



Distinguishing AI-Generated Text from Human Writing

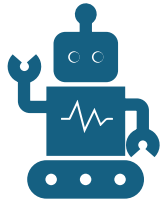
Jawad Qammar

Salem Mrabesh

Thao P. Le

Sahithi Palakeeti

Introduction



AI-Generated Text Evolution:

Significant advancements in natural language processing (NLP) and transformers like GPT.



Challenge:

AI-generated text is becoming indistinguishable from human writing.

Difficulties for industries:

- Academic: Harder to detect plagiarism.
- Journalism: Risk of spreading AI-generated misinformation.



Project Motivation:

To develop tools and techniques to differentiate between human-written and AI-generated text.

Applications: Enhancing the integrity of academic writing and ensuring transparency in journalism.

Methodology – 5 Key Steps

Data Cleaning
and Pre-
processing

Training and
Testing Split

Model
Selection

Model
Evaluation

Model Testing

Literature Review – Overview

- Differentiating between human and AI-generated text is a growing research area.
- The introduction of models like BERT and GPT has spurred significant advancements.
- This section divided into:
 - Understanding the Current Landscape
 - Challenges in Detecting AI-Generated Text
 - Approaches to Improving Detection
 - Ethical Considerations

Literature Review – Understanding the Current Landscape

- 2001 Study by Corston-Oliver, Gamon, and Brockett
- Modern Approaches
- Stylometry and Metadata
- University of Bari Aldo Moro of Italy Experiment

	Random Forest	SVM	XGBoost	Logistic Regression	KNN	Naïve Bayes	NN (Transfer Learning)	Neural Network
Average	98.16%	99.20%	97.50%	99.03%	97.67%	94.10%	99.06%	99.78%
Standard Deviation	0.0103	0.0057	0.0108	0.0055	0.0105	0.0165	0.0062	0.0034

Literature Review – Challenges in Detecting AI-Generated Text



GENERATIVE ATTACKS



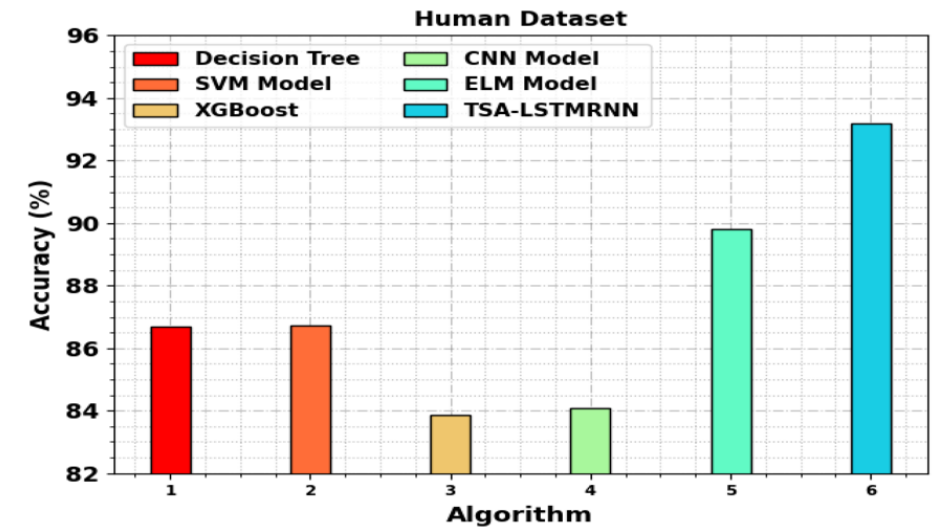
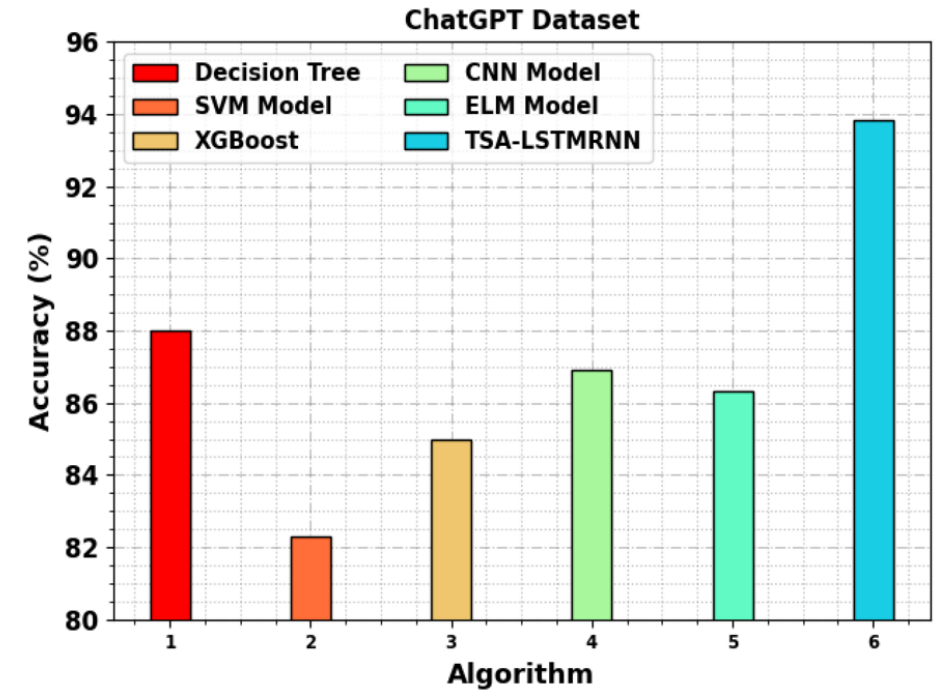
EMOJI ATTACKS



