

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

SC4002 Natural Language Processing
Group 56

Name	Matriculation Number
Bong Jia Hui	U2121174D
Celine Tan	U2120113L
Chong Huai Zhi	U2221000J
Hung Kuo Chen	U2120045A
Rhea Kenneth	U2120878K

Contributions

Group Member	Contributions
Bong Jia Hui	<ul style="list-style-type: none">• Implementation of CNN model• Fine-tuned BERT• Report write-up
Celine Tan	<ul style="list-style-type: none">• Initialization of word embedding matrix• Implementation of RNN model• Report write-up
Chong Huai Zhi	<ul style="list-style-type: none">• Implementation of preprocessing• Implementation of fastText• Report write-up
Hung Kuo Chen	<ul style="list-style-type: none">• Implementation of biGRU model• Research on final enhancement strategy• Report write-up
Rhea Kenneth	<ul style="list-style-type: none">• Implementation of biLSTM model• Implementation of ensemble model• Report write-up

Content Page

Contributions	1
Content Page	2
Introduction	3
1. Preparing Word Embeddings	3
1.1 Data Preprocessing	3
1.2 Initialization of Word Embedding Matrix using GloVe	3
1.3 Dealing with OOV words in Glove	4
2. Model Training & Evaluation - RNN	6
2.1 Transformation of Input Data	6
2.2 Final Configuration of Best Model	6
2.3 Accuracy Score of Validation and Test Set	7
2.4 Methods to Derive Final Sentence Representation to Perform Sentiment Classification	7
2.4.1 Last Hidden State	7
2.4.2 Global Max Pooling	7
2.4.3 Global Average Pooling	8
2.4.4 Concatenation of Max and Average Pooling	8
3. Enhancement	9
3.1 Accuracy Score of Simple RNN Model with Updated Word Embedding	9
3.2 Accuracy Score of RNN Model with Handling of OOV Words	9
3.3 Accuracy Scores of biLSTM and biGRU on test set	9
3.4 Accuracy Scores of CNN on test set	9
3.5 Final Improvement Strategy	10
3.6 Comparison of Solutions and Discussion	10
4. References	12

Introduction

This project aims to develop a robust **sentiment classification** model to determine whether each movie review expresses a positive or negative sentiment. Accurately classifying sentiments in textual data like movie reviews is crucial in understanding audience reactions and opinions, and it has applications in recommendation systems, marketing analysis, and customer feedback interpretation.

The movie review dataset used in this project consists of 8,530 reviews for training, 1,066 for validation, and 1,066 for testing. To achieve an accurate and high-performing sentiment classification model, we follow a structured methodology. Section 2 covers data preprocessing and the preparation of word embeddings. In Section 3, we introduce our baseline Recurrent Neural Network (RNN) model, outlining its structure, initial performance results, and the optimal configurations after fine-tuning hyperparameters. Building on this baseline, Section 4 explores various enhancement techniques to improve the model's accuracy and robustness, including out-of-vocabulary (OOV) handling, advanced architectures, and transfer learning.

1. Preparing Word Embeddings

1.1 Data Preprocessing

We used the **Contractions Library** to convert words with apostrophes into separate words. This is because we intend to remove most punctuations later on. For example, “don’t” will be converted to “do not”.

```
text = contractions.fix(text)
```

However, a word like “Wendy’s” will not be processed because it refers to a subject. Thus, we also removed the substring “ ’s ” from the text data. Firstly, this is because “ ’s ” gets tokenised often; secondly, a word like “Alex’s” might get picked up, but we would like to have “Alex” instead as the subject word itself is more meaningful.

```
text = text.replace("'s", "")
```

Next, we **removed all punctuations except for hyphens** and replaced them with a single space. We removed apostrophes as well due to the occurrence of words enclosed in quotes.

```
# Example
sample_text = "Hello, world! This isn't just any test-string."
result = replace_punctuation_with_space_except_hyphen(sample_text)
print(result)
```

```
→ Hello world This isn t just any test-string
```

Lastly, we removed hyphens only when they are not surrounded by words.

```
result = re.sub(r'(?!\w)-|-(?!\\w)', ' ', text)
```

1.(a) Ans: As a result, the size of the vocabulary formed from our training data is 17761.

