AI – Generated Text Detection using CNN-LSTM

Akhil Karri  
Computer Science Department  
University of IdahoMoscow, Idaho  
karr8512@vandals.uidaho.edu

*Abstract*—This report presents a machine learning-based approach for detecting AI-generated text using deep learning. We leverage a combination of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks to classify text as either human-generated or machine-generated. The dataset consists of text samples labeled accordingly, and we apply preprocessing steps such as text vectorization and sequence padding to prepare the data for model training. The model was evaluated using metrics such as accuracy, precision, recall, and F1 score, with results showing high performance, achieving an accuracy of 99.23%. This approach demonstrates the potential of deep learning models in distinguishing between human and AI-generated content, offering insights into the effectiveness of hybrid models in text classification tasks.

Keywords—Neural Networks, Convolutional Neural Networks, long short-term memory, Recurrent Neural Networks, CNN-LSTM Hybrid Model, TensorFlow, Text Vectorization, Data Pre-Processing, Model Training, Binary Classification, Kernels, Pooling, Layers, Neurons, Accuracy, F1-Score, Precision, Confusion Matrix.

# Introduction

In recent years, the proliferation of artificial intelligence (AI) and machine learning (ML) technologies has led to the development of sophisticated models capable of generating human-like text. These models, such as OpenAI’s GPT series and others, have demonstrated remarkable abilities in producing coherent and contextually appropriate text across various domains, ranging from news articles to creative writing. While the advancements in AI-generated content have immense potential, they also raise significant concerns regarding authenticity, misinformation, and content manipulation. As AI-generated text becomes more indistinguishable from human-written content, the need for robust detection systems to identify AI-generated text has become increasingly important.

This paper proposes a novel approach for detecting AI-generated text using deep learning techniques, specifically combining Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. The proposed model leverages the strengths of CNNs for feature extraction and LSTMs for sequential modeling to distinguish between human-written and machine-generated text. A key challenge in this domain is the imbalance in the dataset, where certain classes (e.g., human-written text) may dominate. To address this, we apply techniques such as data balancing and padding to ensure the model learns effectively from both classes.

The objective of this study is to build a classification model capable of accurately distinguishing between human-written and AI-generated text by training on a large dataset of textual data. The evaluation of the model’s performance is carried out using various metrics, including precision, recall, F1 score, and accuracy, with a particular focus on ensuring a balanced performance across both classes. The contributions of this research are twofold: the development of a hybrid CNN-LSTM model for text classification and the introduction of a reliable and scalable method for identifying AI-generated content.

As AI-generated text continues to improve, the ability to automatically detect such content will be crucial in ensuring transparency and trustworthiness in digital communications, mitigating the spread of fake news, and upholding the integrity of online platforms.

# Tool and Technologies

This section provides an overview of the key technologies and libraries employed in the development of the AI-generated text detection model. The project utilizes a combination of deep learning techniques, data processing libraries, and visualization tools to build, train, and evaluate the model. The following sections describe the tools used in this study:

## Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) are a class of deep learning models primarily used for analyzing visual data but have also proven highly effective in handling text and other sequential data. CNNs are designed to automatically and adaptively learn spatial hierarchies of features from data. They achieve this by using convolutional layers that apply filters to input data, capturing local patterns such as edges in images or n-grams in text.

Key Features of CNNs:

1. Convolutional Layer: The convolutional layer is the core building block of CNNs. It applies a set of filters (also known as kernels) over the input data, performing a convolution operation. Each filter detects specific features (such as edges in images or word patterns in text). The filters are learned during training, allowing the model to automatically identify relevant features.
2. Filter (Kernel): A small matrix that slides over the input data to detect local patterns. The filter is applied to all regions of the input (e.g., a window of text or pixels) to extract features.
3. Stride: The step size with which the filter moves over the input. A larger stride reduces the spatial dimensions of the output.
4. Padding: To preserve the spatial dimensions of the input, padding may be added around the edges of the input data.
5. Activation Function: After the convolution operation, the result is passed through an activation function (usually ReLU, which introduces non-linearity into the network) to produce a feature map. This feature map contains high-level representations of the detected features.
6. Pooling Layer: Pooling layers are used to reduce the spatial dimensions of the feature maps, which helps reduce computation and control overfitting. The most common type of pooling is max pooling, where the maximum value from each region of the feature map is selected. Pooling helps make the model more invariant to small translations in the input.
7. Max Pooling: A common form of pooling where the maximum value from each region of the feature map is retained.
8. Average Pooling: A variant where the average value from each region is retained, though max pooling is more commonly used.
9. Fully Connected Layer: After several convolutional and pooling layers, the final feature maps are flattened into a vector and passed through one or more fully connected layers. These layers perform high-level reasoning based on the features extracted by the convolutional layers. The final fully connected layer produces the output, such as a classification prediction.