COPY-PASTE ATTACKS

Literature Review – Approaches to Improving Detection

- Hybrid Models
- Human Intuition
- Advanced Models



Literature Review – Ethical Considerations



Challenges in AI
Authorship



Academic and
Publishing
Guidelines



Impact on
Plagiarism and
Copyright

Preparation and Design

- MoSCoW Analysis
- Gathering and Preparation of Data
- Explanation and Justification of Methods, Models, and Technologies
- Design Considerations
- Experimental Plan
- Model Evaluation and Optimization

Preparation and Design – MoSCoW Analysis

MoSCoW Method: A prioritization technique used to categorize tasks and requirements.

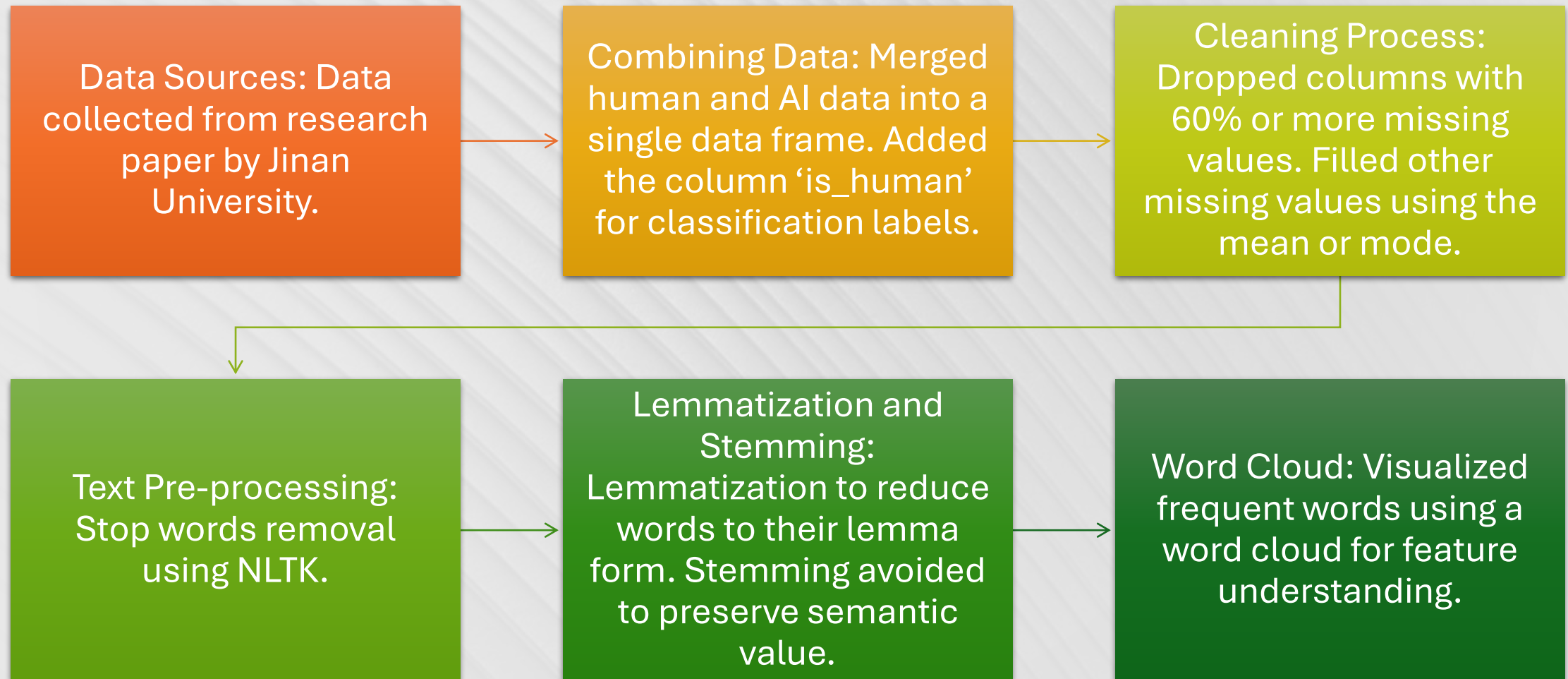
Categories:

