Chapter 2: Methodology and Techniques

2.1 Flowchart/ Block Diagram

In the context of a thesis, the flowchart or block diagram would typically depict the various stages of the research process, including the proposed methodology, data analysis, and conclusions. The flowchart or block diagram can help to provide a clear overview of the thesis structure and can be a useful tool for organizing and presenting information in a logical and coherent manner. The Flowchart of the Proposed work is partition into 2 part- one is for Securing the data using encryption along with blockchain and another one is the diagnosis process. The Figure 2.1 and 2.2 are describe the work flow.

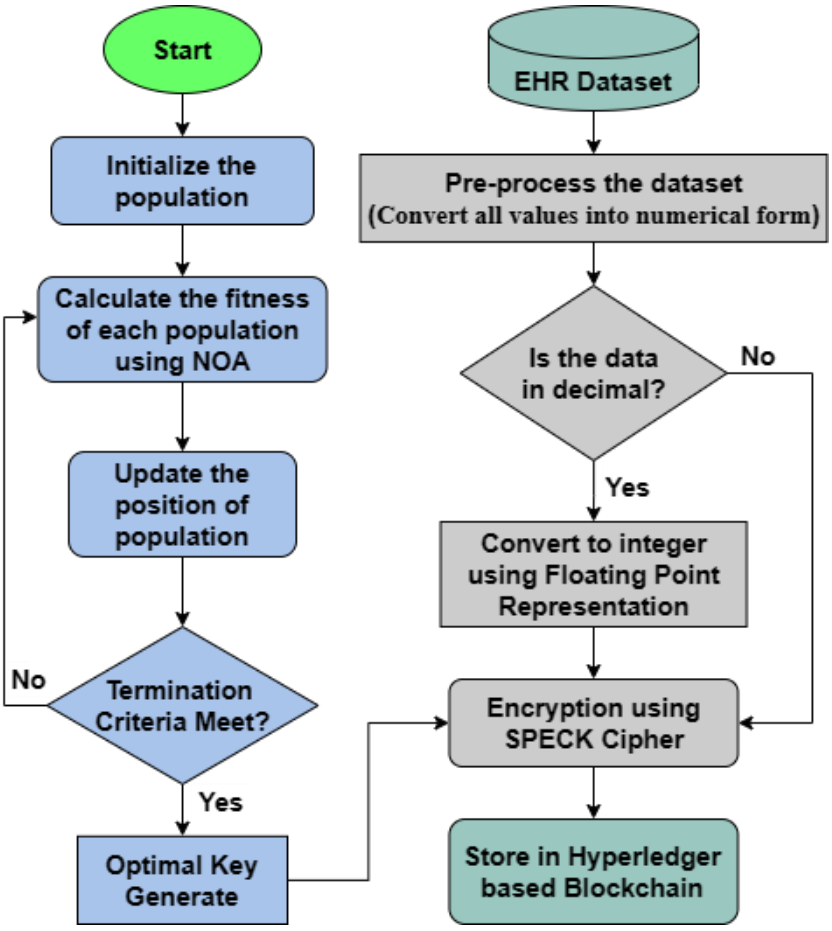

Figure 2.1: 'Flowchart A' for Securing the Medical records

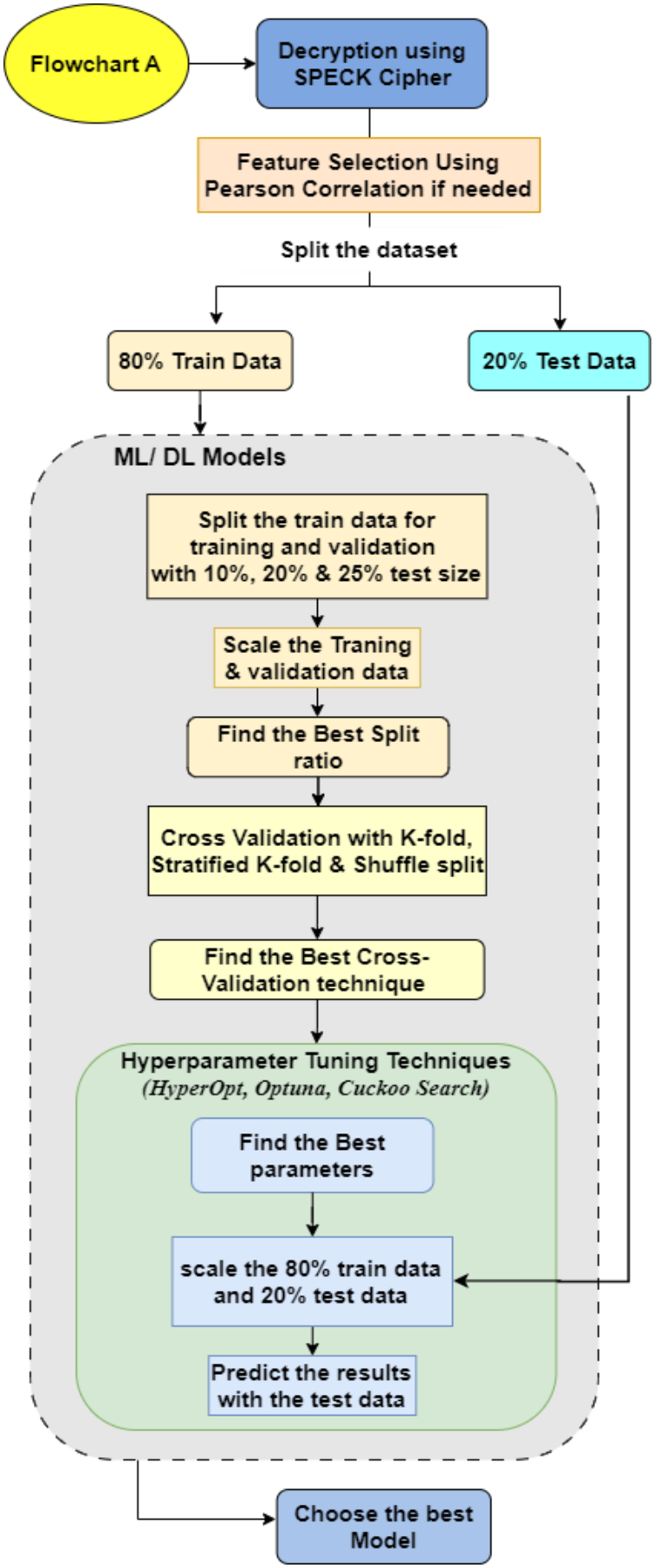

Figure 2.2: Diagnosis process

2.2 Algorithm of the work

The algorithm of the thesis refers to the step-by-step process that follows in order to complete this research thesis. Here is a general algorithm for securing the data and the diagnosis process. The algorithm is following:

Algorithm 1: BESMRM-OAD Model

Input: EHR Dataset
Output: Encrypted Dataset on Hyperledger based Blockchain and best diagnosis model

1. Read the dataset
2. Clean and pre-process the dataset (convert all values in numerical form)
3. If the data value in decimal form, convert it to integer form by using floating point representation.
4. Generate the Optimal Key using NOA following the section 2.2.8.1
5. Set the Optimal Key as the public key of Speck Cipher
6. Encrypt the dataset using Speck Cipher with the public key
7. **Store the dataset in a Hyperledger based blockchain**
8. Share the public key to the doctors and hospital authority
9. Decrypt the dataset using with the public key
10. Do Feature selection using Pearson Correlation if more than 70% of the independent features are correlated at least 5% with the dependent feature
11. Split the dataset into 80% Train Data and 20% Test Data
12. For each ML/ DL model do the following:
    i. Split the train data into training and validation data with 10%, 20% and 25% validation size
    ii. Scale the training and validation data using Robust scaler
    iii. Find the best split ratio using different evaluation metrics
    iv. Do Cross Validation using K-fold, Stratified K-fold and shuffle Split methods
    v. Find the best Cross Validation Technique using different evaluation metrics
    vi. Do the Hyperparameter Tuning with HyperOpt, Optuna, Cuckoo Search. For each technique do the following:
        a) Find the best parameters using training and validation data
        b) Fit the 80% train data after Robust Scaling
        c) Predict different evaluation metrics with the 20% of scaled test data
    vii. Find the best hyperparameter tuning technique
13. **Choose the best diagnosis model basis on the predicted results**

The algorithms follow some different types of techniques that are mention in the below sections. The algorithm may also involve reviewing and revising the thesis multiple times before it is finalized and submitted for evaluation.

2.2.1 Machine Learning and Ensemble Models Description

Machine learning is a subfield of artificial intelligence that involves the use of algorithms to enable computers to learn from data and improve their performance over time. Ensemble models, on the other hand, are a type of machine learning model that combines multiple models to improve predictive accuracy and reduce the risk of overfitting. This section provides an overview of these concepts and their applications, laying the foundation for the subsequent discussions in the document. In this thesis we have used basically four different type of machine learning or ensemble models for different works in Chapter 3 & 4, which are discusses in the following section.

2.2.1.1 K-Nearest Neighbors Classifier

The k-nearest neighbors (KNN) classifier algorithm is a type of supervised machine learning algorithm used for classification tasks. The algorithm works by identifying the k closest data points in the training dataset to a given test data point, based on a distance metric such as Euclidean distance or Manhattan distance.

One of the main advantages of the KNN algorithm is that it does not make any assumptions about the underlying data distribution, making it a non-parametric algorithm. However, its performance can be impacted by the choice of k, the distance metric, and the feature scaling.

The KNN classifier [15] can be mathematically represented as:

Let x be a new data point and D be the training dataset, where $D = (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, where x_i represents the i^{th} feature vector and y_i represents the corresponding class label.

The KNN algorithm can be summarized as follows:

- Calculate the distance between the new data point x and all other data points in D.
- Select the k closest data points to x.
- Determine the class of the new data point by taking the majority class label among the k closest data points.

The distance metric used in KNN can vary, but a commonly used metric is Euclidean distance:

$$dist(x_i, x_j) = \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2}$$

where $x_{i,k}$ and $x_{j,k}$ are the k^{th} features of data points x_i and x_j, respectively, and d is the number of features.

