Journal of Theoretical and Applied Information Technology
15th January 2024. Vol.102. No 1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ELEVATING SENTIMENT ANALYSIS WITH RESILIENT GREY WOLF OPTIMIZATION-BASED GAUSSIAN-ENHANCED QUANTUM DEEP NEURAL NETWORKS IN ONLINE SHOPPING

G.M. BALAJI[1], K. VADIVAZHAGAN[2]

[1]Research Scholar & Assistant Professor, Department of Computer and Information Science, Annamalai University, Chidambaram, Tamilnadu, India.
[2] Assistant Professor, Department of Computer and Information Science, Annamalai University, Chidambaram, Tamilnadu, India.
E-mail: [1]gmbalaji0813@gmail.com, [2]vadivazhagan.k@gmail.com

ABSTRACT

The rise of online shopping reflects a significant change in consumer behavior, with more people drawn to digital marketplaces due to unparalleled convenience, extensive product variety, and competitive pricing offered by online platforms. Product reviews have become a cornerstone within this digital retail landscape, offering invaluable guidance to customers and business proprietors. Shoppers rely on reviews to make well-informed decisions, gaining insights into product quality and functionality, while entrepreneurs utilize this feedback to refine their offerings and elevate customer satisfaction. The analysis of sentiments embedded within product reviews presents formidable challenges due to the intricacies of human language and the sheer volume of data. To address this tough challenge, this paper introduces Resilient Grey Wolf Optimization-based Gaussian-Enhanced Quantum Deep Neural Networks (RGWO-GEQDNN). This novel approach amalgamates the robust, Resilient Grey Wolf Optimization with Gaussian-enhanced Quantum Deep Neural Networks, providing a potent solution for efficient and accurate sentiment analysis within product reviews. RGWO-GEQDNN emphasizes the innovative fusion of nature-inspired optimization and quantum computing principles, promising a breakthrough in sentiment analysis. To assess the performance of RGWO-GEQDNN against state-of-the-art algorithms, Amazon product review dataset is utilized. The results underscore the superiority of RGWO-GEQDNN in accurately classifying sentiment from product reviews, highlighting its transformative potential in the e-commerce landscape.
Keywords: *Sentiment, Reviews, Online Shopping, Classification, Amazon, Neural Network*

1. INTRODUCTION

Online shopping has come a long way since its inception, with evolving e-commerce trends shaping how we buy goods and services. In the ever-advancing landscape of technology and shifting consumer preferences, the online shopping experience continues to transform [1]. One noteworthy trend in online shopping is the remarkable growth of mobile commerce. With the widespread use of smart phones, consumers can shop online, making purchases through mobile apps and responsive websites, ensuring a seamless and convenient shopping experience from virtually anywhere. Personalization is another substantial trend, as retailers utilize data-driven algorithms to customize individual shoppers' recommendations, offers, and product suggestions, enhancing the shopping experience and overall customer satisfaction[2], [3]. In addition to these trends, sustainability is rising in online shopping, with consumers increasingly seeking eco-conscious products and sustainable practices from e-commerce platforms. Retailers respond by introducing sustainable product lines and reducing their environmental footprint through responsible shipping and packaging choices. The future of online shopping is poised to incorporate augmented reality (AR) and virtual reality (VR) experiences, enabling consumers to virtually try on clothing or visualize furniture within their living spaces. These emerging trends promise a more interactive, personalized, and eco-friendly online shopping experience, ensuring that consumers continue to enjoy a dynamic and ever-improving retail environment [4].

In the era of extensive data and digital communication, sentiment analysis has become

essential for unraveling collective opinions, emotions, and attitudes communicated in vast quantities of textual data. Driven by natural language processing and machine learning, this technology empowers us to access valuable insights into public perceptions and reactions concerning various topics, products, and events [5]. Sentiment analysis transcends the mere classification of sentiments as positive, negative, or neutral, offering a nuanced comprehension of the intensity and context of emotions. Its applications are diverse, with significant roles in social media monitoring, customer feedback evaluation, and brand reputation management [6], [7]. By allowing organizations to monitor public sentiment, identify trends, and proactively address emerging concerns, sentiment analysis ensures that businesses make data-driven decisions and that researchers gain a comprehensive understanding of the ever-evolving landscape of human sentiment in the digital age[8].

While fraught with challenges, Sentiment analysis brings significant advantages to customers and companies. Language intricacies, context, and cultural nuances pose challenges, but they can be addressed through advancements in machine learning and deep learning models [9]. Customers benefit from personalized recommendations, improved customer support, and the ability to voice their opinions effectively. On the corporate side, sentiment analysis unlocks insights into customer feedback and market trends, enabling data-driven decision-making, brand reputation management, and competitive analysis. By addressing these challenges, sentiment analysis delivers mutual gains for customers and companies, fostering a symbiotic relationship built on understanding and responsiveness [10], [11].

1.1. Problem Statement

The challenge of dealing with noisy data in sentiment analysis is a persistent and complex problem that significantly hampers the accuracy and reliability of sentiment classification models. Noisy data encompasses a variety of issues, including typographical errors, grammatical inaccuracies, slang, jargon, and informal language. These data imperfections pose substantial obstacles in accurately gauging sentiment, as they introduce ambiguity and confusion into the analysis process. Noisy data leads to misclassifications and erroneous sentiment interpretations, distorting the true sentiment expressed in a text. This problem is exacerbated in user-generated content on social media platforms, where informal language and brevity are common. It challenges sentiment analysis models to differentiate between genuine sentiment expressions and text that deviates from standard language conventions. Addressing the noisy data challenge is crucial for developing more robust sentiment analysis models. Improving the accuracy of sentiment classification by effectively filtering and preprocessing noisy data is an ongoing area of research in natural language processing. This problem statement underscores the critical need to develop techniques that can enhance the reliability of sentiment analysis in the face of noisy and unstructured textual data.

1.2. Motivation

The motivation to address the challenge of noisy data in sentiment analysis stems from its profound impact on the reliability and accuracy of sentiment classification models. In a world where user-generated content on social media platforms and the digital sphere is ubiquitous, understanding public sentiment is essential for businesses, organizations, and researchers. Noisy data, characterized by typographical errors, grammatical inaccuracies, informal language, and more, introduces ambiguity and misclassification, jeopardizing the insights derived from sentiment analysis. Overcoming this challenge is imperative to provide accurate and actionable results for decision-making, brand reputation management, customer service improvement, and more. As sentiment analysis expands into diverse domains, from customer feedback to market trend analysis, the need to enhance the precision of sentiment classification in the face of noisy data becomes increasingly evident. Ultimately, addressing the challenge of noisy data in sentiment analysis is not just a technical endeavor but a strategic imperative for harnessing the full potential of this valuable tool.

1.3. Objectives

This research endeavor aims to develop a bio-inspired optimization-based deep learning classifier for sentiment analysis, specifically tailored to mitigate the challenges posed by noisy data. Building upon the motivations outlined earlier, this novel approach aims to enhance the accuracy and reliability of sentiment analysis by effectively filtering and preprocessing noisy textual data. By integrating bio-inspired optimization techniques with deep learning models, this research seeks to differentiate genuine sentiment expressions from language variations, ensuring that sentiment analysis provides precise and actionable results. The primary goal is to address the complex issue of noisy data, which substantially impacts sentiment analysis

accuracy, particularly in user-generated content on social media platforms. This innovative classifier is envisioned to contribute to the evolution of sentiment analysis, making it more robust and adaptable in real-world, unstructured textual data, ultimately enabling businesses, organizations, and researchers to make more informed and data-driven decisions.

2. LITERATURE REVIEW

"Fuzzy Sentiment in Tweets" [12] tackles the specific challenges of analyzing fuzzy sentiment within tweets. The model combines textual features, such as sentiment lexicons, word embeddings, and syntactic structures, to effectively capture and classify ambiguous sentiment expressions. The Feature Ensemble Model utilizes machine learning and statistical techniques to adaptively weigh and combine these features. "Cross-lingual Aspect-Based Sentiment" [13] introduces a novel approach to address the challenges of cross-lingual aspect-based sentiment analysis. It leverages contrastive learning techniques to align sentiment representations across different languages, enabling the model to understand and classify aspect-based sentiments across language barriers. This involves developing language-agnostic sentiment embeddings and aligning them using contrastive loss functions. "Recursive and Recurrent Networks for Aspect-Based Sentiment" [14] presents a sophisticated approach that combines Recursive and Recurrent Neural Networks (RNNs) to perform Aspect-Based Sentiment Analysis with a particular focus on inter-aspect relations. The model uses the recursive structure to capture relationships between aspects and utilizes recurrent connections to capture sequential dependencies within each aspect.

"Semantic Sentiment Analysis" [15] advances the field of sentiment analysis by introducing a Semantic Conceptualization using Tagged Bag-of-Concepts. The critical scientific contribution lies in integrating semantic tags and conceptual knowledge into sentiment analysis. This approach significantly enhances the contextual understanding of sentiment within text data, paving the way for more precise sentiment analysis. "Summarizing Student Survey Responses" [16] offers a scientific breakthrough in efficiently analyzing open-ended student survey responses. The scientific merit of this research lies in its novel approach to summarizing unstructured survey data. Cluster analysis contributes by grouping similar responses based on common themes, and sentiment analysis assesses the emotional tone within these clusters. "Urdu Sentiment Analysis with Deep Learning" [17] advances the scientific understanding of sentiment analysis in underrepresented languages, such as Urdu, by applying deep learning algorithms. The scientific significance lies in exploring deep learning's capabilities in capturing sentiment nuances in a language with limited resources.

"Attention-Emotion-Enhanced Sentiment Analysis" [18] lies in developing the Attention-Emotion-Enhanced Convolutional LSTM model. It leverages the latest advancements in deep learning and sentiment analysis by incorporating attention mechanisms to enhance information extraction and emotional context for sentiment analysis. This scientific innovation enables accurate sentiment classification by focusing on crucial text segments and understanding emotional cues. "Broad Multitask Transformer Network" [19] is developing a multitask transformer network to streamline sentiment analysis across diverse tasks and domains. It is scientifically significant due to its adaptability and versatility. The model is based on the transformer architecture, representing a scientific breakthrough in natural language processing. "Dynamic Bayesian Network" [20] offers a novel perspective on sentiment analysis. DBNs enable modeling the evolution of topics and sentiments in textual data over time. This technical innovation allows for the dynamic tracking of topic-sentiment dynamics, providing a more sophisticated understanding of how sentiments evolve in response to changing topics.

