National College of Ireland

Configuration Manual

MSc Research Project
Master of Science in Cybersecurity

Ademola Ajayi
Student ID: x23266015

School of Computing
National College of Ireland

Supervisor: Michael Pantridge

National College of Ireland
MSc Project Submission Sheet
School of Computing

Student Name: ……. Ademola Temitayo Ajayi……

Student ID: ……x23266015……..……

Programme: MSc in Cybersecurity…… **Year:** 2024 - 2025..

Module: …Research Project…….……

Lecturer: ……Michael Pantridge…….……

Submission Due Date: ……11-12-2025…….……

Project Title: Enchanced Fileless Malware in Memory Using Convolutional Neural Networks…….……

Word Count: ……1104…… **Page Count:** ……8……..……..……

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: …… Ademola Ajayi ……

Date: ……10-12-2025……

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

Ademola T. Ajayi
X23266015

1 Introduction

This is a configuration manual designed to explain the setup of the project and facilitate the replication of results. This handbook includes all hardware specification, software specifications, and code libraries utilized for the implementation of the artifact. The proposed plan and the research undertaken are heavily dependent on this setup manual.

2 Hardware Specifications

- Central Processing Unit (CPU) – 13th Gen Intel® Core™ i7-1355U, 1700 Mhz, 10 Core(s), 12 Logical Processor
- GPU – Intel ARC Graphics
- System Memory (RAM) – 16.0 GB (15.7 GB usable)
- Operating System (OS) – Microsoft Windows 11 Home, 64-bit Operating System
- OS Version – 10.0.26100
- OS Build – 26100
- Laptop Manufacturer – Acer
- Laptop Model – Swift SFG16-71

3 Software Specifications

For the purposes of this project, python was used as the coding language and the Integrated Development Environment (IDE) used was Microsoft Visual Code. For the purpose of machine learning, data processing, computation and analysis, several python packages were used.

- Microsoft Visual Studio Code (version - Update 1.105.1)
- Python version 3.12.10 installed on PowerShell

