1) Need detail for training and validation, including data cleaning, preprocessing, and normalization steps.

-> The training and validation workflow starts with data cleaning to ensure the dataset is consistent and usable for modeling. Column names are standardized to remove formatting issues, and records with missing target values are excluded to avoid ambiguous labels. Key numerical variables such as age, height, weight, and pre-pain score are converted to numeric types, with invalid or non-convertible values removed to maintain data integrity. Categorical variables are cleaned by trimming whitespace and standardizing text to reduce inconsistencies in category representation. The target variable is then simplified into a smaller number of meaningful exercise categories for exercises and for OTC medications they were left as distinguishable classes only, which helps reduce class fragmentation and improves model learnability, and these categories are encoded into numeric labels for classification.

Next, feature preprocessing prepares the data for model training. Numeric and categorical features are handled separately using a column-wise transformation approach. Numeric features are passed through directly, or optionally normalized to ensure comparable feature scales, while categorical features are transformed using one-hot encoding to convert discrete categories into a machine-readable format. To address class imbalance in the target variable, oversampling is applied so that all exercise categories have similar representation, reducing bias toward majority classes. Model training is conducted using a Random Forest classifier within a unified pipeline that includes all preprocessing steps, ensuring that transformations are consistently applied during both training and prediction.

For validation, stratified k-fold cross-validation is used to evaluate model performance, preserving class distributions across folds and providing a consistent estimate of generalization accuracy. This approach helps prevent overfitting and variance caused by a single train–test split. Model performance is summarized using the mean accuracy and variability across folds. After validation, the final model is trained on the full preprocessed dataset to maximize learning from available data. The complete pipeline, along with the label encoder used for the target variable, is saved to enable reproducible deployment and consistent prediction on new, unseen data.

2) Need the model architecture in detail, including hyperparameters, layers, and fine-tuning parameters.

The model architecture is implemented as an end-to-end machine learning pipeline composed of a preprocessing stage followed by a Random Forest classification model. Input data consists of both numeric features (age, height, weight, and pre-pain score) and categorical features (race, ethnicity, pain timing, and pain location). During preprocessing, numeric features are passed through without scaling, while categorical features are transformed using one-hot encoding. The processed feature vectors are then fed into a Random Forest classifier, which functions as an ensemble of 200 decision trees trained on bootstrapped samples of the data. Each tree learns decision rules based on random subsets of features, and final predictions are produced through majority voting across all trees.

The Random Forest model is configured with key hyperparameters including 200 estimators to reduce variance, balanced class weights to compensate for class imbalance, and a fixed random state to ensure reproducibility. Tree depth and split behavior rely on default settings, allowing trees to grow until stopping criteria are met, which enables the model to capture complex nonlinear relationships in the data. Model fine-tuning primarily involves adjusting hyperparameters such as the number of trees, maximum tree depth, minimum samples per split or leaf, and feature selection strategy at each split, as well as experimenting with class balancing approaches. Validation is performed using stratified k-fold cross-validation to maintain class proportions across folds and provide a reliable estimate of generalization performance. After validation, the finalized pipeline—combining preprocessing and the trained Random Forest model—is fit on the full dataset and saved for consistent deployment and future predictions.

3) For training, need the loss functions, optimizers, and hardware used for training.

Training is performed using a classical machine-learning approach based on a Random Forest classifier rather than a neural network. As a result, there is no explicit loss function or gradient-based optimizer involved in training. Each decision tree in the forest is constructed using a greedy, recursive partitioning process that selects splits by minimizing an impurity measure, with Gini impurity used as the default split criterion. Model learning relies on bootstrap sampling of the training data and random feature selection at each split to introduce diversity among trees, and final predictions are produced through majority voting across the ensemble.

The model is trained on a Mac system using CPU-based computation, as scikit-learn's Random Forest implementation does not use GPU acceleration by default. Parallelism, when enabled, is achieved through multi-core CPU processing during cross-validation and model fitting. This setup is sufficient for efficient training given the model type and dataset size, while ensuring reproducibility and stable performance.

4) Show the metrics used to evaluate performance, such as discrimination, calibration, bias, sensitivity analysis in underrepresented subgroups.

Model performance is evaluated using a combination of overall accuracy and more detailed metrics to capture discrimination, calibration, robustness, and potential bias across subgroups. Discrimination is primarily assessed using classification accuracy obtained from stratified k-fold cross-validation, which measures the model's ability to correctly distinguish between different exercise categories and OTC medications across balanced class distributions; additional discrimination insight can be derived from class-wise precision, recall, and F1-scores computed from the confusion matrix. Calibration is evaluated by comparing predicted class probabilities with observed outcome frequencies, typically using probability calibration curves or metrics such as the Brier score, to assess whether predicted confidence levels align with actual outcomes. Bias and fairness are examined by stratifying performance metrics by sensitive or underrepresented subgroups (such as race or ethnicity) and comparing accuracy, recall, and error rates across these groups to identify systematic disparities in performance. Sensitivity analysis is conducted by evaluating model stability under different data conditions, including varying class balancing strategies, removing or perturbing specific features, and re-running cross-validation to observe changes in performance, with particular attention paid to underrepresented subgroups to ensure that predictions remain consistent and clinically reasonable across populations.

5) Detail the validation methods, such as k-fold cross-validation or a strictly separated test set.

Model validation is conducted using stratified k-fold cross-validation to obtain a unbiased estimate of generalization performance. In this approach, the dataset is divided into five folds while preserving the class distribution of the target variable in each fold. During validation, four folds are used for training and the remaining fold is used for evaluation, and this process is repeated until each fold has served as the validation set once. Performance metrics are then averaged across all folds to reduce variance associated with a single train–validation split. Stratification is especially important in this setting due to class imbalance, as it ensures that minority exercise categories are consistently represented in both training and validation folds.

In addition to cross-validation, the pipeline structure ensures that preprocessing steps—such as one-hot encoding and class balancing—are consistently applied during each training phase, preventing information leakage from the validation folds into training.