"Challenges in Aspect-based Sentiment Analysis" [21] provides an extensive and technically detailed overview of the issues and challenges encountered in aspect-based sentiment analysis. It delves into the intricacies of this specialized field, including data sparsity, aspect identification, and context modeling. "Efficient Adaptive Transfer Network for Aspect-Level Sentiment" [22] designed for aspect-level sentiment analysis. The scientific and technical merit lies in the model's adaptability to varying aspects of text data. EATN optimizes the aspect-level sentiment analysis process by efficiently transferring knowledge and adapting to different aspects. "Weight Distributing Method for Text Sentiment Analysis" [23] investigates the application of a Weight Distributing Method that combines sentiment dictionaries and TF-IDF for text sentiment analysis. The technical advancement is in the method's ability to enhance sentiment classification precision. Systematically weighing the

impact of sentiment words based on their relevance to the document's content provides a more refined and accurate sentiment analysis.

Significant bio-inspired algorithms [25]-[42] also plays an important role in classifications.

3. RESILIENT GREY WOLF OPTIMIZATION - BASED GAUSSIAN - ENHANCED QUANTUM DEEP NEURAL NETWORKS

3.1.1. Define the Problem

In machine learning and quantum deep neural networks, it's essential to start by clearly defining the problem to be solved. This initial step involves specifying the nature of the task, the data, and the objective. To determine the problem mathematically, G-QDNN begins by describing the task at hand. G-QDNN aims to model the relationship between inputs and continuous-valued outputs in regression. Mathematically, the task can be expressed as Eq.(1).

$$\boldsymbol{Y} = f(X) + \varepsilon \quad (1)$$

where Y represents the continuous-valued output or target variable, X denotes the input features or predictors, f(·) represents an unknown function that relates the inputs to the outputs, and ε is the random error term.

In Eq.(2), G-QDNNs ensure the data is compatible with continuous-variable quantum states. The data representation typically consists of a set of input-output pairs, denoted as (x_i, y_i) for i = 1,2,…,N, where x_i represents the input data for the i-th instance, and y_i is the corresponding target value. It's essential to ensure that the input features are continuous and can be encoded in a quantum-friendly manner. The data should be preprocessed and normalized to meet these requirements.

$$x_i = (x_{i1}, x_{i2}, \dots, x_{im}) \quad (2)$$

where m is the number of input features.

Once the task and data are defined, G-QDNN needs to specify an objective function that quantifies the error, or the objective aims to minimize the same. For regression problems, a standard objective function is the mean squared error (MSE). Mathematically, it is defined as Eq.(3).

$$\boldsymbol{C}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad (3)$$

where C(θ) represents the cost or objective function to be minimized, N is the number of data points, y_i is the actual target value, $\hat{y}_i$ is the predicted value, which depends on the parameters θ of the model.

3.1.2. Data Preparation

Data preparation is a crucial step in the development of a G-QDNN. This step ensures that the input data is well-structured, compatible with the quantum framework, and ready for further processing.

In G-QDNNs, the data typically consists of a set of input data points, denoted as x_i , where i ranges from 1 to N. Each x_i is a m-dimensional vector representing m continuous variables. G-QDNN express data preparation as Eq.(4).

$$x_i = (x_{i1}, x_{i2}, \dots, x_{im}), \quad (4)$$

where m represents the number of continuous features in the dataset.

Data preprocessing is essential to ensure the data is well-suited for quantum processing. Common preprocessing steps include normalization and feature scaling. Eq.(5) adjusts the data such that its mean is zero and its standard deviation is one.

$$x_{ij} \leftarrow \frac{x_{ij} - \mu_j}{\sigma_j}, \quad (5)$$

where x_{ij} is the j-th feature of the i-th data point, μ_j is the mean of the j-th feature, and σ_j is the standard deviation of the j-th feature. Having all characteristics normalized to the same scale for quantum circuits is helpful.

G-QDNNs operate with Gaussian quantum states, which are well-suited for representing continuous variables. The steady quantum states can be defined as Eq.(6).

$$\boldsymbol{\rho}(x_i) = \frac{1}{(\pi\hbar)^{m/2}} exp\left(-\frac{1}{2h}\sum_{j=1}^{m} x_{ij}^2\right) \quad (6)$$

where, $\rho(x_i)$ represents the quantum state associated with the i-th data point. The parameter $\hbar$ represents the effective Planck constant, which influences the spread of the quantum state.

To prepare the data for quantum processing, G-QDNN can encode the preprocessed x_i into quantum states using the density matrix formalism. In Eq.(7), encoding takes the normalized feature vectors and maps them into quantum states compatible with a G-QDNN. The quantum state $\rho(x_i)$ encapsulates the information from the original data point and can be used as input for the G-QDNN.

$$\boldsymbol{\rho}(x_i) = \frac{1}{(\pi\hbar)^{m/2}} exp\left(-\frac{1}{2h}\sum_{j=1}^{m} x_{ij}^2\right) \quad (7)$$

Algorithm 1 provides the pseudocode of the data preparation.

Algorithm 1: Data Preparation

Input:
- Input dataset containing N data points, each represented as m-dimensional vectors:

Output:
- A preprocessed dataset suitable for G-QDNN processing.

Procedure:
1. Normalize the input data:
   - For each feature x_{ij} in each data point x_i, calculate the mean μ_j and standard deviation σ_j of that feature across all data points.
   - Update each feature x_{ij} as follows: $x_{ij} \leftarrow \frac{x_{ij}-\mu_j}{\sigma_j}$
2. Encode the preprocessed data into continuous quantum states:
   - For each preprocessed data point x_i, encode it into a continuous quantum state $\rho(x_i)$ suitable for G-QDNN processing.
3. Preprocess the dataset to hold continuous quantum states.

3.1.3. Quantum Circuit

In G-QDNNs, defining the quantum circuit is a pivotal step. The quantum circuit serves as the core element of the G-QDNN, responsible for mapping input data to quantum states and, eventually, generating predictions.

G-QDNNs rely on continuous-variable quantum states, which are well-suited for encoding continuous data. These quantum states can be mathematically described as Eq.(8) (i.e., density matrices). For a single quantum state corresponding to an input data point x_i, the continuous-variable quantum state $\rho(x_i)$ is expressed as Eq.(8).

$$\rho(x_i) = \frac{1}{(\pi\hbar)^{m/2}} exp\left(-\frac{1}{2h}\sum_{j=1}^{m} x_{ij}^2\right) \quad (8)$$

where $\rho(x_i)$ is the density matrix representing the quantum state associated with the i-th data point, $\hbar$ is the effective Planck constant, influencing the quantum state's spread, m is the number of continuous features in the dataset, and x_{ij} represents the j-th feature of the i-th data point.

The quantum circuit within a G-QDNN typically involves Gaussian modes and gates. Gaussian modes represent continuous-variable quantum systems, and gates are operations applied to these modes. Gaussian modes are described as continuous-variable operators in Eq.(9), which can be denoted as $\hat{a}_i$ and $\hat{a}_i^\dagger$ for the i-th mode. These operators satisfy commutation relations:

$$[\hat{a}_i, \hat{a}_i^\dagger] = \delta_{ij}, \quad [\hat{a}_i, \hat{a}_j] = [\hat{a}_i^\dagger, \hat{a}_j^\dagger] = 0, \quad (9)$$

where δ_{ij} is the Kronecker delta [24].

Quantum gates within the G-QDNN manipulate Gaussian modes and their operators. These gates can be mathematically expressed in terms of the operators, including displacement gates, squeezing gates, rotation gates, and more. The effect of a quantum gate on a Gaussian mode can be represented as a transformation in the mode operators, such as $\hat{a}_i$ and $\hat{a}_i^\dagger$. The quantum circuit for a G-QDNN is constructed by composing a sequence of these gates. The parameters of these gates are usually trained during the optimization process to learn the most suitable quantum operations for the given task.

The continuous-variable quantum circuit is denoted as $\hat{U}(\theta)$, is a parameterized sequence of quantum gates. It operates on the continuous-variable quantum states, transforming the input data into a quantum state that encodes information relevant to the task. Eq.(10) mathematically defines the quantum circuit.

$$\rho'(X_i) = \hat{U}(\theta)\rho(X_i)\hat{U}^\dagger + \theta \quad (10)$$

where θ represents the parameters of the quantum circuit and ρ'(X_i)is the quantum state after applying the quantum circuit.

Algorithm 2: Quantum Circuit

Input:
- Parameters θ that define the quantum circuit structure.

Output:
- A parameterized quantum circuit $\hat{U}(\theta)$

Procedure:
1. Initialize an empty quantum circuit $\hat{U}(\theta)$.
2. Define the structure of the quantum circuit, including the number and type of gates, gate

sequences, and connections between Gaussian modes.
3. Parameterize the gates by assigning values to the parameters $\boldsymbol{\theta}$ within the circuit.
4. Optimize the parameters before training.

3.1.4. Quantum Hybrid Model

The Quantum Hybrid Model is a fundamental component in G-QDNNs. It combines classical and quantum elements to create a powerful machine-learning model. At the core of the Quantum Hybrid Model is a classical neural network, which is responsible for handling classical data and performing classical operations. The classical neural network can be represented as a series of layers with associated weights and activation functions. For a single-layer classical neural network, the output z_i can be calculated using Eq.(11).

$$z_i = f\left(\sum_{j=1}^{n} w_{ij}x_j + b_i\right) \quad (11)$$

where z_i is the output of a neuron, w_{ij} represents the weight connecting the input x_j to neuron i, b_i is the b_i as term for neuron i, $f(\cdot)$ is the activation function, such as the sigmoid, ReLU, or any other suitable function.

The Quantum Circuit, as defined in Section 3.1.3, is the quantum component of the hybrid model. It processes continuous-variable quantum states and performs quantum operations. The quantum circuit $\hat{U}(\theta)$ is responsible for encoding classical data into quantum states and conducting quantum transformations. The quantum circuit operates on the density matrix $\rho(X_i)$ for a given input data point x_i as expressed in Eq.(12).

$$\rho'(X_i) = \hat{U}(\theta)\rho(X_i)\hat{U}^{\dagger}\theta. \quad (12)$$

where, $\rho'(X_i)$ represents the quantum state after applying the quantum circuit. The parameters θ are trainable and determine the circuit's behaviour.
The Quantum Hybrid Model fuses traditional neural networks with quantum computing. The output of the quantum circuit $\rho'(X_i)$ is integrated with the classical neural network's output. The integration is expressed in Eq.(13).

$$\boldsymbol{O}(x_i) = g\left(\sum_{i=1}^{N_c} z_i + Tr(\rho'(X_i)M)\right), \quad (13)$$

where $O(x_i)$ represents the model's prediction for input x_i, $g(\cdot)$ is an activation function applied to the combined output, N_c is the number of neurons in the classical neural network, z_i are the outputs of the classical neurons, $Tr(\rho'(X_i)M)$ represents the trace of the product of the quantum state $\rho'(X_i)$ and a measurement operator M.

