Pseudocode Algorithms for Reproducibility

Algorithm 1: Feature Extraction from Video Data

ALGORITHM 1: Behavioral Biomarker Feature Extraction

INPUT: Video frames $V = \{v_1, v_2, ..., v_n\}$

OUTPUT: Feature matrix $F \in \mathbb{R}^{T \times d}$ where T=sequence_length, d=feature_dimensions

```
1. INITIALIZE MediaPipe Pose model
2. FOR each frame vᵢ in V:
3.    landmarks ← EXTRACT_POSE_LANDMARKS(vᵢ)
4.    IF landmarks is None:
5.       landmarks ← INTERPOLATE_MISSING_LANDMARKS()
6.    STORE landmarks[i] ← landmarks
7. END FOR

8. // Calculate movement parameters
9. FOR each landmark point j:
10.   movement[j] ← CALCULATE_EUCLIDEAN_DISTANCE(landmarks[i][j], landmarks[i-1][j])
11.   velocity[j] ← (movement[j] - movement[j-1]) / Δt
12.   acceleration[j] ← (velocity[j] - velocity[j-1]) / Δt
13.   jerk[j] ← (acceleration[j] - acceleration[j-1]) / Δt
14. END FOR

15. // Statistical features
```

```
FOR each movement sequence s:
    mean[s] ← MEAN(movement[s])
    std[s] ← STANDARD_DEVIATION(movement[s])
    skewness[s] ← SKEWNESS(movement[s])
    kurtosis[s] ← KURTOSIS(movement[s])
END FOR

// Frequency domain features
FOR each movement sequence s:
    X[s] ← FFT(movement[s])
    dominant_freq[s] ← ARGMAX(|X[s]|) × fs/N
    spectral_centroid[s] ← Σ(f[k] × |X[s][k]|) / Σ|X[s][k]|
    bandwidth[s] ← SQRT(Σ((f[k] - spectral_centroid[s])^2 × |X[s][k]|) / Σ|X[s][k]|)
END FOR

// Coordination indices
left_movement ← EXTRACT_LEFT_LIMB_MOVEMENTS()
right_movement ← EXTRACT_RIGHT_LIMB_MOVEMENTS()
correlation_coeff ← PEARSON_CORRELATION(left_movement, right_movement)
synchronization_score ← 1 - MIN(1, |σ_left - σ_right| / MAX(σ_left, σ_right))

// Repetitiveness measurement
direction_changes ← COUNT_DIRECTION_CHANGES(movement)
intervals ← CALCULATE_INTERVALS_BETWEEN_CHANGES()
repetitiveness_score ← direction_changes × (1 - MIN(σ_intervals/μ_intervals, 1))
```

38. // Combine all features

39. F ← CONCATENATE([movement, velocity, acceleration, jerk, statistical_features, frequency_features, coordination_features, repetitiveness_features])

40. F ← NORMALIZE_TO_SEQUENCE_LENGTH(F, T=100)

41. RETURN F

Algorithm 2: Hybrid BiLSTM+CNN+Attention Model Architecture

ALGORITHM 2: Hybrid Deep Learning Model Construction

INPUT: Feature sequences $X \in \mathbb{R}^{N \times T \times d}$, Labels $y \in \{0,1\}^N$

OUTPUT: Trained hybrid model M

1. // Model architecture definition

2. input_layer ← INPUT(shape=(T, d))

3. // CNN pathway for spatial feature extraction

4. conv1 ← CONV1D(filters=64, kernel_size=3, activation='relu')(input_layer)

5. conv2 ← CONV1D(filters=64, kernel_size=5, activation='relu')(input_layer)

6. conv3 ← CONV1D(filters=64, kernel_size=7, activation='relu')(input_layer)

7. conv_concat ← CONCATENATE([conv1, conv2, conv3])

8. conv_pool ← MAX_POOLING1D(pool_size=1)(conv_concat)

9. conv_dropout ← DROPOUT(rate=0.3)(conv_pool)

10. // BiLSTM pathway for temporal modeling

11. bilstm1 ← BIDIRECTIONAL_LSTM(units=64, return_sequences=True)(input_layer)

12. bilstm_dropout1 ← DROPOUT(rate=0.3)(bilstm1)

13. bilstm2 ← BIDIRECTIONAL_LSTM(units=32, return_sequences=True)(bilstm_dropout1)

14. bilstm_dropout2 ← DROPOUT(rate=0.3)(bilstm2)

15. // Multi-head attention mechanism

16. attention ← MULTI_HEAD_ATTENTION(num_heads=4, key_dim=16)(bilstm_dropout2, bilstm_dropout2)

17. attention_add ← ADD([attention, bilstm_dropout2])

18. attention_norm ← LAYER_NORMALIZATION()(attention_add)

19. // Feature fusion

20. concat_features ← CONCATENATE([conv_dropout, attention_norm])

21. flatten ← TIME_DISTRIBUTED(FLATTEN())(concat_features)

22. // Global feature extraction

23. global_max ← GLOBAL_MAX_POOLING1D()(flatten)

