

A Hybrid CNN-Based Deepfake Detection Framework for Identifying Machine-Generated Text on Social Media

V. Shanmuka, T. Madhu Hanshika, E. Abhideep, Y. Satyam, K. Srinu
UG Student, UG Student, UG Student, Assistant Professor, Assistant Professor
Information Technology,
CMR Technical Campus, Hyderabad, India

Abstract : In today's world social media platforms are where people talk and share information. As these platforms get bigger they also have more content made by machines like tweets from bots. This kind of content can spread information and trick people into thinking something that is not true. The system uses methods like Naive Bayes, Logistic Regression, Decision Tree, Random Forest, Gradient Boosting and Convolutional Neural Networks to look at tweet data. A set of tweets some written by people and some by bots was used to train and test the system. First the text was cleaned up by taking out words, punctuation and making words simpler. The method used here helps make social media more reliable by finding content made by bots and stopping the spread of information. Social media platforms like these need systems that can detect machine-generated content especially deepfake text to keep people from being tricked. The system that uses machine learning and deep learning techniques is good at detecting machine-generated tweets, on media platforms.

IndexTerms - Deepfake Text Detection, Machine Learning, Deep Learning, Convolutional Neural Networks (CNN), FastText Embeddings, TF-IDF, Natural Language Processing (NLP), Social Media Analysis, Bot Detection, Tweet Classification

I. INTRODUCTION

In years social media platforms like Twitter, Facebook and Instagram have become really important for communication sharing information and interacting with the public. Millions of users consume content every day making these platforms powerful tools for influencing opinions and spreading information. However, the rapid growth of media has also led to the emergence of content generated by machines commonly produced by automated bots. These bots can create text that sounds real and people call it deepfake text. It can trick users. Spread false information. Detecting tweets made by machines is now a challenge. This is because machines have gotten better at generating like language. Modern language models can write text that's very similar to what humans write. This makes it hard to tell what is fake. What is not. This problem is a deal with short messages, like tweets. There is not context and people use casual language. This makes it harder to tell what is real and what is fake. We really need systems to find deepfake text on social media. It is becoming more important to have systems to detect deepfake text, on social media platforms. Traditional machine learning methods are often used for text classification. The text classification tasks are done using these algorithms and techniques. Machine learning approaches work well with text data. Although these methods are simple and computationally efficient they have limitations in capturing meaning and contextual relationships within the text. With the advancement of learning more sophisticated models like Convolutional Neural Networks (CNN) Recurrent Neural Networks (RNN) and transformer-based models like BERT have been introduced. These models can automatically learn patterns and features from data resulting in improved classification performance.

Among these CNN models have gained popularity due to their efficiency and ability to extract features from text data, especially for short and noisy inputs like tweets. In addition to model architectures feature representation plays a role in text classification. Word embedding techniques like Word2Vec, GloVe and FastText have been developed to capture relationships between words. In this work a hybrid deep learning-based approach is proposed for detecting machine-generated tweets. The system combines FastText embeddings with a Convolutional Neural Network (CNN) to improve classification accuracy. A hybrid model integrating CNN with Random Forest is also implemented to enhance performance. The proposed system aims to provide an accurate and scalable solution for deepfake text detection, on social media platforms thereby improving the reliability of information shared on them. A web-based application is developed using the Django framework to provide real-time tweet classification.

II. LITERATURE SURVEY

A. Traditional Machine Learning Approaches

People have been using machine learning for a long time to figure out what kind of text they are dealing with and to find fake content. They use algorithms like Bayes, Support Vector Machines and Logistic Regression. These algorithms work with things like Bag-of-Words Term Frequency and Term Frequency-Inverse Document Frequency to understand the text. They change the text into numbers. Then classify it based on the patterns they find. Naive Bayes is really good at classifying text because it looks at probabilities. Logistic Regression is also good especially when you have to make a decision between two choices. Some people also use methods like Random Forest and Gradient Boosting to make their results better. The problem is that these methods need people to manually make the features and they are not great at understanding what the text really means, especially for short things like tweets on social media. Machine learning approaches, like these have a time capturing what the text is really saying and that is a big problem.