The hybrid model is trained through a combined quantum optimization process. The classical neural network parameters (weights and biases) are optimized using classical optimization algorithms (e.g., gradient descent). In contrast, the quantum circuit parameters θ are optimized using quantum optimization methods or hybrid approaches. The objective function minimized during training typically includes classical and quantum components and the entire model is trained to reduce the overall cost function.

Algorithm 3: Quantum Hybrid Model

Input:
- Classical neural network parameters (weights and biases).
- Quantum circuit parameters (θ).
- Input data x_i.

Output:
- Model prediction for input x_i .

Procedure:
1. Feed the input data $x_i$into the classical neural network to compute classical outputs z_i using the classical parameters.
2. Apply the quantum circuit $\hat{U}(\theta)$ to the input data $x_i$to create a quantum state $\rho'(x_i)$based on the quantum parameters $\boldsymbol{\theta}$.
3. Combine the classical outputs $z_i$with the quantum state $\rho'(x_i)$ to produce a model prediction $\boldsymbol{O}(x_i)$ using an activation function.
4. The model prediction $\boldsymbol{O}(x_i)$ is the output for the input data (x_i) , representing the result of the Classical-Quantum Hybrid Model.

3.1.5. Parameterization

Parameterization is a fundamental aspect of G-QDNNs, as it defines the configurations of both the classical and quantum components. The parameterization sets the starting values for the parameters, which will be fine-tuned during training. G-QDNN will focus on the mathematical aspects of this step. In G-QDNNs, classical neural network parameters include weights (W) and biases (b) for the classical component. These parameters are initially set with random or predefined values before training begins.

The weight matrix W connects the inputs of the classical neural network to its neurons. It's an $N_c \times N$ matrix, where N_c is the number of neurons in the classical network, and N is the number of input features. A normal distribution with a zero mean and

a tiny standard deviation can be used to generate random variables to use as W starting points.

$$\boldsymbol{W} = \begin{bmatrix} w_{11}w_{12} & \cdots & w_{1N} \\ w_{21}w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N_c1}w_{N_c2} & \cdots & w_{N_cN} \end{bmatrix}. \quad (14)$$

The bias vector b corresponds to the biases for each neuron in the classical neural network. It's a N_c dimensional vector, and Eq.(15) initialize it with small random values.

$$\boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{N_c} \end{bmatrix} \quad (15)$$

The parameter vector θ contains all the parameters for the quantum circuit. These parameters determine the settings for the quantum gates and operations. Eq.(16) includes displacement parameters, squeezing parameters, and other gate-specific parameters, depending on the architecture of the quantum circuit.

$$\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_P), \quad (16)$$

where P is the total number of quantum circuit parameters.

The choice of how to initialize these parameters can impact the training and convergence of the G-QDNN. For initial parameterization, G-QDNN generally uses weight initialization and quantum circuit parameter initialization.

(a). Weight Initialization

Randomly initialize the weight matrix W and the bias vector b for the classical neural network using random values:

$$\boldsymbol{W} \sim N(0, \sigma^2), \quad b \sim N(0, \sigma^2). \quad (17)$$

where σ represents the standard deviation of the normal distribution.

(b). Quantum Circuit Parameter Initialization

Set the initial values within predefined ranges or distributions for the quantum circuit parameters.

$$\theta_i \sim Distribution\ (a, b), \text{ for } \boldsymbol{i} = 1,2, \ldots, P \quad (18)$$

where Distribution (a, b) represents a specific distribution with parameters a and b to control the initial values.

The initial parameterization sets the starting conditions for the G-QDNN. During training, these parameters are adjusted to minimize the cost function, leading to a well-tailored model that can effectively represent the data and perform quantum-enhanced machine learning tasks.

Algorithm 4: Parameterization

Input:
- Number of classical neurons N_c.
- Several input features $\boldsymbol{N}$.
- The total number of quantum circuit parameters $\boldsymbol{P}$.

Output:
- Initial parameter values for the classical neural network (weights and biases) and the quantum circuit.

Procedure:
1. Initialize the classical neural network parameters:
   - Create a weight matrix $\boldsymbol{W}$ of dimensions $N_c \times N$.
   - Initialize $\boldsymbol{W}$ with small random values.
   - Create a bias vector $\boldsymbol{b}$ of dimension N_c.
   - Initialize $\boldsymbol{b}$ with small random values.
2. Initialize the quantum circuit parameters:
   - Create a parameter vector $\boldsymbol{\theta}$ of length $\boldsymbol{P}$.
   - Initialize the elements of $\boldsymbol{\theta}$ with random values within predefined ranges or distributions.

3.1.6. Cost Function

The cost function is central in training a G-QDNN, quantifying the error between the model's predictions and the true target values. By minimizing the cost function, the G-QDNN optimizes its parameters, enabling it to make accurate predictions. G-QDNN will discuss the mathematical aspects of the cost function in the context of G-QDNNs.

In G-QDNN, the classical neural network is responsible for handling classical data. The cost function for the classical component, often referred to as the classical cost, is typically expressed as Eq.(19).

$$C_{classical}(W, b) = \frac{1}{N}\sum_{i=1}^{N} L\big(y_i, f(x_i)\big), \quad (19)$$

where $C_{classical}(W,b)$ represents the classical cost function, W denotes the weight matrix of the classical neural network, b represents the bias vector, N is the number of data points, y_i is the true target value for the i-th data point, $f(x_i)$) is the prediction

made by the classical component for the *i*-th data point, a loss function denoted by $L(y_i, f(x_i))$, measures how much predictions deviate from reality. Mean squared error (MSE) is a widespread loss function for regression, whereas cross-entropy is used for classification.

The quantum component in a G-QDNN involves a quantum circuit that processes continuous-variable quantum states. The quantum cost function measures the discrepancy between the model's predictions and the true target values. For regression tasks, the quantum cost is often formulated as Eq.(20)

$$C_{quantum}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \| y_i - \langle \psi(x_i,\theta)|M|\psi(x_i,\theta)\rangle \|^2, \quad (20)$$

where $C_{quantum}(\theta)$ represents the quantum cost function, θ includes the quantum circuit parameters, N is the number of data points, y_i is the true target value, $\langle \psi(x_i,\theta)|M|\psi(x_i,\theta)\rangle$ denotes the expected value of a measurement operator M on the quantum state $\psi(x_i,\theta)$ produced by the quantum circuit.

The overall cost function for a G-QDNN combines classical and quantum costs to create a unified cost function, expressed in Eq.(21).

$$C(W,b,\theta) = C_{classical}(W,b) + \alpha C_{quantum}(\theta). \quad (21)$$

where $C(W, b, \theta)$ represents the total cost function for the G-QDNN, $C_{classical}(W,b)$ is the classical cost, $C_{quantum}(\theta)$ is the quantum cost, α is a hyperparameter that balances the contributions of the classical and quantum components. It determines the relative importance of the quantum and classical costs during training. The value of α can be adjusted to control the trade-off between the classical and quantum parts.

During training, the goal is to minimize the total cost function $C(W, b, \theta)$ by adjusting the classical neural network parameters (W and b) and the quantum circuit parameters (θ). This process is typically performed using optimization algorithms, such as gradient descent, that update the parameters in the direction that reduces the cost function. The gradients concerning the parameters are computed through back propagation for the classical neural network and quantum optimization techniques for the quantum circuit.

Algorithm 5: Cost Function

Input:
- Neural network parameters (W, b).
- Quantum circuit parameters (θ).
- Training dataset with input features x_i and true target valuesy_i .
- Hyperparameter $\boldsymbol{\alpha}$ for balancing classical and quantum costs.

Output:
- Total cost $\boldsymbol{C}(W, b, \theta)$ for the G-QDNN.

Procedure:
1. Calculate the Classical Cost $(C_{classical})$:
   - Use the classical neural network with parameters (W, b) to make predictions $\boldsymbol{f}(x_i)$for each input x_i in the training dataset.
   - Compute the classical cost $C_{classical}(W, b)$ using a suitable loss function, comparing the predictions to the true target values y_i.
2. Calculate the Quantum Cost $(C_{quantum})$:
   - For each input x_i in the training dataset, apply the quantum circuit with parameters $\boldsymbol{\theta}$ to produce a quantum state $\boldsymbol{\psi}(x_i, \theta)$.
   - Measure the quantum state using a specified measurement operator $\boldsymbol{M}$ to obtain an expected value $\langle \psi(x_i,\theta)|M|\psi(x_i,\theta)\rangle$.
   - Calculate the quantum cost $C_{quantum}(\theta)$ by quantifying the difference between the expected values and the true target values y_i.
3. Compute the Total Cost $(C(W, b, \theta))$:
   - Combine the classical cost $C_{classical}(W, b)$and the quantum cost $C_{quantum}(\theta)$ to form the total cost $\boldsymbol{C}(W, b, \theta)$ using the hyperparameter $\boldsymbol{\alpha}$ to balance their contributions:
4. Incorporate both classical and quantum components.

3.1.7. Quantum Circuit Optimization

In G-QDNNs, the parameters of the quantum circuit are adjusted to improve its performance. Before optimizing the quantum circuit, G-QDNN needs a quantum cost function that quantifies the error between the quantum model's predictions and the true target values. In the context

of G-QDNNs, the quantum cost function is often expressed as Eq.(22).

$$C_{quantum}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \|y_i - \langle\psi(x_i,\theta)|M|\psi(x_i,\theta)\rangle\|^2, \quad (22)$$

where $C_{quantum}(\theta)$ represents the quantum cost function, θ includes the quantum circuit parameters, N is the number of data points, y_i is the true target value, $\langle\psi(x_i,\theta)|M|\psi(x_i,\theta)\rangle$ denotes the expected value of a measurement operator M on the quantum state $\psi(x_i,\theta)$ produced by the quantum circuit.

Gradient-based optimization for quantum circuits involves calculating the gradients of the cost function concerning the parameters. The gradient $\nabla_\theta C_{quantum}$ represents how the cost function changes as each parameter ϑ_i is varied. Eq.(23) expresses gradient operation.

$$\nabla_\theta C_{quantum} = \frac{2}{N}\sum_{i=1}^{N} (y_i - \langle\psi(x_i,\theta)|M|\psi(x_i,\theta)\rangle)\,\nabla_\theta \langle\psi(x_i,\theta)|M|\psi(x_i,\theta)\rangle \quad (23)$$

This gradient guides the updates to the quantum circuit parameters in a direction that minimizes the cost function. The parameters of the quantum circuit, θ, are updated in each optimization step. Eq.(24) typically expresses the update rule.

$$\theta_{new} = \theta_{old} - \eta\nabla_\theta C_{quantum}, \quad (24)$$

where θ_{new} represents the updated parameters, θ_{old} represents the current parameters, and η is the learning rate, controlling the size of the parameter updates. It is a hyperparameter that can be adjusted.

Algorithm 6: Quantum Circuit Optimization

Input:
- Quantum circuit with initial parameters θ_{old}.
- Training dataset with input features x_i and true target values y_i .
- Hyperparameters: learning rate(η), maximum iterations, and convergence criteria.