24. global_avg ← GLOBAL_AVERAGE_POOLING1D()(flatten)

25. global_concat ← CONCATENATE([global_max, global_avg])

26. // Classification layers

27. dense1 ← DENSE(units=64, activation='relu')(global_concat)

28. dropout1 ← DROPOUT(rate=0.4)(dense1)

29. dense2 ← DENSE(units=32, activation='relu')(dropout1)

30. dropout2 ← DROPOUT(rate=0.3)(dense2)

31. output ← DENSE(units=1, activation='sigmoid')(dropout2)

32. // Model compilation

33. model ← MODEL(inputs=input_layer, outputs=output)

34. COMPILE(model, optimizer=Adam(lr=0.001), loss='binary_crossentropy', metrics=['accuracy', 'precision', 'recall', 'auc'])

35. RETURN model

Algorithm 3: Ensemble Learning Strategy

ALGORITHM 3: Weighted Ensemble Model Training and Prediction

INPUT: Training data (X_train, y_train), Test data (X_test, y_test)

OUTPUT: Ensemble predictions P_ensemble

1. // Individual model creation

2. model_bilstm_cnn_attention ← CREATE_HYBRID_MODEL()

3. model_gru ← CREATE_GRU_MODEL()

4. model_cnn ← CREATE_CNN_MODEL()

5. // Training configuration

6. callbacks ← [EARLY_STOPPING(patience=15), REDUCE_LR_ON_PLATEAU(patience=5, factor=0.5), MODEL_CHECKPOINT(save_best_only=True)]

7. // Individual model training

8. FOR each model in [model_bilstm_cnn_attention, model_gru, model_cnn]:

9. TRAIN(model, X_train, y_train, epochs=50, batch_size=32, validation_split=0.2, callbacks=callbacks)

10. END FOR

11. // Model saving (UPDATED)

12. SAVE_MODEL(model_bilstm_cnn_attention, 'bilstm_cnn_attention_model.h5')

13. SAVE_MODEL(model_bilstm_cnn_attention, 'bilstm_cnn_attention_model.keras')

14. SAVE_MODEL(model_gru, 'gru_model.h5')

15. SAVE_MODEL(model_gru, 'gru_model.keras')

16. SAVE_MODEL(model_cnn, 'cnn_model.h5')

17. SAVE_MODEL(model_cnn, 'cnn_model.keras')

18. // Validation performance evaluation

19. P_bilstm_cnn_attention ← PREDICT(model_bilstm_cnn_attention, X_validation)

20. P_gru ← PREDICT(model_gru, X_validation)

21. P_cnn ← PREDICT(model_cnn, X_validation)

22. // Weight optimization based on validation performance

23. weights ← OPTIMIZE_WEIGHTS([P_bilstm_cnn_attention, P_gru, P_cnn], y_validation)

24. // Empirically determined optimal weights: w_1=0.5, w_2=0.3, w_3=0.2

25. // Ensemble prediction

26. P_test_bilstm_cnn_attention ← PREDICT(model_bilstm_cnn_attention, X_test)

27. P_test_gru ← PREDICT(model_gru, X_test)

28. P_test_cnn ← PREDICT(model_cnn, X_test)

29. P_ensemble ← w_1 × P_test_bilstm_cnn_attention + w_2 × P_test_gru + w_3 × P_test_cnn

30. P_ensemble_binary ← (P_ensemble > 0.5) ? 1 : 0

```
31. // Save ensemble results (UPDATED)
32. SAVE_ARRAY(P_ensemble, 'ensemble_pred.npy')
33. SAVE_ARRAY(P_ensemble_binary, 'ensemble_pred_binary.npy')
34. SAVE_ARRAY(X_test, 'X_test_advanced.npy')
35. SAVE_ARRAY(y_test, 'y_test_advanced.npy')

36. RETURN P_ensemble, P_ensemble_binary
```

