Optimizing the Topology of Transformer Networks Using Modified Clonal Selection Algorithm: A Bio-Inspired Immunocomputing Approach

Ashish Kharel and Devinder Kaur

Electrical Engineering and Computer Science Department, University of Toledo, OH 43607 USA
Email: akharel@rockets.utoledo.edu (A.K.); dkaur@rockets.utoledo.edu (D.K.)
*Corresponding author
Manuscript received April 7, 2024; revised June 18, 2024; accepted June 28, 2024; published November 26, 2024

***Abstract*—This paper proposes the optimization of the Transformer model for analysis of sequential data using a modified clonal selection algorithm (mCSA). Transformers demonstrate better performance over Long Short-Term Memory (LSTM) deep networks when the input sequence is exceptionally long. They are good at capturing long-term dependencies in comparison to LSTM networks. However, this comparison is valid only if the hyperparameters are optimized correctly. Also, transformers are very sensitive to their hyperparameters. Designing the architecture of the transformer model for better performance is very complex and time-consuming. There have been other efforts using Bayesian, Grid Search, Blackbox, and metaheuristic optimization techniques for the optimization of the architecture of deep learning models. mCSA is a nature-inspired immunocomputing approach. The performance of the optimized transformer model has been compared with an unoptimized transformer model, genetic algorithm optimized transformer, Clonal Selection Algorithm optimized LSTM(CSA_LSTM), Clonal Selection Optimized Hybrid Convolutional Neural Network and LSTM network (CSA-CNN-LSTM, and Random Forest search algorithm. CSA optimized transformer model has consistently shown better performance in comparison to all other models for a variety of datasets such as IMDB-movie, SMS-Spam, and US Twitter Airline datasets. Here we also show that improper optimization of transformer hyperparameters can lead to degraded performance that cannot surpass even traditional ML approaches like random forest. We have also carried out ablation studies to understand the impact of various hyperparameters on the performance of our model.**

***Keywords*—deep learning, clonal selection algorithm, immunocomputing, genetic algorithm, hyperparameters optimization, nature inspired algorithm, topology optimization, transformers**

I. Introduction

Transformer models are new neural network architectures that replace convolution neural networks and recurrent neural networks for machine learning [1]. They utilize a self-attention mechanism [2, 3] to weigh parts of input data higher or lower. While memory-based RNN architectures such as LSTM also gave state of art results after training, the very nature of RNNs slows them down because the input must be repeated for each calculation [1, 4]. Also, when the input data is very long, LSTMs tend to have difficulty learning the features in the data. This can be prevented using attention-based mechanisms [1, 4].

Whenever any deep neural network is designed, many parameters need to be optimized, such as the number of layers, window size, learning rates, size of the input layer, batch size, attention heads and many more depending upon the type of model. However, parameter selection also depends on the

type of data the network is trained on. Usually, the network is either designed by experts or by algorithms. Popular algorithms include Bayesian optimization [5], gradient-based optimization, grid search [6], random search, and evolutionary algorithms which CSA and GA are part of. For a smaller number of hyperparameters, grid search is commonly used. In grid search, each grid holds a discrete value that needs to be to be searched. In random search, the hyperparameters are selected randomly by random sampling. Unfortunately, both search techniques do not take into consideration the results from previous iterations [7]. Bayesian optimization techniques take into consideration the results from previous iterations, but the Bayesian technique is hard to implement [7]. Because of all these constraints, nature-inspired optimization techniques provide a strong alternative to the above techniques.

In statistical modeling, the ability to accurately capture and represent data dependencies is crucial. Datasets such as SMS Spam, IMDB Movie reviews, Twitter US Airline are commonly used to evaluate the performance of new models. In [8], a new method based on discrete hidden Markov model (HMM) was used to identify spam with high accuracy. Transformers excel in this domain by offering a robust framework for modeling complex interactions within data. Transformers have been used on a variety of applications like Natural Language Processing, Time Series Forecasting, Biological Data Analysis, etc.

II. Related Work

Many techniques have been studied for hyperparameter optimization for many generic machine learning models. Grid search-based methods used to be very common. Grid search tries to create a search space for possible solutions to the optimization problem and then tries all the values in the grid. However, this causes the computational cost to increase exponentially as the number of hyperparameters increases. In Bayesian Optimization, the strategy is to fit a Gaussian process model which tries to capture beliefs about the performance of the model. This model is then used to estimate the future distribution of hyperparameters. Another technique used in grid search is the use of early stopping where a bad combination of hyperparameters is discarded early from the information provided by the Bayesian optimization technique. Early stopping thus saves computational time which could be lost while training

unpromising hyperparameters. However, the grid search method does not take into consideration the results of previous iterations. The use of nature-inspired algorithms for the optimization of transformer hyperparameters has not been investigated in the literature. In [9], the authors explored the use of auto-sizing techniques for optimizing the topology of transformers. Auto sizing techniques use regularization to yield the best parameters over a single training run. The main advantage of this approach is that training is only done once to optimize the parameters. In [10], the authors proposed meta-learning hyperparameter optimization algorithms where learning was done from prior experiments. There are also other hyperparameter optimization services such as Google Vizier [11], Amazon Syne Tune [12], and Microsoft Azure ML [13]. In [14], authors compared the prediction accuracy of mCSA with other nature-inspired optimization algorithms such as Particle Swarm Algorithm and Ant Colony Algorithm for Iris dataset. They found that mCSA had better performance. In [15], authors used CSA to optimize the topologies of LSTM networks. They evaluated their methods on IMDB, SMS-Spam and Twitter US Airline datasets.

III. Background

A. The Transformer Model

Transformers were introduced by Vaswani *et al.* and now have widely shown their effectiveness over LSTM networks. Transformers consist of a stack of encoder and decoder layers. Fig. 1 shows the simplified view of transformer models.
In Fig. 1, each input passes through the stack of encoders first, then through the stack of decoders. The output from the last encoder is passed to each decoder block. The output from the last decoder is the final output. Each encoder and decoder block consists of multiple layers.

Because transformers do not use recurrent neural networks, they are free from the sequential processing requirements of RNNs [1]. This makes them very fast because of parallelization.

Fig. 2 shows each encoder and decoder block. Inside each encoder are self-attention layers and feed-forward layers while inside each decoder are self-attention layers [16], feed forward layers, and encoder-decoder attention layers. Fig. 3 shows the overall picture of the transformer. Each layer has its own embedding layer and while all stacks are identical, they do not share weights.