1.2 Initialization of Word Embedding Matrix using GloVe

Firstly, we stored our vocab in a dictionary with its respective index while index 0 is reserved for ‘<PAD>’ and index 1 is reserved for ‘<UNK>’.

```
[ ] vocab_index
{ '<PAD>': 0,
  '<UNK>': 1,
  'encompassing': 2,
  'strangers': 3,
  'kuras': 4,
  'structured': 5,
  'neglects': 6,
  'dorm': 7,
  '15th': 8,
```

We then utilized ‘**glove.6B.100.txt**’ to create the word embedding matrix using the function `embedding_matrix_for_vocab()`. The parameters are the file path for the glove pre-trained embedding, the vocabulary index for our dataset, as well as the embedding dimensions, which we will set to 100.

1.(b) Ans: The number of out-of-vocabulary (OOV) words is 1607.

1.3 Dealing with OOV words in Glove

After matching the words in our training dataset with the pre-trained GloVe embeddings, we proceed to determine the threshold (k) to handle OOV words that occur less frequently. We analyzed the occurrences of each OOV word and observed that 1571 OOV words occur fewer than 3 times. Thus we set **k = 3** and **replaced them with ‘<UNK>’ token**, resulting in the embedding matrix covering 91.15% of the vocabulary.

```
Coverage with k=2: 91.55%
<UNK> tokens count with = 1500
Coverage with k=3: 91.15%
<UNK> tokens count with = 1571
Coverage with k=4: 91.04%
<UNK> tokens count with = 1592
Coverage with k=5: 91.00%
<UNK> tokens count with = 1598
Coverage with k=6: 90.98%
<UNK> tokens count with = 1602
Coverage with k=7: 90.97%
<UNK> tokens count with = 1604
```

The motivation for removing these OOV words and replacing them with the ‘<UNK>’ token is to reduce noise as low-frequency words often add noise to the model during training. By consolidating these rare words we can simplify the word vector representation allowing the model to focus on more informative vocabulary. In addition to this, training meaningful embeddings for infrequent words is challenging as there is not enough data to learn their contextual meaning adequately. Thus, we pooled these words into the ‘<UNK>’ token to train its embedding with contributions from all these rare words, resulting in a more robust representation for handling OOV words during validation and testing.

The remaining relatively **more frequent OOV words were handled using a word embedding technique called FastText**. One of the key features of fastText word representation is its ability to generate vectors for any words, even made-up ones. Indeed, FastText word vectors are **built from character-level substrings vectors of substrings of characters** contained in it. This allows us to build vectors even for misspelled words or concatenation of words.

Considering the word “equal” and $n = 3$, it can be represented by character n-grams: `< eq, equ, qua, ual, al >` and `<equal>`. The word embedding for the word ‘equal’ is then computed as the sum of all vector representations of all its character n-gram and the word itself (Krithika, 2024).

By default, an unknown word is represented as a non-zero vector as shown in the picture below:

[illegible]

```
[ ] print(unsupervised_model.get_word_vector('kitty'))
```

Σ	[-0.05825267	0.26602918	-0.02634677	-0.14058943	0.00676529	0.20693257
	-0.18731262	0.08494161	0.23886073	0.08692317	-0.14115828	0.18758558
	-0.02531155	0.13131389	-0.03649655	-0.12330469	-0.03110471	-0.19174575
	0.08427736	0.04570899	0.17649965	0.39849555	0.15515202	-0.04800915
	-0.15559211	-0.03676648	0.26383626	-0.00285927	-0.14427617	0.05855886
	0.02931981	0.02285934	-0.25070798	-0.12057504	-0.13609374	0.1167496
	-0.02682766	-0.09348433	-0.05731444	0.10181073	-0.03462359	0.0280825
	-0.03984969	0.03600628	0.07600105	-0.0822423	0.07496975	-0.08192611
	-0.00507815	0.21322541	-0.27374524	0.13395123	-0.1773068	-0.05174491
	-0.06063775	-0.11374203	-0.23548594	0.14994724	0.19152151	0.14694316
	-0.04019802	0.11363609	0.2587639	-0.000971539	-0.05771804	0.09764622
	-0.23713805	-0.07806714	-0.00788339	-0.09462493	-0.09035559	0.14218572
	0.07670069	0.01796282	-0.04535525	0.02252533	0.13565251	0.2180725
	0.10603103	-0.06223871	-0.04019866	0.03812748	0.20568773	-0.05159007
	0.08823189	0.01495754	0.05110271	0.05365983	-0.16497928	0.00797845
	0.13107328	0.03119123	0.2716093	-0.18273054	-0.08941116	-0.16262981
	-0.02516318	0.13128364	-0.14601259	-0.10497434		

```
# Training the model (unsupervised)
```

```
MIN_COUNT = 3
excluded = [word for word, count in vocab_counts.items() if word in oov_words and count < MIN_COUNT]

# create new word index to remove OOV with occurrence less than MIN_COUNT
new_vocab_list = list(set(vocab_list) - set(excluded))
new_vocab_index = {
    "PAD": 0,
    "UNK": 1 }
new_vocab_index.update({vocab: idx+2 for idx, vocab in enumerate(new_vocab_list)})
```