Advantages of CNNs for Text:

1. Local Pattern Recognition: In text classification, CNNs can learn to recognize local patterns such as word n-grams or even character-level features that are crucial for distinguishing between different types of text.
2. Efficiency: CNNs have fewer parameters compared to fully connected networks, as filters are shared across the entire input. This results in faster training and lower computational requirements.
3. Hierarchical Feature Learning: CNNs are capable of learning hierarchical features from low-level to high-level, allowing them to capture intricate patterns in complex data.

CNN Architecture for Text:

In text classification, CNNs are typically applied to sequences of words or characters. Each word is represented as a vector, and the convolutional filters learn to detect patterns within the sequence. Here, CNNs can capture essential n-grams and syntactic structures in the text that help in classifying the text as either human-written or AI-generated.

The architecture for text processing in CNNs often consists of:

1. Embedding Layer: Converts words or characters into dense vectors, allowing the CNN to work with dense representations of the text.
2. Convolutional Layers: Detect local patterns in the text.
3. Pooling Layers: Reduce the dimensionality and focus on the most important features.
4. Fully Connected Layer: A final decision layer that classifies the text based on the learned features.

Example CNN Architecture for Text:

1. Input: A sequence of words, where each word is converted to a vector using an embedding layer.
2. Conv1D Layer: Applies a 1D convolution operation with filters of size to detect patterns in the sequence.
3. MaxPooling Layer: Reduces the dimensionality of the feature map.
4. Fully Connected Layers: Used to make the final classification decision based on the extracted features.

In this study, CNN was used to extract local features from text sequences, which were then processed by LSTM layers to capture the sequential dependencies and improve the model's ability to distinguish between human and AI-generated text.

## Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are a class of neural networks designed for processing sequential data. Unlike traditional feedforward neural networks, which assume all inputs are independent of each other, RNNs have the ability to retain information from previous inputs, making them particularly well-suited for tasks where the order of the data matters, such as language modeling, speech recognition, and time-series analysis.

Key Features of RNNs:

1. Memory Mechanism: The core advantage of RNNs lies in their ability to maintain a form of memory. This is achieved through the recurrent connections in the network. At each time step, an RNN takes the current input and combines it with the hidden state (memory) from the previous time step to produce the output and the updated hidden state. This mechanism allows the model to capture sequential dependencies and long-term relationships within the data.
2. Sequential Nature: RNNs are inherently designed to handle data in a sequential manner, making them ideal for tasks involving temporal or ordered data. This is particularly useful when working with sequences of text, as the meaning of a word often depends on its surrounding context.

Challenges: While RNNs are effective in many sequence-based tasks, they suffer from certain limitations:

1. Vanishing Gradient Problem: During backpropagation through time, gradients can shrink exponentially, making it difficult for the model to learn long-range dependencies.
2. Exploding Gradient Problem: Conversely, gradients can grow excessively, leading to instability during training.
3. Limited Memory: Standard RNNs struggle to capture long-term dependencies due to their limited memory, which is addressed by more advanced RNN variants such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs).

Types of RNNs:

1. Vanilla RNN: The basic RNN, which is simple but often ineffective at capturing long-term dependencies.
2. LSTM (Long Short-Term Memory): A more advanced RNN that includes gates to control the flow of information and can remember long-term dependencies more effectively.
3. GRU (Gated Recurrent Unit): A variant of LSTM with a simpler structure, which performs similarly in many tasks but with fewer parameters.

In this project, LSTM networks were used as they can effectively model long-range dependencies in sequential data, such as text sequences, making them well-suited for detecting patterns in human vs AI-generated text.

## Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) designed to capture long-range dependencies in sequential data. LSTMs are particularly suited for tasks involving text, as they can retain information from previous time steps, enabling the model to learn and understand the sequential structure of sentences or paragraphs. In this project, LSTM layers were used after the CNN layers to model the temporal relationships within the text and improve the model's ability to distinguish between human-written and AI-generated text.

