Electronic Supplementary Material 4 (ESM4)

Supervised Machine Learning Baseline and Optimized Models Training and Evaluation Pseudocode and Implementation Details for "Machine Learning–Based Cryptographic Selection System for IoT Devices Based on Computational Resources"

Authors: Qi Ming Ng, Julia Juremi, Nor Azlina Abd Rahman and Vinesha Selvarajah
Version: v0.1 (30 October 2025)

Purpose Note:

This supplemental material includes pseudocode, implementation details, parameter configurations, and the corresponding Python source files for the training, testing, and comparison of the supervised machine learning baseline and optimized models addressed in the main manuscript. This document is for review purposes and will be structured in accordance with the criteria of the Springer journal ESM upon acceptance.

Contents Summary

File	Description
ESM4.docx	Main supplementary document containing pseudocode, implementation details, and supporting explanations
project_root.zip	Contains the Jupyter Notebook source files for the main data-analysis and figure-recreation modules, along with input and output folder structures. The core module (under ML Model/) is ML Models.ipynb , which performs training, testing, and evaluation of supervised machine-learning baseline and optimized models. The submodule (under ML Model/) is Figures_Recreation.ipynb reproduces the figures generated by the core module in accordance with Springer journal guidelines. The input folder (Data_Analysis/Summary/) is empty because the dataset is too large to include in this archive while output folders (ML Models/models_set1_.../) contain the respective trained supervised ML models. Environment files in ML Model/ include generate_env_info_auto.py (environment descriptor module) and its output appendix_environment_concise.txt, plus requirements.txt (pinned packages)
README.txt	Guide for scripts execution and interpretation
ESM4_Input_Summary_Datasets_for_Supervised_Learning_v0.1.zip	Input data generated during the data-analysis and weighted-scoring model development phases.
ESM4_Trained_ML_Models	Exported supervised machine learning models obtained after the training phase.

Reader Guide

This document contains organized pseudocode listings (Algorithms D.1–D.36), parameter configurations, and accompanying explanatory notes related to the supervised machine learning baseline and the optimized model training, testing, and evaluation scripts discussed in Section 4.4 of the main text. The pseudocode prioritizes clarity and reproducibility over precise runtime equivalency. The accompanying Python scripts in the zip file offer the reference implementation utilized for training, testing, and assessing the supervised machine-learning models outlined in the main text. It is recommended to utilize the accompanying requirements.txt file or adhere to the execution environment outlined in the appendix for establishing the necessary environment before executing the attached Python scripts. The enclosed README.txt offers more information and comprehensive instructions for utilizing the Python scripts.

Table of Contents

Section 0 – Algorithm Relationships and Workflow Overview

This section clarifies the relationship between Algorithms D.1 and D.36 utilized in the supervised machine-learning workflow for baseline and optimized model training, evaluation, and inference preparation. The section is segmented into three subsections: the first presents a narrative dependency mapping, the second features a workflow diagram illustrating the complete pipeline, and the final section contains a dependency mapping table for rapid reference.

Section 0.1 – Algorithm Dependency Mapping

This section shows how the machine-learning workflow described in this ESM proceeds through the following algorithmic stages:

- **Algorithm D.1** initializes all required machine-learning libraries, dependencies, project utilities, and optional caching mechanisms.
- **Algorithms D.2–D.6** provide helper functions for bias–variance analysis, ROC-AUC visualizations, confusion-matrix plotting, and class-distribution inspection, all requiring initialized dependencies from D.1.
- **Algorithm D.7** loads the four large dataset segments; it relies on the environment established by D.1.
- **Algorithm D.8** injects priority-weighting attributes (Duration, RAM, ROM, Battery) into each dataset loaded from D.7.
- **Algorithm D.9** merges filtered subsets from D.8 into a consolidated modeling dataset, removing nonessential or simulation-dependent fields.
- **Algorithm D.10** performs exploratory pruning, identifying and removing constant or uninformative columns from the dataset prepared by D.9.
- **Algorithm D.11** constructs the preprocessing pipeline—applying MinMax scaling for numerical features and one-hot encoding for categorical features—based on the column classifications determined by D.10.
- **Algorithm D.12** uses the feature and label definitions from D.10 to create stratified training and testing partitions for supervised learning.
- **Algorithm D.13** normalizes and encodes the training and testing partitions using the preprocessor created in D.11.
- **Algorithm D.14** defines the supervised machine-learning model zoo containing the five baseline estimators.
- **Algorithm D.15** constructs end-to-end ML pipelines by combining column-alignment transforms, the preprocessor from D.11, and the baseline estimators from D.14.
- **Algorithm D.16** trains the baseline models, generates predictions and scores, and evaluates the models using the metric utilities from D.26–D.31.
- **Algorithm D.17** performs Sequential Floating Selection (SFS) optimization on the baseline RandomForest model using preprocessed features generated by D.13.
- **Algorithm D.18** prepares normalized training/testing sets and applies SMOTE for dataset resampling.

- **Algorithm D.19** trains the resampled SMOTE-based model using outputs from D.18.
- **Algorithms D.20–D.24** perform extended optimizations, combining SBS, SFS, SMOTE, and hyperparameter tuning procedures based on the outputs of D.13, D.17, and D.18.
- **Algorithm D.25** initializes helper functions for metric evaluation and visualization.
- **Algorithms D.26–D.31** compute label distribution, confusion matrices, ROC curves, bias–variance statistics, and feature-name extraction, supporting evaluation tasks in D.16 through D.24.
- **Algorithm D.32** initializes utilities required for inference-time dataset alignment.
- **Algorithm D.33** ensures testing/inference DataFrames follow the same column order as the training data.
- **Algorithm D.34** ensures correct feature subset selection for inference.
- **Algorithm D.35** freezes the preprocessor from D.11 for inference-only usage, avoiding retraining.
- **Algorithm D.36** ensures DataFrame structure consistency and conversion during prediction using the final trained models.

Section 0.2 – Workflow Diagram for Algorithm D.1-D.36

This section presents the sequential supervised machine learning model training, optimization, and evaluation workflow executed by Algorithms D.1 through D.36. The complete pipeline is illustrated in Fig. 1.