Output:
- Optimized quantum circuit with updated parameters θ_{new} .

Procedure:
1. Initialize the quantum circuit parameters:
   - Set θ_{old} with initial values.
   - Initialize iteration counter iter to 0.
2. Perform optimization iterations:
   - Increment iter by 1.
   - Calculate the quantum cost $C_{quantum}(\theta_{old})$ using the current quantum circuit.
   - Update the parameters:
   - Check the convergence criteria. If the cost function converges or the maximum number of iterations is reached, exit the loop.
   - Otherwise, set θ_{old} to θ_{new} and repeat the optimization iteration.
   - Compute the gradient $\nabla_\theta C_{quantum}$ of the quantum cost concerning the parameters at θ_{old} .

3.2. Resilient Grey Wolf Optimization

3.2.1. Initialization

Initialization is the foundational step in RGWO, where the initial population of grey wolves is created. In RGWO, grey wolves represent potential solutions to the optimization problem. This step is crucial, as the quality and diversity of the initial population can significantly impact the algorithm's performance. A well-chosen initialization strategy can lead to faster convergence and better solution space exploration. The initialization process in RGWO typically involves randomly generating the initial positions for a population of grey wolves. Let's delve into this crucial step mathematically.

Consider a population of grey wolves, represented as D-dimensional vectors, where D represents the problem space's dimensionality. Therefore, the initialization step aims to generate n initial solutions, each comprising D coordinates. These solutions are usually denoted as $X=\{x_1,x_2,.....,x_n\}$ where x_i is a D-dimensional vector. Eq.(25) represents the same.

$$x_i = [x_{i,1}, x_{i,2}, \dots, x_{i,D}], \text{ for } \boldsymbol{i} = 1,2,\dots,n \quad (25)$$

where, x_i stands for the i-th grey wolf's coordinates and x_{ij} stands for those coordinates in the j-th iteration.

The key aspects to consider during initialization are domain constraints and random initialization.

Domain Constraints ensure that the initial positions of grey wolves adhere to any domain constraints of the optimization problem. If the problem space has defined bounds for each

dimension, the initialization process should generate positions within these bounds. This can be represented as Eq.(26).

$$\text{Domain Constraints: } x_{i,j} \in [l_j, u_j], \text{ for } \boldsymbol{i} = 1,2,\dots,n, \text{ and } \boldsymbol{j} = 1,2,\dots,D \quad (26)$$

where l_j indicates the lower limit and u_j indicates the upper bound.

Random initialization encourages exploration, and the initial positions of grey wolves are typically initialized within the specified domain. This randomness helps in distributing the wolves across the solution space. Eq.(27) expresses the random initialization.

$$x_{i,j} = l_j + (u_j - l_j).rand(0,1), \text{ for } \boldsymbol{i} = 1,2,\dots,n, \text{ and } \boldsymbol{j} = 1,2,\dots,D \quad (27)$$

To generate a random integer within the range of 0 and 1, utilize the *rand*(0,1) function. This function produces a random number between 0 and 1, inclusive.

1. **Population Size:** The population size (n) is a parameter that can be adjusted to influence the search behaviour of GWO. A larger population can enhance exploration, while a smaller size can lead to more focused exploitation.
2. **Initialization Strategy:** Specific initialization strategies can be employed depending on the problem. A stratified initialization method may ensure an even distribution of wolves across the solution space.

Algorithm 7: Initialization

Input:

- $\boldsymbol{n}$ (Population size): The population density of grey wolves.
- $\boldsymbol{D}$ (Dimensionality): The dimensionality of the problem space.
- l_j(Lower bounds for each dimension $\boldsymbol{j}$) A vector holding the bottom boundaries for each dimension.
- u_j (Upper bounds for each dimension$\boldsymbol{j}$): A vector with the maximum values in each dimension.

Output:

- $\boldsymbol{X}$: The initial population of grey wolves, represented as a set of $\boldsymbol{n}$ solutions, where each solution is a $\boldsymbol{D}$-dimensional vector.

Procedure:

1. Initialize an empty set $\boldsymbol{X} = \{\}$ to store the initial population of grey wolves.
2. For $\boldsymbol{i}$ in the range 1 to $\boldsymbol{n}$:
   - Initialize an empty vector $x_i = []$ to represent $\boldsymbol{i}$-th grey wolf position.
3. For $\boldsymbol{j}$ in the range 1 to $\boldsymbol{D}$:
   - Produce an unpredictable integer $\boldsymbol{r}$ between zero and one.
   - Calculate the $\boldsymbol{i}$-th grey wolf position in D dimension
   - Append $x_{i,j}$ to x_i to form the $\boldsymbol{D}$-dimensional position vector for the $\boldsymbol{i}$-th grey wolf.
   - Append x_i to the set $\boldsymbol{X}$to include it in the initial population.
4. Return the set $\boldsymbol{X}$ as the initial population of grey wolves.

3.2.2. Objective Function

The objective function, often considered the heart of any optimization algorithm, is pivotal in guiding the search for optimal solutions. In the RGWO context, this function represents the problem the algorithm aims to solve. The objective function, denoted as $f(x)$, is the mathematical representation of the optimization problem. It takes a potential solution, x, as input and maps it to an real number, representing the quality of that solution. In GWO, the objective function is usually in the form of $f: R^D \rightarrow R$,where $R^D$is the D-dimensional problem space, and R is the set of real numbers.

The primary objective is to either maximize or minimize $f(x)$ based on the nature of the optimization problem. Eq.(28) expresses the objective function.

$$\boldsymbol{Maximize}\ f(x) \quad \boldsymbol{Minimize}\ f(x) \quad (28)$$

The structure of RGWO's objective function in detail is given as Eq.(29). Given a solution, x represented as a D-dimensional vector:

$$\boldsymbol{x} = [x_1, x_2, \dots, x_D] \quad (29)$$

The objective function f(x) maps this D-dimensional vector to a real number that quantifies the quality of the solution as mentioned in Eq.(30).

$$\boldsymbol{f}(x) = Objective\ Function\ (x_1, x_2, \dots, x_D) \quad (30)$$

In the RGWO optimization scenario, the objective function is the basis for fitness evaluation. After initializing the population of grey wolves, an

individual wolf's fitness for its position is computed by applying the objective function. The fitness value essentially represents the "goodness" of the solution. The higher the fitness, the better the solution is concerning the optimization goal (maximization or minimization). Eq.(31) expresses the fitness F_i of the i-th grey wolf.

$$F_i = f(x_i) \quad (31)$$

where x_i is indicates the i-th grey wolf. By evaluating the fitness of each wolf based on the objective function, GWO can effectively distinguish between good and poor solutions in the population.

Algorithm 8: Objective Function

Input:
- $\boldsymbol{x}$ (Solution): A potential solution represented as a $\boldsymbol{D}$-dimensional vector.
- $\boldsymbol{D}$ (Dimensionality): Dimensionality of the problem space.
- Problem-specific parameters (if required for the objective function).

Output:
- $\boldsymbol{f}(x)$: Effective solution

Procedure:
1. Step 1: Define the objective function $\boldsymbol{f}(x)$ corresponding to the optimization problem.
2. Step 2: Calculate the value of the objective function
3. Step 3: Return $\boldsymbol{f}(x)$ as fitness value expressing the quality.

3.2.3. Position Update

The Position Update step aims to modify the positions of grey wolves to facilitate the search for optimal solutions. The new positions are determined using mathematical equations inspired by the hunting and grey wolves' social behaviours. In RGWO, each wolf's position is represented as a D-dimensional vector in Eq.(32):

$$\boldsymbol{x} = [x_1, x_2, \dots, x_D] \quad (32)$$

where D is the dimensionality of the problem space.

The Position Update process considers the influence of these leaders on the entire population. To calculate the updated wolf position, the influences of the alpha, beta, and delta wolves are incorporated into the equation. Eq.(33) assists in identifying the position of a grey wolf .

$$x_i^{t+1} = \frac{x_i^t + A.r_1 - x_i^t}{2} \quad (33)$$

where updated position is indicated as $x_i^{(t+1)}$at the t+1-th iteration, the current position is represented as x_i^t at iteration t, and A is a coefficient representing the influence of the alpha wolf. r_1 is a random vector that introduces stochasticity into the position update. The equation's division by 2 ensures that the updated position lies between the current and alpha wolf positions.

The position updates for the beta and delta wolves are calculated using their respective influence coefficients. Eq.(34) is applied to calculate the same.

$$x_i^{t+1} = \frac{x_i^t + B.r_2 - x_i^t}{2} \ (for\ beta\ wolf)$$
$$x_i^{t+1} = \frac{x_i^t + D.r_3 - x_i^t}{2} \ (for\ delta\ wolf) \quad (34)$$

These equations ensure that the three different types of wolves guide each wolf to explore and exploit the solution space effectively. The random vectors (r_1, r_2 and r_3) introduce randomness, diversifying the search process.

Algorithm 9: Position Update

Input:
- $\boldsymbol{X}$ (Population): The current population of grey wolves.
- Position of A (Alpha wolf's position)
- Position of B (Beta wolf's position)
- Position of D (Delta wolf's position)

Output:
- $\boldsymbol{X}$ (Updated Population): The population of grey wolves with adjusted positions.

Procedure:
1. For each grey wolf x_i in the population $\boldsymbol{X}$:
   - Generate random vectors r_1, r_2 and r_3 to introduce stochasticity.
   - Update the position of x_i as follows:
2. Calculate the new position by averaging the current position x_i, alpha's position A, and a scaled random vector r_1.
3. Repeat the same process for beta and delta wolves using their influence coefficients and random vectors.
   - Ensure the updated position remains within the defined problem space bounds (domain constraints).

4. Return the updated population $\boldsymbol{X}$ with adjusted positions.

3.2.4. Fitness Evaluation

Fitness evaluation in RGWO revolves around the objective function $f(x)$, which represents the optimization problem to be solved. The objective function maps a potential solution x to a real number that signifies the quality or cost of that solution. In mathematical terms, the objective function is expressed as Eq.(35).

$$f(x) = Objective\ Function\ (x_1, x_2, \dots, x_D) \quad (35)$$

where D is the dimensionality of the problem space, and $x_1, x_2, \dots, x_D$ are the coordinates of the solution x.

The goal can vary based on the problem type, such as maximizing profit, minimizing cost, optimizing resource allocation, or achieving any measurable criterion that characterizes the problem. Eq.(36) expresses the Fitness evaluation.

$$\boldsymbol{F}(x) = f(x) \quad (36)$$

where $F(x)$ represents the fitness value of x.