B. Feature Engineering and Representation Learning

Feature engineering is really important for making machine learning models work better. Traditional methods like TF and TF-IDF look at how words are used but they do not get what the words mean. To fix this techniques like Word2Vec, GloVe and FastText have been developed. FastText is good because it looks at parts of words which helps with words that're not in the dictionary and words that are spelled wrong. This is super helpful for media posts, which often have casual language and shortened words. By turning words into vector representations these techniques help models understand how words are related. Studies have shown that using word embeddings with classification models makes text classification work better than old feature extraction methods. FastText and Word2Vec and GloVe are really useful for understanding text. These techniques are especially useful when working with media text. They help models to understand the meaning of words, not their frequency. This leads to performance, in text classification tasks.

C. DEEP LEARNING AND CROSS-PLATFORM MODELS

Deep learning models are really good at helping text classification systems work better. Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) Long Short-Term Memory (LSTM) models, are often used to spot fake content and analyze sentiment. CNN models are great at finding patterns and features in text. LSTM models help understand the sequence and context of text. Research shows that CNN-based models are more accurate at classifying texts, like tweets because they are efficient and can automatically detect important features. This is why CNN models are good for text classification tasks. Cross-platform models are also being explored to detect content, on multiple social media platforms. They also need a lot of computer power. This is a problem because it makes them difficult to use in life. We need deep learning models to be more practical. They should be easy to use in real-life cases. Deep learning models require datasets and They also require a lot of resources. This limitation affects their deployment.

D. Pre-trained Language Models and Specialized Applications

Language models like BERT have done a job in understanding language. (BERT) Bidirectional Encoder Representations from Transformers. BERT models are good, at figuring out what words mean in a sentence and They do this by looking at the text that come after them. It helps, because it allows BERT models to understand what the sentence is actually saying. Language models like BERT are good at detecting news and figuring out how people feel about things. They can also classify text into categories. BERT and other models like it are better at these tasks than models. These models need a lot of power to run and they use a lot of memory. This means they are not great for things that need to happen in time or for systems that do not have a lot of resources. There are some applications that try to detect bots and fake text. These applications use a combination of methods to make them more efficient and accurate. Language models, like BERT are used for these applications. They help with bot detection and deepfake text identification.

To deal with these problems people have been doing research to make these models work better for tasks. They use things like making changes to the models teaching them what they already know and using what they learned before to make them work with less computing power but still get good results. For jobs they mix these models with other things, like CNNs, RNNs or using multiple methods together to get things done faster and more correctly.

When we talk about finding text, the models that are already trained to understand language are really good at spotting patterns and things that do not sound right in text that machines have written. These models can find differences between text

that people have written and text that artificial intelligence has generated and these differences are not always easy to see.. The thing is these models need a lot of power to work so people often use simpler options like FastText with special architectures that use CNN because they are better, for real life and can work fast.

E. Research Gap and Motivation for Proposed Work

Despite advancements in machine learning and deep learning detecting machine-generated text on media remains tough. Traditional methods do not work well because they miss the context and meaning of the text. Advanced deep learning models are often too complex. Need a lot of computer power. Short and noisy texts, like tweets make it harder because they have limited context and use language. Existing systems struggle to detect machine-generated content made by language models. We need a system that's efficient, accurate and can handle short texts in real-time. The proposed system uses FastText embeddings to better represent text and a Convolutional Neural Network (CNN) to automatically extract features and classify text. A hybrid approach combining CNN and Random Forest is also used to improve performance without using much computer power.

