speech detection, Davidson et al. [22] showed that multilingual keywords, part-of-speech tags, the number of sentences, sentiment scores and numbers of mentions and retweets could be effective features for building hate speech classifiers. Although there are disagreements on how hate speech should be defined, most recent studies have relied on well-acknowledged hate speech datasets, for example, Hatebase [22], WaseemA [23] and Stormfront [24].

On the other hand, the essence of online social networks relies on the interaction of accounts. Accordingly, graphs are a common data structure for OSM research issues. Cresci et al. [25] sorted recent studies on trolling content from social bot detection by bipartite graphs, adjacency matrices or peculiar patterns in spectral or temporal behavior. Qian et al. [26] utilized users' historical tweets and tweets that are semantically similar to those labeled as hate speech as features to build an account-based hate speech classifier. Eiman Alothali et al. [27] proposed a live-streaming system for social bot intrusion detection. The system uses the Twitter API to collect tweets, extracts the required features and uses Apache Kafka to stream the data. Dwi Fimoza et al. [28] analyzed positive, negative and neutral comments for YouTube comment sentiments. They automatically classified the comments according to their sentiment polarity and used BERT for sentiment analysis. Ksieniewicz et al. [29] proposed a novel pattern classification system based on feature extraction techniques, addressing the detection of fake news in streaming data.. Roy et al. [30] proposed an automated system using a deep convolutional neural network (DCNN). The proposed DCNN model uses GloVe embedding vectors to capture the semantics of tweet-based text and achieves high precision, recall, and F1-score values, outperforming existing models. Fimoza et al. [31] analyzed the sentiment in Indonesian language movie reviews on YouTube. They utilized a multilingual-cased-model BERT model to achieve the highest accuracy of 68%. Jiang et al. [32] employed techniques for Chinese sentence segmentation, word embedding, and sentiment score calculation to develop and test a sentiment analysis approach for troll detection, focusing on Sina Weibo. Vigna et al. [33] used Facebook as a benchmark and analyzed the textual content of comments on a set of public Italian pages. The authors proposed a variety of hate categories to distinguish different kinds of hate and annotated crawled comments according to the defined taxonomy.

To compare our work with that in related studies, we conducted an extensive review of the literature and identified several key works, as shown in Table 1. While previous studies have primarily focused on account-level features and traditional machine learning algorithms, our approach integrates deep learning and content analysis to more accurately detect and filter malicious content on OSM. Additionally, our system is designed to be scalable and reliable and is capable of processing and storing large streams of data without any loss. Overall, our approach represents a significant advancement in the field of online content filtering and can help mitigate the negative effects of harmful content on OSM.

3 System design and implementation

This section describes the implementation architecture of malicious speech analysis based on the Kafka implementation of API streaming through Kubernetes combined with transfer learning.

Table 1 Comparison with related works

Paper	Data	Features	Online stream	Classifiers	Docker orchestration
[22]	Twitter	Content Hashtags mentions	N/A	Logistic regression naive Baynes decision tree	N/A
[27]	Twitter	Content friend count follower count account level features	Yes	Random forest	N/A
[30]	Twitter	Content	N/A	CNN	N/A
[31]	YouTube	Content	N/A	BERT	N/A
[32]	Weibo	Content	Yes	Hidden Markov model XGBoost, SVM	N/A
[33]	Facebook	Content	N/A	SVM, LSTM	N/A
Ours	Twitter	Content	N/A	CNN	Yes

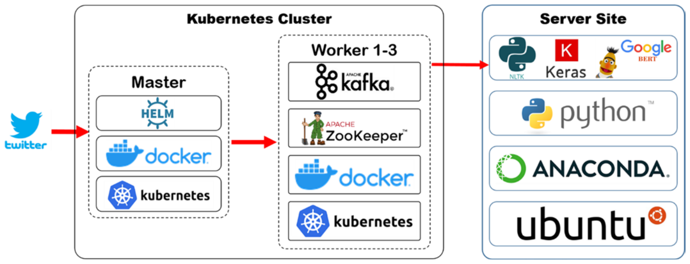


Fig. 2 System architecture diagram

3.1 System architecture

The Kafka system in this experiment is built on Ubuntu 20.04. The producer collects the data from the Twitter API and inputs them into the Kafka broker and then extracts the data through the consumer as the output. An established framework for negative content analysis developed by Keras and NLTK to preprocess the data is combined with transfer learning. Three AI models are trained and compared to find the best one. Figure 2 shows the complete system architecture diagram.

3.2 Kubernetes deployment

In this work, Kafka Cluster is deployed quickly through Kubernetes. The advantage of using Kubernetes is ensuring a powerful horizontal scalability. Kafka characteristics also enable us to quickly increase the number of physical machines to balance work when resources are insufficient. When Kafka Broker fails due to machine

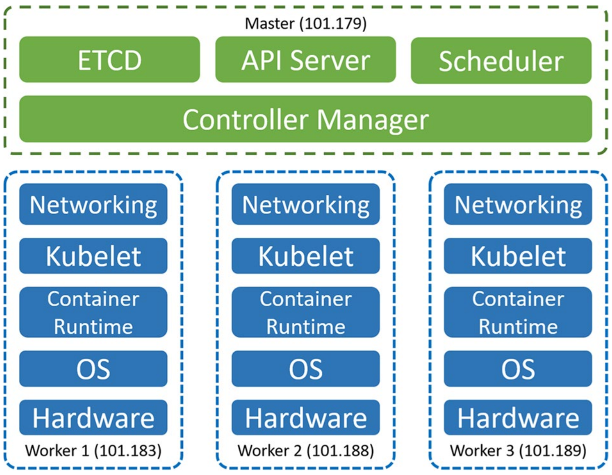


Fig. 3 Kubernetes architecture diagram

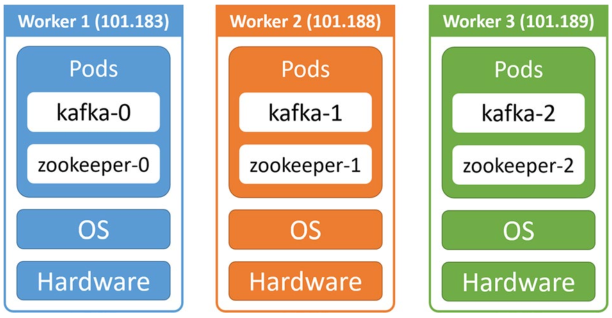


Fig. 4 Pods distribution map

downtime, Kubernetes can also automatically restart the pod service to ensure the normal operation of the system.

