Appendix (Pseudocode):

for each of the hybrid algorithms (HA) used (e.g., biography-based optimization (BBO), backtracking search algorithm (BSA), teaching-learning-based algorithm (TLBO), cuckoo optimization algorithm (COA), multi-verse optimization (MVO),) we need to process like below:

|  |
| --- |
| 1.BBO-MLP |
| Initialize BBO parameters:  Population size (N), migration rate (λ), mutation rate (μ)  Initialize ANN parameters as unknown hyperparameter:  Number of layers (L), neurons per layer (n), learning rate (α)  n1 and n2 for the number of neurons  Create initial population of ANNs with random weights W  For each generation:  Evaluate fitness f\_i of each ANN\_i in the population using AUC on training data  Sort population based on fitness    For each ANN\_i:  For each dimension j:  Perform migration:  If rand < λ:  Select another ANN\_k based on fitness  W\_ij = W\_kj  Perform mutation:  If rand < μ:  W\_ij = W\_ij + δ (where δ is a small random value)  Replace worst-performing solutions with new ones  Select the best ANN based on fitness  Train the best ANN using backpropagation on the training data:  W = W - α \* ∇E (where ∇E is the gradient of the error)  Evaluate the trained ANN on the testing data  Return the trained ANN and its performance metrics (AUC, accuracy) |

|  |
| --- |
| 2. BSA-MLP |
| Initialize BSA parameters:  Population size (N), maximum iterations (T)  Initialize ANN parameters as unknown hyperparameter:  Number of layers (L), neurons per layer (n), learning rate (α)  n1 and n2 for the number of neurons  Create initial population of ANNs with random weights W  For each generation t = 1 to T:  Create trial population W\_trial by perturbing the current population W:  W\_trial = W + F \* (W\_rand - W) (where W\_rand is a random solution, F is a scaling factor)    Evaluate fitness f\_trial of W\_trial using AUC on training data    For each ANN\_i:  If f\_trial\_i is better than f\_i:  W\_i = W\_trial\_i    Select the best ANN based on fitness  Train the best ANN using backpropagation on the training data:  W = W - α \* ∇E (where ∇E is the gradient of the error)  Evaluate the trained ANN on the testing data  Return the trained ANN and its performance metrics (AUC, accuracy) |

|  |
| --- |
| 3. TLBO-MLP |
| Initialize TLBO parameters:  Population size (N), maximum iterations (T)  Initialize ANN parameters as unknown hyperparameter:  Number of layers (L), neurons per layer (n), learning rate (α)  n1 and n2 for the number of neurons  Create initial population of ANNs with random weights W  For each generation t = 1 to T:  Teacher Phase:  Identify the best solution W\_teacher  For each ANN\_i:  W\_i = W\_i + r \* (W\_teacher - T\_mean) (where r is a random number, T\_mean is the mean of the solutions)    Learner Phase:  For each ANN\_i:  Select another random ANN\_j:  If f\_i < f\_j:  W\_i = W\_i + r \* (W\_i - W\_j)  Else:  W\_i = W\_i + r \* (W\_j - W\_i)    Evaluate fitness f\_i of each ANN\_i using AUC on training data    Select the best ANN based on fitness  Train the best ANN using backpropagation on the training data:  W = W - α \* ∇E (where ∇E is the gradient of the error)  Evaluate the trained ANN on the testing data  Return the trained ANN and its performance metrics (AUC, accuracy) |

|  |
| --- |
| 4. COA-MLP |
| Initialize COA parameters:  Population size (N), discovery rate (pa)  Initialize ANN parameters as unknown hyperparameter:  Number of layers (L), neurons per layer (n), learning rate (α)  n1 and n2 for the number of neurons  Create initial population of ANNs with random weights W  For each generation:  Generate new solutions W\_new by random walk and Levy flights:  W\_new = W + α \* Levy(λ) (where α is a step size, Levy(λ) is a random walk)    Evaluate fitness f\_new of W\_new using AUC on training data    For each ANN\_i:  If f\_new\_i is better than f\_i:  W\_i = W\_new\_i    Replace a fraction pa of the worst solutions with new random solutions    Select the best ANN based on fitness  Train the best ANN using backpropagation on the training data:  W = W - α \* ∇E (where ∇E is the gradient of the error)  Evaluate the trained ANN on the testing data  Return the trained ANN and its performance metrics (AUC, accuracy) |

