Exploring the Impact of The Transformer Architecture

Ojasv Singhal
AI-ML Department
Maharaja Agrasen Institute of Technology
Delhi, India
ojasvsinghal11223344@gmail.com

Nalin Khanna
AI-ML Department
Maharaja Agrasen Institute of Technology
Delhi, India
nalinkhanna17@gmail.com

Dr. Neeraj Garg
HOD, AI-ML Department
Maharaja Agrasen Institute of Technology
Delhi, India

Dr. Neelam Sharma
Assistant Professor, AI-ML Department
Maharaja Agrasen Institute of Technology
Delhi, India

***Abstract*—The transformer architecture has revolutionized natural language processing (NLP) tasks, offering significant advancements in various applications such as machine translation, text generation, and sentiment analysis. In this paper , we aim to differentiate between the working and performance of various classical NLP models like LSTM against the novel transformer model (BERT , RoBERTa). We fine-tuned these models on Sentiment analysis with a proposed architecture. We used f1-score and AUC (Area under ROC curve) score for evaluating model performance. We found the BERT model with proposed architecture performed well with the highest f1-score of 0.85 followed by RoBERTa f1- score=0.80. We evaluated the performance of LSTM and BERT models on the IMBD dataset and our accuracy came out to be 0.85 and 0.92 for LSTM and BERT respectively . Despite achieving higher accuracy, BERT required a longer duration for fine-tuning compared to LSTM .**

***Index Terms*—Natural Language Processing (NLP), Deep Learning, Transformers, BERT , LSTM , RoBERTa .**

I. Introduction

The field of natural language processing (NLP) has witnessed remarkable advancements in recent years, driven largely by the development of deep learning architectures. These architectures, characterized by their ability to learn intricate patterns and representations from large-scale data, have revolutionized various NLP tasks, including sentiment analysis, figurative speech recognition, and translation. Among these architectures, transformer-based models have emerged as a breakthrough, offering unparalleled performance and scalability in capturing contextual dependencies and long-range interactions in text data.

Transformer architecture, introduced by Vaswani et al. in the seminal paper ”Attention is All You Need,” has become the cornerstone of state-of-the-art NLP models such as BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pre-trained Transformer), and Transformer-XL. By leveraging self-attention mechanisms and parallelization, transformers have overcome limitations of recurrent neural networks (RNNs) and convolutional neural networks (CNNs), enabling efficient processing of sequences of variable length and capturing global context dependencies in text.

However, amidst the dominance of transformer-based models, traditional approaches such as Long Short-Term Memory (LSTM) networks and other classical methods remain relevant and widely used in various NLP applications. These traditional approaches, characterized by their simplicity, interpretability, and computational efficiency, continue to offer viable alternatives for certain tasks and scenarios.

In this comparative analysis, we aim to explore the landscape of NLP models by examining the strengths and weaknesses of transformer-based models, LSTM networks, and other traditional approaches in the context of sentiment analysis, figurative speech recognition, and translation tasks. By scrutinizing their performance across different datasets, languages, and computational resources, we seek to provide insights into the suitability of each model architecture for specific NLP applications. Through a comprehensive examination of their capabilities in terms of accuracy, computational efficiency, interpretability, and adaptability, we aim to contribute to a deeper understanding of the evolving landscape of NLP models and guide researchers and practitioners in selecting the most appropriate model architecture for their tasks.

II. Literature Survey

The concept of self-attention mechanisms has revolutionized natural language processing tasks. This model has since become the foundation for numerous state-of-the-art models in various domains **[1][3]**. A survey on neural architecture search for transformer models, discussing various techniques and challenges in optimizing these architectures has been performed **[2]**. A novel neural machine translation architecture called the RNN Encoder-Decoder model has been proposed to encode source sentences and decode target sentences **[4]**. The concept of ”vanishing” and ”exploding” gradients, which can hinder the learning process in traditional RNNs has been proposed **[8]**. Xception, a novel deep learning architecture has been introduced that employs depthwise separable convolutions, which significantly reduces the computational cost while maintaining high performance in image recognition tasks **[5][7][10]**. The Adam optimization algorithm has become a popular choice for training deep neural networks due to its adaptive learning rates and efficient convergence properties **[6]**. Google's neural machine translation system employs deep learning techniques to improve the quality of machine translation, bridging the gap between human and machine performance **[9]**.