The Fitness Evaluation step in RGWO as it serves multiple key functions:

1. **Quality Assessment**: It quantifies how well a potential solution performs concerning the optimization goal.
2. **Ranking Wolves**: The fitness values of grey wolves rank them within the population. Wolves with higher fitness values are likelier to become alpha or beta wolves, influencing the algorithm's search dynamics.
3. **Selection Mechanism:** The fitness values are employed in selecting alpha, beta, and delta wolves. These leaders guide the rest of the population towards better solutions, ensuring a balanced exploration-exploitation process.
4. **Convergence Tracking**: Fitness evaluation is crucial for tracking the algorithm's convergence. As the optimization progresses, monitoring the changes in fitness values provides insights into how the algorithm approaches a solution.
5. **Diversity Maintenance**: By distinguishing between solutions with different fitness levels, the algorithm retains diversity within the population, preventing premature convergence to local optima.

Algorithm 10: Fitness Evaluation

Input:
- $\boldsymbol{X}$ (Population): The current population of grey wolves.
- $\boldsymbol{f}(x)$ (Objective Function):
- $\boldsymbol{D}$ (Dimensionality): The dimensionality of the problem space.

Output:
- $\boldsymbol{F}$ (Fitness Values): The fitness scores for every grey wolf in the population are represented as a vector.

Procedure:
1. Initialize an empty vector $\boldsymbol{F}$ to store the fitness values of the grey wolves.
2. For each grey wolf x_i in the population $\boldsymbol{X}$:
   - Calculate the fitness value F_i by applying the objective function $\boldsymbol{f}(x)$
   - Append F_i to the vector $\boldsymbol{F}$ to record the fitness value of x_i.
3. Return the vector $\boldsymbol{F}$ containing grey wolf fitness values.

3.2.5. Dominance Ranking

Dominance ranking is fundamentally a comparison process that determines which solutions in the population are superior to others. It relies on the fitness values calculated in the previous step. In mathematical terms, we have a set of fitness values $F=\{F_1, F_2, \dots, F_n\}$, where n represents the population size. To establish dominance, a solution x_i is compared to another solution x_j, typically in a pair-wise manner. Dominance between two solutions is determined by comparing their fitness values. RGWO defines two critical relationships, which are:

a). **Weak Dominance** ($\prec$): Solution x_i weakly dominates solution x_j (denoted $x_i \prec x_j$) if x_i is at least as good as x_j in all objectives and strictly better in at least one objective.

b). **Strong Dominance** ($\prec_s$): Solution x_i strongly dominates solution x_j (denoted $x_i \prec_s x_j$) if x_i is strictly better than xj in all objectives.

Algorithm 11: Dominance ranking

Input:

- ***F*** (Fitness Values) of the grey wolves in the population.

Output:

- ***R*** (Dominance Ranks): A vector containing the dominance ranks of the grey wolves based on their fitness values.

Procedure:

1. Initialize a vector ***R*** to store the dominance ranks, initially filled with zeros.
2. For each grey wolf x_i in the population with fitness value F_i:
   - Initialize a dominance count C_i to zero.
   - For every other grey wolf x_j in the population with fitness value F_j:
3. If F_i is weakly better than F_j, increment C_i by 1.
4. If F_i is strictly better than F_j, increment C_i by 1 again.
   - Calculate the dominance rank R_i for x_i as $C_i + 1$.
   - Assign R_i to the corresponding position in the ***R*** vector.
   - Higher ranks indicate better solutions, if not satisfied, go to Step 1

3.2.6. Leader Selection

Leader selection in RGWO revolves around dominance and the rankings assigned to grey wolves based on their fitness values. As discussed in the previous step (Dominance Ranking), each solution x_i is assigned a dominant rank R_i, where higher ranks indicate better solutions. The dominance rank means the solution's relative performance within the population. The leader selection process aims to choose the three wolves responsible for guiding the population toward better solutions. These leaders are selected based on their dominant ranks.

The alpha wolf (A) is the solution with the highest dominance rank, indicating the best performance in the population. The beta wolf (B) has the second-highest dominance rank, while the delta wolf (D) has the third-highest dominance rank. Eq.(37) is utilized for calculating the rank.

$$\begin{aligned} A &= \arg\max_i R_i \\ B &= \arg\max_i \{R_i : R_i \neq R_A\} \\ D &= \arg\max_i \{R_i : R_i \neq R_A \text{ and } R_i \neq R_B\} \end{aligned} \quad (37)$$

where $argmax_i$ represents the index i that maximizes the specified condition.

Algorithm 12: Leader Selection

Input:

- R (Dominance Ranks): A vector containing the population's dominance ranks of the grey wolves.

Output:

- A (Alpha Wolf): The grey wolf with the highest dominance rank.
- B (Beta Wolf): The grey wolf with the second-highest dominance rank, excluding the alpha wolf.
- D (Delta Wolf): The grey wolf with the third-highest dominance rank, excluding the alpha and beta wolves.

Procedure:

1. Identify the alpha wolf (A) as the grey wolf with the highest dominant rank from the R vector.
2. Determine the beta wolf (B) as the grey wolf with the second-highest dominance rank from the R vector, excluding the alpha wolf.
3. Find the delta wolf (D) as the grey wolf with the third-highest dominance rank from the R vector, excluding the alpha and beta wolves.
4. The selected alpha, beta, and delta wolves (A, B, and D) are designated as leaders, guiding the population in the subsequent steps of the RGWO algorithm.

3.2.7. Encircle Prey

The Encircle Prey step in RGWO is inspired by the cooperative hunting behaviour of grey wolves in nature. When wolves detect potential prey, they coordinate their movements to encircle it, closing in for a successful hunt. In the context of GWO, this behaviour is mathematically modeled to optimize a problem. The core idea is that grey wolves form a circle around a prey, each wolf adjusting its position and role to contribute to the encirclement. In RGWO, the prey represents the

optimal solution sought by the algorithm, and the grey wolves symbolize the candidate solutions within the population. The wolves' locations are updated mathematically based on a search agent's location relative to the prey. In the RGWO algorithm, the positions of the alpha, beta, and delta wolves play a crucial role in guiding the movement of the remaining wolves. The alpha wolf represents the best solution so far, while the beta and delta wolves represent the second and third-best solutions selected in the Leader Selection step. The encirclement process drives the population toward the vicinity of the optimal solution.

Determine the encircling centres for each leader (C_A, C_B,and C_D) by averaging their positions. These centres represent the focal points around which the rest of the wolves will attempt to encircle. Each non-leader wolf in the population adjusts its position based on the leaders' positions and the encircling centres. The position update equations ensure that wolves move toward forming encirclement. The Eq.(38) - Eq.(40) typically introduce randomness to mimic natural movement:
For a wolf x_i encircling the alpha wolf:

$$x_i^{t+1} = C_a - A.r_1 \quad (38)$$

For a wolf x_i encircling the beta wolf:

$$x_i^{t+1} = C_B - A.r_2 \quad (39)$$

For a wolf $x_i$encircling the delta wolf:

$$x_i^{t+1} = C_D - A.r_3 \quad (40)$$

where the updated position is represented as $x_i^{(t+1)}$at the (t+1)-th iteration. A, B, and D are coefficients that control the influence of the leaders and r_1, r_2 and r_3 introduce randomness.

The Encircle Prey step is an iterative process that aims to guide the population toward encircling the optimal solution. As the iterations progress, wolves adjust their positions to encircle the prey more effectively. This process continues throughout the GWO algorithm, complementing other phases like Exploration and Exploitation to optimize the problem effectively.

Algorithm 13: Encircle Prey

Input:
- $\boldsymbol{X}$ (Population): The current population of grey wolves.
- A (Alpha wolf's position)
- B (Beta wolf's position)
- D (Delta wolf's position)

Output:
- $\boldsymbol{X}$ (Updated Population): The population of grey wolves with adjusted positions.

Procedure:
1. Calculate the encircling centre for the alpha, beta, and delta wolves:
2. For each grey wolf x_i in the population $\boldsymbol{X}$:
   a). Generate random vectors r_1, r_2 and r_3 to introduce stochasticity.
   b). Update the position of x_i based on its role in encircling the prey:
   - If x_i is encircling the alpha wolf, update its position towards $C_A$with the influence of A random vector
   - If $x_i$is encircling the beta wolf, update its position towards $C_B$with the influence of B and random vector
   - If $x_i$is encircling the delta wolf, update its position towards $C_D$with the influence of D and random vector
3. Ensure that the updated positions of the grey wolves remain within the defined problem space bounds (domain constraints).
4. Return the updated population $\boldsymbol{X}$ with adjusted positions.

3.2.8. Update Alpha, Beta, and Delta

The "Update Alpha, Beta, and Delta" step involves re-evaluating the leadership roles of the alpha, beta, and delta wolves based on their performance in the current iteration. The aim is to continually adapting the leadership structure to ensure that the best-performing wolves guide the population towards optimal solutions. The performance of each leader is assessed based on their fitness values. Recall that in RGWO, each solution x_i has a fitness value F_i, which quantifies its quality in terms of the optimization problem. The leadership structure is redefined based on the performance of the current leaders. If a new wolf outperforms one of

the leaders, it takes on the corresponding leadership role.

The pack's alpha wolf is the top performer among a group of solutions. If any wolf *xi* has a higher fitness value than the current alpha wolf. It replaces the alpha wolf. The new alpha wolf (A_{new}) is the wolf with the highest fitness value:

$$A_{new} = \arg\max_i F_i \quad (41)$$

The beta wolf has the second-highest fitness value among the wolves, excluding the alpha wolf. If a wolf x_i surpasses the current beta wolf, it takes over the beta leadership role. The new beta wolf (B_{new}) is the wolf with the second-highest fitness value after the alpha wolf:

$$B_{new} = \arg\max_i \{F_i : x_i \neq A\} \quad (42)$$

The delta wolf holds the third-highest fitness value among the wolves, excluding the alpha and beta wolves. If a wolf x_i exceeds the delta wolf's fitness, it becomes the new delta wolf (D_{new}):

$$D_{new} = \arg\max_i \begin{Bmatrix} F_i : x_i \neq A \; and \\ x_i \neq B \end{Bmatrix} \quad (43)$$

The new leadership structure is based on these updated alpha, beta, and delta wolves, ensuring that the best-performing wolves continue effectively guiding the population.

Algorithm 14: Update Alpha, Beta, and Delta

Input:

- A (Current Alpha Wolf)
- B (Current Beta Wolf)
- D (Current Delta Wolf)
- X (Population): The current population of grey wolves.