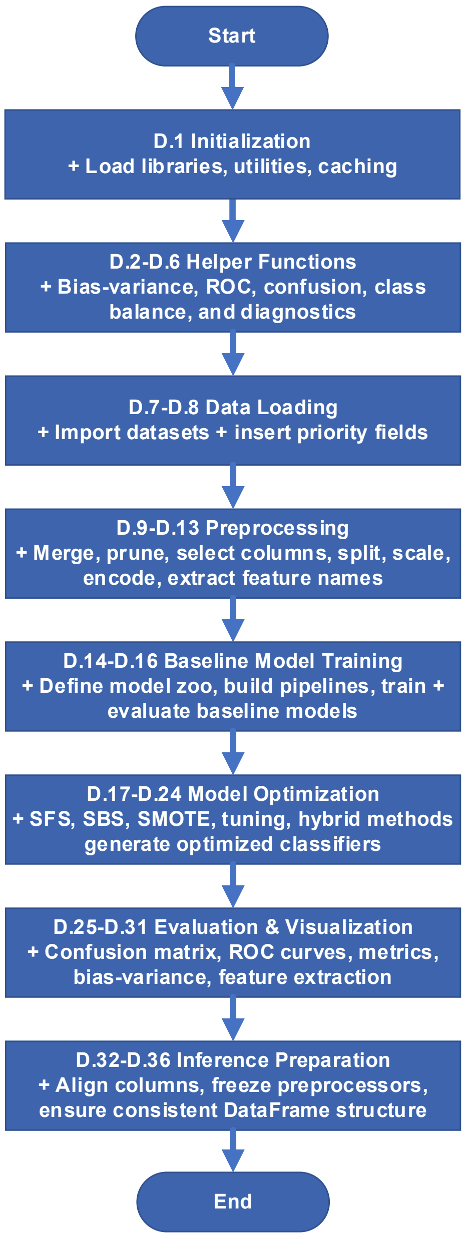
Start
D.1 Initialization
+ Load libraries, utilities, caching
D.2-D.6 Helper Functions
+ Bias-variance, ROC, confusion, class balance, and diagnostics
D.7-D.8 Data Loading
+ Import datasets + insert priority fields
D.9-D.13 Preprocessing
+ Merge, prune, select columns, split, scale, encode, extract feature names
D.14-D.16 Baseline Model Training
+ Define model zoo, build pipelines, train + evaluate baseline models
D.17-D.24 Model Optimization
+ SFS, SBS, SMOTE, tuning, hybrid methods generate optimized classifiers
D.25-D.31 Evaluation & Visualization
+ Confusion matrix, ROC curves, metrics, bias-variance, feature extraction
D.32-D.36 Inference Preparation
+ Align columns, freeze preprocessors, ensure consistent DataFrame structure
End

Fig. 1 Workflow diagram for the supervised machine-learning model training, optimization, and evaluation pipeline (Algorithms D.1–D.36)

Section 0.3 – Algorithm Dependency Table

This section summarizes the dependencies among the algorithms and outlines their primary roles within the supervised machine-learning model training, optimization, and evaluation workflow as shown in Table 1.

Table 1 Algorithm dependencies and functional roles

Algorithm	Depends On	Provides Output For	Functional Role
D.1	—	D.2–D.36	Initialize ML environment, libraries, utilities, caching
D.2	D.1	D.3, D.16–D.24	Consolidate bias–variance results
D.3	D.2	—	Plot bias–variance decomposition
D.4	D.30	—	Wrapper for ROC-AUC visualization
D.5	D.31	—	Wrapper for confusion-matrix visualization
D.6	D.1	—	Class-balance visualization
D.7	D.1	D.8	Load segmented datasets
D.8	D.7	D.9	Add priority feature columns
D.9	D.8	D.10	Merge datasets and drop unused attributes
D.10	D.9	D.11	Column pruning & EDA
D.11	D.10	D.12–D.16	Build preprocessing pipeline (scaling + encoding)
D.12	D.10	D.13–D.24	Train/test partition
D.13	D.11, D.12	D.15–D.24	Normalize & encode train/test data
D.14	D.1	D.15	Define model zoo (baseline classifiers)
D.15	D.11, D.14	D.16	Build ML pipelines
D.16	D.15	D.26–D.31	Train & evaluate baseline models
D.17	D.13	D.20, D.22, D.24	Perform SFS feature selection
D.18	D.13	D.19, D.20, D.23, D.24	Prepare & apply SMOTE
D.19	D.18	—	Train SMOTE-enhanced baseline
D.20	D.17, D.18	—	Train SFS + SMOTE pipelines
D.21	D.13	—	Hyperparameter tuning
D.22	D.13	—	SBS + hyperparameter tuning
D.23	D.18	—	SMOTE + hyperparameter tuning
D.24	D.17, D.18	—	SBS + SMOTE + hyperparameter tuning
D.25	—	D.26–D.31	Initialize metric/visualization helpers
D.26	D.25	D.16–D.24	Label distribution analysis
D.27	D.25	D.16–D.24	Metrics (F1, MCC, confusion matrix)
D.28	D.25	D.2	Bias–variance decomposition

D.29	D.11	D.13, D.17–D.24	Extract feature names
D.30	D.25	D.4	ROC plotting
D.31	D.25	D.5	Confusion-matrix plotting
D.32	—	D.33–D.36	Load ML-utility transformers
D.33	D.32	D.15–D.24	Align columns for inference
D.34	D.32	D.17–D.24	Select columns for inference
D.35	D.32	D.17–D.24	Freeze preprocessor for inference
D.36	D.32	D.17–D.24	Ensure DataFrame structure for inference