III. Methodology

Data Collection and Preprocessing:

- For comparing different transformer models (BERT and roBERTa) , we used Covid19 tweets dataset, publicly available on Kaggle. The train dataset contains 41157 tweets and test dataset contains 3798 tweets. There are 5 classes in the sentiment variable such as Extremely Negative (0), Extremely Positive (1), Negative (2), Neutral (3), and Positive (4).
- To evaluate and compare the performance of LSTM and BERT , we utilized the IMDB dataset, which consists of 50,000 movie reviews. The dataset is evenly divided into 25,000 positive reviews and 25,000 negative reviews, making it suitable for training and testing binary sentiment analysis models.
- Preprocess the datasets by cleaning, tokenizing, and normalizing the text data.
- Split the datasets into training, validation, and test sets.

Experimental Setup:

- Implement or utilize existing implementations of selected models using TensorFlow and PyTorch.
- We used F1-Score and ROC (Receiver Operating Characteristic) curve as evaluation metrics to assess model performance.
- Experiment with different hyperparameters, model configurations, and preprocessing techniques to optimize model performance.

Training and Evaluation:

- Train each model on the respective training datasets for sentiment analysis, figurative speech recognition, and translation tasks.
- Evaluate the trained models on the validation and test datasets to measure their performance.
- Perform cross-validation or k-fold validation to ensure robustness and generalization of results.

Comparison and Analysis:

- Compare the performance of transformer-based models, LSTM networks, across different tasks and datasets.
- Analyze the strengths and weaknesses of each model architecture in terms of accuracy, computational efficiency, interpretability, and adaptability.

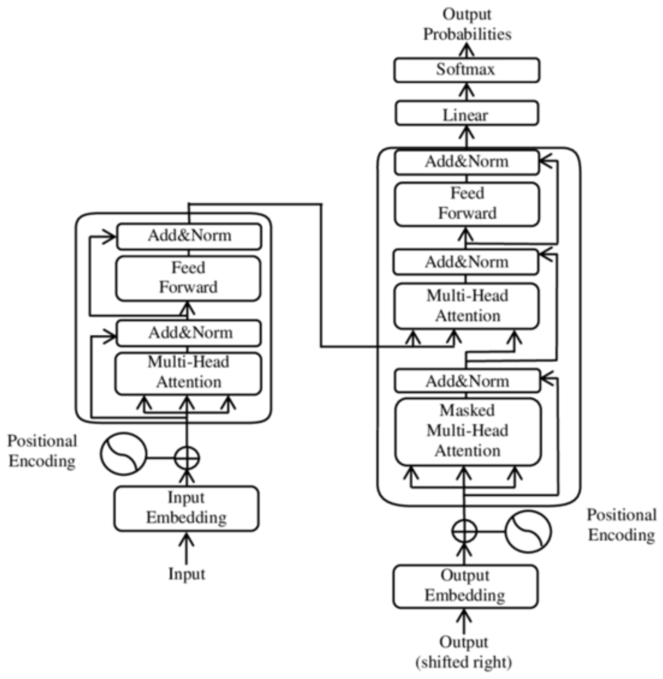

Fig. 1. Transformer Model Architecture

A. Transformer Model Architecture

Most competitive neural sequence transduction models have an encoder-decoder structure [1, 2].

The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 1, respectively.[3]

1) ***Encoder and Decoder stacks :*** **Encoder stack** : The encoder stack processes the input sequence and generates a sequence of context-rich representations. It typically consists of multiple encoder layers stacked on top of each other. Each encoder layer independently processes the input sequence and generates output representations. The input sequence is first tokenized and embedded into a sequence of input embeddings, which are then passed through the encoder layers. Each encoder layer contains two main subcomponents: multi-head self-attention mechanism and feed-forward neural network (FFNN).

Decoder stack : The decoder stack takes the context-rich representations generated by the encoder stack and generates the output sequence, one token at a time. It consists of multiple decoder layers stacked on top of each other. Each decoder layer independently processes the output sequence and generates output representations. Similar to the encoder layers, each decoder layer contains two main subcomponents: multi-head self-attention mechanism and FFNN. In addition to the self-attention mechanism, the decoder layers also include an additional cross-attention mechanism, which attends to the encoder's output representations to capture relevant information for generating the output sequence.