The overall architecture of Kubernetes is shown in Fig. 3. The Kubernetes environment is managed by one master node and three worker nodes in the actual operation. In this cluster, we deploy two services: Zookeeper and Kafka. Kafka is used as the data relay station, while Zookeeper is used as the administrator of Kafka.

The distribution of each service (Pod) on worker nodes is shown in Fig. 4. Kubernetes follows the current computing resources and the operation of each machine (whether it is abnormal or down). By employing dynamic scheduling of services, Kafka can achieve high availability in terms of operation.

3.3 Cluster management through zookeeper

In our proposed system, Zookeeper is mainly responsible for managing the cluster configuration of Apache Kafka, selecting the leader for the Kafka broker, and

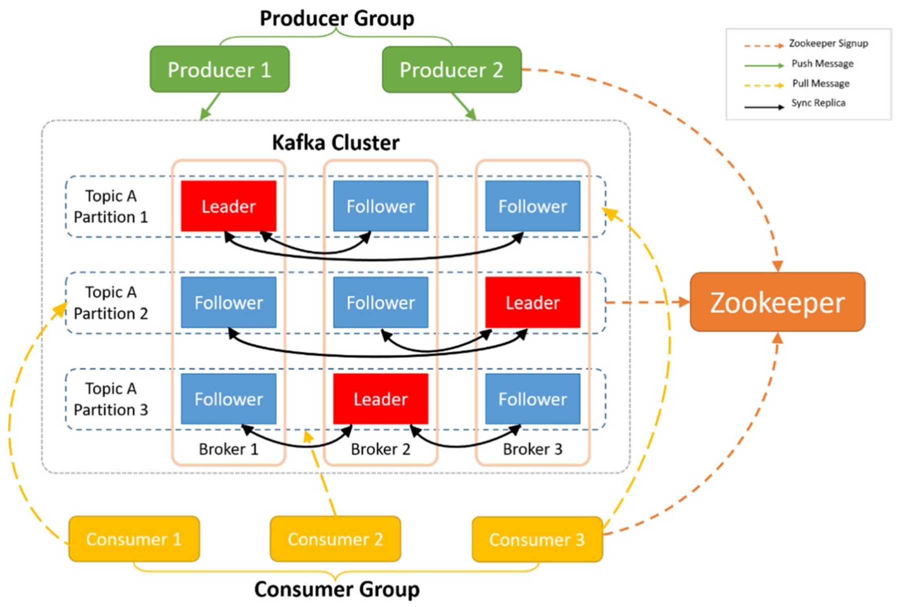


Fig. 5 The role of Zookeeper on Kafka

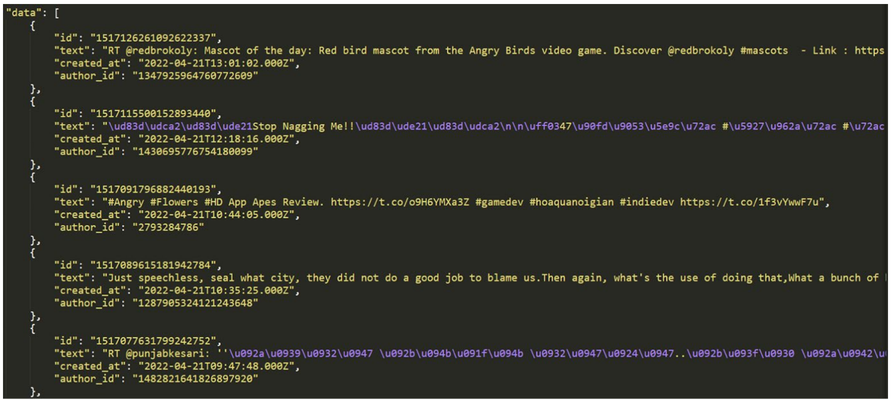
implementing load balancing when the consumer changes. Its operation details are shown in Fig. 5. Next, we will introduce the detailed function of Zookeeper in Kafka:

1. Broker signup

   Zookeeper has a space dedicated to managing Broker's node information. When each Broker starts, it will have a different Broker ID that is registered with Zookeeper and create its own node information under this path "/brokers/ids/[broker_id]". In addition, if a node created by a broker is a temporary node, when the broker fails, the corresponding temporary node will also be subsequently deleted.
2. Topic signup

   In Kafka, the same Topic will be divided into one or more partitions and assigned to different brokers, and these correspondences are maintained by Zookeeper. After the Broker service is started, it registers its own exclusive code on the corresponding path and the number of partitions written to this topic is recorded in the path "/brokers/topics/[topic_name]/[broker_id-partition_id]", while this partition node is also a temporary node.
3. Producer Load Balance