Section 1 – System Initialization

This section specifies the necessary libraries and dependencies for establishing the supervised machine learning model training environment, enabling the trained model to automate the classification task between a cryptographic technique and IoT device specifications. Algorithm D.1 indicates that the libraries and dependencies necessary for training the supervised machine learning model are located at the beginning of the module titled 'ML Models.ipynb'. All library dependencies and version specifications utilized in this work are documented in the supplementary ESM files—requirements.txt and appendix_environment_concise.txt—with an accompanying human-readable summary presented in Appendix A.

```
1: Inputs:
     —

2: Output:
     Preprocessing and modeling environment initialized;
     local utilities imported; optional caching configured

3: Procedure SetupAndImportDependencies()
4:     Add parent project directory to sys.path to enable importing local modules
5:     Import NumPy, pandas, matplotlib, time, Path, gc, and joblib
6:     Import core scikit-learn utilities for training, testing, cloning, pipelines, and preprocessing
7:     Import visualization helpers including display functions, class-balance charts, and sequential feature-selection plotting
8:     Import imbalanced-learn components, including SMOTE and imbalanced pipelines
9:     Import feature-selection utilities such as Sequential Floating Selection (SFS)
10:    Import local utilities: alignment, frozen preprocessors, DataFrame enforcement, and column selectors; reload mylib to apply updates
11:    Define custom transformer SafeSelectByName for selecting categorical dummy columns robustly across folds
12:    (Optional) enable caching via joblib.Memory for expensive preprocessing steps
```

```
13:     Initialize cache object with storage directory
14:     Attach cache to scikit-learn pipelines using the memory argument
15:     Decorate expensive helper functions with @memory.cache where applicable
16:     Return handles to all imported dependencies and utilities, along with
optional caching state
17: endProcedure
```

Algorithm D.1 SetupAndImportDependencies

Section 2 – Helper Function in ML Models.ipynb Module

This section presents the pseudocodes contained in the 'ML Models.ipynb' module, which comprise the auxiliary functions utilized for evaluating and visualizing trained models, as well as for comparing the performance of multiple training models within a single plot, thereby providing a clearer perspective, particularly in determining which model performed superiorly. Algorithm D.2 presents the pseudocode utilized to consolidate the bias and variance decomposition outcomes of various training models into a singular table. Algorithm D.3 presents the pseudocode utilized to view the consolidated bias and variance decomposition of several trained models in a singular plot. Algorithm D.4 presents the pseudocode utilized to generate the ROC-AUC graph for various trained models. Algorithm D.5 presents the pseudocode utilized to compute the distribution of training and testing label data and to generate the class distribution graph, thereby assessing if the dataset experienced class imbalance issues. Collectively, these auxiliary functions provide thorough post-training assessment and visualization, enhancing the clarity of model behavior interpretation and performance comparison.

```
1: Inputs:
      eval_models (list of (name, pipeline)),
      X_train, X_test, y_train, y_test,
      folds = 5

2: Output:
      DataFrame df_bv containing columns: (model, error, bias, variance)

3: Procedure DecomposeErrorBaseline(eval_models, splits, folds = 5)
4:       Initialize empty accumulator list
5:       For each (name, eval_pipeline) in eval_models:
6:             Compute (loss, bias, variance) using BiasVarMetrics1 (Algorithm D.28)
7:             Append {model = name, error = loss, bias = bias, variance = variance} to accumulator
8:       Concatenate all accumulated rows into DataFrame df_bv
9:       Return df_bv
10: endProcedure
```

Algorithm D.2 DecomposeErrorBaseline

```
1: Inputs:
      df_bv

2: Output:
      Composite figure containing:
```

```
      — Stacked bar chart of bias$^2$ and variance
      — Line plot overlay for error

3: **Procedure** *PlotBiasVariance*(df_bv)
4:      Plot stacked bars showing bias$^2$ and variance per model
5:      Overlay a line plot representing model error
6:      Format axes, rotate tick labels, add gridlines and legend
7:      Display final figure and return figure handle
8: **endProcedure**
```

Algorithm D.3 PlotBiasVariance

```
1: **Inputs:**
      results (list of {name, y_score, classes}),
      y_test,
      y_train,
      title

2: **Output:**
      Micro-averaged ROC plot

3: **Procedure** *PlotMicroRocAll*(results, y_test, y_train, title)
4:      Invoke *PlotModelsMicroRoc* (Algorithm D.30) with the provided results and title
5:      Return ROC figure handle
6: **endProcedure**
```

Algorithm D.4 PlotMicroRocAll

```
1: **Inputs:**
      results (list of {name, y_pred, classes}),
      y_test

2: **Output:**
      One confusion-matrix plot per model

3: **Procedure** *PlotConfusionMatricesAll*(results, y_test)
4:      For each record in results:
5:           Generate confusion matrix using *PlotConfusionAfter* (Algorithm D.31) with model name = rec["name"]
6:      Return all generated figure handles
7: **endProcedure**
```

Algorithm D.5 PlotConfusionMatricesAll

```
1: Inputs:
      y_train,
      y_test,
      width = 8 (optional),
      figsize = (15, 6) (optional)

2: Output:
      Class-balance bar plot with wrapped x-tick labels

3: Procedure PlotClassBalance(y_train, y_test, width = 8, figsize = (15, 6))
4:       Compute all_labels ← unique values from combined y_train and y_test
5:       Instantiate Yellowbrick ClassBalance visualizer with all_labels
6:       Fit visualizer with (y_train, y_test)
7:       Set figure size to provided figsize
8:       Render plot without closing figure
9:       Retrieve tick labels and wrap text according to width
10:        Apply updated wrapped labels; adjust layout
11:        Display figure and return handle
12: endProcedure
```

Algorithm D.6 PlotClassBalance

Section 3 – Data Collection

This section presents the pseudocodes provided in the 'ML Models.ipynb' module, which is utilized to import the processed dataset and subsequently prepare it to include all requisite features for supervised machine learning model training. Algorithm D.7 presents the pseudocode utilized for importing the dataset produced in the preceding section, comprising 24 CSV files as seen in ESM 3, into four DataFrames based on the initial four distinct priority rankings, with each dataset containing 23,887,872 rows and 74 columns. The rationale for not consolidating the data into a singular DataFrame, a prerequisite for exploratory data analysis, is that merging all at once requires substantial memory from the host machine, which previously encountered failure. Consequently, this workaround was implemented initially for dataset importation. To further mitigate memory usage, once the four DataFrames were established, the 24 imported CSV files stored in memory were eliminated. Although priority rating is a crucial factor in determining the optimal cryptographic algorithm based on user preferences for IoT device performance, it was absent from the existing dataset. Consequently, prior to merging the four DataFrames into a single DataFrame, Algorithm D.8 was employed to incorporate four integer priority columns representing Duration, RAM, ROM, and Battery. This addition resulted in two additional columns, bringing the total to 78 columns across the four DataFrames, while maintaining the same number of rows (23,887,872).

```
1: Inputs:
      root (path to dataset directory),
      datasize list

2: Output:
      Four merged DataFrames: df1, df2, df3, df4

3: Procedure LoadAndMergeDatasets(root, datasize)
4:     If datasize not provided:
5:         Set datasize ← [0.0078125, 2, 3, 900, 1024, 5120]
6:     Construct lists of file paths for each priority level (priority 1–4)
7:     For each priority list:
8:         Read all CSV files using pd.read_csv
9:         Concatenate into a single DataFrame
10:        Reset index for the merged DataFrame
11:    Print shape and preview of each DataFrame df1..df4
12:    Delete path lists to free memory and run garbage collection
13:    Return (df1, df2, df3, df4)
14: endProcedure
```

