## **Architecture**

* **Data Sources**: The datasets used include APD3, DRAMP, ADAM, CAMP, and dbAMP, which provide sequences of antimicrobial peptides (AMPs). For non-AMPs, sequences were extracted from available datasets and curated to serve as negative examples in the classification task.
* **Preprocessing:** The preprocessing pipeline included multiple steps to ensure data quality. Any sequences containing unknown amino acids (e.g., U, O, B, Z, J) or whitespace were removed. CD-HIT was used to reduce sequence redundancy by clustering similar sequences with a similarity threshold of 90%. Finally, features were normalized using MinMax scaling for uniformity.
* **Feature Extraction:** Four types of features were extracted to represent peptide sequences numerically: Amino Acid Composition (AAC) quantified the frequency of amino acids; Autocorrelation described the distribution of amino acids and their properties; Composition, Transition, and Distribution (CTD) captured sequence properties; and Pseudo-Amino Acid Composition (PseAAC) encoded sequence-order effects and physicochemical properties.
* **Feature Combinations:** Feature subsets were tested in combinations ranging from single methods to pairs and higher-order combinations. This ensured comprehensive exploration of feature interactions and their impact on model performance.
* **Feature Selection:** Various feature selection methods were employed to reduce dimensionality and enhance model performance; Recursive Feature Elimination (RFE), Variance Threshold (VT), Random Forest, and Boruta. These methods were tested individually and in combinations to optimize feature subsets.

**Design Decisions**

### **Data Cleaning:** Sequences were filtered to ensure consistency for downstream analysis. Only sequences with lengths ranging between 10 and 100 amino acids were retained. This range was chosen based on the literature to exclude very short sequences that might lack meaningful structural information, as well as very long sequences that could introduce noise.

### **Feature Selection Thresholds:** The threshold for feature importance in the Random Forest model was set to the mean value of the feature importances. For the Variance Threshold method, features with variances in the top 5% were retained. To minimize computational complexity, the maximum number of iterations for Boruta was set to 10. Recursive Feature Elimination (RFE) selected features based on cross-validation, optimizing for accuracy during the process.

### **Models’ Parameters**: All parameters in this section were determined using grid search to optimize model performance. The Support Vector Machine (SVM) was configured with an RBF kernel, using C=100 and gamma='scale'. The Random Forest Classifier was set with a maximum depth of 40, max\_features='sqrt', and a minimum of one sample per leaf. For the K-Nearest Neighbors (KNN) algorithm, the Manhattan distance metric was employed, with the number of neighbors (k) set to 1.

### **Imbalanced Data:** The Synthetic Minority Over-sampling Technique (SMOTE) was implemented to address class imbalance by generating synthetic data points for the minority class. This approach ensured that the AMP and non-AMP classes were balanced, enhancing the reliability of the machine learning models during training.

## **Algorithms Used:**

**Feature selection Algorithms**

## **Recursive Feature Elimination (RFE):** An iterative feature selection algorithm that fits a model, ranks features based on importance, and removes the least important features

## **Variance Threshold (VT):** A feature selection method that eliminates features with low variance.

## **Boruta:** A wrapper-based feature selection algorithm that uses Random Forest to evaluate the importance of features and identifies all relevant ones.

## **Random Forest Feature Importance:** A feature selection approach that uses the feature importance scores generated by the Random Forest algorithm, which ranks features based on their contribution to model performance.

**ML Models**

## **Support Vector Machine (SVM):** A supervised learning algorithm used for classification tasks.

## **Random Forest Classifier:** An ensemble learning method that builds multiple decision trees and combines their outputs to improve classification accuracy.

## **K-Nearest Neighbors (KNN)**: A classification algorithm that assigns class labels based on the majority vote of the k nearest neighbors.

**Balancing algorithms**

## **SMOTE:** A data augmentation algorithm used to address class imbalance by generating synthetic data points for the minority class.

**Explainability algorithms**

## **LIME:** An explainability algorithm used to interpret the predictions of machine learning models. LIME explains individual predictions by approximating the model **locally** with an interpretable surrogate model.

* **SHAP:** An explainability algorithm used to interpret the predictions **globally** of machine learning models.

## **Dependencies**

* **Python Libraries:** pandas, numpy, matplotlib, scikit-learn, imblearn, lime, propy, shap
* **External Tools:** CD-HIT for sequence clustering
* **Environment:** Google Colab with GPU for efficient computation