1(c) The best strategy to mitigate the OOV limitation on GloVe is to replace OOV words that occur fewer than $k = 3$ times while representing the remaining OOV words using FastText embeddings. This approach ensures that OOV words are effectively handled using its character-level information.

2. Model Training & Evaluation - RNN

2.1 Transformation of Input Data

It is essential to transform the movie reviews from their original string format into a suitable input format for the model. One of the important steps in data transformation is to select a desirable input length for the model training. The minimum length of the movie review in the training dataset is 1 while the maximum length is 53. We plotted a histogram, showing the distribution of review lengths in the training datasets to determine the appropriate length.

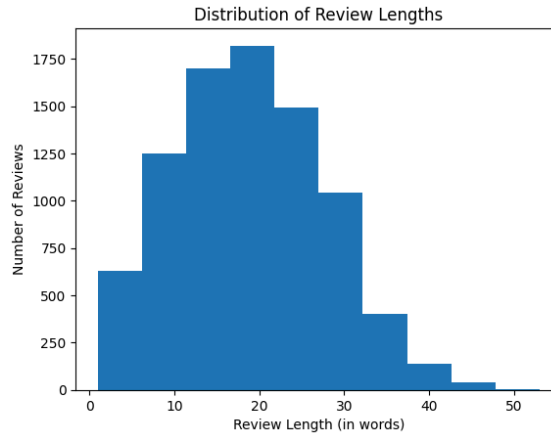


Fig 1. Histogram Showing Distribution of Review Lengths in Training Dataset

The histogram shows that most of the reviews have a length of between 15 and 25 words. To ensure we cover the majority of the full reviews we decided to use the 90th percentile of the review lengths, which turned out to be **31 words, as the input length**.

With the fixed maximum input length, we used the function, **pad_sequences()**, from TensorFlow to maintain the fixed length of 31 words by either truncating longer sentences or padding shorter sentences. This is followed by tokenizing the sentences and representing them at the word level using the respective index for each vocabulary stored in the `vocab_index` dictionary.

2.2 Final Configuration of Best Model

To determine the optimal configuration for the RNN model, we experimented with **different permutations of optimizers (Adam, SGD, RMSprop), learning rates (0.001, 0.005, 0.01) and batch sizes (16, 32, 64)**. We trained each configuration for 50 epochs with **early stopping, using patience of 5**, to determine the best configuration for different final sentence representations using validation accuracy. Early stopping is used to prevent the model from overfitting and save computational costs by halting the training once the model's performance on the validation dataset starts to degrade.

After obtaining the optimal configuration for the respective model, we focused on finding the optimal number of training epochs. We plotted the graph of training and validation loss over the number of epochs, global max pooling is used here as it provides the best accuracy, and observed that the validation loss was fluctuating in an unpredictable manner, suggesting instability in training. Therefore, we implemented **Reduced Learning Rate on Plateau** helping the model to converge smoothly by decaying the learning rate when there is no improvement. We have chosen 15 as the optimal number of epochs because it showed a minimum at the 8th epoch to ensure sufficient training while at the same time avoiding the risk of overfitting.