Output:

- A_{new}(Updated Alpha Wolf)
- B_{new} (Updated Beta Wolf)
- D_{new} (Updated Delta Wolf)

Procedure:

1. Initialize A_{new}, B_{new} and D_{new} with the current alpha, beta, and delta wolves.
2. Calculate the fitness values of all wolves in the population $\boldsymbol{X}$.
3. For each wolf in the population x_i in the population $\boldsymbol{X}$:
   - If the fitness value of x_i is higher than the fitness value of the newA_{new}, update $A_{new}$to be x_i.
   - If the fitness value of $x_i$is higher than the fitness value of B_{new} and $x_i$is not the same as A_{new}, update B_{new}, to be x_i.
   - If the fitness value of $x_i$is greater than the fitness value of D_{new}, and $x_i$is not the same as A_{new} and x_i is not the same as B_{new}, update D_{new} to be x_i.
4. The updated alpha, beta, and delta wolves ($A_{new}, B_{new}\ and\ D_{new}$) are now the leaders guiding the population in the subsequent steps of the RGWO algorithm.

3.2.9. Exploration and Exploitation

The positions of grey wolves in the population are updated based on a mixture of exploration and exploitation methods. The aim is to balance venturing into new territory and exploiting promising areas. The specific equations and strategies used for exploration and exploitation may vary depending on the GWO variant.

Exploration in RGWO involves introducing randomness or stochasticity to encourage the population to explore new regions of the solution space. Common exploration strategies include:

1. **Random Movement:** Wolves adjust their positions within a specific range. This randomness can be controlled using parameters.
2. **Levy Flight:** Levy flights are a special kind of random walk in which large and small steps are taken. It introduces heavy-tailed random steps to promote exploration.

On the other hand, exploitation aims to focus on promising areas of the solution space. Wolves exploit regions with known high fitness values or solutions. Exploitation strategies include:

1. **Deterministic Movement**: Wolves move towards leaders' positions, such as the alpha, beta, or delta wolves, which are expected to be closer to optimal solutions.
2. **Leader Guidance:** Wolves follow the trajectories of leaders, imitating their movements and leveraging their guidance.

Combining exploration and exploitation strategies can mathematically represent the Exploration and Exploitation step. The wolf (x_i)

updated position at the next iteration (t+1) is a result of both exploration (E_i^{t+1}) and exploitation (X_i^{t+1}) components:

$$x_i^{t+1} = E_i^{t+1} + X_i^{t+1} \quad (44)$$

The exploration component (E_i^{t+1}) Introduces randomness and encourages diversity in the population, expressed as Eq.(21).

$$E_i^{t+1} = Exploration\ Strategy \quad (45)$$

The exploitation component (E_i^{t+1}))guides wolves toward promising regions or solutions, often in the direction of leaders, as expressed in Eq.(46).

$$X_i^{t+1} = Exploration\ Strategy \quad (46)$$

Exploration and exploitation strategies vary among GWO variants and are often parameterized to control the balance between the two. Achieving the right balance is crucial for the algorithm's success in finding optimal solutions to complex optimization problems.

3.2.10. Termination

The termination criteria are essential for controlling the algorithm's behaviour and preventing it from continuing indefinitely. RGWO stops after a predetermined number of iterations (maximum) have been reached. The termination decision depends solely on the number of iterations, which can be set based on the available computational resources and the problem's complexity. RGWO stops if the current iteration count (t) exceeds the predetermined maximum iterations (T_{max}) and it is expressed in Eq.(47).

$$\text{If } \boldsymbol{t} > T_{max}, \text{ stop} \quad (47)$$

3.3. Fusion of RGWO and GEQDNN

The fusion of RGWO and GEQDNN represents a synergistic approach to enhance sentiment analysis's overall performance in online shopping. Inspired by the cooperative hunting behavior of grey wolves, RGWO introduces an adaptive mechanism that optimizes the neural network's performance. RGWO excels in finding global optima in complex search spaces, a crucial attribute in refining the parameters of the GEQDNN. This adaptability ensures that the model is finely tuned to the intricacies of sentiment analysis within product reviews. Gaussian enhancement introduces probabilistic distributions, providing a more nuanced representation of uncertainties in sentiment expression. By combining this Gaussian refinement with the adaptive optimization capabilities of RGWO, the model gains an improved ability to capture subtle variations in sentiment, enhancing its precision and accuracy. Incorporating quantum principles into deep neural networks through Gaussian enhancement allows for parallel processing and superposition, exponentially increasing computational efficiency. The synergy of RGWO and Gaussian-Enhanced Quantum Deep Neural Networks addresses the challenges of sentiment analysis, particularly in dealing with the vast amounts of textual data inherent in online shopping, leading to faster and more accurate results.

3.3.1. Advantages of RGWO-GEQDNN

The significant advantages of RGWO-GEQDNN are

- Nuanced Sensitivity: RGWO-GEQDNN's fusion allows for heightened sensitivity, accurately capturing subtle nuances in sentiment expression.
- Adaptive Precision: Resilient Grey Wolf Optimization ensures adaptive tuning, maintaining high accuracy in dynamic sentiment contexts.
- Efficient Processing: Gaussian-enhanced quantum principles enable efficient parallel processing, ensuring faster and more reliable sentiment analysis, particularly in extensive data scenarios.
- Global Optimization: RGWO's capability for global optimization enhances the model's efficiency in navigating complex sentiment landscapes, leading to more accurate classifications.

4. ABOUT DATASET

The Amazon Product Review Dataset, a treasure trove of diverse opinions and user feedback, offers rich insights into consumer preferences and sentiments. With millions of reviews spanning many product categories, it provides a comprehensive view of the collective voice of Amazon customers. Researchers and businesses find immense value in this dataset, as it enables sentiment analysis, trend identification, and market research. Its size and variety make it valuable for training machine learning models and natural language processing algorithms. From understanding customer satisfaction to tracking emerging product trends, the Amazon Product Review Dataset is a vital resource for those seeking to explore the world of e-commerce through the lens of user-generated content.This study investigates the electronics product review dataset from Amazon's collection of

product review datasets, encompassing 6,739,590 customer reviews. The dataset is composed of 11 fields, and detailed descriptions of these fields are provided in Table 1

Table 1. Field Description

Field Name	Description
reviewerID	A unique identifier for the person reviewing the product
asin	An exclusive identifier for the product being reviewed
reviewerName	The name of the individual providing the review
vote	The count of helpful votes received by the review
style	Product metadata is represented as a dictionary.
reviewText	The written content of the review
overall	The numerical rating is given to the product.
summary	A summary or title of the review
unixReviewTime	The time of the review is represented in Unix time.
reviewTime	The time of the review in its raw format
image	Images shared by users after receiving the product.

5. PERFORMANCE METRICS

Sentiment analysis employs the terms TP (True Positive), TN (True Negative), FP (False Positive), and FN (False Negative) to assess the accuracy of sentiment classification. TP represents instances where the model correctly identifies positive sentiment, while TN denotes cases where it accurately recognizes negative sentiment. FP occurs when the model mistakenly detects positive sentiment in negative text, and FN indicates the model incorrectly identifies negative sentiment in positive text. These metrics are crucial in evaluating the precision and effectiveness of sentiment analysis models, allowing for a comprehensive assessment of their performance in categorizing sentiments accurately. TP, TN, FP and FN are variables in the chosen performance metrics below to evaluate the proposed classifier's performance against the state-of-the-art classifiers.

- **Precision**: Precision (PREC) in sentiment analysis indicates the accuracy of positive sentiment predictions. It measures the ratio of correctly identified positive sentiments to all predicted positive sentiments by a model.
- **Recall**: Recall (RCLL), in the context of sentiment analysis, assesses the model's ability to identify actual positive sentiments. It is the ratio of correctly identified positive sentiments to all actual positive sentiments.
- **Classification Accuracy**: Classification accuracy (CL-AC) measures the overall correctness of a model's sentiment predictions in sentiment analysis. It calculates the ratio of correctly predicted sentiments to the total number of sentiments in the dataset.
- **F-Measure**: The F-Measure (F-MSR) is a sentiment analysis metric that balances precision and recall. It comprehensively evaluates a model's ability to make accurate positive sentiment predictions while capturing a high percentage of positive sentiments.
- **Fowlkes–Mallows Index:** The Fowlkes–Mallows Index (FMI) in sentiment analysis measures precision and recall's geometric mean. It assesses a model's ability to accurately predict positive sentiments while considering the trade-off between precision and recall.
- **Matthews Correlation Coefficient**: The Matthews Correlation Coefficient (MCC) is a sentiment analysis metric considering true and false positives and negatives, providing a robust measure of the relationship between predicted and actual sentiments.

6. RESULTS AND DISCUSSION

6.1. Precision and Recall Analysis

Figure 1 and Table 2 present precision and recall metrics for three sentiment analysis algorithms: DLGM, ESDM, and RGWO-GEQDNN. Precision measures the accuracy of positive sentiment identification, while recall assesses the algorithm's ability to capture all positive sentiments in electronic product reviews.

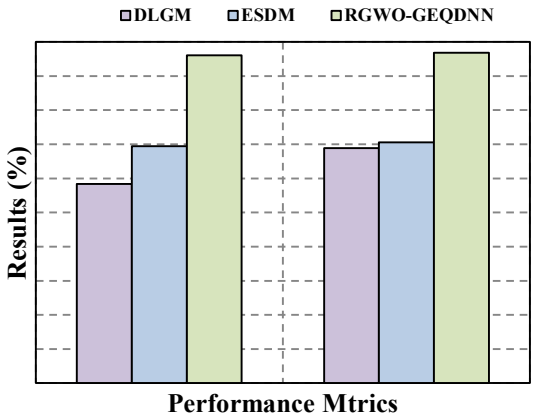

Figure 1. Precision And Recall

DLGM showcases a precision of 58.366% and a recall of 68.869%. Its performance reflects its focus on fine-grained aspect-based sentiment analysis. DLGM's disentangled graph representation dissects complex linguistic structures and precisely identifies aspects and sentiments. It balances precision and recall by emphasizing the explainability of sentiment classifications. This approach enables detailed insights into sentiment relationships. Its intricacy might introduce a trade-off between precision and recall, particularly when handling complex or noisy text data. Despite its contributions, DLGM may suffer from slower processing times, and its disentangled analysis may occasionally struggle with capturing sentiment nuances effectively, which impacts both precision and recall.

ESDM exhibits a precision of 69.403% and a recall of 70.542%, indicating its capability to perform well in sentiment analysis. ESDM excels by effectively utilizing syntactic dependency features and grammatical structure modeling. It captures nuanced sentiment expressions by considering syntactic dependencies. Its focus on aspect-aware sentiment analysis allows for precisely identifying relationships between product features and sentiments. ESDM handles negations efficiently, improving both precision and recall. Due to its intensive syntactic analysis, it may not be as suitable for enormous datasets. It's also limited when dealing with non-standard or informal language, which can affect precision and recall in real-world scenarios.