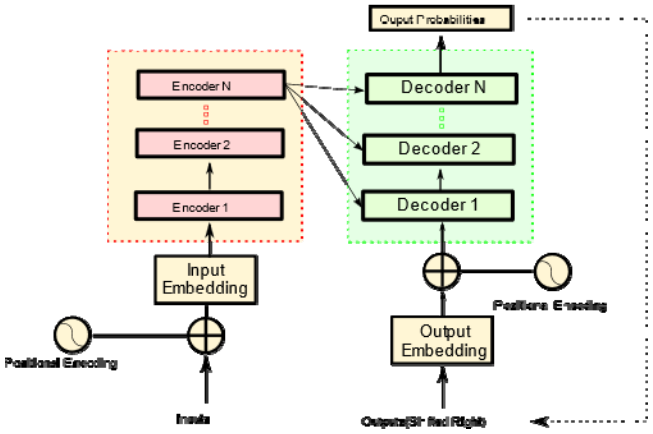

Fig. 1. Simplified overview of transformer model.

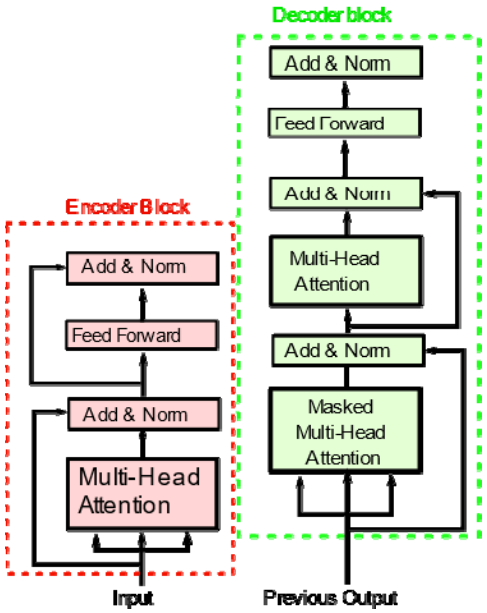

Fig. 2. Internal Architecture of the encoder and decoder.

B. Inside the Transformer

1) Inputs

These are the inputs to the network, such as sentences, words, or numbers. Since computers do not understand words, each of the words is converted into tokens from vocabulary.

2) Input embedding

Instead of using a single number representation for each input to feed into a network, input embedding converts each word into a vector. The vector can be of dimension 256, 512, etc. This action of embedding words into vectors only happens at the first encoder layer. In the case of a transformer having more than one encoder, each encoder receives input as the output from the previous encoder. Each word flows through its own path in the case of transformers, which makes transformers able to parallelize learning. While the multi-head attention layer has dependencies between the paths followed by each word, the feed-forward layer does not have any dependency.

3) Positional encoding

To account for the order of words, position encoders are needed because unlike LSTMS or similar RNNs, transformers process input simultaneously. For each embedding, positional encoding vectors are added which the models are expected to learn.

4) Encoders

The embedded input, which consists of a word vector that is positionally encoded, is passed to the first encoder. The subsequent encoders receive output from the previous encoders as input. Each transformer encoder consists of multiple layers to which input is fed step by step.

a) While LSTMs [17] improve upon earlier architectures for supporting a very long input, the attention mechanism completely removes the limitations of fixed memory and even very long sequences can be accessed. Each attention mechanism consists of a query vector (Q), key vector (K), and scores. The main role of this layer is to enrich each embedding vector with the

context of the entire sentence so that the model has a better understanding of the entire sequence.

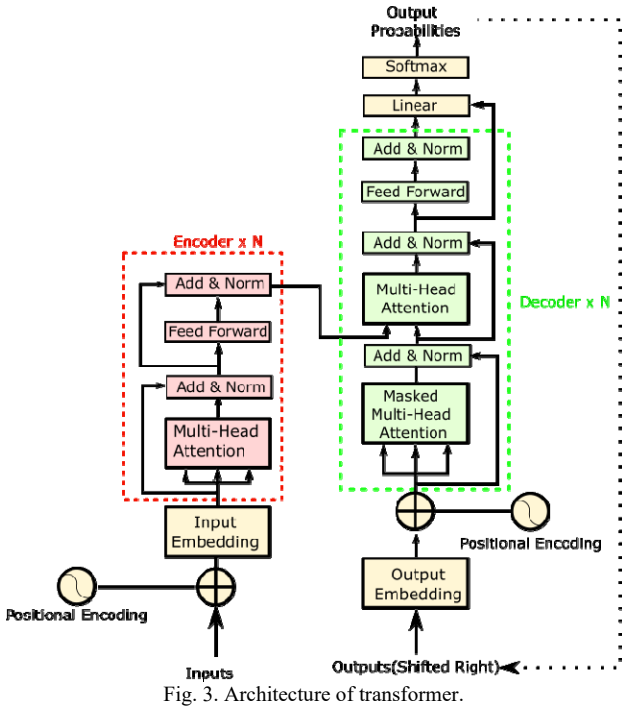

Fig. 3. Architecture of transformer.

b) Add and norm layer: Each Encoder has two Add and Norm layers. One of them comes after the multi-head attention layer while the other comes after the feed forward layer. The main purpose of the add layer is to form a residual connection so that input from both the previous layer and the layer before the previous layer is accounted for enriching the embedded vectors with additional information. The normalization layer is used to normalize the information to reduce the effect of covariant shift. Normalization is done for each embedded vector.

c) Feed-forward layer: This is a fully connected feed-forward layer with two linear layers and a RELU layer for introducing non-linearity. In this layer, each embedding vector is processed independently. The output of this layer is given as:

$$FFN(x) = RELU(x * W1 + b1) * W2 + b2 \quad (1)$$

where $W1$ and $W2$ are the weights in the fully connected layer and b1 and b2 are the biases or offset for the wight values.

5) Decoders

Both the output of the last encoder in the stack and the previous outputs of the network are passed to the decoders. The layers inside the decoders are similar compared to those of encoders, apart from the masked multi-head attention layer.

a) Masked multi-head attention layer [1]: While decoding, we need to make sure that future words are not included in the training. When writing a sentence, we base our next word only on the previous words that have been written, and not on the future words. Masked multi-head attention layer masks certain positions from the input. Usually, to do this, the attention score is set to a large negative number.

b) Output Embedding: Since this is fed to the decoder, the input to the decoders is the previous output from encoders, and the target tokens are decoded up to the current decoding step.

c) SoftMax: This last block of the transformer outputs the predicted next-token probabilities. This gives the predicted distribution of output vocabulary.