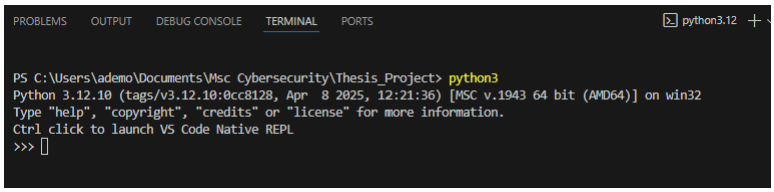

Figure 1: VSCode terminal showing python version

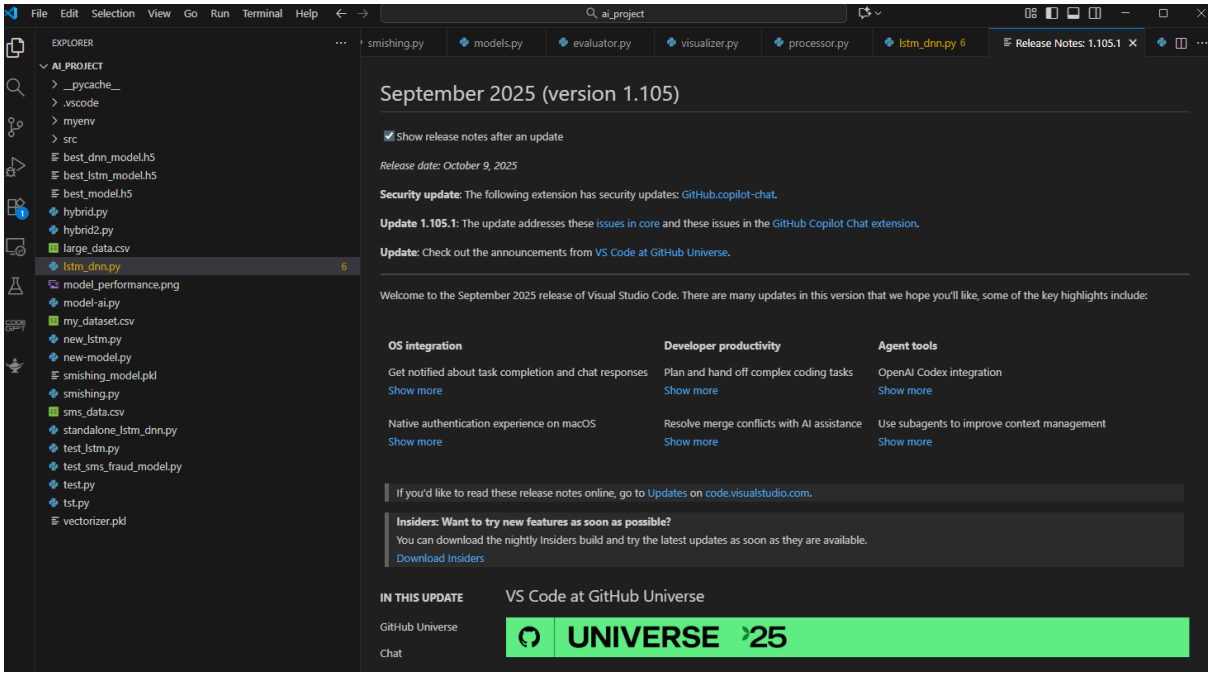

Figure 2: Microsoft VSCode IDE version

4 Dataset Download

The CIC-MALMEM-2022 dataset can be downloaded from Kaggle or from the official website of University of Newbrunswick Canada and saved in a folder. The full file path must be noted so that it can be used in the Python code. The FilelessMalwareDetection.py must have been copied and the dataset relative file path must have been added in the code before executing the file.

5. Python Software Libraries Used

The python libraries used are all in the “requirement.txt” file of the software package. Some of the libraries used are the following

- Numerical and data processing
  - Numpy version 1.26.0
  - Pandas version 2.2.0
- Machine Learning / Deep Learning
  - scikit-learn version 1.5.0
  - xgboost version 2.0.0
  - tensorflow version 2.14.0
- Visualization
  - Matplotlib version 3.8.0
  - Seaborn version0.13.0

- Reproducibility
  - Joblib version 1.3.0

6. Data Preparation

The script expects a CSV file or a zipped CSV file as shown in the figure below

```
78    def load_csv_or_zip(path: str) -> pd.DataFrame:
79        if not os.path.exists(path):
80            raise FileNotFoundError(f"Path not found: {path}")
81        if path.lower().endswith(".zip"):
82            with zipfile.ZipFile(path, 'r') as zf:
83                csvs = [n for n in zf.namelist() if n.lower().endswith(".csv")]
84                if not csvs:
85                    raise ValueError("No CSV inside ZIP.")
86                with zf.open(csvs[0]) as f:
87                    return pd.read_csv(f)
88        return pd.read_csv(path)
89
```

Figure 3: Code block for loading the CSV file

6.1 Data Preprocessing

The script automatically handles the preprocessing of data, label encoding, splitting of data for training, validation and testing. It also handles scale and feature heatmaps.

```
def label_encode(df: pd.DataFrame, label_col: str = "Label") -> pd.DataFrame:
    if label_col not in df.columns:
        raise ValueError(f"Missing label col '{label_col}'.")
    if not pd.api.types.is_numeric_dtype(df[label_col]):
        le = LabelEncoder()
        df[label_col] = le.fit_transform(df[label_col].astype(str).fillna("unknown"))
    return df

def split_and_scale(
    df: pd.DataFrame, label_col: str = "Label",
    test_size: float = 0.2, val_size: float = 0.1
) -> Tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray, np.ndarray, np.ndarray, MinMaxScaler, List[str]]:
    numeric_df = df.select_dtypes(include=[np.number]).copy()
    if label_col not in numeric_df.columns:
        numeric_df[label_col] = df[label_col].values

    feature_names = [c for c in numeric_df.columns if c != label_col]
    X = numeric_df[feature_names].values
    y = numeric_df[label_col].values

    if len(np.unique(y)) < 2:
        raise ValueError("Dataset has only one class after encoding.")

    X_tr, X_te, y_tr, y_te = train_test_split(
        X, y, test_size=test_size, stratify=y, random_state=SEED
    )
    val_fraction = val_size / (1.0 - test_size)
```

Figure 4: Code block showing data preprocessing, split and scale.