RGWO-GEQDNN demonstrates exceptional precision of 96.036% and recall of 96.829%, making it a standout performer. The outstanding results can be attributed to its unique combination of resilient grey wolf optimization and enhanced quantum deep neural networks. This blend allows for robust optimization and fine-grained sentiment modeling. RGWO-GEQDNN efficiently explores solutions with the resilience of grey wolf optimization and the parallelism of quantum computing. The quantum computing principles enable it to capture complex sentiment patterns with exceptional precision and recall. Its adaptability to large datasets and probabilistic output add to its outstanding performance in sentiment analysis, making it a top-performing algorithm in this evaluation.

Table 2. Precision And Recall

Classification Algorithms	PREC (%)	RCLL (%)
DLGM	58.366	68.869
ESDM	69.403	70.542
RGWO-GEQDNN	96.036	96.829

6.2. Classification Accuracy and F-Measure Analysis

Figure 2 and Table 3 provide insights into classification accuracy and F-measure metrics for three sentiment analysis algorithms: DLGM, ESDM, and TABC-TSGMM. Classification accuracy measures the overall correctness of sentiment classification, while the F-measure assesses the balance between precision and recall in sentiment analysis.

DLGM exhibits a classification accuracy of 63.401% and an F-measure of 25.974%. These metrics reveal the algorithm's effectiveness in accurately classifying sentiments and highlight some limitations. The classification accuracy indicates its ability to make correct sentiment predictions, and it may serve well in scenarios where precision is prioritized over recall. The relatively low F-measure suggests room for improvement in balancing precision and recall. DLGM's unique disentangled graph representation and emphasis on explainability contribute to its classification accuracy, but its fine-grained aspect analysis can sometimes hinder overall recall, affecting the F-measure.

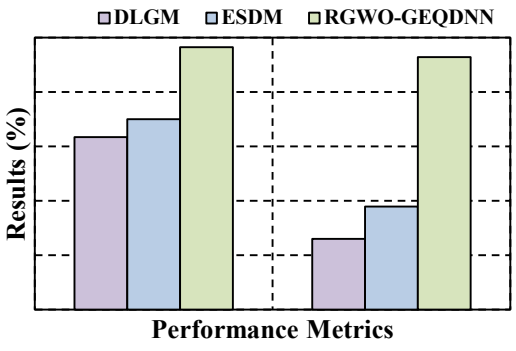

Figure 2. Classification Accuracy And F-Measure

ESDM showcases a classification accuracy of 69.97% and an F-measure of 37.804%, demonstrating its proficiency in sentiment analysis. The classification accuracy highlights its capability to correctly classify sentiments, making it a reliable choice for precision-oriented tasks. The noteworthy

F-measure underlines its balanced approach to precision and recall, indicating an ability to effectively capture sentiments without compromising precision. ESDM's emphasis on syntactic dependency features, grammatical structure modeling, and aspect-aware sentiment analysis collectively contribute to its robust classification accuracy and well-balanced F-measure, making it a versatile and high-performing algorithm.

TABC-TSGMM exhibited exceptional performance, marking a significant milestone in sentiment analysis. The algorithm achieved an impressive classification accuracy of 96.432% and an F-measure of 92.887%, reflecting its unparalleled precision and balanced handling of precision and recall. This outstanding precision makes TABC-TSGMM an ideal choice for applications where the accurate identification of sentiments is paramount. This extraordinary performance is rooted in the algorithm's robust optimization techniques, most notably its utilization of the artificial bee colony (ABC) optimization algorithm, which enables it to tenaciously explore and fine-tune solutions. TABC-TSGMM leverages a Taylor series-based approach within a Gaussian Mixture Model, empowering it to capture even the most intricate sentiment patterns and relationships in the data. This advanced modeling technique equips TABC-TSGMM to excel in nuanced sentiment analysis, elevating it above conventional sentiment analysis methods.

Table 3. Classification Accuracy And F-Measure

Classification Algorithms	FMI (%)	MCC (%)
DLGM	63.401	25.974
ESDM	69.970	37.804
RGWO-GEQDNN	96.432	92.887

6.3. Fowlkes–Mallows Index and Matthews Correlation Coefficient Analysis.

Figure 3 and Table 4 examine two essential evaluation metrics: the Fowlkes–Mallows Index (FMI) and the Matthews Correlation Coefficient (MCC) for three sentiment analysis algorithms: DLGM, ESDM, and RGWO-GEQDNN.

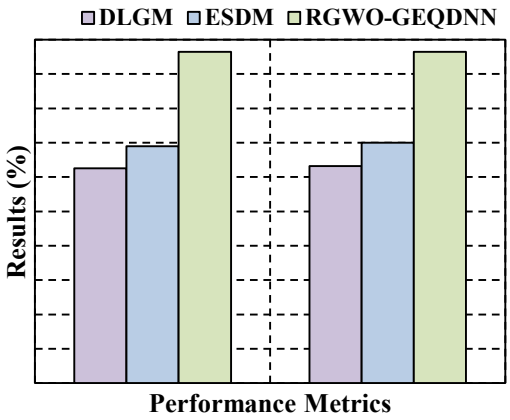

Figure 3. Fowlkes–Mallows Index And Matthews Correlation Coefficient

DLGM exhibits an FMI of 62.554% and a MCC of 63.184%. While DLGM offers some valuable insights into sentiment analysis, it has a few limitations. The MCC indicates a moderate overall performance but leaves room for improvement. One of DLGM's disadvantages is that it can be computationally intensive and may suffer from slower processing times, mainly when dealing with large datasets. Its disentangled analysis might occasionally struggle with capturing sentiment nuances effectively, affecting precision and recall, as reflected in the MCC. DLGM's performance can be improved by addressing these computational and sensitivity issues.

ESDM presents an FMI of 68.934% and an MCC of 69.968%. ESDM showcases notable strengths, but it also has some limitations. The MCC suggests that ESDM performs well but leaves room for further enhancement. One of its challenges is that it may not be as effective when handling non-standard or informal language, which can impact its real-world applicability. ESDM's intensive syntactic analysis can be computationally demanding, making it less suitable for extensive datasets. Addressing these language-dependent limitations and optimizing computational efficiency could be beneficial to further improve its performance.

Table 4. Fowlkes–Mallows Index And Matthews Correlation Coefficient

Classification Algorithms	CL-AC (%)	F-MSR (%)
DLGM	62.554	63.184
ESDM	68.934	69.968
RGWO-GEQDNN	96.442	96.431

RGWO-GEQDNN impressively achieves an FMI% of 96.442% and an MCC of 96.431%. RGWO-GEQDNN represents the state-of-the-art in sentiment analysis with its remarkable performance. However, it's important to note that this algorithm might require substantial computational resources, potentially limiting its practical use in resource-constrained environments. Its exceptional performance may come at the cost of model explainability, as the complex modeling techniques utilized in quantum-deep neural networks can be challenging to interpret. RGWO-GEQDNN's superior performance makes it a compelling choice for high-precision sentiment analysis tasks, provided that computational resources and model transparency are accounted for in the application's context.

The proposed algorithm, RGWO-GEQDNN, outshines the state-of-the-art alternatives. Its advantages lie in its tenacious optimization through resilient grey wolf optimization and the incorporation of enhanced quantum deep neural networks. This unique combination allows it to achieve superior classification accuracy and F-measure. RGWO-GEQDNN stands out by capturing complex sentiment patterns effectively, making it an excellent choice for precision-oriented sentiment analysis tasks. While it may demand more computational resources and pose challenges in model explainability, its remarkable performance makes it a compelling option in scenarios where high precision is critical.

7. CONCLUSION

Resilient Grey Wolf Optimization-based Gaussian-Enhanced Quantum Deep Neural Network (RGWO-GEQDNN) is proposed as a transformative solution in the domain of sentiment analysis for online shopping. This approach blends the power of Resilient Grey Wolf Optimization with quantum computing principles and adeptly addresses the intricate challenges posed by sentiment analysis within product reviews. The outcomes of its evaluation using the Amazon product review dataset unambiguously underscore RGWO-GEQDNN's superiority over existing algorithms. RGWO-GEQDNN's exceptional performance can be attributed to several key factors. The innovative fusion of Resilient Grey Wolf Optimization and Gaussian-enhanced quantum deep neural networks enhances the model's ability to recognize subtle nuances in language and sentiment expression, leading to more accurate sentiment classification. This heightened sensitivity enables RGWO-GEQDNN to extract sentiments from reviews that may be challenging for traditional algorithms to classify. RGWO-GEQDNN's quantum-inspired approach allows it to efficiently process and analyze vast amounts of textual data, making it highly capable of handling the substantial volume of product reviews typically encountered in e-commerce. This efficiency accelerates sentiment analysis, leading to faster and more reliable results. RGWO-GEQDNN's capacity to adapt and optimize, inspired by the Resilient Grey Wolf Optimization algorithm, further contributes to its superior results. This adaptability ensures the model remains effective in the ever-evolving online shopping landscape.

REFERENCES:

[1]. T. Rabeya, E. Khatun, S. R. H. Noori, S. Akter, and I. Jahan, “Bengali Review Analysis for Predicting Popular Cosmetic Brand Using Machine Learning Classifiers,” Lecture Notes in Networks and Systems, vol. 383. pp. 251–260, 2023. doi: 10.1007/978-981-19-4960-9_21.

[2]. N. W. Madinga and J. Lappeman, “Social Media Sentiment Analysis: Online versus ‘Brick and Mortar’ Retailers in South Africa,” J. African Bus., vol. 24, no. 2, pp. 345–362, Apr. 2023, doi: 10.1080/15228916.2022.2069418.

[3]. Tomczyk and L. Eger, “Online safety as a new component of digital literacy for young people,” Integr. Educ., vol. 24, no. 2, pp. 172–184, 2020, doi: 10.15507/1991-9468.099.024.202002.172-184.

[4]. D. Zhang, Z. Shen, and Y. Li, “Requirement analysis and service optimization of multiple category fresh products in online retailing using importance-Kano analysis,” J. Retail. Consum. Serv., vol. 72, 2023, doi: 10.1016/j.jretconser.2022.103253.

[5]. S. Katada, S. Okada, and K. Komatani, “Transformer-Based Physiological Feature Learning for Multimodal Analysis of Self-Reported Sentiment,” in ACM International Conference Proceeding Series, 2022, pp. 349–358. doi: 10.1145/3536221.3556576.

[6]. J. Chen, C. Sun, S. Zhang, and J. Zeng, “Cross-modal dynamic sentiment annotation for speech sentiment analysis,” Comput. Electr. Eng., vol. 106, 2023, doi: 10.1016/j.compeleceng.2023.108598.