KNN is a simple yet effective classification algorithm that is widely used in various applications, such as image recognition, natural language processing, and recommender systems.

2.2.1.2 Logistic Regression

Logistic regression is a statistical algorithm used for binary classification problems, where the goal is to predict the probability of a binary outcome (i.e., 0 or 1) based on a set of input features. The logistic regression algorithm models the relationship between the input features and the probability of the binary outcome using a sigmoid function.

The equation for logistic regression is:

$$P(y = 1|x) = \frac{1}{1 + e^{-\beta^{Tx}}}$$

where $P(y = 1|x)$ represents the probability of the positive class given the input feature vector x, β represents the coefficients or weights associated with each input feature, and e is the base of the natural logarithm.

To make predictions using logistic regression, we first calculate the probability of the positive class based on the input features using the above equation. If this probability is greater than a certain threshold (usually 0.5), we predict that the positive class is present, otherwise we predict the negative class.

Logistic regression is widely used in various domains, such as healthcare, finance, and marketing, for tasks such as disease diagnosis, credit risk assessment, and customer segmentation. It is a simple yet powerful algorithm that can achieve high accuracy with relatively few input features.

2.2.1.3 Random Forest Classifier

Random Forest Classifier is a popular machine learning algorithm used for classification tasks. It is an ensemble method that combines multiple decision trees to create a robust and accurate model. The algorithm works by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Each decision tree is trained on a random subset of the training data, and at each node of the tree, a random subset of the features is considered for splitting. This randomization helps to reduce overfitting and improve the generalization ability of the model.

The final prediction of the random forest is the average of the predictions of all decision trees in the forest. The algorithm can also provide information about the importance of each feature in the classification process.

The random forest classifier can be represented mathematically as follows:

Let X be a training dataset with n samples and m features, where $X = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ and y_i is the class label of x_i. The random forest classifier constructs T decision trees $f_t(x)$, where $t = 1, 2, \dots, T$.

At each node m of the t^{th} decision tree, a feature subset F_m is randomly selected from the full feature set F, where $|F_m| \ll |F|$. The best feature f_k is then selected from F_m to split the node based on some criterion, such as Gini impurity or information gain.

The final prediction of the random forest classifier for a new input x is given by the majority vote of the predictions of all decision trees:

$$\hat{y} = \frac{1}{T}\sum_{t=1}^{T} f_t(x)$$

where $\hat{y}$ is the predicted class label for x.

2.2.1.4 Logit Boost Classifier

The Logit Boost Classifier algorithm is a machine learning technique that is commonly used for binary classification problems. It works by iteratively training a set of weak classifiers and combining them to form a strong classifier. The algorithm seeks to minimize the negative log-likelihood loss function.

The Logit Boost algorithm [16] can be expressed mathematically as:

$$F_m(x) = F_{m-1}(x) + \gamma_m \cdot h_m(x)$$

where $F_m(x)$ is the predicted response for a given input x at iteration m, $F_{m-1}(x)$ is the predicted response from the previous iteration, γ_m is the learning rate for iteration m, and $h_m(x)$ is the weak classifier at iteration m.

At each iteration, the algorithm finds the weak classifier $h_m(x)$ that minimizes the negative gradient of the loss function, given by:

$$r_{im} = -\frac{\partial \mathcal{L}\big(y_i, F_{m-1}(x_i)\big)}{\partial F_{m-1}(x_i)}$$

where $\mathcal{L}$ is the negative log-likelihood loss function, y_i is the true label for input x_i and $F_{m-1}(x_i)$ is the predicted response from the previous iteration for input x_i.

The weak classifier $h_m(x)$ is then trained on the negative gradient r_{im} as the target variable, and the predicted response $h_m(x)$ is added to the overall prediction $F_m(x)$. This process is repeated for a set number of iterations or until a stopping criterion is met.

2.2.2 Deep Learning Models Description

Deep learning models are a class of artificial neural networks that can learn and extract complex representations from raw data through multiple layers of nonlinear transformations. These models have gained popularity in recent years due to their remarkable performance in various tasks, such as image recognition, natural language processing, speech recognition, and many others. Deep learning models are often composed of several layers, each of which performs a specific operation on the input data, allowing the network to learn increasingly complex

features and patterns. Some popular deep learning models we have used in our thesis for different work in chapter 3 & 4, that are describe in the following section.

2.2.2.1 Multi-layer Perceptron

Multi-layer Perceptron (MLP) is a type of artificial neural network that is commonly used for supervised learning tasks, such as classification and regression. It consists of multiple layers of interconnected nodes, with each node using a non-linear activation function to transform its input.

The normal MLP algorithm can be represented mathematically using the following equation:

$$y_k = \sigma\left(\sum_{j=1}^{m} w_{kj}^{(2)} \cdot \sigma\left(\sum_{i=1}^{n} w_{ji}^{(1)} x_i + b_j^{(1)}\right) + b_k^{(2)}\right)$$

In this equation, x_i represents the input features, y_k represents the output of the MLP for the k-th output neuron, $w_{ji}^{(1)}$ and $w_{kj}^{(2)}$ are the weights connecting the input layer to the hidden layer and the hidden layer to the output layer, respectively. $b_j^{(1)}$ and $b_k^{(2)}$ are the biases of the hidden and output layers. The activation function $\sigma(\dots)$ is typically a non-linear function such as the sigmoid function or the Rectified Linear Unit (ReLU) function.

The MLP algorithm learns by adjusting the weights and biases in order to minimize a loss function, such as the mean squared error or cross-entropy loss. This is typically done using a backpropagation algorithm, which calculates the gradient of the loss function with respect to the weights and biases and updates them accordingly.

Overall, the MLP algorithm is a powerful tool for a wide range of machine learning tasks, particularly in areas such as image and speech recognition, natural language processing, and financial forecasting.

2.2.2.2 Bidirectional LSTM

This section focuses on the Bidirectional Long Short-Term Memory (LSTM) algorithm, which is a type of recurrent neural network (RNN) commonly used in natural language processing (NLP) tasks such as speech recognition, sentiment analysis, and language translation. But we can use it in disease diagnosis also.

Bidirectional LSTM [17] can process sequential data in both forward and backward directions. This architecture includes two layers of LSTM, one that processes the input sequence in the forward direction and another that processes it in the backward direction. The outputs from both layers are concatenated to produce the final output.

The mathematical representation of the LSTM algorithm can be expressed as follows:

$$i_t = \sigma(W_{xi} x_t + W_{hi} h_{t-1} + b_i)$$

$$f_t = \sigma\left(W_{xf} x_t + W_{hf} h_{t-1} + b_f\right)$$

$$C_t = f_t * C_{t-1} + i_t * \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

where:

- x_t represents the input at time step t
- h_t represents the hidden state at time step t
- c_t represents the cell state at time step t
- i_t, f_t, and o_t represent the input, forget, and output gates respectively
- σ represents the sigmoid function
- $*$ represents element-wise multiplication
- W and b are weight and bias parameters to be learned during training with *i, f, o* which represents the input, forget, and output gates respectively

The mathematical formulation of the Bidirectional LSTM algorithm can be expressed using the following equation:

$$h_t^f = \text{LSTM}f\left(x_t, h_{t-1}^f, c_{t-1}^f\right)$$

$$h_t^b = \text{LSTM}b\left(x_t, h_{t+1}^b, c_{t+1}^b\right)$$

$$h_t{}' = \left[h_t^f, h_t^b\right]$$

In this equation, h_t^f and h_t^b are the hidden states of the forward and backward LSTM layers, respectively, at time step t. x_t represents the input at time step t, and c_{t-1}^f and c_{t+1}^b are the cell states of the LSTM layers. The final output of the Bidirectional LSTM model is a concatenation of the hidden states from both layers, denoted by $h_t{}'$.

This bidirectional processing allows the model to capture information from both past and future contexts, making it particularly effective for tasks involving sequential data such as natural language processing and speech recognition.

2.2.2.3 Supervised Variational Autoencoder

Supervised Variational Autoencoder (SVAE) [18] is an extension of the standard VAE framework that incorporates labelled training data. SVAE can be used for both supervised and semi-supervised learning tasks, where the goal is to learn a latent representation that captures both the input data and its associated labels.

The SVAE model consists of an encoder network, a decoder network, and a classifier network. The encoder network maps the input data to a latent representation, while the decoder network maps the latent representation back to the original input. The classifier network maps the latent representation to a probability distribution over the possible labels. The encoder, decoder, and classifier networks are typically implemented as neural networks.

The loss function for training the SVAE model can be written as follows using equation:

$$L(\theta, \phi, \psi; x, y) = -D_{KL}[q_\phi(z|x)||p(z)] + E_{q_\phi(z|x)}[\log p_\theta(x|z)] - \alpha E_{q_\phi(z|x)}[\log q_\psi(y|z)]$$

where θ, ϕ, and ψ are the parameters of the decoder, encoder, and classifier networks, respectively; x is the input data; y is the label for the input data; z is the latent variable; $q_\phi(z|x)$ is the approximate posterior distribution over z given x; $p(z)$ is the prior distribution over z; $p_\theta(x|z)$ is the conditional distribution over x given z; $q_\psi(y|z)$ is the conditional distribution over y given z; and α is a hyperparameter that controls the trade-off between the reconstruction error and the classification error.

The KL (Kullback-Leibler) loss equation for can be written as:

$$D_{KL}[q_\phi(z|x)||p(z)] = \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log\left((\sigma_j)^2\right) - (\mu_j)^2 - (\sigma_j)^2\right)$$

where μ_j and σ_j are the mean and standard deviation of the j-th dimension of the latent variable z.

While resolving the reconstruction term, by Monte Carlo evaluation, we attain the succeeding equation

$$E_{q_\phi(z|x)}[\ log\ p_\theta(x|z)]\ = \frac{1}{L}\sum_{l=1}^{L} log\, p_\theta(x|z^l)$$

where L is the total no. of samples and z^l is the sample of L.