   Since the same topic will be partitioned and distributed to multiple nodes, all messages of the producer will be inserted into the same broker. While Kafka supports traditional four-layer load balancing, we need to achieve load balancing through Zookeeper. To do so, each broker will first complete the broker registration when it initiates. Then, the producer will be aware of broker changes. Accordingly, our system realizes a dynamic load balancing mechanism.



```
"data": [
    {
        "id": "1517126261092622337",
        "text": "RT @redbrokoly: Mascot of the day: Red bird mascot from the Angry Birds video game. Discover @redbrokoly #mascots  - Link : https
        "created_at": "2022-04-21T13:01:02.000Z",
        "author_id": "1347925964760772609"
    },
    {
        "id": "1517115500152893440",
        "text": "\ud83d\udca2\ud83d\ude21Stop Nagging Me!!\ud83d\ude21\ud83d\udca2\n\n\uff0347\u90fd\u9053\u5e9c\u72ac #\u5927\u962a\u72ac #\u72ac
        "created_at": "2022-04-21T12:18:16.000Z",
        "author_id": "1430695776754180099"
    },
    {
        "id": "1517091796882440193",
        "text": "#Angry #Flowers #HD App Apes Review. https://t.co/o9H6YMXa3Z #gamedev #hoaquanoigian #indiedev https://t.co/1f3vYwwF7u",
        "created_at": "2022-04-21T10:44:05.000Z",
        "author_id": "2793284786"
    },
    {
        "id": "1517089615181942784",
        "text": "Just speechless, seal what city, they did not do a good job to blame us.Then again, what's the use of doing that,What a bunch of
        "created_at": "2022-04-21T10:35:25.000Z",
        "author_id": "1287905324121243648"
    },
    {
        "id": "1517077631799242752",
        "text": "RT @punjabkesari: ''\u092a\u0939\u0932\u0947 \u092b\u094b\u091f\u094b \u0932\u0947\u0924\u0947..\u092b\u093f\u0930 \u092a\u0942\u
        "created_at": "2022-04-21T09:47:48.000Z",
        "author_id": "1482821641826897920"
    },
```

Fig. 6 Twitter data screenshot

4. Consumer Load Balance

   Similar to that of the producer, a group ID will be registered for each consumer group, and the consumers in each consumer group share the same ID. In Kafka, each partition must only correspond to one of consumer in a consumer group; however, the same partition can be used by different consumer groups at the same time, and the consumers do not interfere with each other. The relationship between them will be recorded by Zookeeper as follows: "/consumers/[group_id]/owners/[topic]/[broker_id-partition_id]". In this way, the variation of consumers is monitored. Once an increase or a decrease in consumers is found, the load balancing of consumers will be triggered.

5. Consumer offset record

   During the process in which a consumer subscribes to a specific partition, it is necessary to regularly save the progress offset of the subscription to Zookeeper. When a consumer restarts after failure or other consumers retake the message processing of the partition, they can start where they left off.

6. Leader election

   When the Kafka Cluster starts, each broker will attempt to register as a controller with Zookeeper. Zookeeper will randomly select a controller, and the controller will designate the leader or followers for all partitions. The controller will select one of the in-sync replicas of the current partition as the leader as long as any replica survives.

3.4 Data collection

3.4.1 Kafka producer data sources

This work utilizes the Twitter API as the data source of Producer in the Kafka system. In specific, we use the public suite Tweepy provided by Twitter officially on Python to realize the scraping of Twitter API data. Through Tweepy, we are able to obtain the content from a specific account or a specific topic tag (Fig. 6).

3.4.2 Model training dataset for transfer learning

This work uses the dataset provided by Kaggle. There are many public or privately owned dataset regarding troll detection [34], in contrast, our primary contribution involves implementing a streaming data processing system capable of filtering obviously harmful content. Therefore, we have selected a dataset with a code implementation number greater than 45, which the community continues to update. Moreover, all data samples have undergone a meticulous review by all authors. After the samples are screened, and the meaningless data is removed, speeches are labeled. Finally, 43136 training samples and 8587 test samples are used, in which 0 is positive speech and 1 is manually verified offensive speech.

3.5 Data preprocessing

3.5.1 Kafka streaming data

The data published by Kafka are sent one by one, and the Twitter data format we obtained is divided into two categories: "data" and "meta." These two categories contain all the original account information, posting time, location, article content, accompanying photos or videos, and more. Therefore, we need to preprocess these data before using them.

3.5.2 Model training

In this work, we first analyze 43,135 pieces of data in the processed dataset. The analysis result suggests that there are 20,709 sentences with positive words and 22,426 sentences with negative words, of which 0 is a positive word and 1 is a negative word from a troll. We also analyze the length of sentences. The cumulative frequency of sentence length is taken as the 0.91 quantile, and the length is approximately 238, as shown in Figs. 7 and 8.

After analyzing the distribution of sentence patterns, NLTK is used to perform several steps on the sentence in sequence:

1. Sentence segmentation: Let our program interpret and split a string into multiple sentences, of which ". (period)" is a good basis for judging the end of a Sentence. However, there are some exceptions when ". " appears in English, such as Mr. Lee. For this tokenized module in NLTK, there are defined functions that can be used to implement sentence segmentation.
2. Word segmentation: The output result after segmentation is used to separate words.
3. Part of speech mark: The same word may have different parts of speech when used, so the results of word segmentation are labeled according to personal pro-