The system aims to improve detection of machine-generated text, on media by handling short text data efficiently. It also focuses on providing real-time detection to address the challenges posed by noisy texts. Recent deep learning models like CNN, LSTM and transformer-based models are really good at classifying text. They have shown to perform in text classification tasks. These models need a lot of data, powerful computers and a lot of time to train. This makes them not so great for applications that need to work in time and on systems that don't have a lot of resources. Many current approaches also struggle with noisy text, which is everywhere on social media sites like Twitter. These types of text make it hard for models to work accurately. Short text can be a few words and noisy text can have lots of mistakes. So we need models that can handle these kinds of text and still work well. Now many models are not good at dealing with these challenges. They are not suitable for use on systems with resources. We need to find a way to make models that're more efficient. This will help us to deploy them on systems. Models like CNN, LSTM and transformer-based models are very useful. We need to make them more efficient. They require a lot of power.

There is a problem with the way we do research now. We do not have models that combine machine learning and deep learning in a smart way. Deep learning models are really good, at finding things in data but they can be too perfect and take a lot of time to work. Machine learning models are faster and easier to understand. They do not really get what is going on in a deep way. We need to find a way to use both machine learning and deep learning so we can get the good things from each. This will help us make models that work well and are easy to understand. Machine learning and deep learning are both important so we should use them together to get the results.

To fix these problems our system uses a mix of methods. It combines FastText embeddings with a Convolutional Neural Network for finding features and a Random Forest classifier to make the final prediction. FastText is good at understanding how words are related even if they are not common or are spelled wrong. This is really helpful. The Convolutional Neural Network finds patterns in the text on its own. A Random Forest model makes the classification more stable. Reduces errors. It does this by combining trees. FastText, Convolutional Neural Network and Random Forest all work together. They make the system more accurate. The system is better, at handling text. It understands the text well. FastText helps with words. The Convolutional Neural Network and Random Forest make predictions. The system they are talking about is supposed to work well and be good for things that need to happen right away. It is made to deal with texts and texts that are not very clear in a good way without using up too much computer power. The system uses a lot of methods to try to find tweets that were written by machines. This system is trying to help people figure out if a text on media is a deepfake text or not and it wants to be a solution that people can trust. The system is designed to detect machine-generated tweets and identify deepfake text on media platforms, like the ones we use every day.

III. SYSTEM ARCHITECTURE

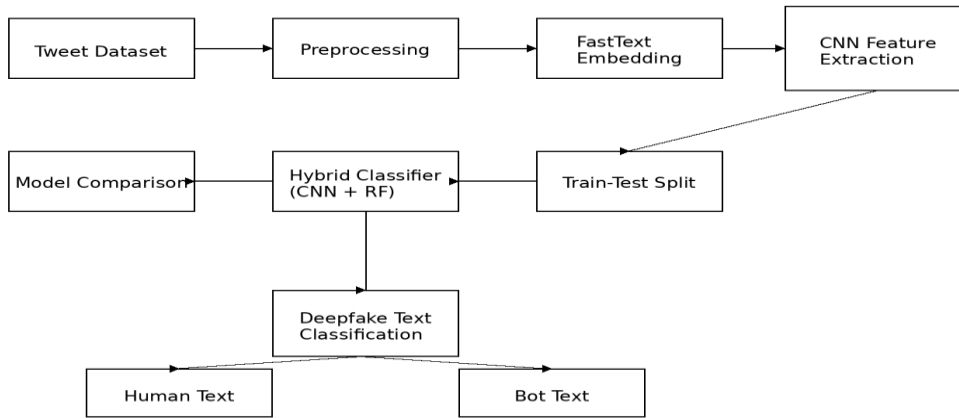


Fig:1 System Architecture Of CNN Based Deepfake Detection

The proposed system architecture is designed to detect machine-generated tweets. It uses a mix of machine learning and deep learning techniques.

The overall workflow starts with collecting a tweet dataset. This dataset contains both human-generated and bot-generated content. The dataset goes through preprocessing steps. These steps include: Removing stopwords, Punctuation, Characters, Converting text into lowercase. This cleaned data is then passed through a feature extraction phase. It uses FastText embeddings. FastText embeddings convert data into numerical vectors. These vectors keep the meaning of the text. These vectors are further processed by a Convolutional Neural Network (CNN) model. This CNN model extracts features from the text and our system is built up of web-based application.