Algorithm D.7 LoadAndMergeDatasets

```
1: Inputs:
      df1, df2, df3, df4

2: Output:
      Updated DataFrames with added priority columns: Duration, RAM, ROM,
Battery

3: Procedure AddPriorityFeatures(df1, df2, df3, df4)
4:     For each DataFrame dfi in {df1, df2, df3, df4}:
5:         Insert integer priority columns (Duration, RAM, ROM, Battery)
             according to the defined priority scenario mapping
6:     Verify added columns and print updated shapes
7:     Return updated DataFrames (df1, df2, df3, df4)
8: endProcedure
```

Algorithm D.8 AddPriorityFeatures

Section 4 – Data Preprocessing (Feature Extraction & EDA)

This section presents the pseudocodes contained in the 'ML Models.ipynb' module, utilized for dataset analysis, feature extraction, exploratory data analysis, and the elimination of characteristics deemed less important or non-contributory to the learning process. Algorithm D.9 was employed to consolidate the four DataFrames into a singular DataFrame, following the elimination of less significant entries from the datasets due to the host machine's memory limitations. Consequently, each dataset comprised 373,248 rows and 78 columns. The columns were subsequently reordered to position priority features after the initial columns, while features pertinent to the data analysis outcomes—such as simulated performance results, feasibility assessments, MinMax scaling, inverted scaling, and both weighted and total weighted scores—were excluded, as these will not be utilized as model inputs, being unknown to the user unless an experiment or simulation has been conducted. This stage results in the consolidated dataset containing 373,248 rows and a reduced total of 12 columns. Algorithm D.10 was employed to identify and eliminate columns with either a unique value or uniform values across all entries, as these features did not contribute to the learning process. Subsequently, the dataset was divided into two subsets: the first containing all features and the second comprising solely the label, prior to data normalization.

```
1: Inputs:
      df1, df2, df3, df4

2: Output:
      df_set1 (Performance dataset)

3: Procedure FilterAndFormSets(df1, df2, df3, df4)
4:       For each DataFrame dfi in {df1, df2, df3, df4}:
5:             Filter rows where "Best Performance Algorithm" == 1 → dfi_set1
6:       Concatenate all dfi_set1 into a single DataFrame df_set1
7:       Reset index of df_set1
8:       Reorder columns to place priority features immediately after "No."
9:       Drop helper, tail, feasibility, and temporary calculation fields
10:      Drop non-modeling fields (simulation-dependent outputs)
11:      Return df_set1
12: endProcedure
```

Algorithm D.9 FilterAndFormSets

```
1: Inputs:
      df_set1

2: Output:
      X_set1, y_set1, num_cols_, cat_cols_

3: Procedure BasicEDAAndColumnPruning(df_set1)
4:       Check null values per column; summarize counts and percentages
5:       Count unique values in each column
6:       Drop columns with:
            (a) only one unique value, or
            (b) unique identifiers with no predictive value
7:       Set label column y_set1 ← "Algorithm Used (Performance)"
8:       Set X_set1 ← all remaining feature columns
```

```
          (Create independent copies to avoid chained-assignment issues)
9:      Identify categorical vs numeric columns based on dtypes
10:      Store lists as cat_cols_ and num_cols_
11:      For categorical columns, print value distributions (value_counts())
12:      Return (X_set1, y_set1, num_cols_, cat_cols_)
13: endProcedure
```

Algorithm D.10 BasicEDAAndColumnPruning

Section 5 – Data Preprocessing

This section presents the pseudocode from the 'ML Models.ipynb' module utilized for preprocessing the dataset, rendering it appropriate for supervised machine learning model training. This encompasses procedures such as numerical attribute scaling, categorical attribute encoding, feature and label list partitioning, and the application of the preprocessor for data normalization and encoding on the training dataset. Algorithm D.11 was employed to construct the preprocessor for MinMax scaling and one-hot encoding for numerical and categorical features, respectively, thereby standardizing the procedure for future application with new data in classification using the trained model. The dataset, including 373,248 items, was partitioned into 80% training data and 20% testing data, employing stratification with a fixed random state of 50, utilizing Algorithm D.12. Subsequent to partitioning, the training dataset was normalized and encoded utilizing the pre-existing preprocessor, as illustrated in Algorithm D.13.

```
1: Inputs:
      num_cols_, cat_cols_, X_set1

2: Output:
      preproc_set1, expected_cols_set1

3: Procedure BuildPreprocessors(num_cols_, cat_cols_, X_set1)
4:      Define OneHotEncoder parameters: handle_unknown = "ignore",
          and correct sparse/sparse_output options
5:      Set expected_cols_set1 ← list of X_set1 column names
6:      Construct a ColumnTransformer containing:
7:            ("num", MinMaxScaler(), num_cols_)
8:            ("cat", OneHotEncoder(), cat_cols_)
9:      Set remainder = "drop"
10:      Return preproc_set1 and expected_cols_set1
11: endProcedure
```

Algorithm D.11 BuildPreprocessors

```
1: Inputs:
      X_set1, y_set1

2: Output:
      X_train1, X_test1, y_train1, y_test1

3: Procedure StratifiedTrainTestSplit(X_set1, y_set1)
4:      Split X_set1 and y_set1 using a stratified split:
          test_size = 0.20, random_state = 50, stratify = y_set1
5:      Reset indices for X_train1, X_test1, y_train1, y_test1
6:      Preview X_test1 for verification
7:      Return all four splits
8: endProcedure
```