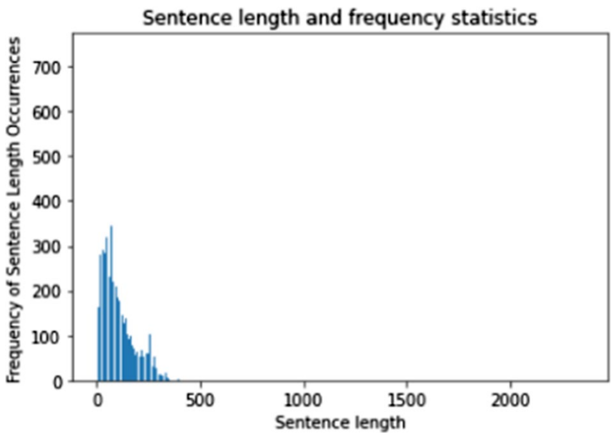


Fig. 7 Sentence length and occurrence statistics chart

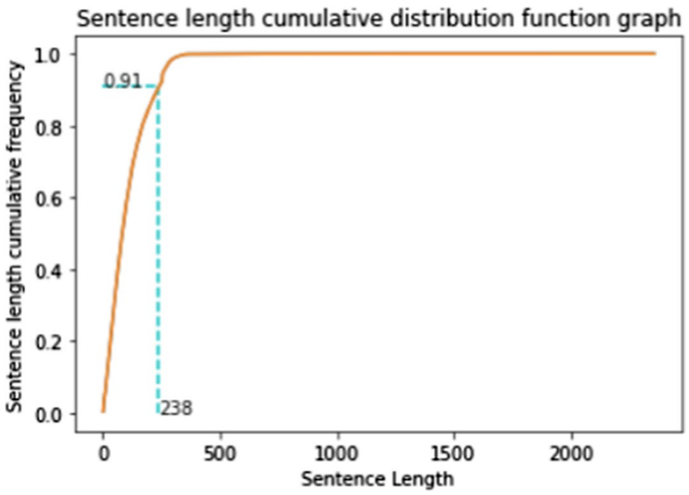


Fig. 8 Sentence length cumulative distribution function graph

nouns, conjunctions, adjectives, verbs, adverbs, etc., thereby providing more accurate information to the machine.

4. Stop words: This step is to remove redundant words, such as "the", "is", "in", "for", "where", "when", "to" and "at". The removal of stop words can reduce the size of the dataset, thereby reducing the training time of the model and simultaneously improving the accuracy of the model because the remaining words are meaningful.

After the data is initially processed by NLTK, the final result will be encoded, and the sentences with insufficient length will be filled with "0"s.

Table 2 Confusion matrix

		True value	
		Positive	Negative
Predicted	Positive	TP	FP
Value	Negative	FN	TN

3.6 Model evaluation

After completing the construction and training of the machine learning model, we need to evaluate the model to understand its learning effect. There are many ways to evaluate a model. However, there is no way to intuitively understand how well the model performs with only the predicted value and the actual value. Therefore, we need several metrics evaluated by the model as a basis for comparison.

There are many ways to evaluate the model. The accuracy rate is one of the most used and observed indicators in classification problems, but only knowing how many prediction results match the actual answer, there is no way to intuitively understand how the model performs. Therefore, we need to have several model evaluation indicators as a basis for comparison. In this study, we use a confusion matrix to calculate other evaluation indicators, including accuracy, recall, precision, and F1 score, which may be of interest in different situations. The confusion matrix contains four elements: true positive (TP), true negative (TN), false-positive (FP), and false-negative (FN), as shown in Table 2.

- TP: Correctly predicts a sample as a positive sample.
- TN: Correctly predicts a sample as a negative sample.
- FP: Falsely predicts a sample as a positive sample.
- FN: Falsely predicts a sample as a negative sample or a positive sample that can not be predicted.

Accuracy is calculated by taking the ratio of the number of correctly predicted samples to the total number of predicted samples, considering all the samples without considering whether the predicted samples are positive or negative. It is calculated as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

Since it is usually difficult to judge the quality of a classification model only by looking at the accuracy rate, we also use indicators such as recall, precision, and F1-score to evaluate the quality of the model. Their calculations are as follows:

$$\text{Recall} = \frac{TP}{TP + FP} \tag{2}$$

$$\text{Precision} = \frac{TP}{TP + FN} \tag{3}$$

Table 3 Kafka cluster device specifications

Quantity	vCPU	Disk	Ram	OS
4	4	500 GB	16 GB	Ubuntu 20.04 LTS

Table 4 Kafka cluster software version

Item	Software version
Master	Docker 19.03.8
	Kubelet 1.17.9
	Helm 3.2.2
	Confluent 5.0.1
	Zookeeper 3.4.13
	Kafka 2.0.1
Worker nodes	Docker 19.03.8
	Kubelet 1.17.9
	Confluent 5.0.1
	Zookeeper 3.4.13
	Kafka 2.0.1

$$F1-\text{score}=\frac{2*\text{Precision}*\text{Recall}}{\text{Precision}+\text{Recall}} \tag{4}$$

4 Experimental results and discussion

4.1 Experimental environment

This section introduces our software and hardware experimental environment. In this experiment, Kafka Cluster uses a total of four hosts, one as the master node and the other three as worker nodes. All instances are Linux-based with Docker, Kubernetes, Helm, Zookeeper and Kafka. The detailed hardware and physical equipment are shown in Table 3, and the software specifications are shown in Table 4.

The role of Kafka consumer is used to identify troll comments. Linux is used as the OS, and Anaconda and its related packages are installed as the Python development environment. The detailed software and hardware specifications are shown in Tables 5 and 6.

4.2 Kafka parameter comparison

To understand the impact of parameters on the Kafka performance, we have written a performance test script to test Kafka Cluster. Thus, we can find the most appropriate results in terms of the overall performance through experiments with different parameters, where the message size represents the size of each data, throughput

Table 5 Model training hardware specifications

Core	Disk	Ram	OS
Intel(R) Core(TM) i7-3970X CPU @ 3.50GHz Nvidia GeForce GTX 2080 Ti	1 TB	64 GB	Ubuntu 20.04 LTS

Table 6 Kafka consumer software version

Software version	
Python 3.7.10	Anaconda 4.8.2
Pandas 1.0.5	Tensorflow 1.14.0
Keras 2.2.5	Matplotlib 3.2.2
Numpy 1.21.2	Kafka 1.3.5
Kafka-python 2.0.2	

represents the maximum number of messages to be transmitted per second, and record amount represents the total number of messages to be transmitted. Through parameter adjustment, we can find the most appropriate result of the overall performance. In terms of performance evaluation, we will focus on the data latency (avg. latency), the number of data reads per second (records/s) and the data file size received per second (MB/s).