Algorithm 4: Cross-Validation and Statistical Analysis

ALGORITHM 4: 5-Fold Cross-Validation with Statistical Testing

INPUT: Dataset D = (X, y), Models M = $\{M_1, M_2, \ldots, M_k\}$

OUTPUT: Performance metrics with statistical significance

```
1. // 5-fold stratified cross-validation
2. folds ← STRATIFIED_K_FOLD(D, k=5, random_state=42)
3. INITIALIZE performance_matrix[k_models][k_folds][n_metrics]

4. FOR fold_i in range(5):
5.    (X_train_fold, y_train_fold), (X_val_fold, y_val_fold) ← folds[fold_i]
6.    FOR model_j in M:
7.       model_j ← TRAIN(model_j, X_train_fold, y_train_fold)
8.       predictions ← PREDICT(model_j, X_val_fold)
9.       // Calculate performance metrics
10.       accuracy ← ACCURACY_SCORE(y_val_fold, predictions)
11.       precision ← PRECISION_SCORE(y_val_fold, predictions)
```

```
      recall ← RECALL_SCORE(y_val_fold, predictions)
      f1_score ← F1_SCORE(y_val_fold, predictions)
      roc_auc ← ROC_AUC_SCORE(y_val_fold, predictions)
      performance_matrix[model_j][fold_i] ← [accuracy, precision, recall, f1_score, roc_auc]
   END FOR
END FOR

// Statistical significance testing
FOR each pair (model_i, model_j) in M:
   performance_i ← MEAN(performance_matrix[model_i], axis=folds)
   performance_j ← MEAN(performance_matrix[model_j], axis=folds)
   // Paired t-test
   t_statistic, p_value ← PAIRED_T_TEST(performance_i, performance_j)
   // Effect size (Cohen's d)
   pooled_std ← SQRT((STD(performance_i)^2 + STD(performance_j)^2) / 2)
   cohens_d ← ABS(MEAN(performance_i) - MEAN(performance_j)) / pooled_std
   // Significance level
   IF p_value < 0.001: significance ← "*"
   ELIF p_value < 0.01: significance ← ""
   ELIF p_value < 0.05: significance ← "*"
   ELSE: significance ← "ns"
   STORE statistical_results[model_i][model_j] ← {p_value, cohens_d, significance}
END FOR

RETURN performance_matrix, statistical_results
```

Algorithm 5: Data Preprocessing and Augmentation

ALGORITHM 5: Data Preprocessing Pipeline

INPUT: Raw feature sequences F_raw, Labels y_raw

OUTPUT: Preprocessed training and test sets

```
1. // Handle missing values
2. FOR each sequence s in F_raw:
3.    missing_indices ← FIND_MISSING_VALUES(s)
4.    IF missing_indices is not empty:
5.       s[missing_indices] ← INTERPOLATE_LINEAR(s, missing_indices)
6.    END IF
7. END FOR

8. // Sequence length normalization
9. TARGET_LENGTH ← 100
10. FOR each sequence s in F_raw:
11.    IF LENGTH(s) > TARGET_LENGTH:
12.       s ← DOWNSAMPLE(s, TARGET_LENGTH)
13.    ELIF LENGTH(s) < TARGET_LENGTH:
14.       s ← PAD_SEQUENCE(s, TARGET_LENGTH, method='zero')
15.    END IF
16. END FOR

17. // Feature standardization
```

18. scaler ← STANDARD_SCALER()

19. F_scaled ← FIT_TRANSFORM(scaler, F_raw)

20. // Train-test split

21. X_train, X_test, y_train, y_test ← TRAIN_TEST_SPLIT(F_scaled, y_raw,

test_size=0.2,

stratify=y_raw,

random_state=42)

22. // Handle class imbalance with SMOTE

23. smote ← SMOTE(random_state=42, k_neighbors=5)

24. X_train_balanced, y_train_balanced ← FIT_RESAMPLE(smote, X_train, y_train)

25. // Data validation

26. ASSERT SHAPE(X_train_balanced)[1] == TARGET_LENGTH

27. ASSERT SHAPE(X_train_balanced)[2] == FEATURE_DIMENSIONS

28. ASSERT UNIQUE(y_train_balanced) == [0, 1]

29. // Save preprocessed data (UPDATED)

30. SAVE_ARRAY(X_train_balanced, 'X_train_advanced.npy')

31. SAVE_ARRAY(X_test, 'X_test_advanced.npy')

32. SAVE_ARRAY(y_train_balanced, 'y_train_advanced.npy')

33. SAVE_ARRAY(y_test, 'y_test_advanced.npy')

34. RETURN X_train_balanced, X_test, y_train_balanced, y_test, scaler

Implementation Notes:

Hyperparameters:

- Sequence Length: 100 frames
- Feature Dimensions: 21 (7 landmarks × 3 features each)
- Learning Rate: 0.001 with ReduceLROnPlateau
- Batch Size: 32
- Dropout Rates: 0.3-0.4 for regularization
- Early Stopping: Patience of 15 epochs
- Ensemble Weights: [0.5, 0.3, 0.2] for [BiLSTM+CNN+Attention, GRU, CNN]

Key Functions:

- MediaPipe Pose: For pose landmark extraction
- SMOTE: For handling class imbalance
- StratifiedKFold: For cross-validation
- Adam Optimizer: For model training
- Multi-Head Attention: With 4 heads and key dimension 16

Reproducibility Requirements:

- Set random seeds: random_state=42 for all stochastic operations
- Use fixed train-test split with stratification
- Apply consistent preprocessing pipeline
- Save model checkpoints and training histories (UPDATED)
- Save all intermediate data files (UPDATED)
- Document all hyperparameter choices and architectural decisions

File Structure (UPDATED):

```
models/
├── bilstm_cnn_attention_model.h5
├── bilstm_cnn_attention_model.keras
├── gru_model.h5
├── gru_model.keras
├── cnn_model.h5
└── cnn_model.keras
```

```
data/
├── X_train_advanced.npy
├── X_test_advanced.npy
├── y_train_advanced.npy
├── y_test_advanced.npy
├── ensemble_pred.npy
└── ensemble_pred_binary.npy
```