Algorithm D.12 StratifiedTrainTestSplit

```
1: Inputs:
      preproc_set1, (X_train1, X_test1, y_train1, y_test1)

2: Output:
      feature_names_set1, normalized previews

3: Procedure NormalizeAndNameFeatures(preproc_set1, splits)
4:      Clone preproc_set1 to avoid modifying the original
5:      Fit the cloned preprocessor on X_train1 only
6:      Transform X_train1 and X_test1 into numeric arrays
7:      Extract feature names using GetFeatureNames (Algorithm D.29)
8:      Wrap transformed arrays into DataFrames using extracted names
9:      Preview the top rows of transformed training and testing sets
10:     Return feature_names_set1 and normalized previews
11: endProcedure
```

Algorithm D.13 NormalizeAndNameFeatures

Section 6 – Algorithm Selection

This section presents the pseudocode referred to as Algorithm D.14, included in the 'ML Models.ipynb' module, utilized for configuring the supervised machine learning algorithms employed to train the baseline models. The minimum quantity of hyperparameters was established to preserve the integrity of the baseline models.

```
1: Inputs:
      —
```

```
2: Output:
      Ordered list of baseline model estimators

3: Procedure DefineModelZoo()
4:     Initialize empty model list
5:     Append LogisticRegression(max_iter = 1000, random_state = 42)
6:     Append SVC(probability = True, random_state = 42)
7:     Append KNeighborsClassifier()
8:     Append RandomForestClassifier(random_state = 42)
9:     Append MLPClassifier(max_iter = 500, random_state = 42)
10:    Return baseline model list
11: endProcedure
```

Algorithm D.14 DefineModelZoo

Section 7 – Performance Evaluation

This section presents the pseudocodes accessible in the 'ML Models.ipynb' module, utilized for constructing the supervised machine learning pipeline for baseline model training and conducting training and performance evaluation. Algorithm D.15 delineates the pseudocode for establishing the baseline model pipeline and facilitates the training of five distinct baseline models, while Algorithm D.16 offers the pseudocode for the training and assessment of these five classifiers.

```
1: Inputs:
      models, preproc_set1, expected_cols_set1

2: Output:
      models_set1 (list of pipelines for dataset 1)

3: Procedure BuildPipelines(models, preproc_set1, expected_cols_set1)
4:     Initialize empty list models_set1
5:     For each (name, estimator) in models:
6:         Construct pipeline consisting of:
7:             (1) AlignColumns(expected_cols_set1)
8:             (2) Cloned preproc_set1
9:             (3) The classifier estimator
10:        Append (name, pipeline) to models_set1
11:    Return models_set1
12: endProcedure
```

Algorithm D.15 BuildPipelines

```
1: Inputs:
      dataset_id, models_dataset, (X_train, X_test, y_train, y_test)
```

```
2: Output:
      results_dataset (list of evaluation records)

3: Procedure BaselineTrainEvalSave(dataset_id, models_dataset, splits)
4:      Create output directory: "models_{dataset_id}baseline"
5:      Initialize empty list results_dataset
6:      For each (name, pipeline) in models_dataset:
7:           Start timing the training process
8:           Fit pipeline on (X_train, y_train)
9:           Predict y_pred ← pipeline.predict(X_test)
10:          Compute y_score using predict_proba or decision_function
11:          Extract classifier classes_ attribute
12:          Stop timer; record runtime
13:          Save fitted pipeline to "models_{dataset_id}baseline" via joblib
14:          Evaluate predictions using ShowMetrics (Algorithm D.27)
15:          Append record {name, y_pred, y_score, classes, runtime} to
results_dataset
16:     Return results_dataset
17: endProcedure
```

Algorithm D.16 BaselineTrainEvalSave

Section 8 – Model Validation

This section delineates the pseudocodes contained within the 'ML Models.ipynb' module, utilized for optimizing the optimal baseline model—the RandomForest classifier—through three distinct optimization strategies: Sequential Backward Selection (SBS), Synthetic Minority Oversampling Technique (SMOTE), and hyperparameter tuning. The process involves using various optimization algorithms, either alone or in conjunction, followed by performance evaluation to identify the optimal model. Algorithm D.17 delineates the implementation of SBS, encompassing model training and testing, and concludes with the evaluation of model performance. Algorithms D.18 and D.19 executed analogous procedures, concentrating on the SMOTE approach as the optimization strategy. The remaining algorithms, specifically Algorithms D.20 to D.24, adhered to identical procedures while employing distinct optimization strategies: a combination of SBS and SMOTE, hyperparameter tuning exclusively, a combination of SBS and hyperparameter tuning, a combination of SMOTE and hyperparameter tuning, and ultimately, a combination of SBS, SMOTE, and hyperparameter tuning.

```
1: Inputs:
      dataset_id; (X_train, X_test, y_train, y_test); preproc; expected_cols

2: Output:
      sfs_results_dataset, sfs_eval_models_dataset
```

```
3: Procedure OptimizeWithSFS(dataset_id, splits, preproc, expected_cols)
4:     Select base classifier: RandomForestClassifier(random_state = 42)
5:     Configure SFS with:
         backward = True, floating = True,
         k_features = 'parsimonious', scoring = 'roc_auc_ovr',
         cv = 5, n_jobs = 10, verbose = 2
6:     Build SFS training pipeline:
         AlignColumns(expected_cols) → preproc → SFS → classifier
7:     Fit pipeline on (X_train, y_train); plot SFS performance trajectory
8:     Resolve selected feature names from SFS output indices
9:     Filter X_train and X_test to selected features; refit base classifier
10:    Predict on X_test; compute scores; evaluate via ShowMetrics
11:    Add record to sfs_results_dataset
12:    Build evaluation pipeline (without SFS):
         AlignColumns → preproc → SelectColumnsByName → classifier
13:    Build final inference pipeline:
         AlignColumns → FrozenPreprocessor → EnsureDataFrame →
SelectColumnsByName → classifier_refit
14:    Save inference pipeline to "models_sfs/{dataset_id}inference.joblib"
15:    Return sfs_results_dataset and sfs_eval_models_dataset
16: endProcedure
```