4.2.1 Topic partition testing

This experiment is aimed at testing different numbers of partitions while maintaining the consistency of the data. The more partitions there are, the less data a single execution sequence needs to process. In addition, the throughput will be improved, and the time required will vary with the reduction. The message size is set to 512 bytes, the throughput is set to −1, which means that there is no limit to the number of messages transmitted per second, and the number of records is set to 2,097,152. The experimental results are shown in Table 7.

From Table 7 and Figs. 9 and 10, it can be found that when the number of partitions is greater than or equal to the number of brokers, the throughput and the transmission efficiency reach their highest values, and the transmission latency is also relatively shorter. Based on the parameter tuning, we have found that 4 partitions is the best. The throughput and the data that can be inserted per second increase. If we keep increasing the number of partitions, there is no significant effect on the performance.

4.2.2 Throughput testing

Next, we turn our attention to throughput issues. To find the best throughput in terms of the best performance, we set the number of partitions to 4 according to the results of the previous experiment and keep the other parameters the same, where

Table 7 Topic partition testing

Partition	Avg latency (ms)	Record/s	MB/s
1	514.94	119,557.151816	58.38
2	374.24	161,929.735156	79.07
3	236.49	250,885.512621	112.5
4	192.23	306,421.975453	149.62
5	230.82	258,588.409371	126.26

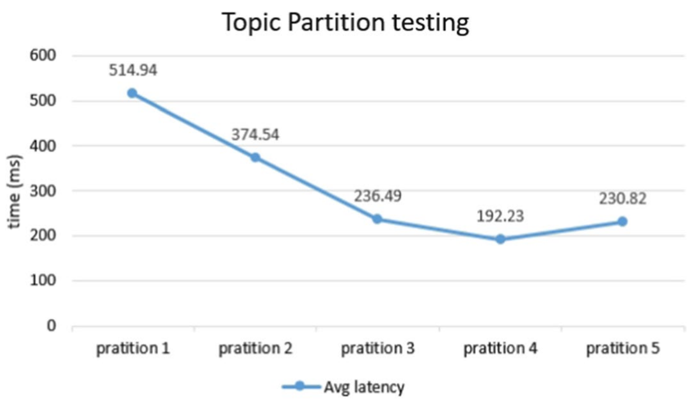


Fig. 9 Topic partition testing—avg latency

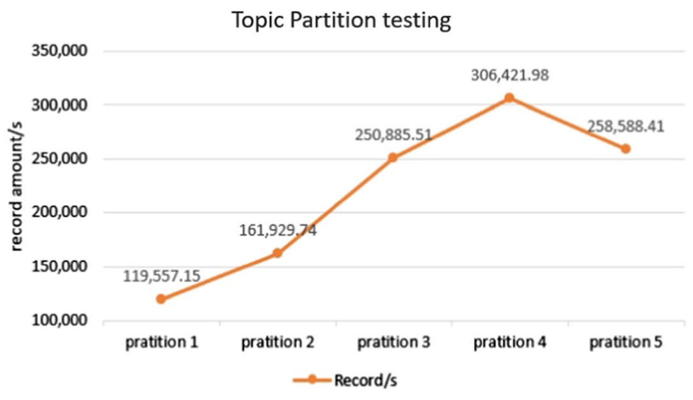


Fig. 10 Topic partition testing—record/s

the message size is set to 512 bytes and the number of records is set to 5,000,000. The experimental results are shown in Table 8.

From Table 8 and Figs. 11, 12, 13 show that when other parameters are fixed, as the throughput increment increases, the number of data (as shown in Fig. 12)

Table 8 Throughput testing

Throughput	Avg latency (ms)	Record/s	MB/s
1000	0.6	1001	0.49
10,000	0.7	10,016	4.89
100,000	1.4	100,040	48.85
1,000,000	166.3	368,286	179.83

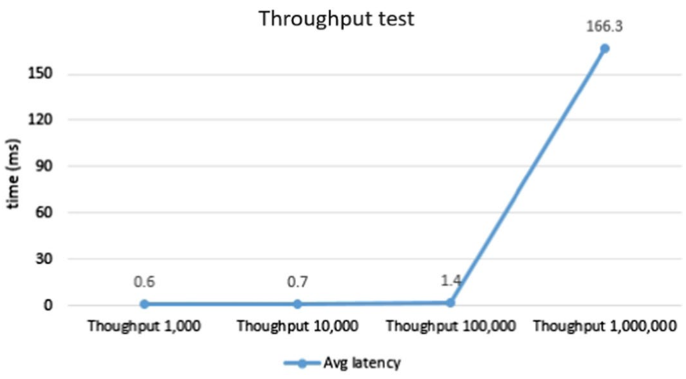


Fig. 11 Throughput testing—avg latency

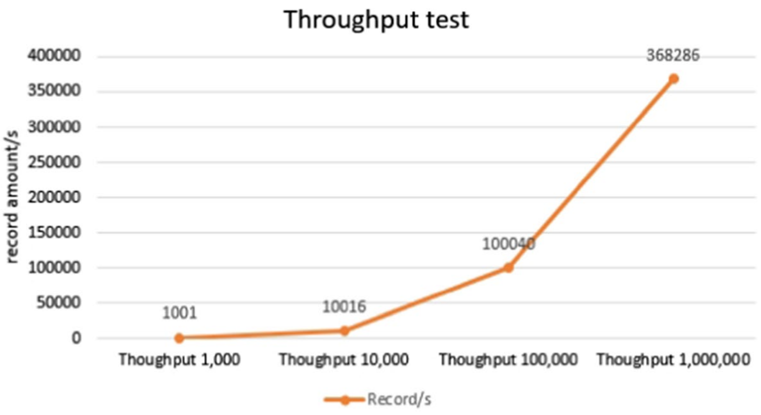


Fig. 12 Throughput testing—record/s

and data size (as shown in Fig. 13) that can be processed per second also increase. When the throughput is set to 1,000,000 and 100,000, the amount of data that can be processed per second is larger, however, the latency also increases (as shown in Fig. 11), and the actual amount of Twitter data that needs to be transmitted in this study is approximately 3–5 MB for every 20,000 transactions. Therefore, in this work, setting the throughput to 10,000 yields the best result.