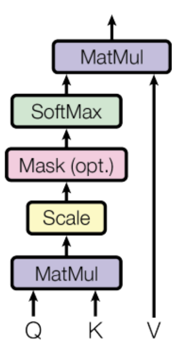

Fig. 2. Multi-head attention

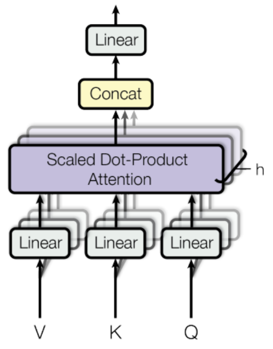

Fig. 3. Scaled-dot product Attention

*2) **Attention**:* The self-attention mechanism is a key component of transformer-based architectures in natural language processing (NLP) and other sequence modeling tasks. It allows the model to weigh the importance of different words or tokens in a sequence when making predictions.

Let's break down how self-attention works:

- Input Representation:

Suppose we have a sequence of input tokens, such as words in a sentence or pixels in an image. Each token is represented as a vector, typically obtained through an embedding layer.

- Query, Key, and Value Vectors:

For each token in the sequence, we generate three vectors: a query vector, a key vector, and a value vector. These vectors are derived from the input token's embedding.

- Attention Scores:

The core of self-attention is computing attention scores, which indicate the importance of each token relative to others in the sequence. To compute attention scores, we take the dot product of the query vector of a token with the key vectors of all other tokens in the sequence. This produces a score for each token, representing how much attention should be paid to it.

- Attention Weights:

The attention scores are then passed through a softmax function to obtain attention weights. These weights determine how much each token contributes to the final output. Tokens with higher attention scores will have higher weights, meaning they are more influential in the output.

- Weighted Sum: Finally, we compute a weighted sum of the value vectors, using the attention weights as coefficients. This weighted sum represents the self-attended representation of the input sequence, where tokens that are more important receive higher weights.
- Multi-Head Attention:

In practice, self-attention is often applied in parallel across multiple "heads" or sets of query, key, and value vectors. This allows the model to capture different aspects of the input sequence's relationships and dependencies.

Overall, the self-attention mechanism enables the model to efficiently capture long-range dependencies in sequences, adaptively attend to relevant information, and produce context-aware representations. This makes it a powerful tool for tasks like language understanding, translation, and generation.

*3) **Positional Encoding**:* Positional encoding is a technique used in the field of natural language processing (NLP) and specifically in the context of transformers, a type of neural network architecture. It addresses the inherent lack of sequential information in transformers by encoding the position of each token in the input sequence. This allows the model to understand the order or position of words in a sequence, which is crucial for tasks like language understanding and translation.

In transformers, positional encoding is typically added to the input embeddings before feeding them into the model. The positional encoding vectors are learned during training and added to the input embeddings to inject positional information. In a computer, when we process text using something like a transformer, the computer doesn't naturally know the order of words. It just sees a bunch of words all mixed up. So, we need a way to tell the computer which word comes first, which comes second, and so on. That's where Positional encoding comes in.

There are multiple methods to perform positional encoding. Some common approaches for positional encoding in transformers are -

1) Sinusoidal Positional Encoding

This approach uses sinusoidal functions to encode the positions of tokens in a sequence. Each dimension of the positional encoding vector corresponds to a sinusoidal wave with a different frequency. This method ensures that positional encodings have a smooth transition and capture positional information effectively. The formula for sinusoidal positional encoding was mentioned earlier. This is the formula for positional encoding -

$$PE(pos, 2i) = sin(pos/10000^{2i/d_{model}})$$

$$PE(pos, 2i+1) = cos(pos/10000^{2i/d_{model}})$$

This formula generates a unique positional encoding vector for each position in the sequence and each dimension of the input embeddings. The sinusoidal functions ensure that the positional encodings have a smooth transition and capture positional information effectively.