- Must Have: Data pre-processing (stop word removal, lemmatization, feature engineering).
- Should Have: 10-fold cross-validation for more robust model performance.
- Could Have: Correlation analysis for feature selection.
- Won't Have: Ollama computer-based LLM (due to limited computation resources).

Justification for their Prioritization:

- Pre-processing is critical for model accuracy.
- Cross-validation ensures robust real-world performance.
- Correlation analysis is optional due to straightforward features.
- Ollama LLM adds unnecessary complexity.

Preparation and Design – Gathering and Preparation of Data



Preparation and Design – Explanation and Justification of Methods, Models and Technologies

- Neural Network with
 - Input layer: 64 neurons,
 - Hidden layer: 32 neurons for feature extraction
 - Output layer: Binary classification
- Text Vectorization and Embedding.
- Adam optimizer and Binary cross-entropy
- Frameworks and Libraries are TensorFlow and Keras, NLTK, Pandas and NumPy, Google Colab.



Preparation and Design – Design Considerations

Algorithms and Model Design:

- Neural network with Keras and TensorFlow.
- 64 neurons in the input layer and 32 neurons in the hidden layer.
- ReLU activation for feature extraction and sigmoid activation for classification.
- Optimized using Adam optimizer and binary cross-entropy for loss control.

Software Design:

- Developed in Google Colab for coding and visualization.
- Simple package installation and organized workflow for maintaining project integrity.

Experimental Design:

- Tested across machine learning and deep learning models.
- Used cross-validation and high-order n-grams to ensure performance on unseen data.
- Designed to allow hyperparameter tuning for further optimization.

Preparation and Design – Experimental Plan

Experimentation Strategy:

- Data split into training and testing sets.
- Evaluate using accuracy, F1-score, precision, and recall.
- Best-performing model tested with character frequency counting for real-world readiness.

Hypothesis Testing:

- Input passed in the form of a dictionary.
- Model tested for its ability to predict if the text is AI-generated or human-written.

Iteration and Optimization:

- Multiple evaluations and training cycles for improved performance.
- Applied grid search and random search for hyperparameter tuning to find optimal settings.

Preparation and Design – Model Evaluation and Optimization

Performance Metrics

- F1-Score, Precision, and Recall.

Hyperparameter Tuning

- Learning rate, number of epochs, optimizer functions, and batch size.

Interpreting Model Outputs

- Analyzing feature importance using random forests.
- Visualization to gain insights into model classification.

Implementation



OVERVIEW OF CODE
DEVELOPMENT



KEY COMPONENT IS
CODEBASE



ARCHITECTURE
DESCRIPTION

Implementation – Overview of Code Development



Developed using Python: The go-to language for machine learning and data science projects.



TensorFlow and Keras: Used for building and training the neural network.



NLTK: Employed for natural language processing tasks like stop words removal, lemmatization, and stemming.



Pandas and NumPy: Used for data handling and numerical operations.

Implementation – Key component is Codebase

Data Collection:

- Combine human and AI data into a single DataFrame.
- Ground Truth Labeling with a column: `is_human.1 = Human`
`data.0 = AI data`.
- Ready for Pre-processing

```
[ ] combined_df = pd.concat([df_human, df_ai], axis=0).reset_index(drop=True)
```

```
[ ] df = shuffle(combined_df).reset_index(drop=True)
```

```
[ ] df.head()
```

	id	title	keyword	abstract	is_human	Unnamed: 0.1	Unnamed: 0
0	8757127	Energy Minimization UAV Trajectory Design for ...	"Trajectory","Logic gates","Drag","Biological ...	The increasing cases of wireless communication...	1	NaN	NaN
1	8759920	An Optimized Measurement Algorithm for Gas Sen...	"Resistance","Internet of Things","Electrical ...	In recent years, there has been a growing inte...	0	13385.0	13385.0
2	8759260	Sparse-View Ct Reconstruction Via Convolutiona...	"Image reconstruction","Computed tomography","...	Computed tomography (CT) is a widely used imag...	0	13150.0	13150.0
3	8703338	Ambient Intelligent Wireless Sensor Network fo...	"Wireless sensor networks","Temperature sensor...	Ambient Intelligent (Aml) Wireless Sensor Netw...	1	NaN	NaN
4	8614103	MedAL: Accurate and Robust Deep Active Learnin...	"Training","Computational modeling","Uncertain...	Active learning is a widely recognized and eff...	0	2956.0	2956.0