The architecture also has a training and evaluation module. The dataset is divided into training and testing sets. Various algorithms are trained on the processed data. Their performance is evaluated using metrics like: Accuracy, Precision, Recall, F1-score. A hybrid classifier is also implemented. It combines CNN with Random Forest. This improves classification performance. And the CNN model extracts text from the data, and hybrid approach enhances the system’s ability to detect patterns in machine-generated text. The system performs deepfake text classification. It categorizes tweets into either human-written or bot-generated classes. The architecture ensures scalability and efficiency. It allows the system to handle volumes of social media data. The integration of FastText embeddings CNN-based feature extraction and hybrid classification provides a solution for detecting deepfake text.

This architecture enables real-time prediction. It improves the reliability of social media platforms and Reduces the spread of machine-generated content. The system uses FastText embeddings to keep the meaning of the text. and the model extracts features from the text. and our system provides an interactive platform & it also finds deepfake text detection. and the system detects machine-generated tweets using a combination of machine learning and deep learning techniques.

IV METHODOLOGY

Table: 1 Summary of Methodology Components and Functional Description

| Module | Input | Processing | Output |
|---------------------------|------------------------------------|---|--|
| Dataset Collection Module | Raw tweet text, Labels (Human/Bot) | Data loading, duplicate removal, class balancing | Structured labeled dataset (TweepFake dataset) |
| Text Preprocessing Module | Raw tweet text | Lowercasing, punctuation removal, tokenization, stopword removal, stemming, lemmatization | Cleaned and normalized text |

| | | | |
|------------------------------|---------------------------|--|---|
| Feature Extraction Module | Cleaned text tokens | TF-IDF vectorization, FastText embedding | Numerical text feature vectors |
| Model Training Module | Feature vectors, Labels | Training Naive Bayes, Logistic Regression, D Tree, Random Forest, Gradient Boosting, CNN | Trained ML/DL models (.pkl / .hdf5 files) |
| Model Evaluation Module | Trained models, Test data | Accuracy, Precision, Recall, F1 Score calculation | Performance metrics, Best model selection |
| Deepfake Detection Module | New tweet input | Text preprocessing + TF-IDF/FastText + CNN prediction | Output label: Human / Bot |
| Hybrid Classification Module | CNN extracted features | Random Forest classification on CNN features | Improved classification result |
| Admin Dashboard Module | Dataset, Model results | Visualization, monitoring model performance | Graphs, statistics, model comparison charts |

Table 1 shows the way the proposed system works is divided into parts each doing a specific job to find out if a tweet is a deepfake or not. It starts with collecting tweets, which includes things written by people and things written by bots. This is done in the dataset collection part. After that the text preprocessing part makes the text clean by taking out words, punctuation and other things that are not needed. It also breaks the text into pieces makes the words simpler and makes sure all the words are in their basic form. The model evaluation part checks how well these models are working by looking at things like how accurate they're how precise they are and how well they can recall things. There is also a part that combines Convolutional Neural Networks with Random Forest to make the predictions more accurate. The whole system is part of a web application made with Django, where people can put in tweets and get answers away.

The final answer says if a tweet is written by a human or a bot. The people in charge can also see how well the system is working and look at the results, in a dashboard, which makes the deepfake detection system work well and can handle a lot of work.

Dataset Collection and preparation:

The dataset for this project includes labeled tweets from media. It has both bot tweets, used to train and test the model. The dataset is cleaned by removing duplicates and irrelevant data. Labels are assigned as Human or Bot for machine-generated tweets. This balanced dataset of human and bot tweets improves classification model performance. The dataset of labeled tweets is split into training and testing sets. This helps to evaluate the model, on human and bot tweets. The model is trained on human-generated and bot-generated tweets. It is then tested on the testing set of bot tweets.