## TensorFlow

TensorFlow is an open-source machine learning framework developed by Google, widely used for building and training deep learning models. TensorFlow provides an efficient and flexible platform for developing machine learning algorithms, and it is especially popular for tasks involving neural networks. In this study, TensorFlow was used to define, compile, and train the hybrid CNN-LSTM model. It also provided tools for managing the data pipeline, handling model evaluation, and optimizing the model using advanced gradient-based optimization techniques like Adam.

## Pandas

Pandas is a powerful data manipulation and analysis library for Python. It provides data structures such as Data Frames, which are ideal for working with structured data. Pandas was used in this project to load, preprocess, and manipulate the dataset. It allowed for easy handling of large amounts of text data, cleaning, filtering, and transforming the data into a format suitable for training the model. With its intuitive interface, Pandas made it straightforward to perform tasks such as data splitting, tokenization, and handling missing values.

## NumPy

NumPy is a fundamental package for scientific computing in Python. It provides support for arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. In this project, NumPy was used for numerical operations such as creating and manipulating matrices for text vectors, implementing padding for input sequences, and preparing data for training and evaluation. NumPy's efficient handling of large data structures made it indispensable for the tasks of vectorization and mathematical computation in this study.

## Seaborn

Seaborn is a Python data visualization library based on Matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn was used in this project for visualizing the performance metrics of the model, including confusion matrices, precision-recall curves, and other relevant plots. It allowed for quick and aesthetically pleasing visualizations to better understand the data distribution, model performance, and evaluation metrics.

## Matplotlib

Matplotlib is a widely used Python plotting library that produces static, interactive, and animated visualizations. In this project, Matplotlib was used for creating various plots, including performance evaluation metrics, loss curves, and visualizations of the data distribution. Its flexibility and wide range of supported plot types allowed for detailed exploration of model performance and data characteristics.

## PyTorch

PyTorch is an open-source deep learning framework developed by Facebook’s AI Research lab. Known for its dynamic computation graph and ease of use, PyTorch was employed in this project for model building and training. PyTorch’s flexibility allowed for efficient implementation of the hybrid CNN-LSTM model and its components. In particular, it was used for constructing the layers, defining the forward pass, and managing the training loop.

## Scikit-learn (sklearn)

Scikit-learn is one of the most popular Python libraries for machine learning. It provides simple and efficient tools for data mining and data analysis, including various utilities for model evaluation and performance metrics. In this project, scikit-learn was used to compute various classification metrics, such as precision, recall, F1-score, and accuracy. It also provided functions for splitting the dataset into training, validation, and test sets.

# Methadology

The dataset used for this project consists of text samples labeled as either human-generated or AI-generated. The dataset was collected from publicly available sources, ensuring a balanced distribution between the two classes. Text samples ranged from 1 to 1700 words, ensuring a diverse representation of various lengths. We initially applied a filtering process to discard any texts outside this range, focusing only on those with a text length between 500 and 3000. This was done to ensure that the model could generalize well across typical text lengths while avoiding overfitting to shorter or excessively long samples.

The preprocessing steps involved multiple stages:

1. Text Cleaning: Checked for any unwanted characters, special symbols, and punctuation marks to standardize the input text and ensure uniformity.
2. Text Filtering: To create a well-defined dataset for training, I focused on texts that fell within a certain range. Specifically, filtered the dataset to include only texts with:
   1. Word Count: Between 150 and 500 words.
   2. Character Length: Between 500 and 3000 characters.

This filtering step was essential for maintaining a manageable range of text lengths, ensuring that the model could generalize well across typical text sizes. To determine the appropriate range for word count and character length, I first visualized the data distributions. By analyzing the distributions of word counts and character lengths, I was able to identify reasonable ranges and apply these constraints to the data. After filtering the dataset, only texts within these bounds were retained, ensuring that the input to the model was consistent and relevant.