6) Output embedding

It is similar to the input embedding layer structurally but different in the role it plays. The embedded vector of the previous output and the embedded vector from the encoder are input together to the decoder to generate a new output sequence.

C. Vanishing Gradient Problem

As the neural network layers are made deeper, they become susceptible to the vanishing gradient problem [18]. Vanishing gradient problems are usually experienced while training networks with gradient-based learning or backpropagation-based learning. During training, the error gradients shrink as they are backpropagated and sometimes may become so small that the network completely stops learning. Many methods have been implemented to fix this issue while training neural networks. Some of the methods include proper weight initialization [19], batch normalization [20], the use of a different activation function [21], having residual connections [22], or for recurrent neural networks, the use of gated recurrent units (GRU) and LSTMs. But even LSTMs suffer from the problem when the dependency is very long.

D. Clonal Selection Algorithm

A clonal selection algorithm (CSA) [23] is a type of artificial immune system (AIS) algorithm that is inspired by the clonal selection theory of acquired immunity. The CSA works by iteratively generating a population of candidate solutions, called antibodies, and then selecting the best antibodies to reproduce and mutate. The process is repeated until a satisfactory solution is found. The CSA is a stochastic algorithm, which means that it does not always find the same solution to a problem. However, it has been shown to be effective in a variety of optimization problems, including function optimization, pattern recognition, and scheduling.
In CSA, the problem to be optimized is encoded as an antigen. The antigen can be represented in binary or decimal format.

This objective function would typically be the performance of the transformer model on a specific task or dataset. We have used Root Mean Squared Error (RMSE) as a measure to test the fitness of the chromosomes.

The parameters of the CSA such as the population size, mutation rate, and convergence criteria are configured for the optimization problem.

Once the algorithm is configured, we run it on the optimization problem to search for the optimal values of the parameters of the Transformer model. The algorithm would iteratively evaluate the objective function and adjust the values of the parameters.

Fig. 4 shows the flowchart of the CSA algorithm. The CSA starts by generating an initial population of antibodies (candidate solutions) randomly or using some heuristic method. Each antibody represents a potential solution to the

optimization problem at hand. The affinity of each antibody is calculated based on its fitness or objective function value. Affinity measures how well an antibody performs in terms of the optimization criterion. Higher affinity indicates better fitness.

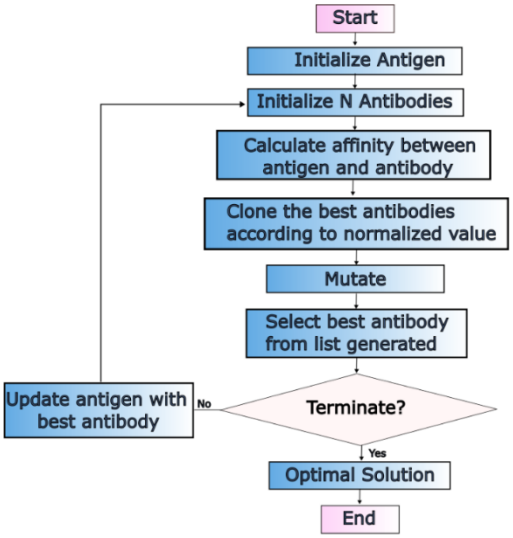

Fig. 4. Flowchart of the CSA algorithm.

Antibodies with higher affinities are selected for clonal expansion, mimicking the selection of B-cells with more effective antibodies in the immune system. The selected antibodies are duplicated to create a larger population. To introduce diversity and explore the solution space, mutation or hypermutation operators are applied to the duplicated antibodies. These operators perturb the antibodies' values or positions to generate new candidate solutions.

The affinity of the mutated antibodies is evaluated, and the antibodies with better fitness are selected for the next generation. This process emulates the selection of B-cells with improved antibodies during affinity maturation. The best-performing antibodies in terms of affinity are retained as memory cells, representing the immune system's long-term memory. These memory cells store high-quality solutions and contribute to the exploration and exploitation of the search space.

Steps 3 to 6 are repeated for a specified number of iterations or until a termination criterion is met. Through repeated clonal expansion, mutation, and selection, the population of antibodies gradually converges toward better solutions.

IV. Proposed Method

A. Antigen Encoding

Fig. 5 shows the encoding scheme of each hyperparameter during training for our experiment. This involves determining the hyperparameter space and the encoding scheme they take. There can be multiple numbers of hyperparameters and all of those should be represented by a 1D vector of antigens. The range and value of each hyperparameter should also be implemented. We will use an integer encoding scheme for antigen encoding because hyperparameters are discrete and integer encoding schemes are intuitive and interpretable. For example, the number of layers or the number of units in a neural network layer are natural choices to encode as integers. This makes the resulting hyperparameter configurations more interpretable and easier to understand.

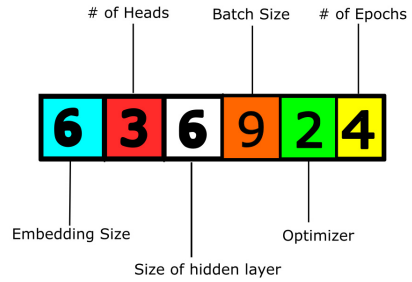

Fig. 5. Encoding of hyperparameters as antigen.

Table 1. Range of each hyperparameter for optimization

Hyperparameter	Range of Hyperparameter
Embedding Size	[1-512]
No. of Heads	[1-8]
Size of hidden layer in FF n/w	[1-2048]
Batch Size	[1-8]
Optimizer type	[Adam, SGD, RMSprop]
No. of epochs to train	[1-16]

Table 1 shows the hyperparameters and their respective ranges chosen for this research. The ranges have been chosen empirically.

B. Antibody Creation

To simulate the CSA for the optimizing transformers, the antibodies are initially generated randomly, constrained by the range of their values. For example, the batch size can be between 16 and 64. The algorithm produces a set of X antibodies, each with a distinct sequence of 1D vectors. Fig. 6 shows the created antibodies with random operation.

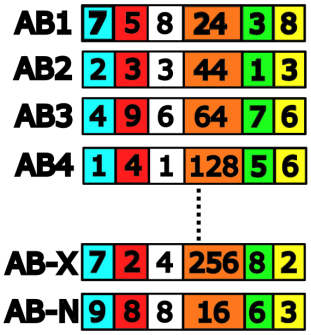

Fig. 6. Initial creation of N antibodies.