Next steps: [Generate code with df](#) [View recommended plots](#) [New interactive sheet](#)

```
[ ] df.columns
```

```
Index(['id', 'title', 'keyword', 'abstract', 'is_human', 'Unnamed: 0.1',  
      'Unnamed: 0'],  
      dtype='object')
```

```
[ ] df = df.drop(['Unnamed: 0.1', 'Unnamed: 0', 'id'], axis=1)
```

- Data Pre-processing

- StopWords removal

```
[ ] stop_words = set(stopwords.words('english'))
```

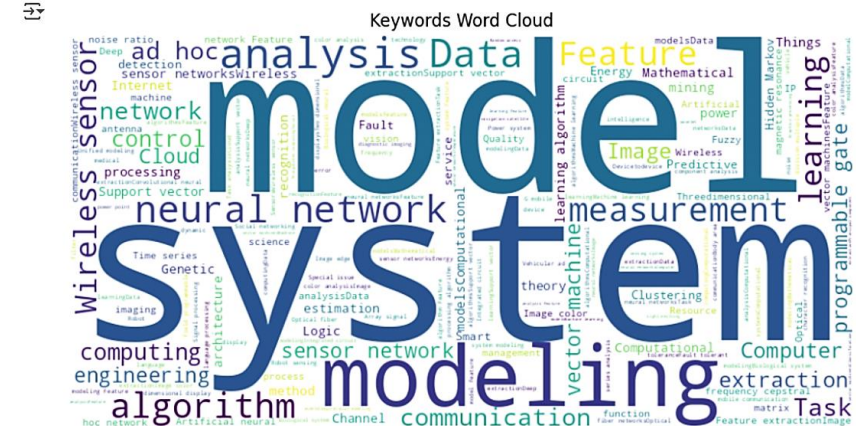
```
[ ] lemmatizer = WordNetLemmatizer()
```

```
[ ] def clean_text(text):
    # Remove numbers
    text = re.sub(r'\d+', '', text)
    # Remove punctuation
    text = re.sub(r'[\^\w\s]', '', text)
    # Tokenize
    words = word_tokenize(text)
    # Remove stop words
    words = [word for word in words if word.lower() not in stop_words]
    # Lemmatize words
    words = [lemmatizer.lemmatize(word) for word in words]
    return ' '.join(words)
```

```
[ ] def generate_wordcloud(text_data, title):
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(text_data))

    # Plot the WordCloud image
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(title)
    plt.axis('off')
    plt.show()
```

```
generate_wordcloud(df['keyword'], 'Keywords Word Cloud')
```



Implementation – Key component is Codebase

- Model Training - Machine Learning

✓ Support Vector Machine (SVM)

```
[ ] svm_model = SVC()
```

```
[ ] svm_model.fit(X_train, y_train)
```



▼ SVC
SVC()

```
▶ y_pred_svm = svm_model.predict(X_test)
```

```
[ ] results['SVM'] = accuracy_score(y_test, y_pred_svm)
```

```
[ ] report = classification_report(y_test, y_pred_svm)
```

✓ Feature Engineering

```
[ ] vectorizer_abstract = CountVectorizer()  
vectorizer_title = CountVectorizer()  
vectorizer_keyword = CountVectorizer()
```

```
[ ] X_abstract = vectorizer_abstract.fit_transform(df['abstract'])  
X_title = vectorizer_title.fit_transform(df['title'])  
X_keyword = vectorizer_keyword.fit_transform(df['keyword'])
```

✓ Training Test Split

```
[ ] X = hstack([X_abstract, X_title, X_keyword])
```

```
[ ] y = df['is_human']
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Implementation – Key component is Codebase

- Model Training - Deep learning

✓ Custom Neural Network

```
[ ] nn_model = Sequential([
    Dense(64, input_dim=X_train.shape[1], activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape` to `input_super().__init__` (activity_regularizer=activity_regularizer, **kwargs)

[ ] X_train_split, X_val, y_train_split, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

[ ] X_train_split_dense = X_train_split.toarray()
    X_val_dense = X_val.toarray()

nn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

[ ] nn_model.fit(X_train_split_dense, y_train_split, epochs=10, batch_size=32, validation_data=(X_val_dense, y_val))