The first two terms in the loss function are the same as those in the standard VAE, which encourage the learned latent representation to follow the prior distribution and reconstruct the input data. The third term is the classification error, which measures how well the model can predict the label from the learned latent representation.

2.2.3 Feature Selection

Feature selection is the process of identifying and selecting a subset of relevant features (variables) from a larger set of available features in a dataset. The goal of feature selection is to improve the performance of a machine learning model by reducing the complexity of the model, avoiding overfitting, and improving the interpretability of the results. There are various methods of feature selection, including filter methods, wrapper methods, and embedded methods, which use different criteria to select the most relevant features for a given problem. Feature selection is an important step in the machine learning pipeline, and its success depends on the quality of the dataset and the appropriate selection of the feature selection method.

Feature selection using Pearson correlation is a commonly used technique in machine learning to identify which features are most strongly correlated with the target variable. This involves calculating the Pearson correlation coefficient between each feature and the target variable, and selecting the features with the highest absolute correlation values.

2.2.3.1 Pearson Correlation

Pearson correlation is a statistical method used to measure the strength of a linear relationship between two variables. It is commonly used in data analysis to determine the degree to which two variables are related to each other. Pearson correlation is also known as Pearson's product-moment correlation coefficient, and is denoted by the symbol "r".

The Pearson correlation coefficient ranges from -1 to 1. A value of -1 indicates a perfectly negative correlation, which means that as one variable increases, the other variable decreases. A value of 1 indicates a perfectly positive correlation, which means that as one variable increases, the other variable also increases. A value of 0 indicates no correlation between the two variables.

The Pearson correlation coefficient is calculated by dividing the covariance of the two variables by the product of their standard deviations. The formula for Pearson correlation is as follows:

r = (sum of (x - mean of x) * (y - mean of y)) / (square root of sum of (x - mean of x)^2 * square root of sum of (y - mean of y)^2)

where x and y are the two variables being analysed, and the mean is the average value of each variable.

Pearson correlation can be used to identify the relationship between any two variables, such as the relationship between temperature and ice cream sales, or the relationship between age and income. It can also be used to determine the significance of a relationship, which helps in making predictions based on the data.

One of the limitations of Pearson correlation is that it only measures the strength of a linear relationship. If the relationship between the variables is not linear, then Pearson correlation may not be an accurate measure of the relationship. Additionally, Pearson correlation only measures the strength of the relationship between two variables, but it does not determine causation between them.

2.2.4 Feature Scaling

Feature scaling is a technique used in machine learning to standardize or normalize the range of features or variables in a dataset. It involves transforming the values of the features so that they fall within a specific range, which can help improve the performance of certain algorithms, particularly those that are sensitive to the scale of the input features. There are various methods of feature scaling, including min-max scaling, z-score normalization, and logarithmic scaling, Robust scaling and Standard scaling. In the below section we discuss about the last two methods.

2.2.4.1 Robust Scaling

Robust scaling is a technique used in data preprocessing to rescale numerical features so that they have the same range or variance. It is often used in machine learning and statistical

modeling to normalize data and improve model performance. The goal of robust scaling is to make the data more comparable across features, while also reducing the impact of outliers.

In traditional scaling methods, such as min-max scaling and standardization, the scaling factors are based on the mean and standard deviation of the data. However, these methods can be sensitive to outliers, as the scaling factors are heavily influenced by extreme values. Robust scaling addresses this issue by using robust statistics, such as the median and interquartile range (IQR), to calculate the scaling factors.

The most common robust scaling technique is the median absolute deviation (MAD) scaling. MAD scaling first computes the median of each feature, and then scales each value by the median absolute deviation of that feature. The MAD is a robust measure of variability that is less affected by outliers than the standard deviation.

Another popular robust scaling [19] method is the interquartile range (IQR) scaling, which uses the IQR instead of the standard deviation to calculate the scaling factors. The IQR is the difference between the 75th percentile (Q3) and the 25th percentile (Q1) of the data. This method is also less sensitive to outliers than standardization.

Robust scaling can improve the performance of machine learning models, especially when dealing with data that contains outliers or extreme values. By reducing the impact of outliers, robust scaling can help prevent the model from being biased towards the outliers and overfitting to the training data.

2.2.5 Cross Validation Techniques

Cross-validation techniques are used to assess the performance of machine learning models and to select the best model for a given task. These techniques involve dividing the available data into training and validation sets, and then evaluating the model on the validation set. There is some common Cross validation technique that are used in this thesis work which are mentioned in the below subsection.

2.2.5.1 K-Fold

K-fold cross-validation is a popular technique for evaluating the performance of a machine learning model. It involves splitting the dataset into k equally sized folds, and using each fold once as a validation set while the model is trained on the remaining k-1 folds.

The process can be summarized as follows:

- Split the dataset into k folds
- For each fold, use it as the validation set and train the model on the remaining k-1 folds
- Record the model's performance on the validation set
- Repeat steps 2-3 for each fold
- Calculate the average performance across all folds as the final estimate of the model's performance.

The advantage of k-fold cross-validation is that it provides a more reliable estimate of the model's performance by reducing the variance in the evaluation metric. This is because the model is evaluated on multiple validation sets rather than just one, which helps to capture the variability in the data.

One disadvantage of k-fold cross-validation is that it can be computationally expensive, especially for large datasets or complex models. Additionally, it can be difficult to interpret the results of k-fold cross-validation, as it can be challenging to determine the source of any performance differences between folds.

2.2.5.2 Stratified K-Fold

Stratified k-fold cross-validation is a variant of k-fold cross-validation that is particularly useful for imbalanced datasets, where the distribution of classes in the dataset is uneven. In this technique, the dataset is split into k folds such that each fold contains roughly the same proportion of each class as the overall dataset.

The process can be summarized as follows:

- Split the dataset into k folds
- For each fold, use it as the validation set and train the model on the remaining k-1 folds
- Ensure that the class distribution in the validation set is representative of the overall class distribution in the dataset
- Record the model's performance on the validation set
- Repeat steps 2-4 for each fold
- Calculate the average performance across all folds as the final estimate of the model's performance.

The advantage of stratified k-fold cross-validation is that it ensures that the model is evaluated on validation sets that are representative of the overall class distribution in the dataset. This can help to reduce the risk of bias in the model's evaluation, particularly when dealing with imbalanced datasets.

One disadvantage of stratified k-fold cross-validation is that it can be more computationally expensive than regular k-fold cross-validation, particularly when the number of classes in the dataset is large. Additionally, it can be difficult to interpret the results of stratified k-fold cross-validation, particularly when the class distribution in the dataset is complex.

2.2.5.3 Shuffle split

Shuffle-split cross-validation is a technique used to create multiple train/test splits of a dataset, allowing for more flexibility in the evaluation of machine learning models. In this technique, the dataset is randomly split into a specified number of train/test sets, with the size of each set determined by the user.

The process can be summarized as follows:

- Split the dataset into k test sets, with a specified test size and train size for each set.

- For each train/test split, train the model on the training set and evaluate its performance on the test set.
- Record the model's performance on each test set.
- Repeat steps 2-3 for each train/test split.
- Calculate the average performance across all train/test splits as the final estimate of the model's performance.

The advantage of shuffle-split cross-validation is that it allows for more control over the size of the train and test sets, and the number of train/test splits. This can be useful in situations where the size of the dataset is limited or where there are specific requirements for the distribution of the train and test data.

One disadvantage of shuffle-split cross-validation is that it can result in less statistically robust estimates of model performance, particularly if the number of train/test splits is small. Additionally, it can be more computationally expensive than other cross-validation techniques, particularly when the size of the dataset is large.

2.2.6 Metrices for Evaluation

Evaluation metrics are tools used to measure the performance of a machine learning model. The selection of the evaluation metrics depends on the type of problem being solved and the objectives of the project. In the thesis, we used some evaluation metrics for analysing the results that's are Precision, recall, F1 score, specificity, ROC-AUC score, balanced accuracy, accuracy, negative predictive value, F1 measure, and kappa score, False Positive Rate and False Negative Rate. Each metric provides different insights into the performance of a model, and their selection depends on the specific problem and objectives of the analysis.

1. Precision: Precision is the proportion of true positives among the total predicted positives. It is a measure of the accuracy of positive predictions.

2. Recall: Recall is the proportion of true positives among the total actual positives. It is a measure of the completeness of positive predictions.

3. F1 Score: F1 score is a harmonic mean of precision and recall. It is a single metric that provides a balance between the two measures and is useful when both precision and recall are important.

4. Specificity: Specificity is the proportion of true negatives among the total actual negatives. It is a measure of the ability of a model to correctly identify negative cases.

5. ROC-AUC Score: ROC-AUC score measures the area under the receiver operating characteristic (ROC) curve, which is a graphical representation of the trade-off between true positive rate and false positive rate at different probability thresholds. It is a measure of the overall performance of a binary classifier.

6. Balanced Accuracy: Balanced accuracy is the average of sensitivity (recall) and specificity. It is useful when the classes are imbalanced.

7. Accuracy: Accuracy is the proportion of correct predictions among the total predictions. It is a commonly used metric for classification problems with balanced classes.

8. Negative Predictive Value: Negative predictive value is the proportion of true negatives among the total predicted negatives. It is a measure of the accuracy of negative predictions.

9. F1 Measure: F1 measure is a weighted average of precision and recall. It is useful when both precision and recall are important, and the classes are imbalanced.

10. Kappa Score: Kappa score measures the agreement between the predicted and actual classifications, taking into account the agreement that could occur by chance. It is a useful metric for evaluating the performance of classifiers in multi-class problems.

11. False Positive Rate (FPR): It is the proportion of actual negative cases that are incorrectly classified as positive by the model. In other words, it measures the percentage of cases that are true negatives but are classified as false positives by the model. FPR is calculated using the following formula: FPR = False Positives / (False Positives + True Negatives)

12. False Negative Rate (FNR): It is the proportion of actual positive cases that are incorrectly classified as negative by the model. In other words, it measures the percentage of cases that are true positives but are classified as false negatives by the model. FNR is calculated using the following formula: FNR = False Negatives / (False Negatives + True Positives)

In summary, depending on the specific problem and objectives, a combination of these metrics may be used to provide a more comprehensive evaluation of the model's performance.