Fig 2: Plot of Training and Validation Loss Over Epochs for RNN Using Global Max Pooling



Fig 2.1: Plot of Training and Validation Loss Without the Use of Reduced Learning Rate on Plateau

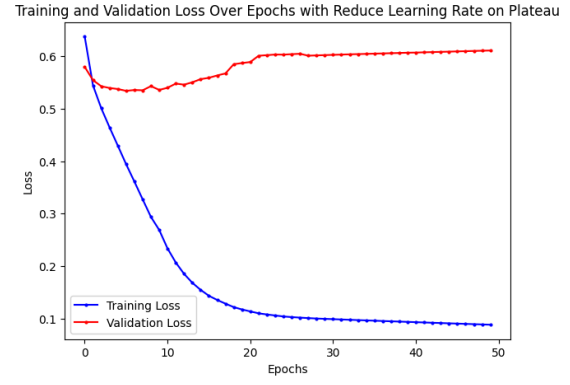


Fig 2.2: Plot of Training and Validation Loss With the Use of Reduced Learning Rate on Plateau

2.(a) Ans: The optimal configuration: Optimizer: 'Adam', Learning Rate: 0.001, Batch Size: 32, Epochs: 15

2.3 Accuracy Score of Validation and Test Set

Epoch	1	2	3	4	5	6	7	8	9	10	11	12
Validation Accuracy	0.6839	0.7242	0.7392	0.7486	0.7439	0.7411	0.7448	0.7430	0.7373	0.7280	0.7355	0.7280

2.(b) Ans: Final Validation Accuracy: 74.48% ; Final Test Accuracy: 73.92%

2.4 Methods to Derive Final Sentence Representation to Perform Sentiment Classification

We experimented with four different methods to represent the sentence information, each aiming to aggregate the word-level representations into a single sentence-level representation to derive a final sentence representation for sentimental classification. Below are the methods along with their explanations and performance:

2.4.1 Last Hidden State

The hidden state at any timestamp t can be computed based on both the input at $x(t)$ and the previous hidden state $h(t-1)$. Thus, the last hidden state of the RNN is often used to represent the entire sentence assuming the last hidden state $h(T)$ **captures all the relevant information from the sequence**.

$$h_{\text{sentence}} = h_T$$

2.4.2 Global Max Pooling

Global max pooling selects the maximum value across all timestamps for each feature dimension $h(t, f)$. This approach captures the strongest signal for each feature across the entire sequence, highlighting the **key features that are most important to represent the sentence**.

$$h_{\text{sentence},f} = \max_{t=1,\dots,T} (h_{t,f}) \quad \text{for } f = 1, 2, \dots, N$$

2.4.3 Global Average Pooling

Global average pooling computes the average value for each feature across all timestamps. This method **provides a balanced representation of the entire sequence** by averaging the contribution from each word. Although it captures the general context well, specific or important key features.

$$h_{\text{sentence}} = \frac{1}{T} \sum_{t=1}^T h_t$$

2.4.4 Concatenation of Max and Average Pooling

The motivation for concatenating the results from both max pooling and average pooling is to leverage both their strengths, which is to **capture both the most significant features and the overall context**. However, the performance is slightly worse might be due to the increased complexity of the sentence representation, thus leading to minor overfitting.

$$h_{\text{sentence}} = \left[\max_{t=1, \dots, T} (h_{t,f}); \frac{1}{T} \sum_{t=1}^T h_{t,f} \right] \quad \text{for } f = 1, 2, \dots, N$$

2.(c): Ans

Sentence Representation	Test Accuracy (%)
Last Hidden State	64.95
Global Max Pooling	73.92
Global Average Pooling	73.17
Concatenation of Max and Average Pooling	73.64

The best method for representing the sentence information is using global max pooling.

3.Enhancement

3.1 Accuracy Score of Simple RNN Model with Updated Word Embedding

We updated the embedding layer to be trainable during model training by setting `model.add(Embedding(trainable = True))`. This allowed the model to fine-tune the word embeddings based on a specific dataset. This results in an **improvement in the test accuracy of the model from 73.92% to 77.39%**, demonstrating the benefit of allowing the embeddings to adapt to the data during training.

3.(a) Ans: Test accuracy for the RNN model with updated word embedding is 77.39%

3.2 Accuracy Score of RNN Model with Handling of OOV Words

After incorporating a trainable embedding layer and implementing the strategy mentioned above (Sec 1.3) to handle OOV words, we further observed the **model accuracy score increased to 78.99%**. This highlighted the positive impact of handling unseen words properly.

3.(b) Ans: Test accuracy for the RNN model with FastText to handle OOV words is 78.99%

3.3 Accuracy Scores of biLSTM and biGRU on test set

After hyper-parameter tuning, we found that the **optimal number of layers stacked** for biLSTM and biGRU is **2** and found that the accuracy decreases as the stacked layers increase. This could be due to the vanishing gradients and exploding gradients issue where the gradients propagated through the network become too small or too large.