A. Text Processing Module

We remove spaces and deal with words that are repeated. We also change words into a standard format. We find that social media text has a lot of things like emojis and hashtags that do not really help us. So we. Remove these things or process them in a way that makes sense. This helps the model focus on the text that actually means something. We also get rid of numbers and characters that're not letters. This helps to reduce the noise in the data. We have to be careful with words that are misspelled because this happens a lot on media. We use techniques like stemming and lemmatization to group words that mean things together. This helps to make the data smaller. It makes the model work better. These steps also help to remove information and make the data more consistent. After we clean up the text we put it into a format that's good for getting features from it. Cleaning the text is very important for making the model work well. It directly affects how good the input data is. If we remove information that's not relevant and noisy the system gets better at finding patterns and telling the difference, between tweets written by humans and tweets written by machines.

B. Feature Extraction

The text is cleaned up. Then it is turned into numbers using special methods. In this project we use TF-IDF and FastText embeddings to show what the text data is like. TF-IDF is helpful because it finds the words in a document by giving more importance to words that are not used very often and less importance to words that are used all the time. FastText embeddings are also useful because they understand how words are related to each other and they can even handle parts of words which's really useful when we are dealing with text from social media. These methods change the text into something that the computer can understand which's, like a set of features that the computer can use to learn from. We can then use these features to train machine learning and deep learning models.

C. Machine Learning Models

We tested six machine learning and deep learning classifiers in this study.

1) Naive Bayes

It thinks that features are not connected to each other. This is not always true with world data.. It works well for classifying text because it is simple and fast. The Naive Bayes classifier is really useful when we are working with a lot of data like text features.

2) Logistic Regression

Logistic Regression is a way to classify things into two groups. It uses a function to predict which group something belongs to. This method is simple and fast. It works well when the features and labels are connected in a way. People use Logistic Regression a lot for classifying text.

3) Decision Tree

Decision Tree classifier makes choices based on feature values. It divides data into branches. Creates a tree, like structure. Each branch ends in a label. This method is easy to understand.. If we are not careful it can make mistakes by fitting the data too closely.

4) Random Forest

Random Forest is a way to combine Decision Trees to get better results. It uses parts of the data and features to build each tree. Then it makes a prediction based on what most of the trees say.

5) Gradient Boosting

Gradient Boosting is a technique that builds models one after the other. Each model tries to fix the mistakes of the one. It combines weak models to make a strong one.

6) Convolutional Neural Network(CNN)

In this project we used CNN to analyze tweets and get results than traditional machine learning models. The Convolutional Neural Network is particularly good, at handling unstructured data like tweets.

D. Model Training and Evaluation

One part is for training. The other part is, for testing. We use the training part to train our model. The testing part is used to see how well the model works. The dataset is split into training and testing parts. This helps us to evaluate the models performance. This is done to see how well the models work. Each model is trained with the training data. Then tested with the testing data. The models are judged on how they do. We use things like accuracy and precision and recall and F1-score to do this. Accuracy is like a score that shows how correct the model is overall. Precision is about how many of the things the model says are positive're actually positive.

V. EXPERIMENTAL RESULTS AND EVALUATION

A. Dataset Description

The dataset for this project comes from media platforms. It is a collection of tweets. The goal is to tell human-written tweets from machine-generated ones. This data is from a dataset. Many researchers use it to study deepfake text detection. It has tweets from users. It also has tweets from automated bots. This mix helps train the model in a way. The model can learn to spot the differences between machine-generated content. Tweets are used because they show how people and machines write in a setting. The dataset is useful, for studying how to detect text.

B. Experimental Setup

This project is going to see how different machine learning and deep learning models can find tweets that machines have written. The system is built using Python and the Django framework is used to make a website that can predict things in time. We use the training data to build models. The testing data helps us see how well these models work. Before training the models we clean up the data. We remove words and unnecessary things to make the data better for our models. The data cleaning step is important. We use Scikit-learn for building models. NLTK and NLTK help us work with text data. NumPy and Pandas are useful.