2.2.7 Data Encryption Algorithm

Encryption algorithms are a set of mathematical formulas and rules that are used to convert plain text into ciphertext, making it unreadable without the use of a specific key or password. Encryption is an essential part of modern security systems, as it ensures that sensitive information remains private and protected from unauthorized access. There are many different encryption algorithms, each with its own strengths and weaknesses, and they are constantly evolving as technology advances and new threats emerge.

Block ciphers are an essential part of many encryption algorithms because they provide a way to securely transform plaintext into ciphertext. In many encryption systems, the plaintext is divided into fixed-length blocks, and each block is processed using a block cipher algorithm with a secret key. This process results in a block of ciphertext that is as long as the original plaintext block. The use of a block cipher adds an extra layer of security by making it difficult for an attacker to determine the relationship between the plaintext and ciphertext. Block ciphers can be used in various encryption modes, such as ECB (Electronic Codebook), CBC (Cipher Block Chaining), CFB (Cipher Feedback), and OFB (Output Feedback), each offering unique advantages and disadvantages. Overall, block ciphers are an important building block in modern encryption systems and play a crucial role in protecting sensitive data.

There are many lightweight block ciphers that use a reduced block size or a reduced number of rounds to reduce the computational overhead. Some popular lightweight block cipher algorithms include SIMON, SPECK, PRESENT, and PRINCE. These ciphers are designed to be fast, efficient, and secure, while also being suitable for low-power and low-memory environments. Lightweight block ciphers are typically used in resource-constrained environments, such as embedded systems, wireless sensor networks, and Internet of Things

(IoT) devices, where the use of standard block ciphers like AES or DES may be impractical due to their high resource requirements.

2.2.7.1 Speck Cipher

The National Security Agency (NSA) released Speck [20], a family of lightweight block ciphers in June 2013. This family includes ten block ciphers with different key and block sizes. Previous block ciphers were designed to perform well on a specific platform, but not on a range of devices. Speck was created to address the need for secure, flexible, and analysable lightweight block ciphers. It provides excellent performance on both software and hardware platforms, and can be implemented in various ways. Additionally, this block ciphers can be analysed using existing techniques.

Speck block is represented as Simon2n/mn, where n is the word size and m are the number of key words which is discussed in Table 2.1.

Table 2.1: Speck Parameters

Block Size *2n*	**Key Size** *mn*	**Key Words** *m*	**Word Size** *n*	**Rot** *α*	**Rot** *β*	**Rounds** *T*
32	64	4	16	7	2	22
48	72	3	24	8	3	22
	96	4				23
64	96	3	32	8	3	26
	128	4				27
96	96	2	48	8	3	28
	144	3				29
128	128	2	64	8	3	32
	192	3				33
	256	4				34

The Speck Cipher design for encryption mainly comprises of two different functions-

1. The round function and
2. The Key Expansion functions.

Round Function: The Speck2n encryption round function utilizes the following operations on n-bit words:

- bitwise XOR, $\oplus$,
- addition modulo 2n, +,
- left and right circular shifts, S^{j} and S^{-j}, respectively, by j bits.

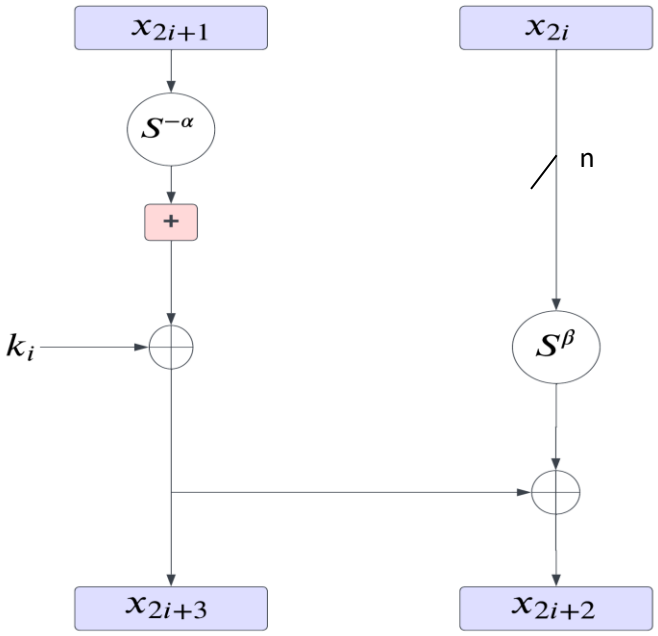

Figure 2.3: Speck encryption round function; (x_{2i+1}, x_{2i}) denotes the sub-cipher after i steps of encryption

The SPECK round function (used for encryption) can be expressed as:

$$R_k(x, y) = ((S^{-\alpha}x + y) \oplus k,\ S^{\beta}y \oplus (S^{-\alpha}x + y) \oplus k)$$

And its inverse (used for decryption) as:

$$R_k^{-1}(x, y) = (S^{\alpha}((x \oplus k) - S^{-\beta}(x \oplus y)),\ S^{-\beta}(x \oplus y))$$

Where x is the left-most word of a given block, y the right-most word and *k* the appropriate round key. The Speck key schedules take a key and from it generate a sequence of T key words $k_0, ..., k_{T-1}$, where T is the number of rounds. The effect of the single round function R_{ki} is shown in Figure 1. Encryption is then the composition $R_{kT-1} \circ ... \circ R_{k1} \circ R_{k0}$, read from right to left. Note that Speck can be realized as the composition of two Feistel-like maps with respect to two different types of addition, namely,

$$(x, y) \rightarrow (y, (S^{-\alpha}x + y) \oplus k) \text{ and } (x, y) \rightarrow (y, S^{\beta}x \oplus y).$$

This decomposition is pictured in Figure 2.4.

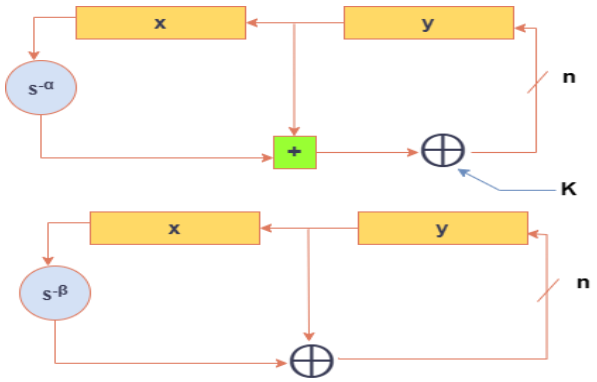

Figure 2.4: Speck round function decomposed into Feistel-like steps

Key Expansion function: The Speck key schedules use the own round function to generate round keys k_i. This is useful cause we don't need to implement a new method. Let K be a key for a Speck2n block cipher. We can write $K = (l_{m-2}, \ldots, l_0, k_0)$, where $l_i, k_0 \in GF(2)$n, for a value of m in 2, 3, 4. Sequences k_i and l_i are defined by

$$l_{i+m-1} = (k_i + S^{-\alpha} l_i) \oplus i$$

and

$$k_{i+1} = S^{\beta} k_i \oplus l_{i+m-1}.$$

The value k_i is the i^{th} round key, for $0 \leq i < T$. See Figure 2.5.

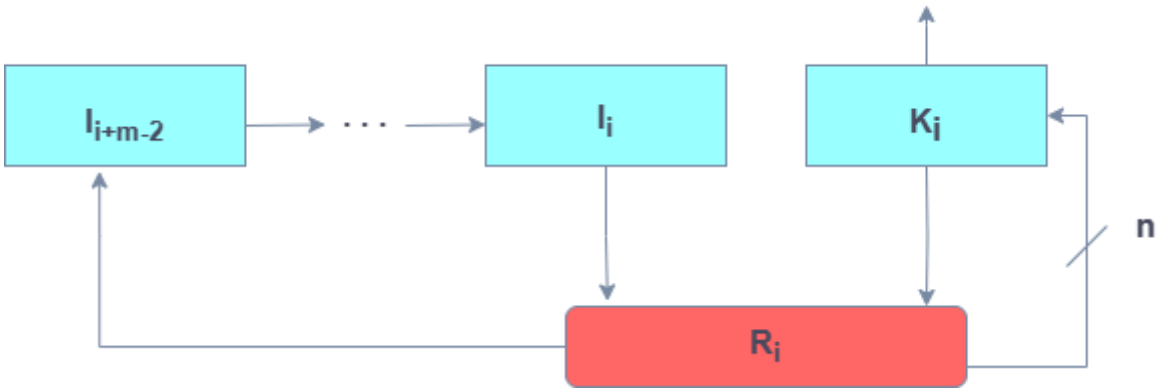

Figure 2.5: Speck key expansion, where R_i is the Speck round function with i acting as round key

2.2.8 Optimization Algorithms

Optimization algorithms are mathematical procedures used to find the best solution for a problem by minimizing or maximizing a specific objective function. In machine learning and deep learning, optimization algorithms are used to optimize the parameters of a model to minimize the loss function. There some common optimization algorithm that are used for model optimization in this thesis that are describe in the below section. A Nature Inspired Optimization Algorithm are used to generate the optimal key of the encryption technique that is also describe in the following section.

2.2.8.1 Nutcracker Optimization Algorithm

This section introduces a new released bio-inspired Nutcracker optimization algorithm (NOA) [21] based on the cunning actions of the nutcracker. The nutcracker's behaviour can be divided into two primary categories: the first involves collect and storing pine seeds (food), and the second involves searching for and retrieving storage locations. These two activities are distinguished by the fact that they occur during two different time periods. The initial activity takes place during the summer and fall seasons. The second habit, meanwhile, takes place during the winter and spring seasons. Based on the two primary behaviours, the proposed

algorithm simulates nutcracker behaviour. Foraging and storage strategy and cache-search and recovery strategy are the two basic tactics.