1. Tokenization: Text samples were split into tokens (words) using the Tokenizer from Keras. This process converts raw text into a sequence of tokens that can be processed by the model.
2. Vectorization: Using Keras Tokenizer, we converted the sequence of tokens into integer sequences, where each word in the text is mapped to a unique integer index. This integer sequence was then used as input to the model.
3. Padding: To ensure that the input text sequences have consistent lengths, we applied zero-padding to each text sequence. This step is necessary for feeding the sequences into the neural network, where each sequence is padded to the median length.
4. Data Balancing: Initially, I analyzed the distribution of the classes in the dataset. It was found that there was an imbalance in the number of samples for human-generated and AI-generated texts. The imbalance can affect the model's ability to correctly classify the minority class.

I randomly selected a smaller subset of instances. This reduced the number of samples in the majority class, bringing the distribution closer to even.

After applying these balancing techniques, the dataset was re-checked for class balance before proceeding with model training. This ensured that the model had an equal opportunity to learn from both classes, improving its classification performance across all categories.

After all the preprocessing, the data was split into training and validation sets. The final dataset consisted of 17548 samples, with an even distribution of human and AI-generated texts. The training set consisted of 70% of the samples while the validation and testing set consisted of 15% of the samples.

# Model Architecture

The model developed for detecting AI-generated text is based on a hybrid architecture combining Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks. This combination leverages the strength of CNNs in feature extraction from the input text and the capability of LSTMs to capture long-term dependencies in the sequential nature of the text. Below is a detailed description of the model architecture.

## Input Layer

The input layer receives sequences of text represented as integer-encoded tokens. The input shape is (seq\_length), where seq\_length is the maximum sequence length of the text (set to 340). Each input sequence corresponds to a processed text sample.

## Embedding Layer

The embedding layer transforms the integer-encoded input text into dense vectors of fixed size (embedding\_dim = 340). This representation captures semantic relationships between words in the vocabulary and is fundamental for learning patterns in the input data. The embedding layer has an input dimension of vocab\_size, which represents the size of the vocabulary (the total number of unique words in the dataset).

Input Dimensions: vocab\_size

Output Dimensions: embedding\_dim = 340.

## Convolutional Layers

Convolutional layers are used to capture local features in the input text. The model contains two convolutional layers, each followed by a batch normalization and max-pooling layer to extract hierarchical features and reduce dimensionality.

First Convolutional Layer:

Filters: 128

Kernel Size: 5

Activation: ReLU

Batch Normalization: This layer normalizes the activations to speed up training and reduce internal covariate shifts.

Max-Pooling Layer: This layer reduces the sequence length by taking the maximum value from the pooled region, which helps in reducing computational complexity.

Second Convolutional Layer:

Filters: 256

Kernel Size: 3

Activation: ReLU

Batch Normalization & Max-Pooling: Similar to the previous layers, these layers ensure stable training and further reduce sequence length.

## LSTM Layers

LSTM layers are employed to capture long-term dependencies and sequential relationships in the text. The model includes two LSTM layers, where the first LSTM layer returns sequences to pass the processed information to the next layer, and the second LSTM layer outputs the final representation of the sequence.

First LSTM Layer:

Units: 128

Return Sequences: True

Dropout: 0.3

Recurrent Dropout: 0.3

Second LSTM Layer:

Units: 64

Return Sequences: False

Dropout: 0.3

Recurrent Dropout: 0.3

The LSTM layers allow the model to effectively process sequential data, maintaining the context of words and their relationships in the text.

## Fully Connected Layers

The dense layers refine the learned features and generate the final output. These layers help in transforming the sequential data into a classification decision.

First Dense Layer:

Units: 256

Activation: ReLU

Dropout Layer: Dropout rate of 50% is applied to prevent overfitting by randomly disabling some neurons during training.

Second Dense Layer:

Units: 128

Activation: ReLU

Dropout Layer: Another 50% dropout layer to further regularize the network.

Third Dense Layer:

Units: 64

Activation: ReLU

These fully connected layers ensure that the model learns high-level representations from the features extracted by the CNN and LSTM layers.

## Output Layers

The output layer is a fully connected layer with a single unit and a sigmoid activation function, making it suitable for binary classification. The model outputs a probability score between 0 and 1, where values closer to 1 indicate AI-generated text, and values closer to 0 represent human-generated text.

Activation: Sigmoid.

## Model Compilation

After constructing the model architecture, the following compilation steps were applied:

Optimizer: Adam, known for its efficiency in large-scale training.

Loss Function: Binary Cross-Entropy, as the task is a binary classification problem.