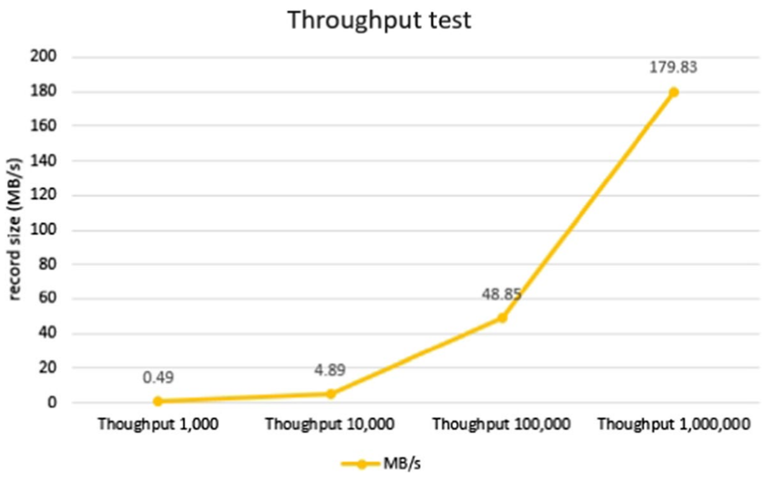


Fig. 13 Throughput testing—MB/s

4.2.3 Replication and ACK testing

In this section, the effect of different numbers of replications and ACK settings on the performance are simultaneously examined. ACK refers to the message sending confirmation mechanism of the producer. It is a data backup strategy of the Kafka cluster, which affects the data durability. This parameter is divided into three options as follows:

- ACKs = 0: The producer transmits the data through the network, and it is considered to be successfully inserted. The producer will not wait for a confirmation of the synchronization completion of the broker and continues to transmit the next piece of information. Although it provides the least delay, it has the weakest data durability. When the server fails, data loss is likely to occur.
- ACKs = 1: The producer confirms that the leader replica has received a message before continuing to transmit the next one but will not confirm whether the leader has received the message. This configuration provides better sustainability and higher latency. When the leader fails, but the follower has not replicated, the data will be lost.
- ACKs = − 1: This configuration is also referred to as ACK = all. In this configuration, the producer will wait for the follower to confirm that it has received the data before continuing to send the next one. Although the data durability and security are the best, the relative latency will be improved and the throughput will be decreased.

In this experiment, we set the number of partitions to 4, message size to 512 bytes, record amount to 5,000,000, throughput to 10,000, and batch size to 1 MB. The experimental results are shown in Table 9.

Table 9 Replication and Acks testing

Replication	Acks	Records/s	Avg latency (ms)
2	0	10010.0	0.3
2	1	10005.4	1.0
2	−1	9992.0	4.4
3	0	10008.0	0.5
3	1	10004.0	1.7
3	−1	9990.0	9.6

Table 10 Twitter API testing

Records/s	MB/s	Avg latency (ms)
9946.95	1.5	2.04

According to the experimental results, it can be seen that the latency is the lowest when ACK = 0, followed by when is ACK = 1, which is when the producer needs to wait for the response of the leader, and finally when is ACK = −1, which is when the producer needs to wait for the reply of the follower, and the more replications there are, the higher the latency. For the throughput part, there is no significant difference due to the small amount of data, but it can still be found that when there are more programs, the throughput will decrease.

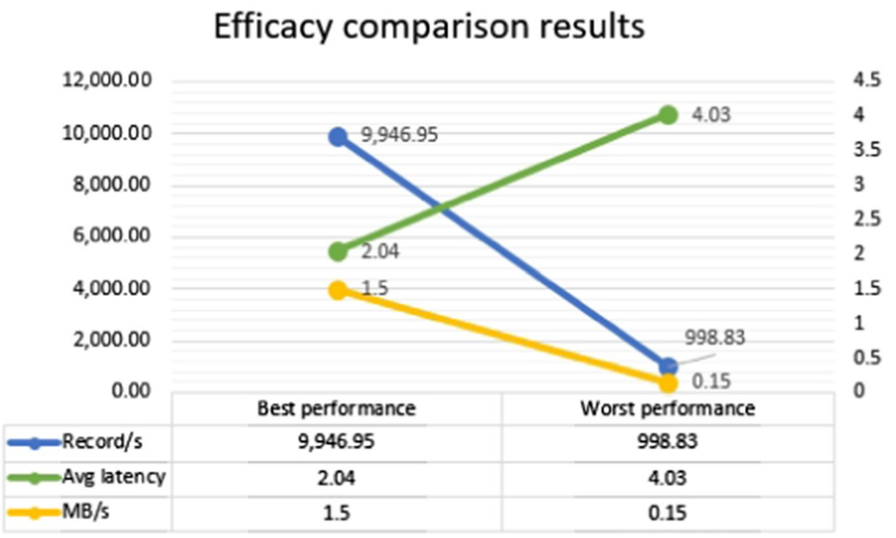
4.2.4 Practical application

Finally, all the best parameters obtained from the above experiments are actually set in a real system, and the Twitter API is used to obtain relevant data as the Kafka producer. The message size is 3.44 MB, the record amount is 20,000, and the other parameters are set as follows according to the above experimental results: the number of partitions is set to 4, throughput is set to 10,000, batch size is set to 5 MB, and replication is set to 2. Since this study does not require strict data storage, we hope that all data transmissions are indeed received by the leader, so we set ACK to 1. The experimental results are shown in Table 10.