C. Mapping Immune System Terminologies to Hyperparameter Optimization Problem

To develop a computational model based on CSA for the sentiment analysis dataset, we need to map the terminologies of the immune system to the structure of the sentiment analysis domain. Table 2 shows how CSA can be interpreted in terms of application to transformer hyperparameter optimization.

D. Modified CSA

We implemented a modified version of CSA (mCSA) for our cloning and mutation operation. In an unmodified CSA algorithm, the number of clones and mutations is fixed for all selected antibodies. In modified CSA, the number of clones is given by Eq. (2).

$$N_c = ceil\left(N_i \frac{x_i - x_{min}}{x_{max} - x_{min}}\right) \quad (2)$$

where N_C is the number of clones to be generated, x_i is the affinity value of the i_{th} antibody, x_{min} is the lowest affinity value from the list of antibodies and x_{max} is the highest affinity value from the list of antibodies. N_i is fixed to a higher bound of normalization.

Each of the cloned antibodies goes through hypermutation. If the affinity of the antibody is low, the number of mutation points is set to high and vice versa. The mutation points for modified CSA are calculated under the formula given in Eq. (3).

$$Mp = 2 * ((Ni + 1) - Nc) \quad (3)$$

where M_p is the number of mutation points, N_i is the higher bound of normalization and N_c is the number of clones.

Table 2. Interpretating CSA Terminologies as a hyperparameter optimization problem

Immune System	Transformer model
Antigen (Ag)	Initial hyperparameters to be optimized
Antibody (Ab)	Potential solutions to hyperparameter problem
Affinity	Proximity between antigen and antibody
Cloning	Creation of multiple copies of the antibody
Mutation	Change in one or more hyperparameters of antibody
Population	Total number of antibodies
Generation	Number of Iteration

E. Selection

Selection is the process of selecting the best antigens from the pool of created antibodies. It is based on the fitness value evaluated during the affinity creation process. These antibodies are considered to be the best solutions as they are more effective in dealing with the given antigen. The CSA aims to improve the population of antibodies by selecting and reproducing those with higher affinity, similar to how the immune system generates more effective antibodies in response to antigens.

V. Experiment Setup

This section describes the implementation details of carrying out this research. It gives details of the datasets used, steps involved in data processing, and implementation, and the various performance metrics to compare the results. Apart from the already existing techniques such as Blackbox, Bayesian, grid search, and random optimization as mentioned earlier to correctly optimize the hyperparameters of the transformer, we hereby propose a biologically inspired, mCSA algorithm to optimize the hyperparameters of transformers.

A. Datasets

We used three datasets for comparing the performance, the IMDB movie dataset, the SMS spam dataset, and the Twitter US Airline dataset. We included database mixes such that we can estimate model performance in a dynamic environment. The SMS spam dataset is unbalanced with comparatively few number of data. The IMDB consists of movie reviews as a binary sentiment, which is whether the movie was favorable or unfavorable for the audience. It consists of a total of 50k movie reviews. The SMS spam dataset consists of approximately 5.5k SMS data, each labeled as whether the data was spam or not spam (ham). The Twitter US Airline dataset consists of approximately 11k sentiment data.

We then benchmarked the results by comparing them to the GA-optimized Transformer model, the existing general transformer model without optimization (the default transformer), LSTM, and various other ML models.

B. Dataset Preparation

Since our dataset mixture consisted of small, large, and unbalanced datasets, we used K-fold cross validation in order to robustly predict the performance of the model, and to better assess the generalizability to unseen data. K-fold cross-validation was used with the number of folds (K) set to 10 and the number of repetitions set to 3. Since the transformer works with tokens, tokenization was carried out for every word and then embedded as real-valued vectors.

Since computers cannot understand words and sentences directly, they need to be converted into a sequence of numeric vectors. Furthermore, sentences in the dataset can be inconsistent and can have missing values, for example, an empty row in the dataset or invalid characters. Hence, to transform the dataset into computer-readable numeric vector sequence, first, the data is cleaned up and then encoding techniques like BagOfWord, TF-IDF, and Word2Vec are used [24, 25]. The following steps were used for data preprocessing.

1) Data Filtration: We need to make sure that invalid characters and empty tables are removed before processing the data.
2) Data shuffle: Shuffling data randomly ensures that the training dataset does not lean heavily to one side or the other during the data-gathering process. Usually, train and test data splits are created by shuffling randomly.
3) Train-test data split: Train and test datasets are separated to ensure that the model is not remembering the dataset. Very complex models can remember the dataset features and thus will have a very high training accuracy but have a degraded performance for the test dataset.
4) Removing irrelevant words: We need to remove any words or sentences that contain invalid characters or numbers.
5) Tokenization: Tokenization is the process of cutting data into parts so that they can be represented as vectors. This process also ensures that repeated words as represented as the same vector.
6) Padding/truncating: Inputs to the network should be of the same size. In the dataset, some reviews might contain short sentences while some might contain longer sentences. Truncating and padding operations ensure that they are of the same length, either by removing some portions of the sentences or by adding a filler word, respectively.
7) Word embedding: The tokenized data is then encoded as vectors known as word embedding. This is usually done to increase the dimension of the dataset so that complex representations can be learned easily.

C. Performance Metrics

For comparing the performance of various deep and

shallow machine learning models, the selection of proper metrics plays a crucial role. Various metrics can be used in evaluating machine learning model performance. For regression, metrics such as mean square errors (MSE), and root mean squared error (RMSE) are used. In the case of classification metrics, accuracy, confusion matrix, precision, recall, F1-score, Area Under the ROC Curve (AUC-ROC) or Area Under the Precision-Recall Curve (AUPR) are used [26]. AUC measures the ability of the classifier to distinguish between classes. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. It is useful when the class distribution is relatively balanced. AUPR measures the trade-off between precision and recall for different thresholds. AUPR is particularly useful when dealing with imbalanced datasets where the positive class is rare. Since this research is a classification problem related to sentiment analysis as positive or negative, the performance metric of the confusion matrix, AUC and AUPR were chosen. An average of performance metrics such as accuracy, precision, recall, and F1-score are computed from the confusion matrix across all the folds and number of repetitions.

Before we delve into confusion matrix parameters, we must discuss true positive (TP), true negative (TN), false positive (FP), and false negative (FN). Let us consider the case of the SMS-spam dataset, where all spam sentences are labeled as positive (1), and all ham sentences are labeled as negative (0).