Keras is used for building networks. We use these libraries to make our models. We also break down the words into parts and make them simpler. After that we use techniques, like TF-IDF and FastText embeddings to turn the text into numbers that the models can understand. Machine learning models are used to find machine-generated tweets and deep learning models are used to find machine-generated tweets.

C. System Interface

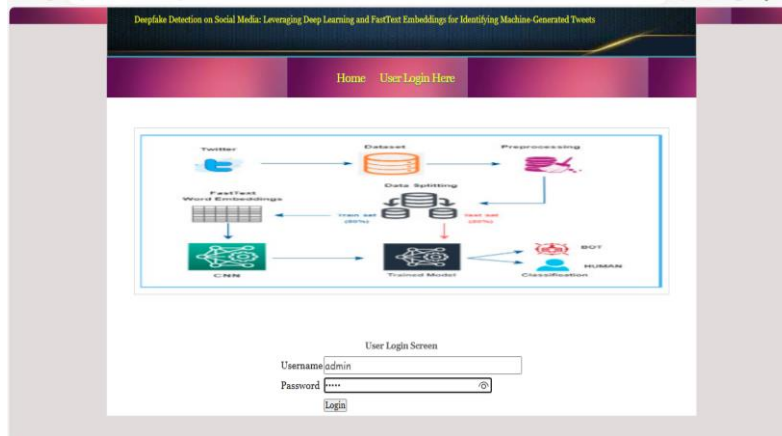


Figure 2. Screenshot of the Deepfake Detection system home page showing the main interface with User Login and system workflow diagram.

Figure 2 shows the home page of our Deepfake Detection web application. The interface has a login section for users to enter their details to access the system. The home page also shows how the system works. It includes steps like inputting tweet data, cleaning up the data, using FastText to understand the text, using a CNN model to process it and classifying it as human or bot-generated.

| Algorithm | Accuracy | Precision | Recall | FScore |
|----------------------|----------|-----------|--------|--------|
| Naive Bayes | 58.00 | 57.83 | 56.67 | 55.65 |
| Logistic Regression | 57.00 | 57.37 | 57.35 | 57.00 |
| Decision Tree | 61.50 | 61.50 | 61.56 | 61.45 |
| Random Forest | 60.50 | 60.43 | 60.48 | 60.42 |
| Gradient Boosting | 61.00 | 65.34 | 62.57 | 59.69 |
| CNN Algorithm | 87.11 | 87.19 | 87.11 | 87.10 |
| Extension Hybrid CNN | 94.57 | 94.55 | 94.73 | 94.56 |

Figure 3: Performance Comparison of Machine Learning Classifiers

Figure 3 presents the complete performance comparison of all classifiers on the test set.

D. Performance Visualization

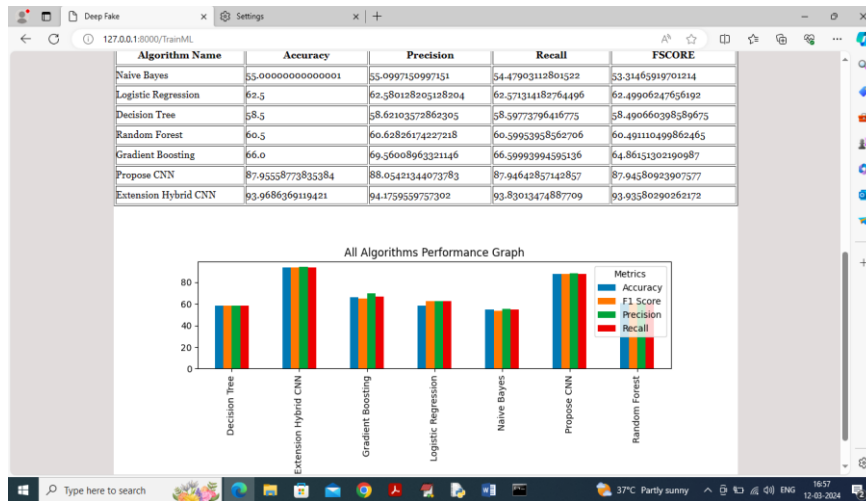


Figure 4. Screenshot of the Performance Visualization page showing the model performance table and bar chart comparing Accuracy, Precision, Recall, and F1-Score across all classifiers.