In addition, the worst performance is obtained with the Twitter API and the following settings: the number of partitions is set to 1, throughput is set to 1,000, replication is set to 3 and ACK is set to −1. The best performance is compared with the worst performance to confirm the actual difference in operational effectiveness. The comparison results are shown in Fig. 14.

4.3 Transfer learning results

In the experiments of this study, we select three neural networks for training, namely, RNN, LSTM and BERT. First, each model is trained and verified by using the processed dataset, and the model is evaluated.



	Best performance	Worst performance
Record/s	9,946.95	998.83
Avg latency	2.04	4.03
MB/s	1.5	0.15

Fig. 14 Efficacy comparison result

Table 11 Important configurations for deep learning experiments

Input unit	Batch	Epochs	Output dimension	Train:Test	Fold	Activation	Dropout
180	150	60	20	9:1	5	Sigmoid	0.1

Each model training has an Earlystopping mechanism and a ModelCheckpoint mechanism. For the Earlystopping mechanism, we set the training to be stopped in advance when the loss value of the model exceeds 10 epochs and does not drop. While the ModelCheckpoint module occurs during the model training, TensorFlow will monitor the epoch of each round. The loss function saves the model weight of the epoch with the lowest loss value in this complete training. As there are typically many parameters involved in neural network experiments, we present all of the significant ones in Table 11 for easy reference.

Table 12 depicts the accuracy, recall, precision and F1 score for all three transfer learning models. BERT has the best performance in each evaluation index, and its accuracy rate can be as high as 83.61% with a relatively balanced dataset with high throughput and low latency.

5 Conclusion and future works

5.1 Conclusions

In this paper, we present how to utilize Kubernetes to rapidly deploy Kafka Cluster without many manual configurations. When the architecture cannot be loaded, it can

Table 12 Comparison of the results of the three models

	RNN	LSTM	BERT
Accuracy	0.6182	0.7986	0.8361
Recall	0.6786	0.7946	0.8235
Precision	0.6198	0.8165	0.8545
F1-score	0.6479	0.8054	0.8387

quickly expand. When a Kafka broker goes down, Kubernetes can also automatically restart the Pod service to ensure the normal operation of the system. Moreover, we compare the data transmission performance of Kafka and find the best results under this system architecture. According to our experimental results, in this hardware environment, we can divide a topic into four partitions: the throughput is set to 10,000, batch size is set to 5 MB, replication is set to 2, and ACK = 1 to achieve the optimum throughput with the lowest latency and maintain data reliability.

In addition, we use transfer learning to identify offensive content. First, sentence segmentation, word segmentation, and part-of-speech tagging (pos) are preprocessed through NLTK. Then, the processed dataset is input into different neural network models for comparison. These models include RNN, LSTM and BERT, and model evaluation is used to find the most appropriate prediction model. Finally, BERT has the best performance, with an accuracy rate of 83.61%.

In conclusion, we demonstrate a highly scalable and reliable framework to process a large amount of OSM data. Accordingly, several consumers can rely on a pretrained AI model to pull and detect online offensive content in almost real time, which reduces the harm caused by emerging trolls and offensive content if it is removed in a timely manner.

5.2 Future work

According to the characteristics of permanent data storage in the Kafka system, a large amount of data can collected and stored in the future with the advancement of time. Combined with current data analysis technology, more datasets can be obtained, and these large datasets can be used as model training data to conduct more detailed identification and classification of negative speech, such as discriminatory speech, hate speech, negative statements and other inappropriate speech based on race or sexual orientation. In the future, we will import multiple AI models and languages for training and increase the grading of the severity of negative remarks. After the system is improved, it is expected that it can be directly imported into social media, and a warning will be issued immediately when a user makes a post. This allows users to reduce inappropriate speech, thereby reducing offensive social media quarrels.

Author Contributions CM Lai conceived research design, structuring the paper and interpreting the findings. TW Chang collected the data, conducted experiment evaluation and wrote a part of the manuscript. CT Yang supervised the research and reviewed the manuscript.

Funding This work was partially supported by the National Science and Technology Council (NSTC), Taiwan (R.O.C.), under Grants Number 111-2622-E-029-003-, 111-2811-E-029-001-, 111-2621-M-029-004-, and 110-2222-E-029-001-

Data Availability The datasets generated during and/or analyzed during the current study are available from the corresponding author upon reasonable request.

Declarations

Ethical approval Not applicable.