For biLSTM model, we used 2 stacked biLSTM layers, each with units = 128. The dropout layer after each biLSTM layer uses dropout rate = 0.5 to reduce overfitting. After the biLSTM layers, max-pooling layer was added at the end to retain the most important features. Lastly, the dense layer maps the output to the final predictions using a Sigmoid function. The optimal batch size used was `batch_size = 16` and was trained with Adam optimizer at `learning_rate = 0.001`. The model achieved a **test accuracy of 80.11%**.

For biGRU model, the optimal configuration is 2 stacked biGRU layers each with units = 16, `batch_size = 32` and trained with Adam optimizer at `learning_rate = 0.002`. Similar to biLSTM, a global max-pooling layer was added before the Sigmoid layer. The **test accuracy achieved 79.64%** for biGRU model. It is observed that when increasing the number of units in each biGRU layer, the validation accuracy decreased. This might be caused by overly complex models which lead to overfitting.

3.(c) Ans: Test accuracy for the biLSTM model is 80.11% and that of biGRU model is 79.64%.

3.4 Accuracy Scores of CNN on test set

For CNN, the optimal configuration used was a `batch_size` of 32 and `epochs = 10`. It uses 1 convolutional layer and a concatenation of Average and Max pooling layers.

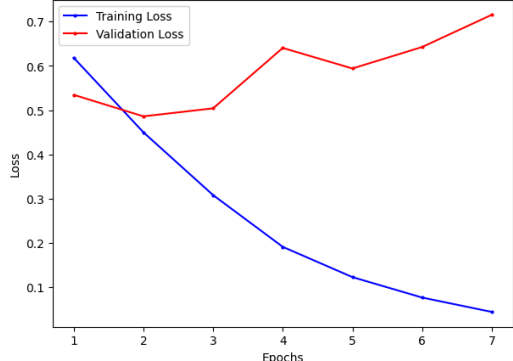
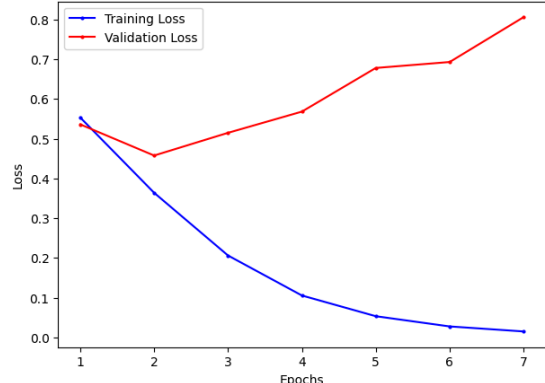
3.(d) Ans: Test accuracy for the final CNN model is 78.04%.

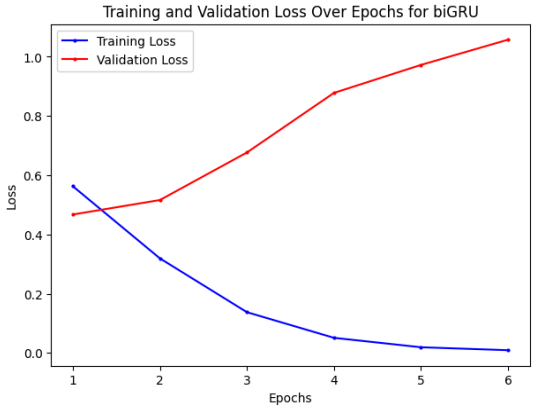
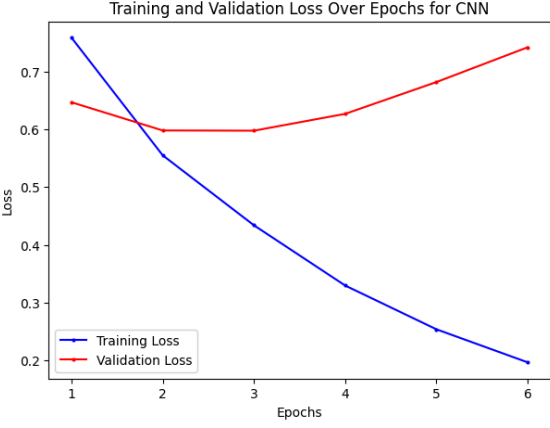
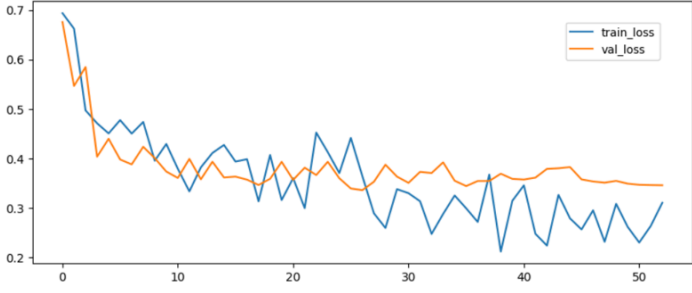
3.5 Final Improvement Strategy