1) True positive (TP)**:** When the prediction of the model is the same as that of the positive ground truth, the result is called to be true positive. For example, the model correctly predicts that the given sentence is spam.
2) True Negative (TN): When the prediction of the model is correct for the negative ground truth, it is known as true negative. For example, if the model correctly predicts that the given sentence is not spam.
3) False positive (FP): When the prediction of the model is incorrect for the positive ground truth, the result is called false positive. For example, if the model incorrectly predicts that the given sentence is spam.
4) False negative (FN): When the prediction of the model is incorrect for the negative ground truth, the result is false negative. For example, if the model incorrectly predicts that the given sentence is not spam.

The other performance metrics such as Accuracy, Precision Recall, and F1 Score are calculated from the confusion matrices.

1) Accuracy

Accuracy shows how many of the classifications were correctly classified. Accuracy is given by:

$$\frac{TP+TN}{TP+TN+FP+FN} \quad (4)$$

2) Precision

When the class distribution is imbalanced, precision can be used as a good classification metric indicator. Precision is calculated as the ratio between the total number of positive examples correctly classified to the total number of examples classified correctly or incorrectly as positive.

$$\text{Precision} = \frac{TP}{TP+FP} \quad (5)$$

3) Recall

It can be defined as the ratio between the number of correctly classified positive examples to the total number of positive samples. In other words, the recall metric measures the ability of the model to recognize positive examples. The higher the recall, the more positive examples are recognized.

$$\text{Recall}=\frac{TP}{TP+FN} \quad (6)$$

4) F1 score

It helps to have a measurement that represents both recall and precision. It is calculated as the harmonic mean of precision and recall.

$$\text{F1-score}= 2\times\frac{\text{Precision}*\text{Recall}}{\text{Precision}+\text{Recall}} \quad (7)$$

D. Experiment Setup

This research project was implemented using Python 3.7 with high-level neural networks API called Keras library running on top of Tensor Flow [27]. The hardware platform used was a GPU computer on AMD Ryzen 5800H processor and RTX 3070 GPU with 8GB of GPU memory and 32GB of RAM.

The classification process using the transformer is illustrated in Fig. 9. First, the hyperparameters are set for the transformer using default initialization parameters and training is carried out for a given dataset. Then RMSE is calculated from the classification operation from the transformer and used as fitness criteria for the mCSA. The mCSA algorithm outputs a new set of parameters to be used for training by selecting new parents and performing mutation operations. The training is now done using the optimal hyperparameters discovered by the mCSA. The process continues until the fitness criteria are reached. Lastly, a prediction is made on test data, and the performance is evaluated based on the metrics described above.

The proposed CSA-Transformer was benchmarked using the IMDB dataset. Table 3 shows the features of IMDB, SMS SPAM, and Twitter US Airline datasets that have been used in this experiment.

Table 3. Dataset statistics

	IMDB	**SMS Spam**	**Twitter USAirline**
Input Length	500	100	200
Vocabulary length	181,556	8024	13,234
Classes	2	2	2
K-Fold Cross Validation and Repetitions	*K*=10 *R*=3	*K*=10 *R*=3	*K*=10 *R*=3

E. Results

We have compared the performance of the optimized transformer using mCSA and GA with unoptimized transformer model. The hyperparameters of the unoptimized transformer were chosen as the most sensible default value. We further compared the performance of the optimized transformer with optimized LSTM using the Clonal Selection Algorithm and CSA_CNN_LSTM, which is the optimized architecture of a hybrid Convolution Neural Network and

LSTM using CSA [28].

1) Performance Comparison for the IMDB dataset

Table 4 shows the comparison of accuracies of the various ML models for the IMDB validation data set. It shows that compared to the unoptimized transformer; the optimized transformer has higher accuracy. Our unoptimized transformer consisted of the initialization values as shown in Table V. The optimized model exhibits superior performance over other algorithms such as CSA-LSTM, CSA_CNN_LSTM, and Random Forest. It also shows that the performance metrics of the unoptimized transformer are subpar compared to optimized LSTM networks. However, optimization of the hyperparameters yields a vastly different conclusion. When optimized, transformers have shown superior performance.

Table 4. Performance comparison of mCSA Transformer with other techniques for IMDB validation Dataset

Model	Accuracy	precision	recall	F1-score
mCSA Optimized Transformer	90.48%	93.15%	91.47%	92.30%
GA optimized Transformer	89.72%	92.15%	87.88%	89.96%
Transformer (unoptimized)	88.56%	87.34%	90.06%	88.67%
CSA-LSTM [27]	89.52%	88.58%	90.77%	89.66%
CSA-CNN-LSTM [27]	89.88%	89.37%	90.05%	89.71%
Random Forest [27]	54.66%	54.38%	52.34%	53.34%

Table 5 shows the new hyperparameters found by our mCSA algorithm for the IMDB dataset. The number of epochs and the type of optimizer have not changed from the initial unoptimized value. It means that for this IMDB dataset, training for 2 epochs yielded the best result without overfitting.

Figs. 7 and 8 show the average of AUC and AUPR curves for the IMDB dataset across all folds. It shows that our classifier has a very good discriminatory power.

Table 5. the final optimized hyperparameters discovered by our proposed model for the IMDB dataset

Hyperparameter	optimized value	initialization value (unoptimized)
#Epochs	2	2
#Heads	2	8
hidden layer size in feed-forward layer	39	32
embedding size for each token	16	32
optimizer	ADAM	ADAM

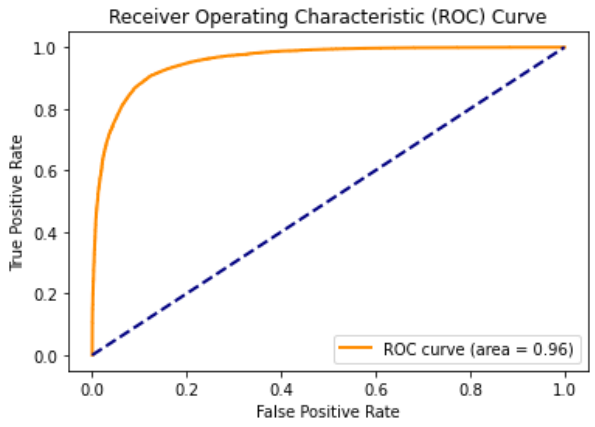

Fig. 7. Average AUC Curve for IMDB Dataset across all folds.