Algorithm D.17 OptimizeWithSFS

```
1: Inputs:
     preproc_set, (X_train, X_test, y_train, y_test)

2: Output:
     Normalized X_train/X_test; (X_train_smote, y_train_smote)

3: Procedure SmotePrepAndResample(preproc_set, splits)
4:     Clone and fit preproc_set on X_train
5:     Transform X_train and X_test to numeric arrays
6:     Extract feature names using GetFeatureNames (Algorithm D.29)
7:     Wrap transformed arrays into DataFrames using extracted names
8:     Apply SMOTE(random_state = 42, k_neighbors = 5) to training set
9:     (Optional) Visualize class balance
10:    Return normalized data and SMOTE-resampled training sets
11: endProcedure
```

Algorithm D.18 SmotePrepAndResample

```
1: Inputs:
     dataset_id; preproc_set; (X_train, X_test, y_train, y_test)
```

```
2: Output:
      smote_results_dataset; saved inference pipelines

3: Procedure TrainWithSmoteAndSave(dataset_id, preproc_set, splits)
4:      Define smote_models ← { ("RandomForest", RandomForestClassifier(random_state = 42)) }
5:      For each (name, base_clf) in smote_models:
6:          Construct ImbPipeline: preproc_set → SMOTE(k_neighbors = 5) → base_clf
7:          Fit on training data; predict on untouched test set
8:          Compute y_score and obtain classes_
9:          Evaluate with ShowMetrics (Algorithm D.27)
10:         Extract fitted preprocessor and model
11:         Build inference-only pipeline (preproc → clf), no SMOTE
12:         Save inference pipeline to results directory
13:         Append evaluation record to smote_results_dataset
14:     Return accumulated SMOTE results
15: endProcedure
```

Algorithm D.19 TrainWithSmoteAndSave

```
1: Inputs:
      dataset_id; splits; preproc; expected_cols; models

2: Output:
      results_sfs_smote; saved inference pipelines

3: Procedure TrainEvalSfsSmote(dataset_id, splits, preproc, expected_cols, models)
4:      Create output folder "models_sfs_smote"
5:      Initialize empty list results_sfs_smote
6:      For each (name, base_clf) in models:
7:          Build pipeline:
              AlignColumns → preproc → SMOTE → SFS(forward, floating) → base_clf
8:          Fit pipeline on X_train; extract selected index set
9:          Map indices to names using GetFeatureNames (Algorithm D.29)
10:         Predict and compute y_score; evaluate via ShowMetrics
11:         Build inference-only pipeline:
              AlignColumns → FrozenPreprocessor → EnsureDataFrame → SelectColumnsByName → cloned classifier
12:         Save inference pipeline
13:         Append record including selected features and runtime
14:     Return results_sfs_smote
15: endProcedure
```

Algorithm D.20 TrainEvalSfsSmote

```
1: Inputs:
      dataset_id; splits; preproc; expected_cols; models

2: Output:
      results_hyper

3: Procedure TrainEvalHyper(dataset_id, splits, preproc, expected_cols, models)
4:     Define hyperparameter search spaces for each model type
5:     Initialize results_hyper ← []
6:     For each (name, base_clf) in models:
7:         Build pipeline: AlignColumns → preproc → base_clf
8:         Initialize RandomizedSearchCV with model-specific search space
9:         Fit search on training data; obtain best_estimator_
10:        Predict and compute scores; extract classes_
11:        Evaluate via ShowMetrics
12:        Save tuned pipeline via joblib
13:        Append evaluation record including best parameters
14:    Return results_hyper
15: endProcedure
```

Algorithm D.21 TrainEvalHyper

```
1: Inputs:
      dataset_id; splits; preproc; expected_cols; models; sbs_k_options

2: Output:
      results_sbs_hyper

3: Procedure TrainEvalSbsHyper(dataset_id, splits, preproc, expected_cols,
models, sbs_k_options)
4:     Initialize results_sbs_hyper ← []
5:     For each (name, base_clf) in models:
6:         Build pipeline:
             AlignColumns → preproc →
             SFS(backward, floating = False, k_features='parsimonious') →
base_clf
7:         Define parameter grid including sbs_k_options + model hyperparameters
8:         Initialize RandomizedSearchCV with scoring='roc_auc_ovr'
9:         Fit search; extract best estimator
10:        Resolve selected feature names via GetFeatureNames
11:        Predict and evaluate using ShowMetrics
12:        Build and save inference-only pipeline
13:        Append evaluation record
14:    Return results_sbs_hyper
15: endProcedure
```

Algorithm D.22 train_eval_sbs_hyper

```
1: Inputs:
      dataset_id; splits; preproc; expected_cols; models

2: Output:
      results_smote_hyper

3: Procedure TrainEvalSmoteHyper(dataset_id, splits, preproc, expected_cols,
models)
4:      Initialize results_smote_hyper ← []
5:      For each (name, base_clf) in models:
6:          Build pipeline:
              AlignColumns → preproc → SMOTE → base_clf
7:          Define search space including:
              SMOTE k_neighbors options + model-specific hyperparameters
8:          Initialize RandomizedSearchCV with n_iter = 40
9:          Fit search; extract best_pipe
10:         Predict and compute y_score; evaluate using ShowMetrics
11:         Build inference-only pipeline (no SMOTE)
12:         Save tuned inference pipeline
13:         Append evaluation record
14:     Return results_smote_hyper
15: endProcedure
```

Algorithm D.23 TrainEvalSmoteHyper

```
1: Inputs:
      dataset_id; splits; preproc; expected_cols; models; sbs_k_options

2: Output:
      results_sbs_smote_hyper

3: Procedure TrainEvalSbsSmoteHyper(dataset_id, splits, preproc, expected_cols,
models, sbs_k_options)
4:      Initialize results_sbs_smote_hyper ← []
5:      For each (name, base_clf) in models:
6:          Build pipeline:
              AlignColumns → preproc → SMOTE →
              SFS(backward, floating=False) → base_clf
7:          Define combined search space including:
              sbs_k_features, smote__k_neighbors, and model hyperparameters
8:          Initialize RandomizedSearchCV with scoring='roc_auc_ovr'
9:          Fit search; extract best estimator; resolve selected features
10:         Predict; compute y_score; evaluate via ShowMetrics
11:         Build inference-only pipeline (no SMOTE/SFS) and save
12:         Append evaluation record
```

```
13:     Return results_sbs_smote_hyper
14: endProcedure
```