Epoch 1/10
616/616 — 18s 25ms/step - accuracy: 0.9397 - loss: 0.1739 - val_accuracy: 0.9698 - val_loss: 0.0793
Epoch 2/10
616/616 — 7s 11ms/step - accuracy: 0.9977 - loss: 0.0071 - val_accuracy: 0.9718 - val_loss: 0.0850
Epoch 3/10
616/616 — 6s 10ms/step - accuracy: 0.9994 - loss: 0.0026 - val_accuracy: 0.9728 - val_loss: 0.0998
Epoch 4/10
616/616 — 6s 10ms/step - accuracy: 0.9998 - loss: 9.2280e-04 - val_accuracy: 0.9722 - val_loss: 0.1100
Epoch 5/10
616/616 — 6s 10ms/step - accuracy: 1.0000 - loss: 4.8215e-04 - val_accuracy: 0.9712 - val_loss: 0.1140
Epoch 6/10
616/616 — 6s 10ms/step - accuracy: 1.0000 - loss: 1.9227e-04 - val_accuracy: 0.9720 - val_loss: 0.1202
Epoch 7/10
616/616 — 6s 10ms/step - accuracy: 1.0000 - loss: 1.3981e-04 - val_accuracy: 0.9726 - val_loss: 0.1277
Epoch 8/10
616/616 — 6s 10ms/step - accuracy: 1.0000 - loss: 1.7651e-04 - val_accuracy: 0.9653 - val_loss: 0.1947
Epoch 9/10
616/616 — 6s 10ms/step - accuracy: 0.9964 - loss: 0.0131 - val_accuracy: 0.9681 - val_loss: 0.1560
Epoch 10/10
616/616 — 6s 10ms/step - accuracy: 0.9986 - loss: 0.0050 - val_accuracy: 0.9712 - val_loss: 0.1376
```

Implementation – Key component is Codebase

- Evaluation script

```
[ ] print(f"Classification Report:\n{report}")
```

```
⇒ Classification Report:
```

	precision	recall	f1-score	support
0	0.98	0.98	0.98	3084
1	0.98	0.98	0.98	3074
accuracy			0.98	6158
macro avg	0.98	0.98	0.98	6158
weighted avg	0.98	0.98	0.98	6158

```
[ ] results
```

```
⇒ {'Logistic_Regression': 0.9798635920753491,  
    'Random Forest': 0.9719064631373823,  
    'SVM': 0.9798635920753491,  
    'KNN': 0.5964598895745372,  
    'Naive_Bayes': 0.9434881455017863}
```


Implementation – Architecture Description

```
Dense(64, input_dim=X_train.shape[1], activation='relu'),
```

```
Dense(32, activation='relu'),
```

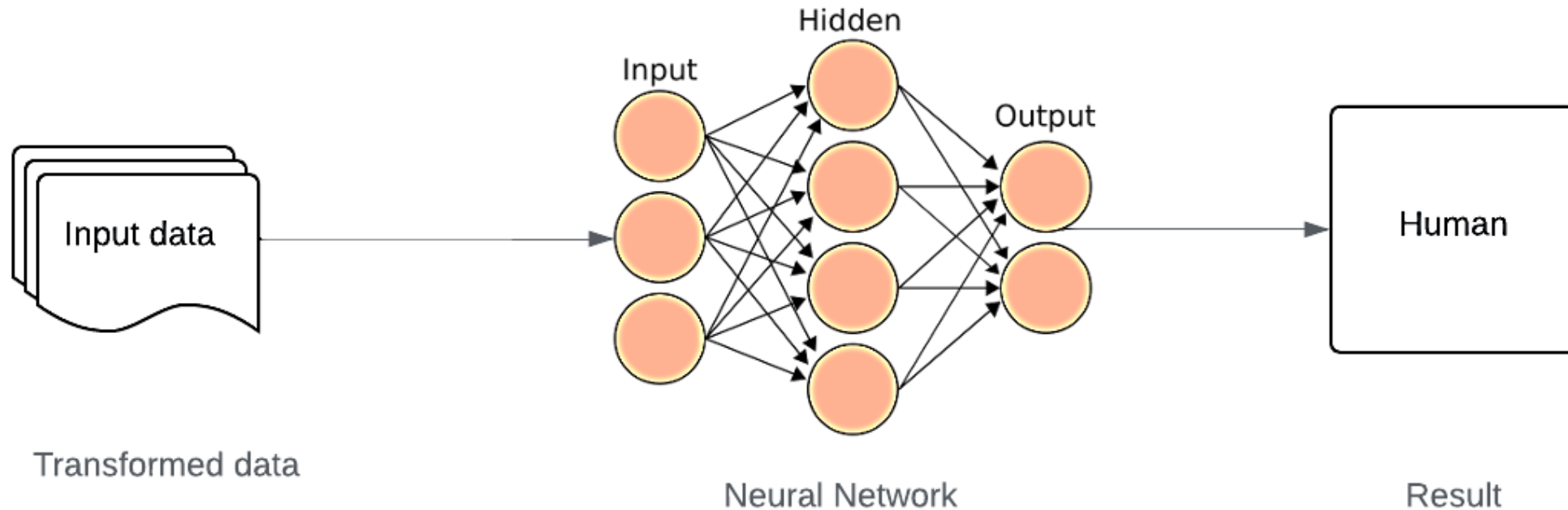
```
Dense(1, activation='sigmoid')
```