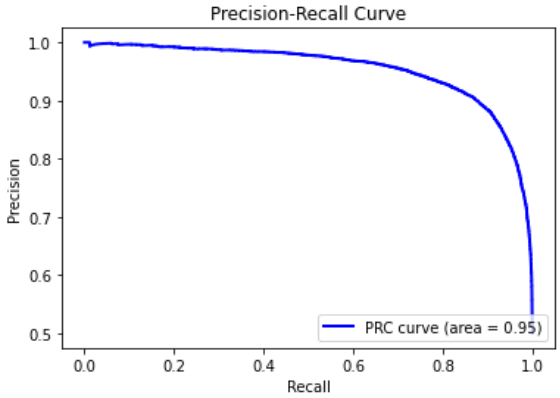

Fig. 8. Average AUPR Curve for IMDB Dataset across all folds.

2) Performance Comparison for SMS SPAM dataset

In the SMS spam dataset, there is more ham compared to spam, thus making it unbalanced. Training deep neural networks for these kinds of datasets is difficult. Making small changes to the hyperparameters can have a drastic effect on the performance.

This once again strengthens the need for a proper hyperparameter optimization technique. Note that the hyperparameters were optimized based on the validation dataset. Table 6 shows the comparison of accuracies of the various ML models for the SMS spam validation data set. In this dataset too, our model performed better than the unoptimized transformer and other NIA-optimized LSTM models. In all the four performance metric domains (accuracy, precision, recall and F1-score), the mCSA optimized transformer performed better. Figs. 9 and 10 show the AUC and AUPR curve for the SMS Spam dataset during the fourth fold of K-fold operation. Given that the dataset is imbalanced, the AUPR curve is vital for assessing the performance of our model. Our AUPR score for this dataset is 0.98, which is very good.

Table 6. Performance comparison of mCSA Transformer with other techniques for SMS spam validation dataset

Model	Accuracy	precision	recall	F1-score
mCSA Optimized Transformer	99.25%	99.75%	99.55%	99.64%
GA optimized Transformer	98.87%	99.65%	99.15%	99.40%
Transformer (unoptimized)	98.55%	99.25%	98.77%	99.01%
CSA-LSTM [27]	98.48%	97.05%	91.03%	93.95%
CSA-CNN-LSTM [27]	98.74%	93.33%	96.55%	94.91%
Random Forest [27]	92.64%	78.89%	59.31%	67.71%

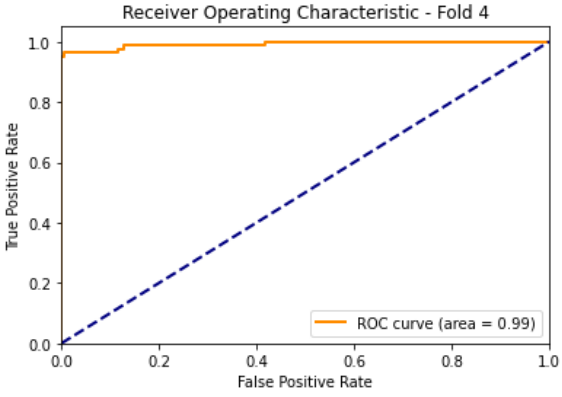

Fig. 9. AUC Curve for SMS-SPAM Dataset for the 4th fold.

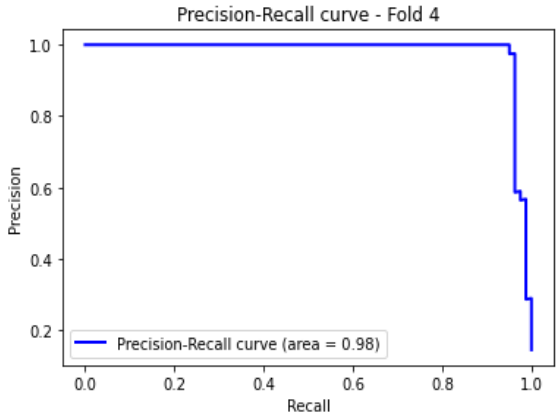

Fig. 10. AUPR Curve for SMS-SPAM Dataset for the 4th fold.

Table 7 shows the final topology found for the optimized transformer model for the SMS spam dataset. While the type of optimizer used in training remained the same, our research found better values for the other hyperparameters.

Table 7. The final optimized hyperparameters discovered by our proposed model for the spam dataset

Hyperparameter	optimized value	initialization value (unoptimized)
#Epochs	4	2
#Heads	2	8
hidden layer size in feed-forward layer	18	32
embedding size for each token	58	32
optimizer	ADAM	ADAM

3) *Performance Comparison for Twitter US Airline dataset*

Table 8 shows the comparison of accuracies of the various ML models for the Twitter US Airline data set. It shows that compared to the other optimization techniques, the mCSA-optimized transformer has higher performance in all of the metrics. Our unoptimized transformer consisted of the initialization values as shown in table IX. The optimized model exhibits superior performance over other algorithms such as CSA-LSTM, CSA_CNN_LSTM, and Random Forest. It also shows that the performance metrics of the unoptimized transformer are subpar compared to optimized LSTM networks. However, optimization of the hyperparameters yields a vastly different conclusion. When optimized, transformers have shown superior performance.

Table 8. Performance comparison of mCSA Transformer with other techniques for Twitter US airline dataset

Model	Accuracy	precision	recall	F1-score
mCSA Optimized Transformer	94.56%	95.67%	97.52%	96.58%
GA optimized Transformer	92.84%	94.67%	96.64%	95.64%
Transformer (unoptimized)	90.48%	87.58%	88.25%	87.91%
CSA-LSTM [27]	92.25%	94.06%	95.94%	94.99%
CSA-CNN-LSTM [27]	92.77%	94.50%	95.88%	95.19%
Random Forest [27]	83.32%	83.85%	89.05%	90.40%

Table 9. The final optimized hyperparameters discovered by our proposed model for the Twitter US Airline Dataset

Hyperparameter	optimized value	initialization value (unoptimized)
#Epochs	2	2
#Heads	6	8
hidden layer size in feed-forward layer	48	32
embedding size for each token	24	32
optimizer	ADAM	ADAM

Table 9 shows the new hyperparameters found by our mCSA algorithm for the US Twitter Airline dataset. The number of epochs has not changed from the initial unoptimized value. It means that for this Twitter US Airline dataset, training for 2 epochs yielded the best result without overfitting.

Figs. 12 and 13 show the AUC and AUPR curve for the US Twitter Airline dataset. Since this is a multiclass classification, the AUC and AUPR curves are plotted for each class individually. Classes 0, 1 and 2 refer to positive, neutral, and negative classification respectively.