Algorithm D.24 TrainEvalSbsSmoteHyper

Section 9 – System Initialization and Helper Function in mylib.py Module

This section outlines the necessary libraries and dependencies for establishing an environment conducive to the development of helper functions, as well as defines the requisite helper functions for comprehending the dataset utilized in supervised machine learning, model performance evaluation, and visualization. Algorithm D.25 delineates the libraries and dependencies necessary for the helper functions required for the assessment and visualization of supervised machine learning model performance, which are positioned at the beginning of the module titled 'mylib.py'. All library dependencies and version specifications utilized in this work are documented in the supplementary ESM files—requirements.txt and appendix_environment_concise.txt—with an accompanying human-readable summary presented in Appendix A. Algorithm D.26 presents the pseudocode employed to compute the class distribution of the dataset utilized in supervised machine learning models. Algorithm D.27 is employed to compute the confusion matrix and additional performance metrics, including accuracy and MCC. Algorithm D.28 presents the pseudocode utilized for the computation of bias and variance decomposition. Algorithm D.29 presents the pseudocode utilized to generate the sequential feature name list, ensuring accurate alignment of feature order for both training and testing data within the preprocessor. Algorithm D.30 presents the pseudocode utilized for plotting the ROC-AUC curve and its corresponding values. Algorithm D.31 presents the pseudocode utilized for the visualization of the confusion matrix. Collectively, these helper functions enable methodical assessment and visualization of model efficacy during the research.

```
1: Inputs:
     —

2: Output:
     Dependencies imported; helper functions available for metrics, plots,
     and bias–variance analysis

3: Procedure SetupAndImportDependencies_Mylib()
4:     Import re globally and math internally for certain plotting utilities
5:     Import NumPy and pandas
6:     Import matplotlib.pyplot for ROC and confusion-matrix plotting
7:     Import sklearn metrics:
           classification_report, matthews_corrcoef, roc_curve, auc,
           confusion_matrix, ConfusionMatrixDisplay
```

```
8:      Import sklearn preprocessing utilities: LabelEncoder, label_binarize
9:      Import sklearn validation utilities: check_is_fitted and NotFittedError
10:     Import bias–variance decomposition from mlxtend.evaluate
11:     Define project helper functions:
          show_labels_dist, show_metrics, bias_var_metrics, bias_var_metrics1,
          get_feature_names, safe, plot_models_micro_roc, plot_confusion_after
12:     Return loaded modules and helper function handles
13: endProcedure
```

Algorithm D.25 SetupAndImportDependencies_Mylib

```
1: Inputs:
     X_train, X_test, y_train, y_test

2: Output:
     Label frequency and percentage table

3: Procedure ShowLabelsDist(X_train, X_test, y_train, y_test)
4:      Print the shapes of X_train, X_test, y_train, y_test
5:      Convert y_train and y_test to DataFrames
6:      Compute value-count tables for each
7:      Calculate percentage distribution for train and test labels
8:      Concatenate frequency and percentage tables side by side
9:      Print “Frequency and Distribution of labels” followed by the table
10:     Return the table object
11: endProcedure
```

Algorithm D.26 ShowLabelsDist

```
1: Inputs:
     y_true, y_pred, labels, digits = 3

2: Output:
     Confusion table; per-class and aggregate FPR/FNR;
     classification report; MCC values

3: Procedure ShowMetrics(y_true, y_pred, labels, digits = 3)
4:      Compute confusion matrix using fixed label order
5:      Format the matrix into a DataFrame for readability
6:      Extract TP, FP, FN, and TN per class
7:      Compute per-class false-positive rate and false-negative rate
8:      Compute macro and weighted averages
9:      Print classification report with precision, recall, F1-score
10:     Compute overall MCC and class-wise one-vs-rest MCC
11:     Print all metrics
```

```
12:     Return a success indicator
13: endProcedure
```

Algorithm D.27 ShowMetrics

```
1: Inputs:
    name, X_train, X_test, y_train, y_test, classifier,
    folds = 200, loss = "0-1_loss", seed = 44

2: Output:
    (avg_loss, avg_bias, avg_variance)

3: Procedure BiasVarMetrics1(name, X_train, X_test, y_train, y_test, clf,
    folds = 200, loss = "0-1_loss", seed = 44)
4:     Encode y_train and y_test using LabelEncoder
5:     Convert features to NumPy arrays if DataFrames
6:     Perform bias–variance decomposition using mlxtend
7:     Print average bias, variance, expected loss, and accuracy (1 − loss)
8:     Return (avg_loss, avg_bias, avg_variance)
9: endProcedure
```

Algorithm D.28 BiasVarMetrics1

```
1: Inputs:
    preprocessor, input_cols

2: Output:
    feature_names_out (list)

3: Procedure GetFeatureNames(preprocessor, input_cols)
4:     Initialize empty list out
5:     For each transformer (name, trans, cols) in preprocessor.transformers_:
6:         Skip remainder drops and empty column sets
7:         Resolve original column names from indices, slices, or masks
8:         Fetch fitted transformer (e.g., OneHotEncoder)
9:         If transformer supports get_feature_names_out:
10:            Append expanded names
11:        Else:
12:            Append original column names
13:    Extend output list with resolved names
14:    Return out
15: endProcedure
```

Algorithm D.29 GetFeatureNames

```
1: Inputs:
      results, y_test, y_train, title = None, decimals = 3

2: Output:
      Micro-average ROC curve (multiple models compared)

3: Procedure PlotModelsMicroROC(results, y_test, y_train, title = None, decimals = 3)
4:       Flatten y_test; compute global class set from y_train
5:       Binarize true labels for multi-class ROC calculation
6:       Align model score columns to global class order
7:       For each model:
8:             Compute micro-averaged ROC and AUC
9:             Plot ROC curve line
10:      Draw diagonal reference line
11:      Label axes, set title, and show legend
12:      Issue warning if no curves were plotted
13:      Return figure handle
14: endProcedure
```