We utilised a pre-trained transformer model, BERT. The pre-trained model scored an accuracy of 50.0% on the test dataset which was used as a baseline to measure the effectiveness of fine tuning. Due to limited resources, we opt for Parameter Efficient Fine Tuning techniques, **LoRA**. The target modules were the query, key and value linear layers. Using LoRA, we are also able to reduce the risk of catastrophic forgetting as the original model weights are frozen and only a separate adaptor is trained which can be merged to the base during inference. We trained the LoRA adapter with the following configurations: $\{r=4, \alpha=32, \text{dropout}=0.1\}$, while monitoring the training and validation loss to ensure that there was no overfitting or underfitting.

3.(e) Ans: The optimal fine-tuned model obtained had an accuracy score of 84.6% on the test dataset.

3.6 Comparison of Solutions and Discussion

Models	Test Accuracy (%)	Comparisons of Training and Validation Loss
RNN (with OOV handling)	78.99	<p>Training and Validation Loss Over Epochs for RNN with Trainable Embedding Layer and OOV Handling</p>  <p>Fig 3: Plot of Training and Validation Loss for RNN</p>
biLSTM	80.11	<p>Training and Validation Loss Over Epochs for biLSTM</p>  <p>Fig 4: Plot of Training and Validation Loss for biLSTM</p>

biGRU	79.64	 <p><i>Fig 5: Plot of Training and Validation Loss for biGRU</i></p>
CNN	78.04	 <p><i>Fig 6: Plot of Training and Validation Loss for CNN</i></p>
Enhancement 1: BERT	84.61	 <p><i>Fig 7: Plot of Training and Validation Loss for BERT</i></p>
Enhancement 2: biGRU-CNN-biLSTM Ensemble	81.05	-

The RNN seems to perform slightly better than CNN and this could be due to the fact that **RNN is better at learning sequential data** (Craig, 2024).

The performance of biLSTM and biGRU models are slightly better than the RNN model. This could be due to **RNN’s limitation that it is prone to vanishing gradient problems**, especially across long sequences. biLSTM and biGRU models have complex architectures that allow them to **capture important information over long sequences effectively**, overcoming this limitation (Shaikh & Ramadass, 2024).

BERT shows the best performance in sentiment analysis. It is pre-trained on large text corpora which allows it to capture rich semantic representations. This provides a better starting point for training compared to the other models which would require extensive training from scratch.

4. References

- Craig, L. (2024, July 29). *CNN vs. RNN: How are they different?: TechTarget*. Search Enterprise AI. <https://www.techtarget.com/searchenterpriseai/feature/CNN-vs-RNN-How-they-differ-and-where-they-overlap>
- Krithika. (2024, October 11). Introduction to FastText embeddings and its implication. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2023/01/introduction-to-fasttext-embeddings-and-its-implication/>
- Luay, M. (2023, September 22). Sentiment analysis using recurrent neural Network(RNN),Long short term Memory(LSTM) and Convolutional Neural Network(CNN) with Keras. Medium. <https://medium.com/@muhammadluay45/sentiment-analysis-using-recurrent-neural-network-rnn-long-short-term-memory-lstm-and-38d6e670173f>
- Shaikh, Z. M., & Ramadass, S. (2024, July). *Unveiling deep learning powers: LSTM, BiLSTM, gru, BiGRU, RNN comparison*. Indonesian Journal of Electrical Engineering and Computer Science. <https://ijeecs.iaescore.com/index.php/IJEECS/article/view/35633/18407>