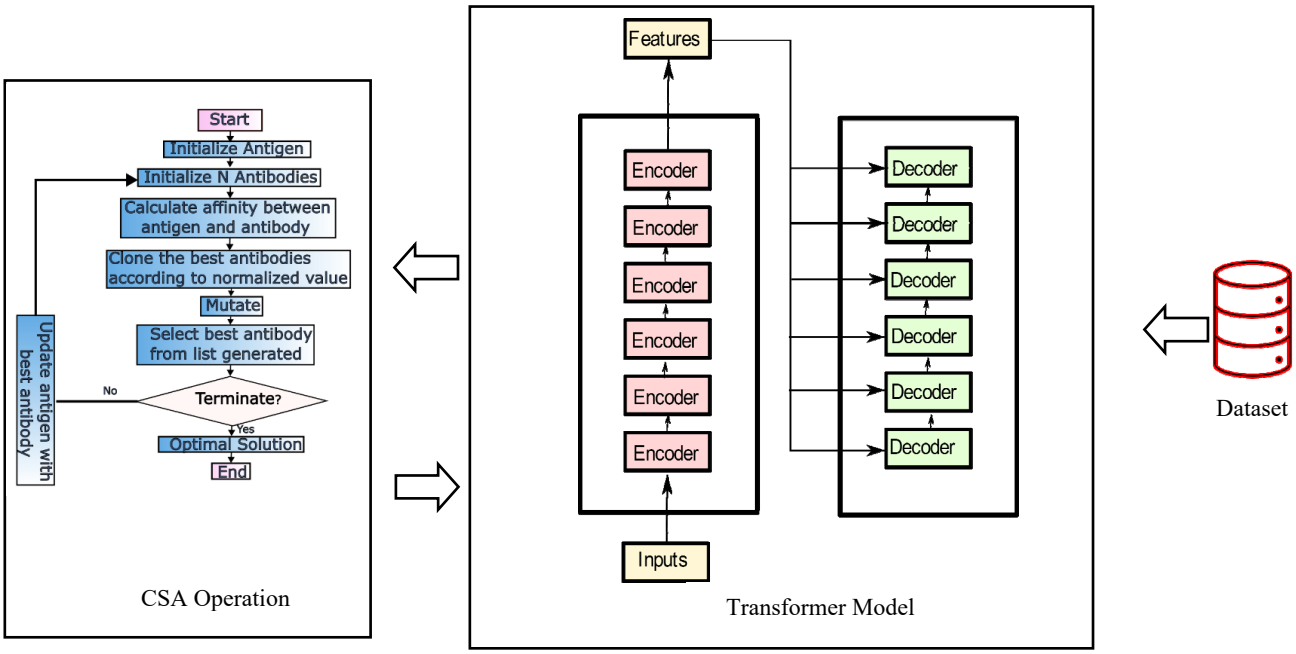

Fig. 11. Optimization process of transformers using CSA.

Table 10. Summary of accuracy comparison of mCSA transformer with other techniques for all datasets

Dataset/Model	MCSA-Transformer	GA-Transformer	Unoptimized Transformer	CSA-LSTM	CSA-CNN-LSTM	Random Forest
IMDB	90.48%	89.72%	88.56%	89.52%	89.88%	54.66%
SMS Spam	99.25%	98.87%	98.55%	98.48%	98.74%	92.64%
US Twitter Airline	94.56%	92.84%	90.48%	92.25%	92.77%	83.32%

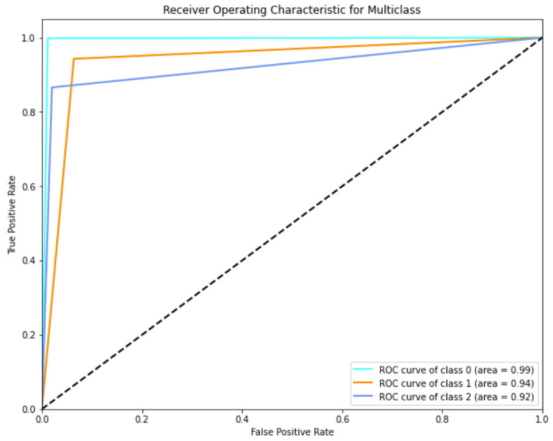

Fig. 12. AUC curve for US Twitter airline dataset.

VI. Ablation Study

To evaluate the role of different hyperparameters in improving the accuracy of the transformer model, we conducted ablation study by varying the optimal hyperparameter values for each case, keeping everything else constant. Fig. 14 shows our ablation study result. It is seen that the number of epochs, the size of hidden layer in feed forward layer, and the type of optimizer used impact the performance of our model significantly, compared to the number of heads and embedding size.

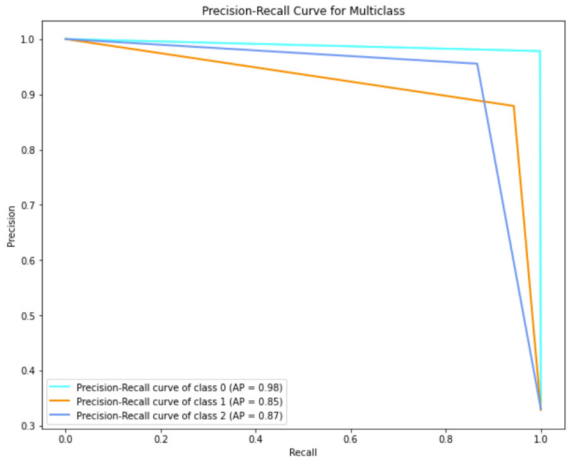

Fig. 13. AUPR curve for US Twitter airline dataset.

VII. Conclusion

Table X shows the comparative analysis of model accuracies among various other optimization algorithms and models that have been compared in this research. It is shown that MCSA-optimized transformer has the best accuracy. The performance of the original transformer has been highly improved by optimizing its architecture using evolutionary approaches to evolve its hyperparameters. Bio-inspired Modified Clonal Selection Algorithm (mCSA) was used for evolving the hyperparameters of transformers. In this paper, the optimized transformer architecture manifested an increase in the prediction accuracy to 90.48% from 88.56% for the IMDB dataset, an improvement to 99.25% from 98.55% for the SMS spam dataset, and an improvement to 94.56% from 90.48% for the US Twitter Airline dataset, using 10-fold cross-validation.