2) Learned Positional Encoding

Instead of using predefined positional encodings, this approach learns the positional embeddings as part of the model training process. In this method, the model learns to generate unique positional embeddings for each position in the sequence. These embeddings are optimized along with other parameters of the model to capture the positional information effectively. Let's denote the learned positional embedding for position ***pos*** and dimension ***i*** as ***PE(pos,i)***

During training, the model learns the values of ***PE(pos,i)*** along with other parameters of the model.

So, the formula is simply: ***PE(pos,i)***

Where:

pos is the position of the token in the sequence.

i is the dimension of the positional embedding.

In this approach, the values of *PE*(*pos*,*i*) are initialized randomly or using some other initialization strategy (such as Xavier or He initialization) at the beginning of training. Then, during the training process, the model updates these values through backpropagation along with other parameters to minimize the loss function and improve performance on the task at hand.

3) Fixed Positional Encoding

In this approach, fixed positional embeddings are added to the input embeddings before feeding them into the model. These positional embeddings are predefined and remain constant throughout the training process. Fixed positional embeddings can be generated using various techniques such as random initialization, predefined patterns, or learned embeddings from pretraining.

Fixed positional embeddings are typically generated using various techniques such as random initialization, predefined patterns, or learned embeddings from pretraining. Once generated, these embeddings are not updated during training and remain fixed.

For example, if you're using random initialization, the formula would be:

PE(*pos*,*i*)**=Random Initialization**

Or if you're using predefined patterns (such as sine or cosine waves), the formula would depend on the specific pattern used.

Similarly, if you're using learned embeddings from pretraining, the formula would be:

PE(*pos*,*i*)**=Pretrained Embedding Value**

Where:

pos is the position of the token in the sequence.

i is the dimension of the positional embedding.

Fixed positional embeddings provide a simple and efficient way to incorporate positional information into the model. However, they may not adapt to different patterns in the data as effectively as learned positional embeddings.

B. The BERT Model

BERT (Bidirectional Encoder Representations from Transformers) is a bidirectional transformer, meaning that it uses both left and right contexts in all layers as in Fig 4

It utilizes a specific input representation scheme designed to capture bidirectional contextual information from the input text. The input representation for BERT consists of three main components as shown in Fig 5:

1) Token Embeddings:
   - BERT tokenizes the input text into a sequence of subword tokens using a pre-trained vocabulary.
   - Each subword token is mapped to a unique token embedding vector through an embedding lookup table.
   - BERT supports subword tokenization techniques like WordPiece or SentencePiece, which enable it to handle out-of-vocabulary words and capture morphological variations in languages.
2) Segment Embeddings:
   - BERT incorporates segment embeddings to distinguish between different segments or sentences in the input text.
   - In tasks involving sentence pairs (e.g., question answering, natural language inference), BERT appends special segment embedding tokens to each tokenized input, indicating the segment to which each token belongs.
   - For single-sentence tasks, BERT assigns the same segment embedding to all tokens in the input sequence.
3) Positional Embeddings:
   - BERT incorporates positional embeddings to convey positional information about the order of tokens in the input sequence.
   - Unlike recurrent neural networks (RNNs) or convolutional neural networks (CNNs), which inherently capture sequential order, transformers like BERT do not have built-in mechanisms for sequential processing.
   - Positional embeddings are added to the token embeddings to indicate the position of each token in the input sequence. They enable BERT to capture sequential relationships and attend to token positions effectively.
   - Positional embeddings are typically learned embeddings or sinusoidal embeddings, which encode positional information based on the position of each token in the input sequence.

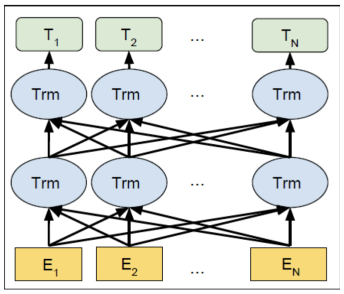

Fig. 4. BERT Architecture

To achieve better performance and accuracy, we have proposed a method for BERT. We fine-tuned the BERT model on pre-processed tweets data using a dropout layer, a hidden layer, a fully connected layer and a SoftMax layer for classification on top of BERT embeddings. We have considered BERT-base uncased pre-

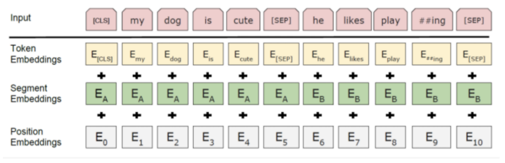

Fig. 5. Input representation in BERT

trained model for this task, which has 12 layers, 768 hidden size, 110 M parameters.