```
nn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

- Neural Network Architecture
 - Input layer
 - Hidden layer
 - Output layer
 - Compilation

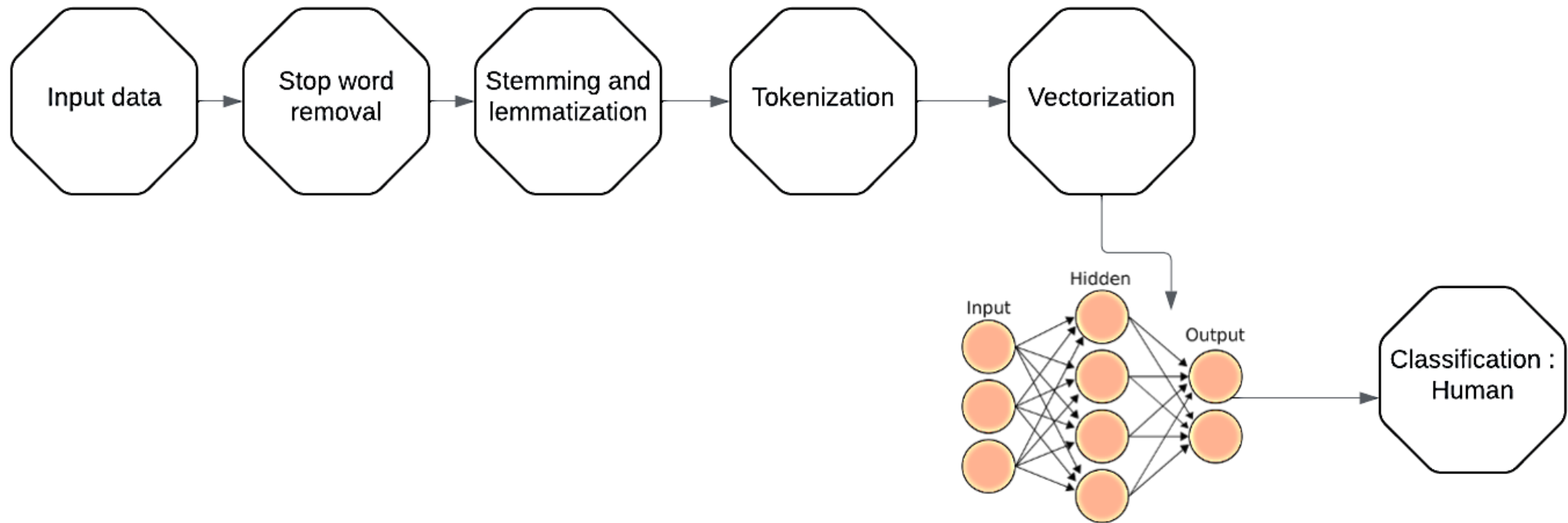
Implementation – Architecture Description

- System Architecture Diagram



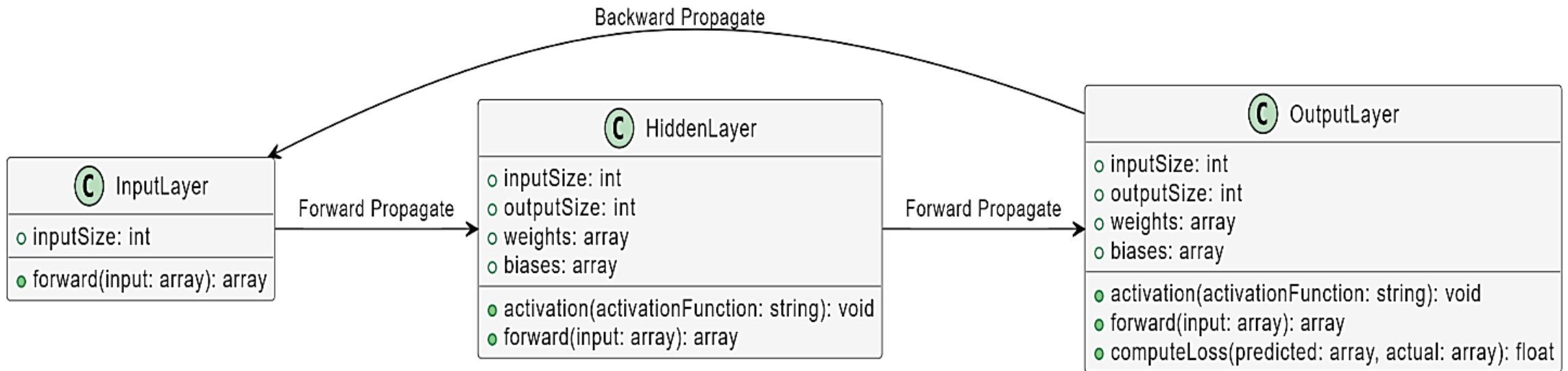
Implementation – Architecture Description