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Rosa H, Pereira N, Ribeiro R, Ferreira P, Carvalho J, Oliveira S, Coheur L, Paulino P, Simão A, Trancoso I (2019) Automatic cyberbullying detection: a systematic review. Comput Hum Behav 93:333–345
2. 2021 Online Social Anxiety and Cyberbullying Experiences among Children in Taiwan Survey https://www.children.org.tw/english/news_detail/bully2021
3. Hinduja S, Patchin J (2019) Connecting adolescent suicide to the severity of bullying and cyberbullying. J Sch Violence 18:333–346
4. Sawhney R, Agarwal S, Neerkaje A, Aletras N, Nakov P, Flek L (2022) Towards suicide ideation detection through online conversational context. In: Proceedings Of The 45th International ACM SIGIR Conference On Research And Development In Information Retrieval. pp 1716-1727
5. Hossain E, Sharif O, Hoque M (2021) NLP-CUET@DravidianLangTech-EACL2021: investigating visual and textual features to identify trolls from multimodal social media memes. In: Proceedings of the First Workshop on Speech and Language Technologies for Dravidian Languages. pp 300-306 (2021,4), https://aclanthology.org/2021.dravidianlangtech-1.43
6. Stewart L, Arif A, Starbird K (2018) Examining trolls and polarization with a retweet network. In: Proc ACM WSDM, Workshop On Misinformation And Misbehavior Mining on the Web. 70
7. Ali R, Farooq U, Arshad U, Shahzad W, Beg MO (2022) Hate speech detection on twitter using transfer learning. Comput Speech Lang 74:101365
8. Kumar DA, Chinnalagu A (2020) Sentiment and emotion in social media covid-19 conversations: Sab-lstm approach. In: 2020 9th International Conference System Modeling and Advancement in Research Trends (SMART), pages 463-467
9. Devlin J, Chang M, Lee K, Toutanova K (2018) Bert: pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805
10. Mendhe C, Henderson N, Srivastava G, Mago V (2020) A scalable platform to collect, store, visualize, and analyze big data in real time. IEEE Trans Comput Soc Syst 8:260–269
11. Alothali E, Alashwal H, Salih M, Hayawi K (2021) Real time detection of social bots on Twitter using machine learning and Apache Kafka. In: 2021 5th Cyber Security In Networking Conference (CSNet). pp 98-102
12. Lai CM, Chen MH, Kristiani E, Verma VK, Yang CT (2022) Fake news classification based on content level features. Appl Sci 12(3):1116
13. Fathoni H, Yen HY, Yang CT, Huang CY, Kristiani E (2021) A container-based of edge device monitoring on kubernetes. In: Chang JW, Yen NL, Hung JC (eds) Frontier Computing. Springer, Singapore, pp 231–237
14. Dewi L, Noertjahyana A, Palit H, Yedutun K (2019) Server scalability using kubernetes. In: 2019 4th Technology Innovation Management and Engineering Science International Conference (TIMES-iCON). pp 1-4
15. Hugo A, Morin B, Svantorp K (2020) Bridging mqtt and kafka to support c-its: a feasibility study. In: 2020 21st IEEE International Conference on Mobile Data Management (MDM), pages 371-376

16. van Dongen G, Van Den Poel D (2021) A performance analysis of fault recovery in stream processing frameworks. IEEE Access 9:93745–93763
17. Wu H, Shang Z, Wolter K (2020) Learning to reliably deliver streaming data with apache kafka. In: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 564-571
18. Wu H, Shang Z, Peng G, Wolter K (2020) A reactive batching strategy of apache kafka for reliable stream processing in real-time. In: 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), pp 207-217
19. Xiao J, Zhou Z (2020) Research progress of RNN language model. In: 2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA). pp 1285-1288
20. Eker A, Eker K, Duru N (2021) Multi-Class Sentiment Analysis from Turkish Tweets with RNN. In: 2021 6th International Conference on Computer Science and Engineering (UBMK). pp 560-564
21. Saha D, Das A, Nath TC, Saha S, Das R (2022) Detection of Fake News and Rumors in Social Media Using Machine Learning Techniques With Semantic Attributes. In: Convergence Of Deep Learning In Cyber-IoT Systems And Security. pp 85
22. Davidson T, Warmsley D, Macy M, Weber I (2017) Automated hate speech detection and the problem of offensive language. Proc Int AAAI Conf Web Soc Media 11:512–515
23. Waseem Z, Hovy D (2016) Hateful symbols or hateful people? Predictive features for hate speech detection on twitter. In: Proceedings Of The NAACL Student Research Workshop. pp 88-93
24. De Gibert O, Perez N, Garcia-Pablos A, Cuadros M (2018) Hate speech dataset from a white supremacy forum. arXiv:1809.04444
25. Cresci S (2020) A decade of social bot detection. Commun ACM 63:72–83
26. Qian J, ElSherief M, Belding E, Wang W (2018) Leveraging intra-user and inter-user representation learning for automated hate speech detection. arXiv:1804.03124
27. Alothali E, Alashwal H, Salih M, Hayawi K (2021) Real time detection of social bots on twitter using machine learning and apache kafka. In: 2021 5th Cyber Security in Networking Conference (CSNet), pp 98-102
28. Fimoza D, Amalia A, Harumy TH (2021) Sentiment analysis for movie review in bahasa indonesia using bert. In: 2021 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA), pp 27-34
29. Ksieniewicz P, Zyblewski P, Choraś M, Kozik R, Giełczyk A, Woźniak M (2020) Fake news detection from data streams. In: 2020 International Joint Conference On Neural Networks (IJCNN). pp 1-8
30. Roy P, Tripathy A, Das T, Gao X (2020) A framework for hate speech detection using deep convolutional neural network. IEEE Access 8:204951–204962
31. Fimoza D, Amalia A, Harumy T (2021) Sentiment analysis for movie review in Bahasa Indonesia using BERT. In: 2021 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA). pp 27-34
32. Jiang Z, Di Troia F, Stamp M (2021) Sentiment analysis for troll detection on Weibo. In: Malware Analysis Using Artificial Intelligence and Deep Learning. pp 555-579
33. Del Vigna12 F, Cimino23 A, Dell'Orletta F, Petrocchi M, Tesconi M (2017) Hate me, hate me not: Hate speech detection on facebook. In: Proceedings of the First Italian Conference on Cybersecurity (ITASEC17). pp 86-95
34. Wagh R, Punde P (2018) Survey on sentiment analysis using twitter dataset. In: 2018 Second International Conference on Electronics, Communication And Aerospace Technology (ICECA). pp 208-211