7. Model Training and Evaluation

7.1 CNN Model

This function constructs a Convolutional Neural Network intended to categorise memory artefacts as benign or malicious. The model accepts an 8×8 feature map as input and employs two convolutional layers to identify spatial behavioural patterns linked to fileless malware. Batch normalisation

enhances training stability, but max pooling minimises dimensionality. The collected features are flattened and subsequently processed through a thick layer with dropout to mitigate overfitting. A terminal sigmoid output layer generates a probability score for binary classification. The model employs the Adam optimiser and utilises binary cross-entropy loss, with accuracy, precision, and recall measured throughout training to assess detection efficacy.

```python
def build_cnn(input_shape=(8, 8, 1)) -> tf.keras.Model:
    inputs = layers.Input(shape=input_shape, name="input")
    x = layers.Conv2D(32, (3,3), activation="relu", name="conv1")(inputs)
    x = layers.BatchNormalization(name="bn1")(x)
    x = layers.MaxPooling2D((2,2), name="pool1")(x)
    x = layers.Conv2D(64, (3,3), activation="relu", name="conv2")(x)
    x = layers.BatchNormalization(name="bn2")(x)
    x = layers.Flatten(name="flatten")(x)
    x = layers.Dense(128, activation="relu", name="dense1")(x)
    x = layers.Dropout(0.5, name="drop1")(x)
    outputs = layers.Dense(1, activation="sigmoid", name="out")(x)
    model = models.Model(inputs, outputs, name="cnn_malware")
    model.compile(
        optimizer=tf.keras.optimizers.Adam(1e-3),
        loss="binary_crossentropy",
        metrics=["accuracy", tf.keras.metrics.Precision(name="precision"), tf.keras.metrics.Recall(name="recall")]
    )
    return model
```

Figure 5: Code block showing CNN model build

7.2 CNN Training

This section of the scripts trains the CNN model. It first builds the CNN, then sets up callbacks for early stopping and saving the best model during training.

```python
    model = build_cnn(input_shape=(8,8,1))
    cbs = [
        callbacks.EarlyStopping(monitor="val_loss", patience=6, restore_best_weights=True),
        callbacks.ModelCheckpoint(os.path.join(outdir, "best_cnn.keras"), monitor="val_loss", save_best_only=True)
    ]
    hist = model.fit(
        X_tr_img, y_tr,
        validation_data=(X_va_img, y_va),
        epochs=args.epochs, batch_size=args.batch_size, verbose=2, callbacks=cbs
    )
    h = pd.DataFrame(hist.history)
    h.to_csv(os.path.join(outdir, "cnn_training_history.csv"), index=False)
```

Figure 6: Code block showing CNN training

7.3 CNN Evaluation

This code block generates predictions from the trained CNN on the test dataset and evaluates the model's performance. It computes key binary classification metrics using the predicted probabilities, then creates and saves a confusion matrix plot.

```python
    yprob_cnn = model.predict(X_te_img, verbose=0).ravel()
    met_cnn = evaluate_binary(y_te, yprob_cnn)
    plot_confmat(y_te, met_cnn["y_pred"], "CNN — Confusion Matrix", os.path.join(plots_dir, "cnn_confusion_matrix.png"))
    save_json(met_cnn, os.path.join(outdir, "cnn_metrics.json"))
```