C. *RoBERTa*

RoBERTa, an advanced version of the BERT model boasts robust optimization techniques and extensive pre-training on vast text corpora. By refining BERT's training process with dynamic masking and a larger dataset, RoBERTa achieves superior performance in natural language understanding tasks. Its bidirectional context encoding enables nuanced understanding of textual nuances and context dependencies.
For this task, we proposed a method to fine tune the model on preprocessed tweets data using a dropout layer, a hidden layer, a fully connected layer and a SoftMax layer on top of RoBERTa embeddings as shown in Figure 4. We have chosen the distil RoBERTa -base pre-trained model for this task, which has 6 layers, 768 hidden size,12 heads, 82 M parameters.

D. *Long Short-Term Memory(LSTM)*

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to overcome the limitations of traditional RNNs in capturing long-range dependencies in sequential data. LSTM networks consist of memory cells and gates that regulate the flow of information through the network, allowing them to retain and propagate information over long sequences.

- Configurations

For LSTM, the model takes in a text sequence along with the corresponding lengths of each sequence as input. It embeds the text (embedded dimension = 20), processes it through the LSTM layer (size = 64), passes the last hidden state through fully connected layers with ReLU activations, and finally applies a sigmoid activation to produce a single output value between 0 and 1. (number of epochs: 10, learning rate: 0.001, optimizer: Adam)

For BERT, I used DistilBertForSequenceClassification, which is based on the DistilBERT architecture. DistilBERT is a smaller, distilled version of the original BERT model. It is designed to have a smaller number of parameters and reduced computational complexity while maintaining a similar level of performance. (number of epochs: 3, learning rate: 5e-5, optimizer: Adam)

IV. RESULTS AND OBSERVATIONS

Table I shows the results and accuracy for BERT and RoBERTa models for sentiment analysis. The learning rate 2e-5 and sentiment length, 120 are constant for both the models.

Model	f1-score	lr	Dropout	Batch
BERT	0.85	2e-5	0.35	8
RoBERTa	0.80	2e-5	0.32	32

TABLE I
COMPARING BERT AND ROBERTA

Table II shows the comparison between BERT and LSTM for sentiment analysis. The accuracy for BERT came out to be more than that of LSTM model.

Model	Accuracy	Training time
LSTM	0.85	4.56 min
BERT	0.92	118.89 min

TABLE II
COMPARING BERT AND LSTM

A. *BERT*

BERT got best results at batch size of 8 and drop out of 0.35. Fig 6 and Fig 7 shows the precision vs recall curve and ROC Curve for BERT respectively.

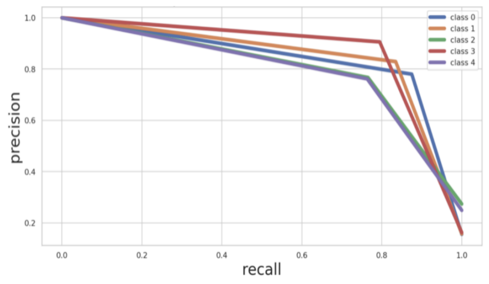

Fig. 6. Precision-Recall Curve for BERT

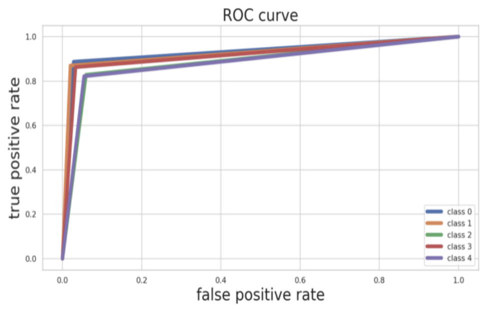

Fig. 7. ROC Curve for BERT

The Figure 6 shows that class 4 (Positive) has low f1-score and class 0 (Extremely Negative) has high f1-score. This means that class 1 classifies well with low misclassification error compared to all other classes.

Figure 7 shows that class 0 (Extremely Negative) has high AUC compared to all other models. These results show that BERT is having difficulty in classifying class 2 and 4 correctly.