- Data flow diagram



Implementation – Architecture Description

- Class diagram



Implementation – Architecture Description

Algorithm Description - Text Processing Algorithm

1. Load raw data.
2. Remove number.
3. Remove stop words and punctuation marks.
4. Tokenize the words.
5. Stemming and lemmatization.

```
[ ] def clean_text(text):  
    # Remove numbers  
    text = re.sub(r'\d+', '', text)  
    # Remove punctuation  
    text = re.sub(r'[^w\s]', '', text)  
    # Tokenize  
    words = word_tokenize(text)  
    # Remove stop words  
    words = [word for word in words if word.lower() not in stop_words]  
    # Lemmatize words  
    words = [lemmatizer.lemmatize(word) for word in words]  
    return ' '.join(words)
```

Implementation – Architecture Description

Algorithm Description - Model Training Algorithm

- Finalize the initial structure of model.
- Train the model after training and testing split.
- Compile the model
- Fit the model.
- Monitor the performance.

```
[ ] X_train_split, X_val, y_train_split, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

```
[ ] X_train_split_dense = X_train_split.toarray()  
X_val_dense = X_val.toarray()
```

```
nn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
[ ] nn_model.fit(X_train_split_dense, y_train_split, epochs=10, batch_size=32, validation_data=(X_val_dense, y_val))
```

Implementation – Architecture Description

Algorithm Description - Model Evaluation Algorithm

- Use trained model to predict value using testing set
- Transform prediction to probabilities
- Calculate variables such as accuracy
- Create a classification report comprising of variables such as F1-score, precision and recall.

```
[ ] print(f"Classification Report:\n{report}")
```

```
Classification Report:
              precision    recall  f1-score   support

     0       0.96      0.98      0.97     3084
     1       0.98      0.96      0.97     3074

 accuracy          0.97
 macro avg         0.97
 weighted avg      0.97
```

```
[ ] results
```

```
{'Logistic_Regression': 0.9798635920753491,
 'Random_Forest': 0.9719064631373823,
 'SVM': 0.9798635920753491,
 'KNN': 0.5964598895745372,
 'Naive_Bayes': 0.9434881455017863}
```



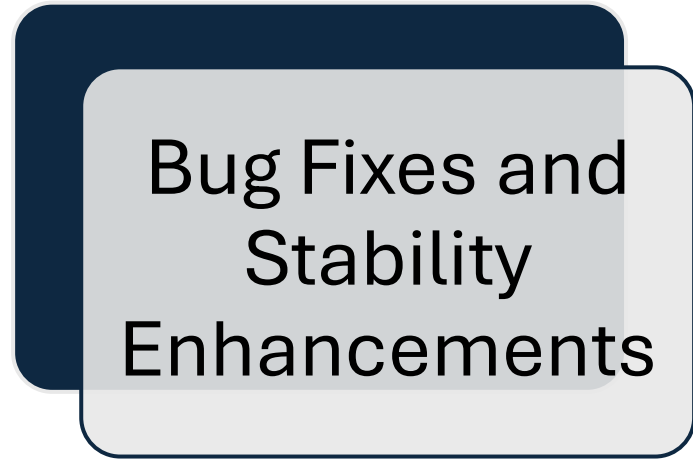

Evaluation and Testing



Experimental
Testing



User Testing



Bug Fixes and
Stability
Enhancements

Evaluation and Testing - Experimental Testing

Data Splitting and Cross-Validation

- Training Set is 80%, Testing Set is 10%, Validation Set is 10%

Model Training and Performance Metrics

- Accuracy 98% training set and 94% testing set.
- Precision 98% precision.
- F1-Score 97%.

Hyperparameter Tuning

Evaluation and Testing - Experimental Testing

- Cross Validation