Figure 7: Code block showing the evaluation of CNN model

7.4 Benchmarked Baselines (SVM, RF, XGBoost and Naïve Bayes)

This code block prepares the baseline ML models that will be compared against the CNN. It creates a dictionary of classifiers such as Random Forest, Naïve Bayes, and SVM with an RBF kernel, and XGBoost. It then merges the training, and validation sets into a single dataset, which will be used to train these baseline models for fair comparison with the CNN.

```
249     baseline_map = {
250         "RandomForest": RandomForestClassifier(n_estimators=250, max_depth=12, n_jobs=-1, random_state=SEED),
251         "NaiveBayes": GaussianNB(),
252         "SVM_RBF": SVC(kernel="rbf", probability=True, random_state=SEED),
253     }
254     if XGB_AVAILABLE:
255         baseline_map["XGBoost"] = XGBClassifier(
256             n_estimators=350, learning_rate=0.1, max_depth=6,
257             subsample=0.9, colsample_bytree=0.9, random_state=SEED, eval_metric="logloss"
258         )
259
260     X_all = np.vstack([X_tr, X_va])
261     y_all = np.concatenate([y_tr, y_va])
262
```

Figure 8: Code block showing other ML baseline models

7.5 Evaluation of Benchmarked ML Baselines

This section trains and evaluates all baseline ML models. The metrics and confusion matrix for each model are saved to disk. The results such as accuracy, precision, recall, F1-score, and ROC-AUC are stored for later comparison, and ROC curve data is collected to enable multi-model visual analysis.

```
263     # Evaluate all baselines
264     results, roc_curves = [], []
265     for name, mdl in baseline_map.items():
266         print(f"[+] Training and evaluating {name} ...")
267         mdl.fit(X_all, y_all)
268         if hasattr(mdl, "predict_proba"):
269             yprob = mdl.predict_proba(X_te)[:, 1]
270         else:
271             s = mdl.decision_function(X_te)
272             yprob = (s - s.min()) / (s.max() - s.min() + 1e-8)
273         met = evaluate_binary(y_te, yprob)
274         save_json(met, os.path.join(outdir, f"{name}_metrics.json"))
275         plot_confmat(y_te, met["y_pred"], f"{name} — Confusion Matrix", os.path.join(plots_dir, f"{name}_confusion.png"))
276         results.append({
277             "Model": name,
278             "Accuracy": met["accuracy"],
279             "Precision": met["precision"],
280             "Recall": met["recall"],
281             "F1": met["f1"],
282             "ROC_AUC": met["roc_auc"]
283         })
284         roc_curves.append({"label": name, "fpr": met["fpr"], "tpr": met["tpr"], "auc": met["roc_auc"]})
```

Figure 9: Code block showing evaluation of benchmarked models

7.6 Visualization and Metric Plotting

These functions generate visual evaluation plots for model performance. This visual shows how many benign and malware samples were correctly or incorrectly classified.

```python
def plot_confmat(y_true, y_pred, title, path):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(4.8,4.2))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=["Benign","Malware"],
                yticklabels=["Benign","Malware"])
    plt.xlabel("Predicted"); plt.ylabel("Actual"); plt.title(title)
    plt.tight_layout(); plt.savefig(path, dpi=200); plt.close()

def plot_roc_curve(curves: List[Dict], path: str, title="ROC Curves"):
    plt.figure(figsize=(6,4.5))
    for c in curves:
        plt.plot(c["fpr"], c["tpr"], label=f'{c["label"]} (AUC={c["auc"]:.4f})')
    plt.plot([0,1],[0,1],"--", color="gray")
    plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
    plt.title(title); plt.legend(loc="lower right")
    plt.tight_layout(); plt.savefig(path, dpi=220); plt.close()
```

Figure 10: Code blocking showing the visualization functions

References

Canadian Institute for Cybersecurity, 2022. *Malware memory analysis (CIC-MalMem-2022)*. [Online] Available at: https://www.unb.ca/cic/datasets/malmem-2022.html [Accessed 11 October 2025]

Godoy, L., 2022. *Malware Memory Analysis / CIC-MalMem-2022*. [Online] Available at: https://www.kaggle.com/datasets/luccagodoy/obfuscated-malware-memory-2022-cic/data [Accessed 12 October 2025].

Powers, D.M. (2020) 'Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation,' arXiv (Cornell University), 2(1), pp. 37–63. https://doi.org/10.48550/arxiv.2010.16061.