Foraging and storage strategy:

This can be described as follows:

- **Foraging stage: Exploration phase 1**

At this phase, the nutcrackers start moving into the search space to occupy their beginning positions/food spots that were created by Eq. (19). (Collection area). Every nutcracker starts by inspecting the cone that houses the seeds in their initial placement. When it comes across quality seeds, the nutcracker will carry them to the storage area and bury them in a cache. The nutcracker will search for another cone in a different location within pine trees or other trees if it is unable to obtain good seeds. The position update approach can be used to model this behaviour analytically as follows:

$$\vec{X}_i^{t+1} = \begin{cases} X_{i,j}^t & if\ \tau_1 < \tau_2 \\ \begin{cases} X_{m,j}^t + \gamma \cdot \left(X_{A,j}^t - X_{B,j}^t\right) \\ \quad +\mu.\left(r^2.U_j - L_j\right), & if\ t \le T_{max}/2.0 \\ X_{C,j}^t + \mu \cdot \left(X_{A,j}^t - X_{B,j}^t\right) \\ \quad +\mu.(r_1 < \delta).\left(r^2.U_j - L_j\right), Otherwise \end{cases} & otherwise \end{cases} \quad (1)$$

where X_i^{t+1} is described as new position of the ith nutcracker in the current generation t; $X_{i,j}^t$ is denoted as j[th] position of the i[th] nutcracker in the present generation; U_j and L_j are two vectors, including the upper and lower bound of the j[th] dimension in the optimization problem;γ is a random number generated according to the levy flight; $X_{best,j}^t$ is the jth dimension of the best solution obtained up to this; A, C, and B are three distinct indices that were arbitrarily chosen from the population to aid in the search for a high-quality food source; τ_1, τ_2, r and r_1 are random real numbers in the range of [0,1]; $X_{m,j}^t$ is the mean of the jth dimensions of all solutions of the current population in iteration t; and μ is a number generated based on the normal distribution(τ_4), levyflight(τ_5) and randomly between one(τ_3) and zero shown in the below equation:

$$\mu = \begin{cases} \tau_3 & if\ r_1 < r_2 \\ \tau_4 & if\ r_2 < r_3 \\ \tau_5 & if\ r_1 < r_3 \end{cases} \quad (2)$$

where [0,1]-ranged random real values r_2 and r_3 are used. Eq. (1) was suggested to investigate high-quality food sources. The first state of Eq. (1) simulates the probability that the nutcrackers discover good seeds on the first try. This notion means that the nutcrackers will not change their original j[th] dimension in the solution. The second state of Eq. (1) was proposed to allow the nutcrackers to globally explore random positions in the search space to enhance the search capability of NOA for exploring the search space as must as possible to avoid being stuck into local minima and reaching the promising regions that might include the near-optimal solution. In addition to moving globally in the search space with a probability δ to cover

intractable regions by the second state of Eq. (1), the third state of Eq. (1) was introduced to allow the nutcrackers to investigate places surrounding a solution randomly chosen from the population (1). Later on, the sensitivity analysis for determining the ideal value for δ is shown. It was proposed that A and B inform the nutcrackers of the new food position. As scaling factors to control the exploration capacity in NOA, τ_1 and τ_2 were proposed. The suggested algorithm supports a variety of small and large values for τ_1 and τ_2, and can switch between local and global searches. μ explains to the nutcracker the necessary direction and different step sizes to explore a new area.

- **Storage stage: Exploitation phase 1**

In the first step of investigation, phase 1, nutcrackers move the food they have collected to temporary storage areas (storage area). In this phase, referred to as "exploitation phase 1," the nutcrackers harvest and store pine seed crops. Mathematically, this behaviour can be described as follows:

$$\vec{X}_i^{t+1(new)} = \begin{cases} \vec{X}_i^t + \mu \cdot \left(\vec{X}_{best}^t - \vec{X}_i^t\right) \cdot |\lambda| + r_1 \cdot \left(\vec{X}_A^t - \vec{X}_B^t\right) & if\ \tau_1 < \tau_2 \\ \vec{X}_{best}^t + \mu \cdot \left(\vec{X}_A^t - \vec{X}_B^t\right) & if\ \tau_1 < \tau_3 \\ \vec{X}_{best}^t \cdot l & Otherwise \end{cases} \quad (3)$$

where $\vec{X}_i^{t+1(new)}$ is a fresh location in the nutcrackers' storage region in iteration t, λ is a number produced using the levy flight, and τ_3 is a random number between 0 and 1. To diversify NOA's exploitation behaviour, the component l was linearly reduced from 1 to 0. This variation in the NOA's exploitation operator will speed up convergence while also preventing the possibility of becoming caught in local minima when searching in one direction.

To maintain the balance between exploration and exploitation operators during the optimisation process, the exchange between the foraging phase and the cache is applied using the following formula:

$$\vec{X}_i^{t+1} = \begin{cases} \text{Eq. (1)}, & if\ \varphi > P_{a_1} \\ \text{Eq. (3)}, & otherwise \end{cases} \quad (4)$$

where φ is a probability value that decreases linearly from one to zero based on the current generation and is a random number between zero and one. This step can drive the bulk of the solutions in one direction, covering the solutions there as much as possible while looking for the nearly ideal solution, according to Fig. 4(b), which illustrates the direction of the solutions generated using this stage. The first suggested strategy's exploration and exploitation procedures are flow-charted in Fig. 5 and listed in Algorithm 1.

Reference memory

The nutcrackers start the spatial memory after putting the seeds in the cache. In order to recall their cache, the nutcrackers create a mental image of it and use certain items as cues. To make it through the winter, these nutcrackers almost entirely rely on their recollection of where those caches are. Nutcrackers meticulously inspect and sketch the site of storage.

Their single chance of reproduction and survival is hidden food. Furthermore, because they are covered with snow during the winter, local characteristics are useless while nutcrackers appear to interpret towering trees or prominent topographical features as signs. The behaviour of nutcrackers retrieving their cache will then be simulated.

Cache-search and recovery strategy

There are two primary phases to this strategy: Stages of cache-search and recovery, which are covered in depth in the next two subsections:

- **Cache-search stage: Exploration phase 2**

When winter arrives, the trees become naked, signalling that it is time to leave hiding mode and transition to exploring and searching. The nutcrackers start looking for their stashes. This stage is known as the second exploration. To find their caches, the nutcrackers employ a spatial memory technique. Many objects are most likely used by nutcrackers as signals for a single cache. For the sake of simplicity, we'll assume that each cache contains just two objects. The markers or objects will be hidden at various angles from the concealing site. At this phase, the nutcrackers begin to assume their beginning positions/cache positions in the search space, which were generated by Eq. (19). (Storage area). These things are what we refer to as Reference Points (RPs). Using the following matrix, two RPs for each cache/nutcracker in the population can be defined in NOA:

$$\mathrm{RPs} = \begin{bmatrix} \overrightarrow{RP}^{t}_{1,1} & \overrightarrow{RP}^{t}_{1,2} \\ \vdots & \vdots \\ \overrightarrow{RP}^{t}_{i,1} & \overrightarrow{RP}^{t}_{i,2} \\ \vdots & \vdots \\ \overrightarrow{RP}^{t}_{N,1} & \overrightarrow{RP}^{t}_{N,1} \\ \vdots & \vdots \end{bmatrix} \tag{5}$$

Thus, for the i^{th} nutcracker in the current generation t, stand for RPs (objects) of the cache position (). We suppose that the three items represent the vertices of a triangle to demonstrate the relationship between the nutcracker, cache, and reference point. Nutcrackers can be seen from a variety of locales and perspectives. Three different 3D positions are shown for the nutcracker in Fig. 6, along with three different RP sites for a single cache. We presume that the cache is located at the origin point in 3D space. Nutcrackers are seen from different perspectives. The angles at which the first, second, and third locations are situated are acute, obtuse, and right, respectively. In 3D space, a nutcracker can be found on either the diagonal or coordinate axes. For RPs, the same is true.

Nutcrackers are highly accurate in finding hidden caches. However, pertinent research show that 20% of nutcracker attempts are unsuccessful on the first try [22]. The second RP will be used to identify the meal if the nutcracker is unable to find it using the first RP. The nutcracker will utilise the third reference if it is unable to get its cache a second time. To facilitate nutcracker exploration while looking for hidden caches, two different equations are created to generate the first and second RPs. In order to locate hidden caches near the nutcrackers, the current position is updated within the surrounding regions to create the first RP. The first RP is created using the following mathematical formula:

$$\overrightarrow{RP}_{i,k}^{t} = \vec{X}_i^t + \alpha \cdot \cos(\theta) \cdot \left(\left(\vec{X}_A^t - \vec{X}_B^t\right)\right), k = 1 \quad (6)$$

To assist the nutcrackers in investigating various areas in quest of the hidden caches, the second RP is created by modifying the present resolution within the search space of the problem. The following formula is used to calculate the second RP:

$$\overrightarrow{RP}_{i,k}^{t} = \vec{X}_i^t + \alpha \cdot \cos(\theta) \cdot \left(\left(\vec{U} - \vec{L}\right) \cdot \tau_3 + \vec{L}\right) \cdot \overrightarrow{U_2}, k = 2 \quad (7)$$

$$\overrightarrow{U_1} = \begin{cases} 1 & \overrightarrow{r_2} < P_{rp} \\ 0 & otherwise \end{cases} \quad (8)$$

where k stands for the RP index, $RP^{t}_{i,k}$ for the cache position of the i^{th} nutcracker in the current iteration t, and U and L for the higher and lower bounds of a D-dimensional problem, respectively. X^{t}_{A} is the cache position of the A^{th} nutcracker in the current iteration t; r2 is a vector with values randomly generated between zero and one; r3 is a random number of the second RP in the range [0, 1]; r is a random in the range [0, 1]; and P_{rp} is a probability used to calculate the proportion of globally investigating different areas within the search area.

