National College of Ireland

Configuration Manual

MSc Research Project
Cybersecurity

Vishwa Ramesh
X23328266

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Vishwa Ramesh		
Student ID:	X23328266		
Programme:	Cybersecurity	**Year:**	2025
Module:	MSc Research Project		
Lecturer:	Vikas Sahni		
Submission Due Date:	09/08/2025		
Project Title:	Configuration manual		
Word Count:	690	**Page Count:**	7

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Vishwa Ramesh

Date: 09/08/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Vishwa Ramesh
X23328266

1. System Requirements

Operating System:

- Windows 11

Python Version:

- Python 3.11.13

Hardware:

- RAM:16 GB (KDD datasets can be large)
- Disk Space: At least 2 GB free for dataset storage and processing

Libraries/Dependencies:

- pandas, numpy, matplotlib, seaborn
- scikit-learn (for RandomForest, SVM, Logistic Regression, metrics, preprocessing, train-test split)
- tensorflow (Keras Sequential and Dense layers)

2. Installation

Since this notebook runs on Google Colab, most libraries come pre-installed, following commands must be run first
No additional setup for hardware is required because Colab provides:
Pre-configured Windows environment
Python 3.8+
Free access to GPU/TPU (optional for deep learning models)

```
# Upgrade pip
!pip install --upgrade pip

# Install required libraries
!pip install pandas numpy matplotlib seaborn scikit-learn tensorflow
```

2.1 Dataset Import

The notebook uses the KDD dataset (kdd_train.csv and kdd_test.csv). The dataset is imported https://www.kaggle.com/datasets/kiranmahesh/nslkdd?select=kdd_train.csv

```
import pandas as pd
df_train = pd.read_csv('https://your-dataset-url/kdd_train.csv')
df_test = pd.read_csv('https://your-dataset-url/kdd_test.csv')
```

Data Loading

```
[ ] df_train = pd.read_csv('kdd_train.csv')
    df_train.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_diff_
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	0.17	
1	0	udp	other	SF	146	0	0	0	0	0	...	1	0.00	
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	0.10	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	1.00	
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	1.00	

5 rows × 42 columns

```
[ ] df_test = pd.read_csv("/content/kdd_test.csv")
```

```
[ ] df_test.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_diff_srv_rate	dst_host_same_src_port_rate	dst_h
0	5	1	54	9	2429	475	0	0	0	0	...	0.02	0.01	
1	0	2	12	9	45	134	0	0	0	0	...	0.02	0.01	
2	0	2	12	9	45	80	0	0	0	0	...	0.00	0.01	
3	1979	2	44	9	145	105	0	0	0	0	...	0.84	1.00	
4	14462	1	44	4	1	0	0	0	0	0	...	0.68	1.00	

5 rows × 44 columns

2.2 Evaluation settings

#Logistic Regression model

```
LOGISTIC_REGRESSION_EVAL = {
    "classification_metrics": ["accuracy", "precision", "recall", "f1-score", "auc"],
    "plots": {
        "roc_curve": True,
        "confusion_matrix": True
    },
    "test_data_used": True,
    "model_params": {
        "max_iter": 1000,
        "solver": "lbfgs"
    }
}
```

```
[22] print("Logistic Regression Report:")
     print(classification_report(y_test, y_pred_lr))
```

```
Logistic Regression Report:
              precision    recall  f1-score   support

           0       0.76      0.94      0.84     11245
           1       0.92      0.70      0.80     11299

    accuracy                           0.82     22544
   macro avg       0.84      0.82      0.82     22544
weighted avg       0.84      0.82      0.82     22544
```

#Keras Neural Network model

```
KERAS_MODEL_EVAL = {
   "classification_metrics": ["accuracy", "precision", "recall", "f1-score", "auc"],
   "plots": {
      "roc_curve": True,
      "confusion_matrix": True
   },
   "test_data_used": True,
   "model_params": {
      "layers": [64, 32, 1],
      "activation_functions": ["relu", "relu", "sigmoid"],
      "optimizer": "adam",
      "loss": "binary_crossentropy",
      "epochs": 10,
      "batch_size": 32
   }
}
```

#SVM model

#Model: SVC (Support Vector Classifier)

Kernel: rbf (Radial Basis Function)

Random State: 42 (for reproducibility)

#Data Handling & Preparation:

#Labels converted to binary:

attack_flag = 1 → Attack

attack_flag = 0 → Normal

Features scaled:

#Used StandardScaler to standardize training and test data.

#Training Data:

X_train_scaled, y_train

#Test Data:

X_test_scaled, y_test

#Evaluation Metric:

#classification_report — includes:

Precision

Recall

F1-score

Support (number of samples per class)

```
# Predictions
y_pred = svm_model.predict(X_test_scaled)

# Classification report
print("Classification Report for SVM:")
print(classification_report(y_test, y_pred))
```

```
Classification Report for SVM:
              precision    recall  f1-score   support

           0       0.88      0.99      0.93     11245
           1       0.99      0.86      0.92     11299

    accuracy                           0.93     22544
   macro avg       0.93      0.93      0.93     22544
weighted avg       0.93      0.93      0.93     22544
```

- Model training
- Predictions
- AUC score
- Actual vs predicted heatmap

```
[35] nn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
     nn_model.fit(X_train_scaled, y_train, epochs=10, batch_size=128, validation_split=0.2)
```

```
Epoch 1/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9323 - loss: 0.1575 - val_accuracy: 0.9908 - val_loss: 0.0373
Epoch 2/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9894 - loss: 0.0326 - val_accuracy: 0.9918 - val_loss: 0.0273
Epoch 3/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9913 - loss: 0.0259 - val_accuracy: 0.9917 - val_loss: 0.0266
Epoch 4/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9924 - loss: 0.0230 - val_accuracy: 0.9923 - val_loss: 0.0237
Epoch 5/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9929 - loss: 0.0209 - val_accuracy: 0.9938 - val_loss: 0.0223
Epoch 6/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9939 - loss: 0.0179 - val_accuracy: 0.9939 - val_loss: 0.0188
Epoch 7/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9938 - loss: 0.0186 - val_accuracy: 0.9938 - val_loss: 0.0214
Epoch 8/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9941 - loss: 0.0168 - val_accuracy: 0.9945 - val_loss: 0.0186
Epoch 9/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9947 - loss: 0.0147 - val_accuracy: 0.9940 - val_loss: 0.0207
Epoch 10/10
788/788 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9942 - loss: 0.0162 - val_accuracy: 0.9944 - val_loss: 0.0212
<keras.src.callbacks.history.History at 0x791f0cbfbd10>
```

3.Troubleshooting

Mismatched feature names:

Ensuring train and test data use the same encoders and scaling.

Dropping labels, attack_flag, and any extra columns like network_status before scaling.

predict_probability errors for SVM:

Using SVC(probability=True) to enable probability estimates.

Graph execution errors (Keras):

Graph execution errors in Keras often occur due to incompatible input shapes, missing layers, or incorrect tensor operations. Carefully inspect model architecture and layer connections. Debug by running the model step-by-step in eager execution mode or using model.summary() to verify layer outputs.

Scale data before passing to neural networks. Neural networks are sensitive to the scale of input features. Use StandardScaler, MinMaxScaler, or RobustScaler to normalize or standardize your data. Scaling ensures faster convergence and improves training stability by preventing large gradients and vanishing/exploding activation values.

Checking input shape: input_shape=(X_train_scaled.shape[1],).