B. *RoBERTa*

RoBERTa got best results at batch size 32 and drop out of 0.32. Figure 8 and Figure 9 shows the Precision vs Recall Curve and ROC Curve for RoBERTa respectively.

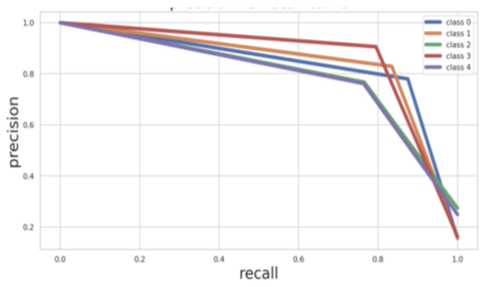

Fig. 8. Precision-recall Curve for RoBERTa

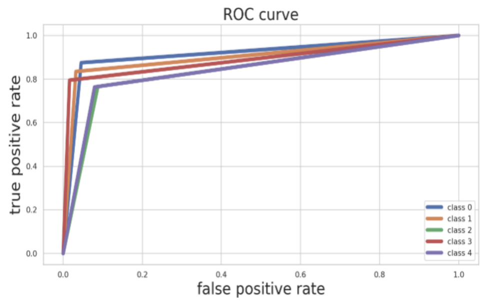

Fig. 9. ROC Curve for RoBERTa

Figure 8 shows that class 3 (Neutral) has high precision and low recall (low f1-score) and class 0 (Extremely Negative) has high precision and high recall (high f1- score). This means that class 0 classifies well with low misclassification error among all classes.

Figure 9 (ROC curve) shows that the RoBERTa model generalizes well for class 0 compared to all classes. RoBERTa model also has difficulty in classifying class 2 and 4 correctly

V. Conclusion and Future Scope

In this study, we fine-tuned Transformer-based pretrained models, including BERT and RoBERTa, using a proposed method for Multiclass Sentiment analysis on a Covid19 tweets dataset. Our experiments revealed that BERT achieved the highest accuracy but required longer training time (batch size=8), while RoBERTa yielded acceptable results with faster training time (batch size=32). Notably, both models struggled with accurately classifying sentiments labeled as Negative and Positive. The study was conducted with specific batch size and dropout settings for 5 epochs, suggesting that model performance may vary with different settings. We also evaluated and compared the performance of 2 deep learning algorithms (BERT and LSTM) for conducting binary classification in sentiment analysis. We concluded that BERT does take a significantly longer time to be fine-tuned compared with LSTM due to its more complex architecture and larger parameter space but it also exhibits superior performance, achieving higher accuracy and F1-score compared to vanilla LSTM.

Future research could explore the performance of these models with varying batch sizes and dropout values, potentially leading to further improvements in performance. Ultimately, this research contributes to the selection of the most suitable pre-trained models for Sentiment analysis based on a trade-off between accuracy and speed.

References

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., et and all (2017). Attention is All You Need. In Advances in Neural Information Processing Systems (pp. 5998-6008).

[2] K. T. Chitty-Venkata, M. Emani, V. Vishwanath and A. K. Somani, ”Neural Architecture Search for Transformers: A Survey,” in IEEE Access, vol. 10, pp. 108374-108412, 2022, doi: 10.1109/ACCESS.2022.3212767.

[3] B. Abibullaev, A. Keutayeva and A. Zollanvari, ”Deep Learning in EEG-Based BCIs: A Comprehensive Review of Transformer Models, Advantages, Challenges, and Applications,” in IEEE Access, vol. 11, pp. 127271-127301, 2023, doi: 10.1109/ACCESS.2023.3329678.

[4] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. CoRR, abs/1406.1078, 2014.

[5] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. arXiv preprint arXiv:1610.02357, 2016.

[6] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In ICLR, 2015.

[7] H. Xiong, K. Jin, J. Liu, J. Cai and L. Xiao, ”Deep Learning-based Image Text Processing Research*,” 2023 IEEE 9th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), New York, NY, USA, 2023, pp. 163-168, doi: 10.1109/BigDataSecurity-HPSC-IDS58521.2023.00037.

[8] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[9] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144, 2016.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778, 2016.