Metrics: Accuracy, to evaluate the overall performance of the model

##### A screenshot of a computer program Description automatically generated

##### Total params: 39,533,169 (150.81 MB)

##### Trainable params: 39,532,401 (150.80 MB)

##### Non-trainable params: 768 (3.00 KB)

# Model Performance

A graph with numbers and a line

Description automatically generated

## Accuracy graph:

The training accuracy starts at 72.7% in the first epoch and rapidly improves to 99.95% by the sixth epoch, indicating that the model learns quickly and fits the training data well.

Validation accuracy begins at a baseline of 49.8%, suggesting that the model initially struggles to generalize. By the 4th epoch, validation accuracy improves drastically to 98.21%. This sharp increase indicates that the model starts to generalize well on unseen data.

The highest validation accuracy of 99.66% is achieved in the 9th epoch, showing that the model effectively distinguishes between human-generated and AI-generated text.

The convergence of both training and validation accuracy lines after the 6th epoch suggests that the model avoids overfitting, maintaining strong generalization.

The steep increase in validation accuracy between epochs 3 and 4 shows that the model successfully generalizes after this point.

The training accuracy plateaus close to 100% early on, highlighting efficient learning.

A graph with numbers and lines

Description automatically generated

## Loss graph:

Training loss starts at 0.4558 in the first epoch and quickly drops to 0.0044 by the 10th epoch, showing a steady and consistent reduction in loss.

Validation loss decreases from 0.7376 in the first epoch to 0.0177 at the 10th epoch. The dramatic drop after the 4th epoch aligns with the accuracy improvement, indicating that the model successfully minimizes the error on unseen data.

The final test loss of 0.0134 and a test accuracy of 99.35% further confirm the model's outstanding performance in real-world evaluation scenarios.

Both training and validation losses decrease significantly, indicating that the model is converging and minimizing classification errors.

A minimal difference between training and validation loss demonstrates the absence of overfitting.

## Confusion Matrix:

A blue squares with white text

Description automatically generated

The confusion matrix demonstrates that the model makes very few errors, with only 17 misclassifications out of 2633 predictions.

## Performance Metrics:

precision recall f1-score support

Human 0.99 1.00 0.99 1346

Generated 1.00 0.99 0.99 1287

accuracy 0.99 2633

macro avg 0.99 0.99 0.99 2633

weighted avg 0.99 0.99 0.99 2633

* The model achieves 99.5% precision, meaning very few false positives occur.
* The recall of 99.1% indicates that the model misses very few AI-generated texts.
* With an overall F1-Score of 99.3%, the model demonstrates excellent classification performance.

# Key Observations

High Accuracy: The overall accuracy of 99.35% reflects the model's strong capability to differentiate between human-generated and AI-generated texts.

Class Balance: The performance is consistent across both classes, as reflected by the near-identical precision, recall, and F1-scores.

Low Misclassification: The small number of false positives (6) and false negatives (11) shows the model's reliability and minimal error rate.

# Conclusion

The CNN-LSTM hybrid model achieves outstanding results with a 99.35% accuracy, high precision, recall, and F1-scores across both classes. The confusion matrix confirms that misclassifications are rare, making this model a robust solution for detecting AI-generated text.

##### References

Sunagar, P., Sowmya, B.J., Pruthviraja, D. et al. Hybrid RNN Based Text Classification Model for Unstructured Data. SN COMPUT. SCI. 5, 726 (2024). <https://doi.org/10.1007/s42979-024-03091-x>

CS231N convolutional neural networks for visual recognition,

<https://cs231n.github.io/convolutional-networks/#overview>

Y. Kim, “Convolutional neural networks for sentence classification,” ACL Anthology,

https://aclanthology.org/D14-1181/ (accessed Dec. 13, 2024).

X. She and D. Zhang, "Text Classification Based on Hybrid CNN-LSTM Hybrid Model," 2018 11th International Symposium on Computational Intelligence and Design (ISCID), Hangzhou, China, 2018, pp. 185-189, doi: 10.1109/ISCID.2018.10144. keywords: {Feature extraction;Text categorization;Logic gates;Convolutional neural networks;Training;Classification algorithms;text classification;word2vec;convolutional neural network;long short-term memory network;hybrid model},

“RNN vs. CNN: Understanding key differences in text classification,” ARTiBA, https://www.artiba.org/blog/rnn-vs-cnn-understanding-key-differences-in-text-classification (accessed Dec. 13, 2024).