[7]. N. Zhong and J. B. Ren, "Using sentiment analysis to study the relationship between subjective expression in financial reports and company performance," Front. Psychol., vol. 13, 2022, doi: 10.3389/fpsyg.2022.949881.
[8]. Y. Wang, J. Guo, C. Yuan, and B. Li, "Sentiment Analysis of Twitter Data," Appl. Sci., vol. 12, no. 22, 2022, doi: 10.3390/app122211775.
[9]. H. Kwak and J. An, "Revealing the hidden patterns of news photos: Analysis of millions of news photos through GDELT & deep learning-based vision APIs," in AAAI Workshop - Technical Report, 2016, vol. WS-16-16-, pp. 99–107. doi: 10.1609/icwsm.v10i2.14840.
[10]. Y. Sun, Q. Sun, and S. Zhu, "Prediction of Shanghai Stock Index Based on Investor Sentiment and CNN-LSTM Model," J. Syst. Sci. Inf., vol. 10, no. 6, pp. 620–632, 2022, doi: 10.21078/JSSI-2022-620-13.
[11]. T. Aditya Sai Srinivas, K. Govinda, S. Ramasubbareddy, and E. Swetha, "Sentimental analysis of demonetization over twitter data using machine learning," J. Comput. Theor. Nanosci., vol. 16, no. 5–6, pp. 2055–2058, 2019, doi: 10.1166/jctn.2019.7849.
[12]. H. T. Phan, V. C. Tran, N. T. Nguyen, and D. Hwang, "Improving the Performance of Sentiment Analysis of Tweets Containing Fuzzy Sentiment Using the Feature Ensemble Model," IEEE Access, vol. 8, pp. 14630–14641, 2020, doi: 10.1109/ACCESS.2019.2963702.
[13]. N. Lin, Y. Fu, X. Lin, D. Zhou, A. Yang, and S. Jiang, "CL-XABSA: Contrastive Learning for Cross-Lingual Aspect-Based Sentiment Analysis," IEEE/ACM Trans. Audio Speech Lang. Process., vol. 31, pp. 2935–2946, 2023, doi: 10.1109/TASLP.2023.3297964.
[14]. C. R. Aydln and T. Gungor, "Combination of recursive and recurrent neural networks for aspect-based sentiment analysis using inter-aspect relations," IEEE Access, vol. 8, pp. 77820–77832, 2020, doi: 10.1109/ACCESS.2020.2990306.
[15]. Y. S. Mehanna and M. Bin Mahmuddin, "A Semantic Conceptualization Using Tagged Bag-of-Concepts for Sentiment Analysis," IEEE Access, vol. 9, pp. 118736–118756, 2021, doi: 10.1109/ACCESS.2021.3107237.
[16]. H. Kim and G. Qin, "Summarizing Students' Free Responses for an Introductory Algebra-Based Physics Course Survey Using Cluster and Sentiment Analysis," IEEE Access, vol. 11, pp. 89052–89066, 2023, doi: 10.1109/ACCESS.2023.3305260.
[17]. U. Sehar, S. Kanwal, K. Dashtipur, U. Mir, U. Abbasi, and F. Khan, "Urdu Sentiment Analysis via Multimodal Data Mining Based on Deep Learning Algorithms," IEEE Access, vol. 9, pp. 153072–153082, 2021, doi: 10.1109/ACCESS.2021.3122025.
[18]. F. Huang, X. Li, C. Yuan, S. Zhang, J. Zhang, and S. Qiao, "Attention-Emotion-Enhanced Convolutional LSTM for Sentiment Analysis," IEEE Trans. Neural Networks Learn. Syst., vol. 33, no. 9, pp. 4332–4345, 2021, doi: 10.1109/TNNLS.2021.3056664.
[19]. T. Zhang, X. Gong, and C. L. P. Chen, "BMT-Net: Broad Multitask Transformer Network for Sentiment Analysis," IEEE Trans. Cybern., vol. 52, no. 7, pp. 6232–6243, 2022, doi: 10.1109/TCYB.2021.3050508.
[20]. H. Liang, U. Ganeshbabu, and T. Thorne, "A Dynamic Bayesian Network Approach for Analysing Topic-Sentiment Evolution," IEEE Access, vol. 8, pp. 54164–54174, 2020, doi: 10.1109/ACCESS.2020.2979012.
[21]. A. Nazir, Y. Rao, L. Wu, and L. Sun, "Issues and Challenges of Aspect-based Sentiment Analysis: A Comprehensive Survey," IEEE Trans. Affect. Comput., vol. 13, no. 2, pp. 845–863, 2022, doi: 10.1109/TAFFC.2020.2970399.
[22]. K. Zhang et al., "EATN: An Efficient Adaptive Transfer Network for Aspect-Level Sentiment Analysis," IEEE Trans. Knowl. Data Eng., vol. 35, no. 1, pp. 377–389, 2023, doi: 10.1109/TKDE.2021.3075238.
[23]. H. Liu, X. Chen, and X. Liu, "A Study of the Application of Weight Distributing Method Combining Sentiment Dictionary and TF-IDF for Text Sentiment Analysis," IEEE Access, vol. 10, pp. 32280–32289, 2022, doi: 10.1109/ACCESS.2022.3160172.
[24]. C. F. V. Loan, "The ubiquitous Kronecker product," J. Comput. Appl. Math., vol. 123, no. 1, pp. 85–100, 2000, doi: https://doi.org/10.1016/S0377-0427(00)00393-9.
[25]. D. Jayaraj, J. Ramkumar, M. Lingaraj, and B. Sureshkumar, "AFSORP: Adaptive Fish Swarm Optimization-Based Routing Protocol for Mobility Enabled Wireless Sensor Network," Int. J. Comput. Networks Appl., vol. 10, no. 1, pp. 119–129, 2023, doi: 10.22247/ijcna/2023/218516.

[26]. R. Jaganathan and V. Ramasamy, "Performance modeling of bio-inspired routing protocols in Cognitive Radio Ad Hoc Network to reduce end-to-end delay," Int. J. Intell. Eng. Syst., vol. 12, no. 1, pp. 221–231, 2019, doi: 10.22266/IJIES2019.0228.22.

[27]. J. Ramkumar, K. S. Jeen Marseline, and D. R. Medhunhashini, "Relentless Firefly Optimization-Based Routing Protocol (RFORP) for Securing Fintech Data in IoT-Based Ad-Hoc Networks," Int. J. Comput. Networks Appl., vol. 10, no. 4, pp. 668–687, Aug. 2023, doi: 10.22247/ijcna/2023/223319.

[28]. J. Ramkumar and R. Vadivel, "Improved frog leap inspired protocol (IFLIP) – for routing in cognitive radio ad hoc networks (CRAHN)," World J. Eng., vol. 15, no. 2, pp. 306–311, 2018, doi: 10.1108/WJE-08-2017-0260.

[29]. M. Lingaraj, T. N. Sugumar, C. S. Felix, and J. Ramkumar, "Query aware routing protocol for mobility enabled wireless sensor network," Int. J. Comput. Networks Appl., vol. 8, no. 3, pp. 258–267, 2021, doi: 10.22247/ijcna/2021/209192.

[30]. R. Vadivel and J. Ramkumar, "QoS-enabled improved cuckoo search-inspired protocol (ICSIP) for IoT-based healthcare applications," Inc. Internet Things Healthc. Appl. Wearable Devices, pp. 109–121, 2019, doi: 10.4018/978-1-7998-1090-2.ch006.

[31]. J. Ramkumar and R. Vadivel, "Improved Wolf prey inspired protocol for routing in cognitive radio Ad Hoc networks," Int. J. Comput. Networks Appl., vol. 7, no. 5, pp. 126–136, 2020, doi: 10.22247/ijcna/2020/202977.

[32]. A. Senthilkumar, J. Ramkumar, M. Lingaraj, D. Jayaraj, and B. Sureshkumar, "Minimizing Energy Consumption in Vehicular Sensor Networks Using Relentless Particle Swarm Optimization Routing," Int. J. Comput. Networks Appl., vol. 10, no. 2, pp. 217–230, 2023, doi: 10.22247/ijcna/2023/220737.

[33]. J. Ramkumar and R. Vadivel, "Whale optimization routing protocol for minimizing energy consumption in cognitive radio wireless sensor network," Int. J. Comput. Networks Appl., vol. 8, no. 4, pp. 455–464, 2021, doi: 10.22247/ijcna/2021/209711.

[34]. R. Jaganathan and R. Vadivel, "Intelligent Fish Swarm Inspired Protocol (IFSIP) for Dynamic Ideal Routing in Cognitive Radio Ad-Hoc Networks," Int. J. Comput. Digit. Syst., vol. 10, no. 1, pp. 1063–1074, 2021, doi: 10.12785/ijcds/100196.

[35]. P. Menakadevi and J. Ramkumar, "Robust Optimization Based Extreme Learning Machine for Sentiment Analysis in Big Data," 2022 Int. Conf. Adv. Comput. Technol. Appl. ICACTA 2022, pp. 1–5, Mar. 2022, doi: 10.1109/ICACTA54488.2022.9753203.

[36]. J. Ramkumar and R. Vadivel, CSIP—cuckoo search inspired protocol for routing in cognitive radio ad hoc networks, vol. 556. 2017. doi: 10.1007/978-981-10-3874-7_14.

[37]. J. Ramkumar, C. Kumuthini, B. Narasimhan, and S. Boopalan, "Energy Consumption Minimization in Cognitive Radio Mobile Ad-Hoc Networks using Enriched Ad-hoc On-demand Distance Vector Protocol," in 2022 International Conference on Advanced Computing Technologies and Applications, ICACTA 2022, 2022. doi: 10.1109/ICACTA54488.2022.9752899.

[38]. L. Mani, S. Arumugam, and R. Jaganathan, "Performance Enhancement of Wireless Sensor Network Using Feisty Particle Swarm Optimization Protocol," ACM Int. Conf. Proceeding Ser., pp. 1–5, Dec. 2022, doi: 10.1145/3590837.3590907.

[39]. R. Jaganathan, V. Ramasamy, L. Mani, and N. Balakrishnan, "Diligence Eagle Optimization Protocol for Secure Routing (DEOPSR) in Cloud-Based Wireless Sensor Network," 2022, doi: 10.21203/rs.3.rs-1759040/v1.

[40]. J. Ramkumar, R. Vadivel, and B. Narasimhan, "Constrained Cuckoo Search Optimization Based Protocol for Routing in Cloud Network," Int. J. Comput. Networks Appl., vol. 8, no. 6, pp. 795–803, 2021, doi: 10.22247/ijcna/2021/210727.

[41]. J. Ramkumar, S. S. Dinakaran, M. Lingaraj, S. Boopalan, and B. Narasimhan, "IoT-Based Kalman Filtering and Particle Swarm Optimization for Detecting Skin Lesion," in Lecture Notes in Electrical Engineering, 2023, vol. 975, pp. 17–27. doi: 10.1007/978-981-19-8353-5_2.

[42]. J. Ramkumar and R. Vadivel, "Multi-Adaptive Routing Protocol for Internet of Things based Ad-hoc Networks," Wirel. Pers. Commun., vol. 120, no. 2, pp. 887–909, Apr. 2021, doi: 10.1007/s11277-021-08495-z.