To restore its cache, every nutcracker needs two RPs, which are produced using equations (6) and (7). The RP can quickly recover it when it is near the cache. In the first generation, it is assumed that the nutcrackers lack the expertise to choose the proper RPs (i.e., those that are close to the cache). However, with time, nutcrackers develop more expertise in the act of storage. To make it simple to retrieve items later, nutcrackers picture locations nearby their warehouses. It was suggested in these calculations that the nutcrackers be trained and given enough experience to pick an appropriate location for RPs. The early convergence of RPs towards the correct position is prevented in NOA by (cache site). The nutcracker's angle-of-view, which ranges from 0 to, is determined at random. The various nutcracker viewing angles are: (a) $\theta = 0$; (b) $0 < \theta < \pi/2$; (c) $\theta = \pi/2$; (d) $0 < \theta < \pi$; (e) $\theta = \pi$. showing that the RP position is on the same cache position. Eqs. (6) and (7) can be rewritten as follows to address this

$$\overrightarrow{RP}_{i,1}^{t} = \begin{cases} \vec{X}_i^t + \alpha \cdot \cos(\theta) \cdot \left(\left(\vec{X}_A^t - \vec{X}_B^t\right)\right) + \alpha \cdot RP, & if\ \theta = \pi/2 \\ \vec{X}_i^t + \alpha \cdot \cos(\theta) \cdot \left(\left(\vec{X}_A^t - \vec{X}_B^t\right)\right), & otherwise \end{cases} \quad (9)$$

and

$$\overrightarrow{RP}_{i,2}^{t} = \begin{cases} \vec{X}_i^t + \left(\alpha \cdot \cos(\theta) \cdot \left(\left(\vec{U} - \vec{L}\right) \cdot \tau_3 + \vec{L}\right) + \alpha \cdot RP\right) \cdot \overrightarrow{U_2}, & if\ \theta = \pi/2 \\ \vec{X}_i^t + \alpha \cdot \cos(\theta) \cdot \left(\left(\vec{U} - \vec{L}\right) \cdot \tau_3 + \vec{L}\right) \cdot \overrightarrow{U_2}, & otherwise \end{cases} \quad (10)$$

where $RP^{t}_{i,1}$ denotes the i^{th} row, first column, in the matrix given in Eq. (5); () denotes the i^{th} row, second column, in Eq. (5). The cache/nutcracker index is represented by each row index in the matrix. In the meantime, the RP index is represented by each column index in the matrix. A random position is RP. To prevent the early convergence of RPs towards a potential solution, the third term of the initial state of Equations (9) and (10) (i.e., $\theta = \pi/2$) has been added, makes

sure the NOA converges frequently, enabling the nutcracker to enhance RP selection in upcoming generations. α can be determined using the following formula:

$$\alpha = \begin{cases} \left(1 - \frac{t}{T_{max}}\right)^{2\frac{t}{T_{max}}}, & if\ r_1 > r_2 \\ \left(\frac{t}{T_{max}}\right)^{\frac{2}{t}}, & otherwise \end{cases} \quad (11)$$

where the current and maximum generations are denoted by t and T_{max}, respectively. To speed up the proposed algorithm's convergence, the first state in Eq. (11) drops linearly with each iteration. In the meantime, the second state grows linearly to prevent falling into local minima that could happen as a result of the first state.

The optimisation method gives all nutcrackers sufficient practise in choosing the right RPs. According to the applicable RPs, these nutcrackers update the storage locations/solutions. All nutcrackers in NOA will use the exploration mechanism to look for the most promising regions that could have a nearly ideal answer. To prevent becoming stuck in local minima, the algorithm will search and utilise locations nearby caches with the proper RPs with each generation that passes. The following equation can be used to modify a nutcracker's position:

$$\vec{X}_i^{t+1} = \begin{cases} \vec{X}_i^t, & if\ f\left(\vec{X}_i^t\right) < if\ (\overrightarrow{RP}_{i,1}^t) \\ \overrightarrow{RP}_{i,1}^t, & otherwise \end{cases} \quad (12)$$

where $X^{t+1}{}_i$ is the new location/new cache of the i^{th} nutcracker at iteration (t+1), X_i is the current position/current cache of the i^{th} nutcracker in the present iteration t, and $RP^t_{i,1}$ is the first RP of the current cache of the i^{th} nutcracker at iteration t. Eq. (12) was proposed to guide NOA to explore and exploit the promising areas of the $RP^t_{i,1}$. If NOA is unable to obtain the optimal locations around $RP^t_{i,1}$, it will investigate them in other regions, such as those near the reference $RP^t_{i,2}$ which are covered in more detail below.

- **Recovery stage: Exploitation phase 2**

The first significant possibility is that a nutcracker can recall his cache's location using the first RP. The following are the two scenarios. The initial scenario is that there is food, and the second is that there isn't any. The following equation can be used to mathematically model this behaviour.

$$X_{i,j}^{t+1} = \begin{cases} X_{i,j}^t, & if\ \tau_3 < \tau_4 \\ X_{i,j}^t + r_1 \cdot \left(X_{best,j}^t - X_{i,j}^t\right) + r_2 \cdot \left(\overrightarrow{RP}_{i,1}^t - X_{C,j}^t\right), & otherwise \end{cases} \quad (13)$$

where denotes the new position or cache of the i^{th} nutcracker at iteration (t+1), $X^t_{i,j}$ denotes the present dimension or cache of the i^{th} nutcracker in iteration t, X^t_b denotes the best position or cache in iteration t, and $RP^t_{i,1}$ denotes the first RP of the present location or cache of the i^{th} nutcracker in iteration t.

Eq. (13)'s first state simulates the initial scenario (food exists). This idea implies that there is a potential that some dimensions in this cache/solution will persist until the future generation.

The nutcracker will put more emphasis on food storage the following time because a cache that preserves pine seeds is actually beneficial. Eq. (13)'s second state simulates the second scenario. (Food does not exist). Since the solution region in this case does not look promising, the algorithm will use an escape strategy to prevent becoming caught in local minima. Food from the cache is lost for a variety of causes, including theft by another nutcracker or bird, natural disasters like rain and snow, or human error.

Eq. (13) state (1) enables a nutcracker to take advantage of potential search space areas and improve the local search capabilities of NOA. To improve the global search capabilities of NOA, eq. (13) state (2) enables a nutcracker to explore random areas in searching area.

The second main possibility is that the nutcracker will use the second RP to look for his hidden food because he cannot recall where it is using the first RP. A nutcracker memorises a lot of the RPs he will take during early storage. Nutcrackers restore caches on the first attempt (with the first reference), but the suggested technique takes the likelihood of failing on the first attempt into account. When the weather changes between autumn (storage time) and winter, the nutcrackers who can't discover their cache rely on nearby landmarks that may vanish. (Recovery time). The second RP is used for updating the nutcracker's spatial memory, Eq. (12):

$$\vec{X}_i^{t+1} = \begin{cases} \vec{X}_i^t, & if\ f\left(\vec{X}_i^t\right) < if\ (\overrightarrow{RP}_{i,2}^t) \\ \overrightarrow{RP}_{i,2}^t, & otherwise \end{cases} \tag{14}$$

where $RP^t_{i,2}$ is the second RP of the present cache of the i^{th} nutcracker at iteration t and X^t_i is the current position/current cache of the i^{th} nutcracker in the current iteration t. The NOA has the chance to investigate uncharted territory around the second RP and take advantage of attractive places where a viable solution might be located thanks to Eq. (14). In NOA, it is expected that a nutcracker will use the second RP to locate its cache, hence Eq. (13) is revised utilizing the second RP:

$$X_{ij}^{t+1} = \begin{cases} X_{ij}^t, & if\ \tau_5 < \tau_6 \\ X_{ij}^t + r_1 \cdot \left(X_{best,j}^t - X_{ij}^t\right) + r_2 \cdot \left(\overrightarrow{RP}_{i,2}^t - X_{Cj}^t\right), & otherwise \end{cases} \tag{15}$$

where [0,1] is the range for the random values r_1, r_2, τ_5, and τ_6. The algorithm can increase the local search around the most suitable areas that contain the nearly optimal solution by entering the initial state of Eq. (15). The second condition of Equation (15) in the contract permits the algorithm to expand its ability to perform global searches by looking for new locations in the search space. In conclusion, the recovery behaviour may be expressed as the following equation:

$$\vec{X}_i^{t+1} = \begin{cases} \text{Eq. (13)}, & if\ \tau_7 < \tau_8 \\ \text{Eq. (15)}, & otherwise \end{cases} \tag{16}$$

where [0,1] is a range of random numbers, and τ7 and τ8 is one among them. A nutcracker who recalls the secret store is represented by the first instance in the equation above, whereas a nutcracker who forgets the hidden store is represented by the second case. Now, the subsequent

equation describes how the exploring behaviours related to the initial and secondly RPs are balanced:

$$\vec{X}_i^{t+1} = \begin{cases} \text{Eq. (12)}, & if\ f(\text{Eq. (12)}) < f(\text{Eq. (14)}) \\ \text{Eq. (14)}, & otherwise \end{cases} \quad (17)$$

Finally, to maintain a balance between exploration and exploitation, the exchange between the cache-search stage and the recovery stage is applied using the following formula:

$$\vec{X}_i^{t+1} = \begin{cases} \text{Eq. (16)}, & if\ \phi > P_{a_2} \\ \text{Eq. (17)}, & otherwise \end{cases} \quad (18)$$

where is a chance number between 0 and 1, Pa2 is a probability value equal to 0.2, and this value was determined through additional trials.

$$\vec{X}_{i,j}^{t} = +\left(\vec{U}_j - \vec{L}_j\right) \cdot \overrightarrow{RM} + \vec{L}_j,\ I = 1, 2, \ldots, N,\ j = 1, 2, \ldots, D \quad (19)$$

$$\vec{X}_i^{t+1} = \begin{cases} \vec{X}_i^{t+1}, & if\ f(\vec{X}_i^{t+1}) < f(\vec{X}_i^{t}) \\ \vec{X}_i^{t}, & otherwise \end{cases} \quad (20)$$