|  |
| --- |
| 5. MVO-MLP |
| Initialize MVO parameters:  Population size (N), maximum iterations (T), wormhole existence probability (WEP), white hole probability (WHP)  Initialize ANN parameters as unknown hyperparameter:  Number of layers (L), neurons per layer (n), learning rate (α)  n1 and n2 for the number of neurons  Create initial population of ANNs with random weights W  For each generation t = 1 to T:  Evaluate fitness f\_i of each ANN\_i using AUC on training data    For each ANN\_i:  For each dimension j:  Update position based on wormhole and white hole probabilities:  If rand < WHP:  Select another ANN\_k based on fitness  W\_ij = W\_kj  If rand < WEP:  W\_ij = W\_ij + TDR \* (rand - 0.5) \* (Xmax - Xmin) (where TDR is the traveling distance rate, Xmax and Xmin are boundaries)    Perform exploitation and exploration based on interaction of universes    Select the best ANN based on fitness  Train the best ANN using backpropagation on the training data:  W = W - α \* ∇E (where ∇E is the gradient of the error)  Evaluate the trained ANN on the testing data  Return the trained ANN and its performance metrics (AUC, accuracy) |

The pseudo-code outlines integrating an ANN with a Backtracking Search Algorithm to optimize its weights and biases. The specifics of the crossover, mutation, and evaluation functions would need to be tailored to the particular problem and the ANN architecture being used.

1. **Initialize ANN and** HA
2. **BSA Phases**
3. **Train ANN with optimized weights**

|  |
| --- |
| 1. Initialize ANN and HA |
| Define ANN structure (number of layers, neurons per layer)  Initialize weights and biases of ANN randomly  Define HA parameters:  population\_size  max\_iterations  crossover\_rate  mutation\_rate  Initialize population:  For i = 1 to population\_size:  Initialize individual i with random ANN weights and biases |
| 2. HA Phases |
| For iteration = 1 to max\_iterations:  # Phase 1: Initialization  If iteration == 1:  Save the initial population as historical\_population  End If    # Phase 2: Selection-I  For each individual in the population:  Select a random individual from historical\_population  End For    # Phase 3: Mutation  For each individual in the population:  Create a mutant individual by adding random variations to the individual  Ensure mutant individual is within bounds  End For    # Phase 4: Crossover  For each individual in the population:  Generate a crossover individual by combining the mutant and the current individual based on crossover\_rate  End For    # Phase 5: Selection-II  For each individual in the population:  If the crossover individual is better than the current individual:  Replace the current individual with the crossover individual  End If  End For    # Update historical\_population with the current population  Update historical\_population with current population  End For |
| 3. Train ANN with Optimized Weights |
| plaintext  Copy code  Select the best individual from the final population  Set ANN weights and biases to those of the best individual  Train ANN on the training dataset using optimized weights and biases  Evaluate ANN performance on the validation dataset |

**Detailed Explanation**

1. Initialize ANN and HA:

Define the structure of the ANN (e.g., number of layers, neurons in each layer).

Randomly initialize the weights and biases of the ANN.

Define the parameters for the BSA, including population size and the number of iterations.

Initialize a population of individuals, each representing possible weights and biases for the ANN.

1. HA Phases:

Initialization: Store the initial population.

Selection-I: Select random individuals from the historical population.

Mutation: Create mutant individuals by introducing variations.

Crossover: Combine mutant individuals with current individuals to generate crossover individuals.

Selection II: Replace current individuals with crossover individuals if they perform better.

Update the historical population for the next iteration.

1. Train ANN with Optimized Weights:

After HA optimization, select the best individual (set of weights and biases).

Use these weights and biases to initialize the ANN.

Train the ANN on the training dataset and evaluate its performance on a validation dataset.