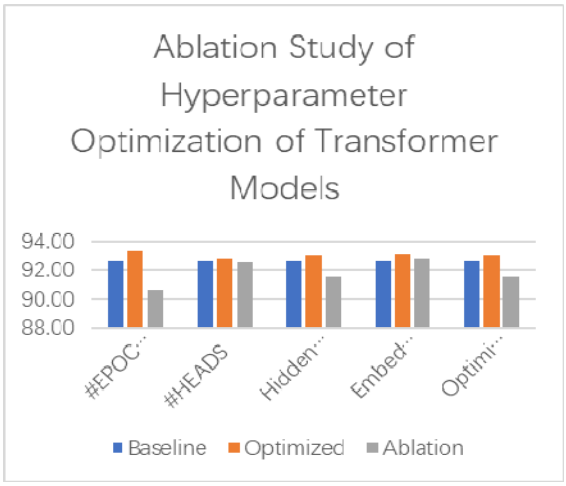

Fig. 14. Ablation study of our proposed model.

The effectiveness and utility of advanced machine learning models, such as the CSA-Transformer, extends beyond traditional data analysis and predictive tasks. First, accurately identifying spam messages can help reduce processing load, improve network efficiency, save storage space, and even help extend life of electronic components. From a sustainability perspective, properly identifying whether a message is spam or ham is analogous to identifying relevant and irrelevant energy data. In terms of energy optimization, this relates to accurately predicting energy usage and identifying potential areas of improvement. Ecologically disturbed locales, such as areas affected by deforestation, pollution, or climate change, require innovative and efficient energy management solutions. The proposed Transformer model can be used in energy applications such as predictive maintenance, renewable energy management, and adaptive energy policies.

A novel framework using mCSA has been proposed that simplifies the selection of various hyperparameters of transformers, including but not limited to the number of epochs to train, batch size, number of embeddings, number of heads, percentage of validation data, etc. In addition, using mCSA also saves time in comparison to other black box or random methods since mCSA is based on the survival of the fittest paradigm. In the future, other nature-inspired algorithms such as various variants of Swarm intelligence such as Grey Wolf Optimization or Cuckoo Search Algorithms can be used for optimizing the hyperparameters of transformers.

Conflict of Interest

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

Ashish Kharel conducted the research, carried out the experiments and wrote the paper; Devinder Kaur advised and guided the research, and helped in proofreading the paper; all authors had approved the final version.

REFERENCES

[1] V. Ashish, S. Noam, P. Niki, U. Jakob *et al.*, "Attention is all you need," arXiv:1706.03762 [cs.CL], 2017.

[2] S. B. Ł. Kaiser, "Can active memory replace attention?" *Advances in Neural Information Processing Systems*, 2016.

[3] A. Parikh, O. Täckström, D. Das, and J. Uszkoreit, "A decomposable attention model," *In Empirical Methods in Natural Language Processing*, 2016.

[4] O. Kuchaiev and B. Ginsburg, "Factorization tricks for LSTM networks," arXiv preprint arXiv:1703.10722, 2017

[5] P. I Frazier, "A tutorial on Bayesian optimization," arXiv preprint arXiv:1807.02811, 2018.

[6] P. W. Glynn *et al.*, "Stochastic optimization via grid search," *Lectures in Applied Mathematics-American Mathematical Society*, vol. 33, pp. 89–100, 1997.

[7] A. Hussain and S. A. Ludwig, "Hyperparameter optimization: comparing genetic algorithm against grid search and bayesian optimization," in *Proc. 2021 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2021.

[8] X. Tian and X. M. Chen, "A discrete hidden Markov model for SMS spam detection," *Applied Sciences*, vol. 10, no. 14, p. 5011, 2020.

[9] M. Kenton *et al.*, "Auto-sizing the transformer network: Improving speed, efficiency, and performance for low-resource machine translation," arXiv preprint arXiv:1910.06717, 2019.

[10] Y. T. Chen *et al.*, "Towards Learning Universal Hyperparameter Optimizers with Transformers," arXiv preprint arXiv:2205.13320, 2022.

[11] G. Daniel *et al.*, "Google vizier: A service for black-box optimization," in *Proc. the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.

[12] S. David *et al.*, "Syne tune: A library for large scale hyperparameter tuning and reproducible research," in *Proc. International Conference on Automated Machine Learning*, 2022.

[13] M. P. Ranjit *et al*., "Efficient deep learning hyperparameter tuning using cloud infrastructure: Intelligent distributed hyperparameter tuning with Bayesian optimization in the cloud," in *Proc. 2019 IEEE 12th International Conference on Cloud Computing*, IEEE, 2019.

[14] A. A. Bataineh and D. Kaur, "Immuno-computing-based neural learning for data classification," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 6, 2019. http://dx.doi.org/10.14569/IJACSA.2019.0100632

[15] A. A. Bataineh and D. Kaur, "Immunocomputing-Based Approach for Optimizing the Topologies of LSTM Networks," *IEEE Access*, vol. 9, pp. 78993-79004, 2021, doi: 10.1109/ACCESS.2021.3084131

[16] J. K. Chorowski *et al.*, "Attention-based models for speech recognition," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[17] H. Sepp and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] H. Sepp, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, 1998.

[19] Y. H. Hu *et al.*, "Overcoming the vanishing gradient problem in plain recurrent networks," arXiv preprint arXiv:1801.06105, 2018.

[20] S. Shibani *et al*., "How does batch normalization help optimization?" *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[21] H. H. Tan and K. H. Lim, "Vanishing gradient mitigation with deep learning neural network optimization," in *Proc. 2019 7th International Conference on Smart Computing & Communications (ICSCC)*. IEEE, 2019.

[22] V. Andreas, M. J. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[23] D. Dasgupta, "An overview of artificial immune systems and their applications," *Artificial Immune Systems and Their Applications*, Springer, pp. 3–21, 1993.

[24] W. Chong and K. Q. Huang, "How to use bag-of-words model better for image classification," *Image and Vision Computing*, vol. 38, 2015, pp. 65–74.

[25] M. Long and Y. Q. Zhang, "Using Word2Vec to process big text data," in *Proc. 2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 2015.

[26] D. Jesse and M. Goadrich, "The relationship between Precision-Recall and ROC curves," in *Proc. the 23rd International Conference on Machine Learning*, 2006.

[27] G. Aurélien, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*, O'Reilly Media, Inc, 2022.

[28] A. A. Bataineh and D. Kaur, "Immunocomputing-based approach for optimizing the topologies of LSTM networks," *IEEE Access*, vol. 9, pp. 78993–79004, 2021, doi: 10.1109/ACCESS.2021.3084131.