Algorithm D.30 PlotModelsMicroROC

```
1: Inputs:
      model_name; results list; y_true; (styling options optional)

2: Output:
      Annotated confusion-matrix figure

3: Procedure PlotConfusionAfter(model_name, results, y_true, …)
4:       Locate evaluation record based on model_name; raise error if missing
5:       Determine full label set = union(true labels, model classes)
6:       Compute confusion matrix using full label ordering
7:       Compute row-wise, column-wise, and overall percentages
8:       Adjust figure size, label layout, and wrapping based on label length
9:       Draw heatmap; annotate counts and percentages
10:      Set plot title, axis labels, and enforce fixed aspect ratio
11:      Display figure
12: endProcedure
```

Algorithm D.31 PlotConfusionAfter

Section 10 – System Initialization and Helper Function in ml_utils.py Module

This section outlines the requisite libraries and dependencies for establishing an environment appropriate to developing helper functions, alongside specifying the functions essential for structuring and aligning the DataFrame and regulating the preprocessor's operations during model inference. Algorithm D.32 delineates the libraries and dependencies requisite for the helper functions necessary for DataFrame and preprocessor structuring, ensuring that the testing data aligns with the training data to prevent errors during the prediction process, positioned at the commencement of the module titled 'ml_utils.py'. All library dependencies and version specifications utilized in this work are documented in the supplementary ESM files—requirements.txt and appendix_environment_concise.txt—with an accompanying human-readable summary presented in Appendix A. Algorithm D.33 presents the pseudocode utilized to guarantee that the freshly generated DataFrame, representing the testing data, is organized in the identical column sequence as the training data. Algorithm D.34 presents the pseudocode utilized to guarantee the selection of accurate feature data for input in the trained model during prediction. Algorithm D.35 presents the pseudocode employed to immobilize the preprocessor post-training, enabling its application for generating predictions with the testing data. Algorithm D.36 presents the pseudocode utilized to guarantee that the new DataFrame maintains the identical order as the training DataFrame. Collectively, these helper functions sustain structural consistency between training and testing datasets, hence avoiding alignment discrepancies and guaranteeing dependable model inference.

```
1: Inputs:
      —

2: Output:
      NumPy, pandas, and Sklearn base classes imported;
      custom ML utility transformers defined

3: Procedure SetupAndLoadDependencies_ForMLUtils()
4:      Import future/typing helpers: from __future__ import annotations;
           from typing import Iterable
5:      Import NumPy and pandas
6:      Import BaseEstimator and TransformerMixin from sklearn.base
7:      Define transformer AlignColumns(expected_cols)
8:      Define transformer EnsureDataFrame(colnames)
9:      Define transformer SelectColumnsByName(names)
10:     Define transformer FrozenPreprocessor(fitted_preproc)
11:     Return handles to all defined transformers
12: endProcedure
```

Algorithm D.32 SetupAndLoadDependencies_ForMLUtils

```
1: Inputs:
      expected_cols (list of desired column names)
```

```
2: Output:
     Transformer that guarantees fixed column set and order
     (missing columns filled with NaN)

3: Procedure AlignColumns(expected_cols)
4:      fit:
5:           Store expected_cols; return self
6:      transform:
7:           Add any missing columns to X with NaN values
8:           Return X reordered as X[expected_cols]
9: endProcedure
```

Algorithm D.33 AlignColumns

```
1: Inputs:
     names (list of feature names to select)

2: Output:
     Transformer returning DataFrame subset with the specified columns

3: Procedure SelectColumnsByName(names)
4:      fit:
5:           Store names; return self
6:      transform:
7:           Filter X to columns present in both X and names
8:           Return X.loc[:, selected_columns]
9: endProcedure
```

Algorithm D.34 SelectColumnsByName

```
1: Inputs:
     fitted_preproc (already-fitted preprocessing object)

2: Output:
     Transformer wrapper providing a frozen (non-trainable) preprocessor

3: Procedure FrozenPreprocessor(fitted_preproc)
4:      fit:
5:           Return self (no training performed)
6:      transform:
7:           Return fitted_preproc.transform(X)
8: endProcedure
```

Algorithm D.35 FrozenPreprocessor

```
1: Inputs:
     colnames (list of expected output column names)

2: Output:
     Transformer that ensures output is always a pandas DataFrame

3: Procedure EnsureDataFrame(colnames)
4:      fit:
5:           Store colnames; return self
6:      transform:
7:           If X already has .columns:
8:                Return X unchanged
9:           Else:
10:               Wrap X into a DataFrame using stored colnames
11:               Return the new DataFrame
12: endProcedure
```

Algorithm D.36 EnsureDataFrame

Appendix A. Execution Environment

Scope

This appendix summarizes the execution environment used to perform the machine learning and data analysis experiments described in the manuscript. The complete machine-readable metadata is provided in appendix_environment_concise.txt, automatically generated by generate_env_info_auto.py.
All Python package versions are pinned in requirements.txt to ensure reproducibility.

Table A.1 Runtime Environment Summary

Area	Description
Operating system	Windows 11 (10.0.26200, 64-bit, SP0)
CPU	Intel64 Family 6 Model 183 Stepping 1 (GenuineIntel)
Python	3.13.2 (build tags/v3.13.2:4f8bb39, Feb 4 2025 15:23:48)
Virtual environment	venv
Environment snapshot	2025-10-26 17:14:01

Key Third-party Packages

- IPython v9.4.0
- imbalanced-learn v0.14.0
- joblib v1.5.1
- matplotlib v3.10.5
- mlxtend v0.23.4
- numpy v2.2.6
- packaging v25.0
- pandas v2.3.0
- scikit-learn v1.7.1
- yellowbrick v1.5
  (Complete list available in requirements.txt.)

Local Project Modules

Tools_ml_app.ml_utils — custom utility functions for data preprocessing and model evaluation.
mylib — supplementary scripts and helper methods for the ML pipeline.

Standard Library / Built-in Modules Used

__future__, argparse, datetime, importlib, pathlib, platform, re, site, sys, sysconfig

Reproducibility Notes

This environment summary provides the human-readable overview only.
For exact replication, recreate the Python environment using requirements.txt and rerun generate_env_info_auto.py to regenerate appendix_environment_concise.txt.
If discrepancies arise, the machine-readable files (appendix_environment_concise.txt and requirements.txt) are considered authoritative.