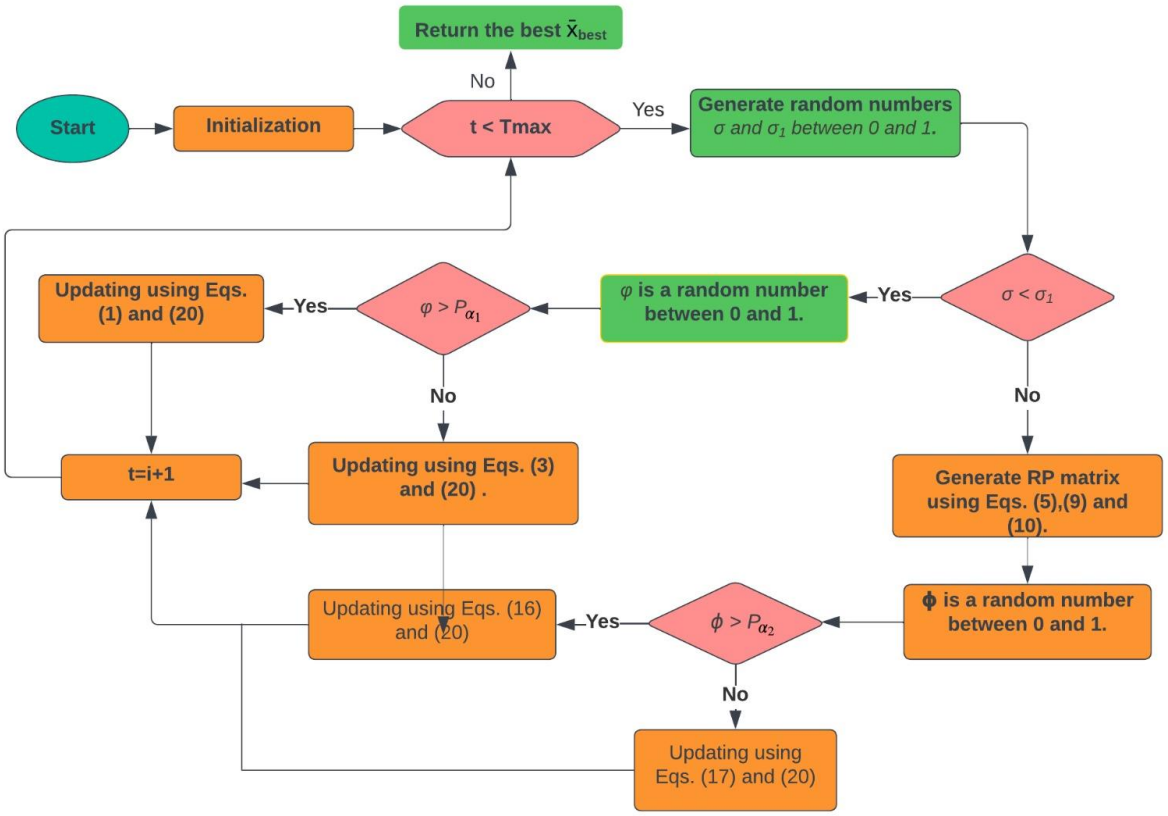

Figure 2.6: Flowchart of NOA

They first take into account population, decision parameter, and fitness value relating to the students in order to alter this approach to be suited for optimization procedure [28]. Subsequently, a straightforward group teaching module is developed without losing generalizability. The approach achieves consensus via transaction witness and pseudorandom sortition. Throughput, latency, and latency three dimensions have been added to the formula to boost the algorithm's scalability. Using a consensus technique with strong scalability, low latency, high throughput, and decentralised properties is recommended.

The NOA algorithm is used in this work to offer an enhanced hybrid consensus algorithm. Verifiable cryptographic sortition chooses the consensus node in a dynamic manner, allowing a large number of nodes to fairly participate in the consensus while assuring low latency and high throughput.

The Function is used as a objective function in our thesis is following:

$$F14 = \left(\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^2} \right)^{-1}$$

The domain range of the above function is (-65 to 65) and the global optima is 1. The dimension of the function is 2. Among 2 values we select the maximum value as the optimal key.

2.2.8.2 HyperOpt

HyperOpt [23] is an open-source Python library used for optimizing the ML and DL models. The algorithm works by iteratively exploring a hyperparameter space to find the set of hyperparameters that minimizes the objective function.

The objective function is defined as a function of the hyperparameters to be tuned and represents the performance metric of the machine learning model. HyperOpt uses a tree-structured Parzen estimator to model the objective function and search for the optimal hyperparameters.

The algorithm uses the Bayesian optimization approach to update the model at each iteration. The model is updated using the posterior distribution of the objective function, which is estimated using the prior distribution and the observations made so far.

The hyperparameter search space is defined using a prior distribution over the hyperparameters. The prior distribution can be any probability distribution that captures the prior knowledge about the hyperparameters. HyperOpt uses a Gaussian process to model the prior distribution over the hyperparameters.

The algorithm iteratively samples new hyperparameters from the posterior distribution of the objective function and evaluates the performance of the machine learning model using these hyperparameters. The process continues until a stopping criterion is met, such as a maximum number of iterations or a threshold performance level.

The optimization problem can be formulated as follows:

$$\min_{\theta} f(\theta)$$

where θ represents the hyperparameters to be tuned and $f(\theta)$ is the objective function to be minimized.

The Bayesian optimization approach updates the posterior distribution of the objective function at each iteration using Bayes' rule:

$$P(f|D_{1:t},\theta) = \frac{P(D_t|f,\theta)P(f|D_{1:t-1},\theta)}{P(D_t|D_{1:t-1},\theta)}$$

where $P(f|D_{1:t},\theta)$ is the posterior distribution of the objective function given the observations $D_{1:t}$ and the hyperparameters θ, $P(D_t|f,\theta)$ is the likelihood of the new observation given the objective function and the hyperparameters, $P(f|D_{1:t-1},\theta)$ is the prior distribution of the objective function given the previous observations and the hyperparameters, and $P(D_t|D_{1:t-1},\theta)$ is the marginal likelihood of the new observation given the previous observations and the hyperparameters.

HyperOpt uses a tree-structured Parzen estimator to model the posterior distribution of the objective function. The estimator is defined as:

$$\hat{p}(f|\theta) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{V}\sum_{j=1}^{V} cN\left(f\middle|f_i,\sigma_j^2\right)$$

where $\hat{p}(f|\theta)$ is the estimated posterior distribution of the objective function given the hyperparameters θ, N is the number of observations, V is the number of samples in the estimator, f_i is the ith observation of the objective function, σ_j^2 is the variance of the jth component of the estimator, and $\mathcal{N}\left(f\middle|f_i,\sigma_j^2\right)$ is the normal distribution with mean f_i and variance σ_j^2.

The algorithm uses the posterior distribution to sample new hyperparameters to evaluate the machine learning model. The sampling is done using an acquisition function, which balances the exploration of the hyperparameter space and the exploitation of the best hyperparameters found so far. The acquisition function is defined as:

$$a(\theta) = E_{f|\theta}[max(f_{1:t}) - f(\theta)]$$

where $a(\theta)$ is the acquisition function, $max(f_{1:t})$ is the maximum observed value of the objective function so far, and $E_{f|\theta}$ is the expectation over the posterior distribution of the objective function given the hyperparameters θ.

HyperOpt uses the Expected Improvement (EI) acquisition function, which is defined as:

$$EI(\theta) = E_{f|\theta}[max(f_{1:t}) - f(\theta)]\Phi(Z) + \sigma_{f|\theta}\phi(Z)$$

where $EI(\theta)$ is the Expected Improvement at hyperparameters θ, $\Phi(Z)$ and $\phi(Z)$ are the cumulative distribution function and probability density function of the standard normal distribution, respectively, and Z is the standard normal random variable:

$$Z = \frac{max(f_{1:t}) - f(\theta)}{\sigma_{f|\theta}}$$

where $\sigma_{f|\theta}$ is the standard deviation of the posterior distribution of the objective function given the hyperparameters θ. The algorithm samples new hyperparameters by maximizing the acquisition function. The optimization problem can be formulated as follows:

$$\theta_{t+1} = \arg\max_{\theta} a\,(\theta)$$

HyperOpt has been shown to be effective in optimizing the hyperparameters of various machine learning models and is widely used in research and industry.

2.2.8.3 Optuna

Optuna is a Python library for hyperparameter optimization [24], which is the process of tuning the parameters of a machine learning model to improve its performance. Optuna uses a Bayesian optimization algorithm to efficiently search the hyperparameter space.

The Bayesian optimization algorithm in Optuna uses a probabilistic model, called a surrogate model, to estimate the objective function (e.g., the accuracy of a machine learning model) at unexplored points in the hyperparameter space. The surrogate model is then used to guide the search towards promising regions of the hyperparameter space.

The Bayesian optimization algorithm in Optuna can be described by the following equation:

$$\alpha_{t+1} = \arg\max_{\alpha} E\,[y|\mathcal{D}_t, \alpha]$$

where α represents the hyperparameters being optimized, y represents the objective function (e.g., the accuracy of a machine learning model), $\mathcal{D}_t$ represents the set of observations at iteration t, and $E[y|\mathcal{D}_t, \alpha]$ represents the expected value of the objective function at hyperparameters α given the observations $\mathcal{D}_t$.

The Bayesian optimization algorithm in Optuna uses a probabilistic model to estimate $E[y|\mathcal{D}_t, \alpha]$. This probabilistic model is typically a Gaussian process or a tree-structured Parzen estimator (TPE).

The Bayesian optimization algorithm in Optuna iteratively selects hyperparameters to evaluate by optimizing the acquisition function:

$$\alpha_{t+1} = \arg\max_{\alpha} \kappa_t\,(\alpha)$$

where $\kappa_t(\alpha)$ represents the acquisition function at iteration t and hyperparameters α. The acquisition function balances exploration (i.e., selecting hyperparameters in unexplored regions of the hyperparameter space) and exploitation (i.e., selecting hyperparameters in regions of the hyperparameter space with high expected objective function values).

In summary, Optuna uses a Bayesian optimization algorithm to efficiently search the hyperparameter space by iteratively selecting hyperparameters to evaluate based on a probabilistic model and an acquisition function.