Figure 4 shows the Performance Visualization page of the Deepfake Detection system we proposed. This page gives a comparison of all the models we trained. We used metrics like Accuracy, Precision, Recall and F1-Score to evaluate them. The results are, in a table, which lists how well each algorithm did. We also used a bar chart to compare the models performance. This makes it easier to see how effective each Deepfake Detection model. The Performance Visualization page helps us understand the results of our Deepfake Detection system better. It shows us which model works best for detecting Deepfakes.

E. Real-Time Detection



Figure 5 Screenshot of the Real-Time Detection page showing tweet input interface and classification result as Human or Bot using the trained model.

Figure 5 shows the Real-Time Detection page of the Deepfake Detection system. This is where users can type in some tweet text and get a classification result away. So when someone puts in a tweet the Deepfake Detection system does some work on it. The Deepfake Detection system cleans it up. Breaks it down into smaller pieces. Then the Deepfake Detection system makes sure everything is consistent. After that the Deepfake Detection system turns the text into numbers using TF-IDF and FastText embeddings. The Deepfake Detection system then uses these numbers to make a prediction, with the trained CNN and hybrid models.

VI. FUTURE SCOPE AND CONCLUSION

The system we propose works well for finding tweets made by machines. It uses a mix of machine learning and deep learning methods. The results of our tests show that our model does better than machine learning methods. It is more accurate. Works better overall. We also made a web app. This makes it easier to use our system and classify tweets in time. The proposed system demonstrates an approach for detecting machine-generated tweets. It uses a combination of machine learning and deep learning techniques. By integrating FastText embeddings with a Convolutional Neural Network the system captures both contextual patterns in data. The system can also be made better by adding things like looking at what users do checking the network and looking at metadata. If we look at what people write and what they do on the site we can find bots that're really good at hiding.

In conclusion our deepfake detection system offers an efficient way to spot fake tweets made by machines. It uses learning and advanced methods to pull out key features leading to high accuracy and fast performance. The results show that combining CNN and FastText embeddings works well for classifying text. This system can help a lot in fighting misinformation and making social media safer and more trustworthy if we make some improvements and scale it up. Our deepfake detection system and fake tweets are a concern. The deepfake detection system is very important.

VII. REFERENCES

- [1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," Proc. International Conference on Learning Representations (ICLR), 2013.
- [2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," Proc. NAACL-HLT, 2019, pp. 4171–4186.
- [3] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of Tricks for Efficient Text Classification," Proc. EACL, 2017, pp. 427–431.
- [4] Y. Kim, "Convolutional Neural Networks for Sentence Classification," Proc. EMNLP, 2014, pp. 1746–1751.
- [5] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
- [6] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information," Transactions of the Association for Computational Linguistics, vol. 5, pp. 135–146, 2017.
- [7] R. Zellers, A. Holtzman, H. Rashkin, Y. Bisk, A. Farhadi, F. Roesner, and Y. Choi, "Defending Against Neural Fake News," Proc. NeurIPS, 2019.
- [8] J. Pennington, R. Socher, and C. Manning, "GloVe: Global Vectors for Word Representation," Proc. EMNLP, 2014, pp. 1532–1543.
- [9] D. Jurafsky and J. H. Martin, Speech and Language Processing, 2nd ed., Pearson, 2009.
- [10] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, no. 3, pp. 273–297, 1995.
- [11] L. Breiman, "Random Forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.
- [12] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," Annals of Statistics, vol. 29, no. 5, pp. 1189–1232, 2001.
- [13] [T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," Proc. KDD, 2016, pp. 785–794.
- [14] S. Bird, E. Klein, and E. Loper, Natural Language Processing with Python, O'Reilly Media, 2009