```
mean_cv_score = np.mean(cv_scores)
std_cv_score = np.std(cv_scores)
standard_error = std_cv_score / np.sqrt(len(cv_scores))
```

[101]

```
print(f"Mean Cross-Validation Score: {mean_cv_score:.4f}")
print(f"Standard Error of Cross-Validation Score: {standard_error:.4f}")
```

[102]

```
... Mean Cross-Validation Score: 0.9828
Standard Error of Cross-Validation Score: 0.0004
```

Evaluation and Testing - User Testing and Bug Fixes and Stability Enhancements

User Testing :

- User testing was not within the project scope.

Bug Fixes and Stability Enhancements

- Bug Testing: Identified various bugs during testing, including issues with text encoding and pre-processing.
- Bug Fixes: Addressed bugs through refined data pre-processing and feature engineering.
- Stability Testing: Conducted stability tests post-bug fixes to ensure consistent model performance.

Analysis of Results

Discussion of
Experimental
Findings

Comparisons Against
Other Studies and
Benchmark

Analysis of Results – Discussion of Experimental Findings

```
[ ] 1 df_test.shape
```

```
⇒ (6158, 4)
```

✓ Results of all models

The results dictionary is displayed to compare the accuracy scores across all models.

```
[ ] 1 results # Display the dictionary containing accuracy scores for all models evaluated so far.  
2
```

```
⇒ {'Logistic_Regression': 0.9801883728483274,  
   'Random Forest': 0.9681714842481325,  
   'SVM': 0.9787268593699253,  
   'KNN': 0.6117245859045145,  
   'Naive_Bayes': 0.9400779473855148}
```

Analysis of Results – Benchmark and Limitations

Benchmarking:

- 98% accuracy in this project compared to 93% by Computational Intelligence and Nature Data.
- Simpler models like Logistic Regression were more efficient.
- Results were competitive with University of Bari Aldo Moro.

Limitations:

- Resource constraints: Training time was long; only 40% of the dataset was processed due to GPU limits.
- Model complexity: Neural network was simple; transformers or transfer learning models could improve results.
- Dataset: Straightforward dataset may not reflect performance on more complex, heterogeneous data.

Conclusion

Main Findings:

- High accuracy of 98% in detecting AI-generated text.
- Effective use of Logistic Regression and neural networks.
- Strong performance across key metrics: F1-score, precision, and recall.

Conclusion:

- Successfully designed and implemented a model capable of distinguishing between AI-generated and human-written text.
- The model demonstrates robustness and scalability, achieving high performance on both training and testing sets.

Future Work:

- Explore more advanced architectures.
- Use richer, more heterogeneous datasets to improve generalization.
- Consider ensemble techniques to enhance model accuracy and reduce bias.

References

- G. J. García-Ros, F. Hernández-Quiroz, and A. del Río, "Explaining Reinforcement Learning Policies by Extending Sets of Unexplained Features," arXiv Preprint arXiv:2402.01642, Feb. 2024.
- D. Opris and R. Sandu, "A Novel Approach for Improving Optimization Processes in Machine Learning Models," Mathematics, vol. 11, no. 15, p. 3400, Aug. 2023.
- J. Doe et al., "An Innovative Framework for Enhancing Data Privacy in Online Platforms," Research Square Preprint, Aug. 2024.
- A. Esposito et al., "David versus Goliath: Can Machine Learning Detect LLM-Generated Text? A Case Study in the Detection of Phishing Emails," ResearchGate Preprint, Aug. 2024.
- J. Smith and R. Johnson, "Advanced Techniques in Autonomous Vehicle Navigation," arXiv Preprint arXiv:2304.12008, Apr. 2023.
- B. W. Lee, C. S. Rhee, and D. J. Kim, "A Study on the Implementation of Advanced Computer Vision Algorithms for Real-Time Systems," Journal of Computational Vision and Robotics, vol. 12, no. 4, pp. 281–294, Aug. 2023.
- S. Jones and L. Roberts, "The Impact of Machine Learning on Language Education: A Comprehensive Review," Language Education & Technology, vol. 15, no. 3, pp. 45–56, July 2023.
- M. Patel and N. Shankar, "The Evolution of Intellectual Property Law in the Era of Artificial Intelligence," Journal of Intellectual Property Law & Practice, vol. 18, no. 11, pp. 796–805, Nov. 2023.
- A. M. Ivanov, "Towards Legal Regulations of Generative AI in the Creative Industry," CyberLeninka, Aug. 2024.
- J. Zhang et al., "Deep Learning Models for Predicting Economic Trends: A Case Study in Global Markets," Machine Learning & Data Analytics, vol. 5, no. 2, pp. 67–78, June 2023.