2.2.8.4 Cuckoo Search

Cuckoo Search is a nature-inspired optimization algorithm that is based on the breeding behaviour of cuckoo birds. The algorithm was first proposed by Xin-She Yang and Suash Deb in 2009 [25].

The Cuckoo Search algorithm is based on the concept of brood parasitism, where cuckoo birds lay their eggs in the nests of other bird species. The host birds may either incubate the cuckoo eggs or reject them. The cuckoo chicks that hatch from the eggs may then push the host bird's eggs out of the nest or outcompete the host bird's chicks for food.

The Cuckoo Search algorithm starts by randomly generating an initial population of candidate solutions. Each solution is represented as a vector of decision variables. The algorithm then iteratively improves the candidate solutions by mimicking the following three behaviours of cuckoo birds:

1. **Levy Flight**: Cuckoo birds are known to perform a type of random flight called a Levy flight, which is characterized by long jumps with occasional short steps. In the Cuckoo Search algorithm, Levy flights are used to generate new candidate solutions by perturbing the current solutions in a random direction.
2. **Egg Laying**: Cuckoo birds lay their eggs in the nests of other birds based on a certain probability. In the Cuckoo Search algorithm, each candidate solution has a certain probability of being replaced by a new solution generated by a Levy flight.
3. **Host Rejection**: If a host bird discovers a cuckoo egg in its nest, it may reject the egg or abandon the nest altogether. In the Cuckoo Search algorithm, some of the generated solutions may be rejected based on a certain probability.

The Cuckoo Search algorithm can be formulated as follows:

Let $f(x)$ be the objective function to be minimized, where $x \in R^n$ is the decision variable vector.

1. Initialize the population of N candidate solutions x_i.
2. For each iteration t:
    a. Generate a new solution u_i by performing a Levy flight with step size s:

$$u_i = x_i + s \cdot L_\beta(u) \odot \epsilon,$$

where ϵ is a vector of random numbers drawn from a standard normal distribution, $\odot$ denotes element-wise multiplication, and $L_\beta(u)$ is the Levy flight step size calculated as:

$$L_\beta(u) = \left(\frac{\Gamma(1+\beta)\sin(\pi\beta/2)}{\beta 2^{(\beta-1)/2}\Gamma\left((\beta+1)/2\right)} \right)^{1/\beta},$$

where Γ is the gamma function and β is a parameter that controls the step size distribution.

b. Generate a new candidate solution v_i by combining u_i and x_j, where $j \neq i$ is a random index:

$$v_i = u_i + \alpha \cdot (x_j - u_i),$$

where α is a parameter that controls the step size.

c. Replace x_i with v_i with probability p_i, where p_i is a parameter that controls the egg laying rate:

$$p_i = 0, \quad \text{if } rand(0,1) < p_a$$
$$1, \quad otherwise$$

where $rand(0,1)$ is a randomly generated value between 0 and 1, and p_a is the probability of a cuckoo's egg being discovered and rejected by a host bird, which is also a parameter of the algorithm.

d. Abandon v_i with probability q, where q is a parameter that controls the host rejection rate:

$$q = 1 - \left(1 - e^{-\lambda t}\right)^{\frac{1}{2}},$$

where λ is a parameter that controls the rate of discovery of cuckoo eggs, and t is the current iteration number.

e. Evaluate the fitness of the new candidate solutions v_i. If v_i is better than x_i, replace x_i with v_i.

3. Repeat step 2 until a stopping criterion is met (e.g., a maximum number of iterations or a desired level of solution quality).

The Cuckoo Search algorithm has been shown to be effective in solving a wide range of optimization problems, including function optimization, feature selection, and parameter tuning for machine learning algorithms. The algorithm's ability to balance exploration and exploitation makes it a popular choice for solving complex optimization problems.

2.2.9 Hyperledger Blockchain

Blockchain is a shared ledger implementation method. It provides decentralisation [26], appropriateness, security, accessibility, and all of the above. Data stored in the Blockchain can now be replicated across a number of machines, eliminating the single point of failure associated with a centralized server. While the data is accessible when needed, very few computers really malfunction. Integrity with data upkeep, preserving it from unwanted changes. The capacity to track all stored data in blockchains is possible.

Finally, privacy allows the members to remain anonymous. Technically speaking, the blockchain is made up of a collection of well-ordered and trustworthy blocks of chain, each of which includes a header and stores data. The header is made up of several elements, including the previous block, the identification, and the signature. The identification represents an

internationally distinctive value with a mathematical function that encloses each block of data. Chaining blocks is managed by the preceding block. A logical chain of connections would be created since each novel block in the chain would include the identifier value of the block before it. One of the open-source blockchains provided by the Hyperledger management is called Hyperledger Fabric. It seeks to create a decentralised setting. It involves committed peer, client, certificate authority, order, and endorser peer. Additionally, the components communicate through channels that have been set up to enable transactions in a private and hidden manner, dividing various application domains. There are two methods that the fabric certificate authority is in charge of. First, it ensures that different components (users or smart contracts) can use the specified system. Then, it verifies the component and grants permission to use it for a certain function (such as carrying out a transaction) or access another part as a result of the authorization. The chain sent by the system-generated channel must be continued by the committing peer [27]. As a result, they maintain different blockchains for every channel that an individual has formed. Scalability and anonymity are provided by this 'individual chain per channel' method [28].

When compared to privacy, a component that lacks easy access to a network cannot access a chain from a committed peer connected to the channel. Scalability states that individual for each channel permits the exchange of various transactions and data stored with in various committed nodes, increasing the demanded amount where a node gets fulfilled and to raise quantity of data, thus enhancing the system's sustainability. Peers who are approving are in charge of two processes. The first step is to collect the transaction from the client. After that, it is examined using a smart contract system because the transaction contains a number of linked rules that must be adhered to. Two processes are carried out by gathering peers, such as obtaining consumer transactions and organising transactions for assessing the blockchain's dependability. As a result, every ordering peer on a given chain needs to attest that the transaction was added to the committing peers. It is claimed that even though a person may frequently visit identical medical facilities, this blockchain only records one visit. The EHR pertinent to the individual is stored on the blockchain for each medical facility (also known as the local blockchain). It is assumed that the medical facility maintains the minimal structure needed to operate the Hyperledger network in order to put the blockchain.

A smart contract is a chain code application in Hyperledger fabric. It is typical practise to construct network-agreed business logic using chain codes. The state that results from the generation of a chain code belongs to that chain code alone and is unavailable to other chain codes. If you have the required authorization, you can call another chain code. It is useful to think about the following two sorts of chain code while discussing them: a chain of application-specific code for the entire system in general, chain code is in charge of processing system-related transactions, including lifecycle management and policy configuration. Nonetheless, users have access to the system's chain code API and are free to modify it as needed. Application states, such as digital assets or arbitrary data input, must be maintained on the ledger by the application chain code. A chain code begins with a package that contains metadata, which is used to ensure the consistency of the code and metadata, such as the name, version, and counterparty signatures. The software is installed automatically on the counterparties' local computers when the chain code package has been loaded on the counterparties' network nodes. A registration function is carried out within the smart contracts (chain code) for the private health authority designated by the fabric network administrator to

administer and manage the fabric network in order to register the members. For enrolled parties, the healthcare authorities establish a private, permissioned network to which only they have accessibility. A virtual private network connection will be used by everyone to access the registration system with added security (VPN). The patient will just provide enrolment data at the time of registration, including name, social security number, address, and contact details. Also, the principal physician, hospital, laboratory, pharmacy, researcher, and insurance will all register with the body that regulates the healthcare industry. The public health authority checks the record when they have finished the enrolment process and issues a chain code address. The registration procedure has been finished by all parties, and all transactions on the network have been finished.

The Hyperledger blockchain procedure in the healthcare industry is shown in Figure 2.7. The modular nature of the Hyperledger Fabric model offers security, resilience, flexibility, and scalability. It provides plug-in implementation of a variety of components and adapts to the economic ecosystem's complexity and subtleties. The main components of the Figure 2.7's fundamental fabric technology concepts are as following.

Chain codes: It is a self-executing software that is currently developed in Go (identical to smart contracts).

Channels: It is a confidential "subnet" of communication between specific network users (or hospitals), with the intention of facilitating private transactions.

Ordering service: It guarantees the regularity and planning of transactions

Endorsement policy: It consists of the set of guidelines that a node can use to determine whether or not a transaction is accepted.

Application SDK: Based on the endorsement policy outlined in the chain codes, it approves transactions prior to commitment

Endorsing peers: To validate the blocks, it obtains them from the ordering service.

Committing peers: It also updates the state of the data in the State DB and the ledger.

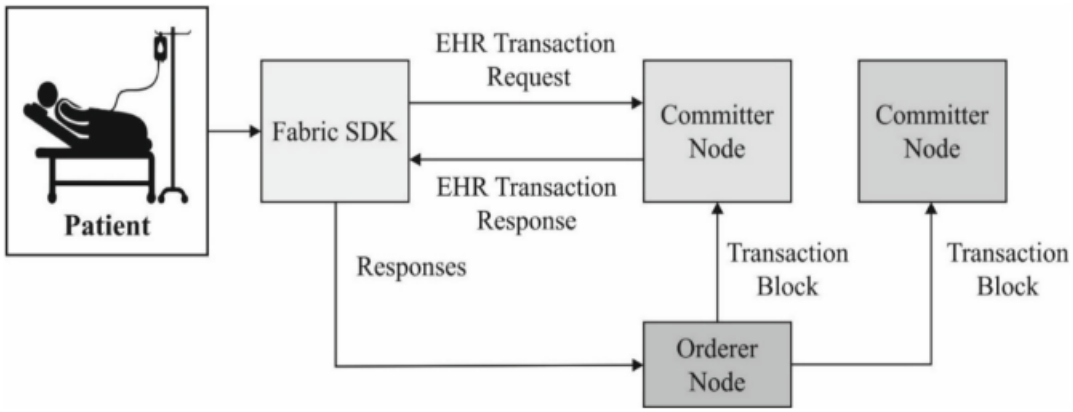

Figure 2.7: Hyperledger blockchain in healthcare