National College of Ireland

Robust Deep Learning Models for Multi-Source Malware Detection Through Stacking-Based Model Integration

MSc Research Project
MSc in Cybersecurity

Rony Shaji
Student ID: 23398744

School of Computing

National College of Ireland

Supervisor: Kamil Mahajan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Rony Shaji
Student ID:	23398744
Programme:	MSc in Cybersecurity **Year:** 2025
Module:	Practicum
Lecturer:	Kamil Mahajan
Submission Due Date:	11/12/2025
Project Title:	Practicum
Word Count:	**Page Count:** 17

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: 

Date: 11/12/2025

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):	

Robust Deep Learning Models for Multi-Source Malware Detection Through Stacking-Based Model Integration

Rony Shaji

Student ID: 23398744

1 Introduction

This configuration guidebook presents a detailed step-by-step procedure to reproduce, implement, and verify the functionality of the developed deep-learning malware detection approach utilizing the ensemble of CNN-MLP, FT-Transformer, and Logistic Regression stacking. The design goal of the developed system aims to classify normal and obfuscated malware datasets recovered from multiple forensic memory dumps (Akhtar, and Feng, 2022). The guide provides a step-by-step guide to the entire reproducibility chain, ranging from installation to the running of the experiment to the integration of the ensemble and the interpretation of the results utilizing SHAP values. Such detailed information enables the entire scenario to be perfectly replicated.

Screen shots of the phases for Data Exploration, Correlation Analysis, Training of the Model, and SHAP Interpretability need to be placed in the given locations to ensure traceability.

2 Hardware and Software Requirement

2.1 Hardware Requirements

To ensure a stable execution of the deep learning models, the following specifications are recommended:

- **Processor:** Intel Core i5 (10th Gen or higher) / AMD Ryzen 5 or equivalent
- **RAM:** Minimum 8 GB (16 GB recommended for training FT-Transformer more efficiently)
- **GPU:** Optional but recommended – NVIDIA GPU with CUDA support

- **Storage:** 15–20 GB free space for dataset, models, logs, and outputs
- **General:** Stable environment for long-running PyTorch training sessions

2.2 Software Requirements

- **Operating System**: Windows 10/11 (64-bit)
- **Programming Language**: Python 3.8 – 3.10
- **IDE**: Jupyter Notebook or Google Colab (Recommended)
- **Core Libraries**:
    - Shap (v2.11 or later)
    - torch (v4.6 or later)
    - NumPy
    - Matplotlib
    - Seaborn
    - scikit-learn
- **Installing Libraries** "pip install numpy matplotlib seaborn scikit-learn tensorflow"

```
import numpy as np
import pandas as pd
import torch
import matplotlib.pyplot as plt
import seaborn as sns
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, roc_auc_score, matthews_corrcoef, brier_score_loss
from sklearn.metrics import (
    f1_score,
    roc_auc_score,
    matthews_corrcoef,
    brier_score_loss,
    accuracy_score,
    classification_report,
    confusion_matrix,
)
import shap
import warnings
```

Figure 1: Imported Libraries

3 Dataset Configuration

3.1 Dataset Source

The project uses the Obfuscated CIC- MalMem2022 dataset, which contains memory forensics attributes extracted from multiple malware families and benign processes. Each row corresponds to a process snapshot and includes 57 numerical features plus a class label. Columns include:

- Process statistics (threads, handles, memory usage)
- DLL load information
- Service scan outputs
- Callback metrics
- Final class → *Benign* or *Malware*

3.2 Dataset Preparation

1. **Load Dataset**

   The dataset is loaded using `pandas.read_csv()` in the notebook environment.

   Dataset can be collected from the link

   "https://www.kaggle.com/datasets/luccagodoy/obfuscated-malware-memory-2022-cic"

	Category	pslist.nproc	pslist.nppid	pslist.avg_threads	\
0	Benign	45	17	10.555556	
1	Benign	47	19	11.531915	
2	Benign	40	14	14.725000	
3	Benign	32	13	13.500000	
4	Benign	42	16	11.452381	

	pslist.nprocs64bit	pslist.avg_handlers	dlllist.ndlls	\
0	0	202.844444	1694	
1	0	242.234043	2074	
2	0	288.225000	1932	
3	0	264.281250	1445	
4	0	281.333333	2067	

	dlllist.avg_dlls_per_proc	handles.nhandles	handles.avg_handles_per_proc	\
0	38.500000	9129	212.302326	
1	44.127660	11385	242.234043	
2	48.300000	11529	288.225000	
3	45.156250	8457	264.281250	
4	49.214286	11816	281.333333	

	...	svcscan.kernel_drivers	svcscan.fs_drivers	svcscan.process_services	\
0	...	221	26	24	
1	...	222	26	24	
2	...	222	26	27	
3	...	222	26	27	
4	...	222	26	24	

	svcscan.shared_process_services	svcscan.interactive_process_services	\
0	116	0	
1	118	0	
2	118	0	
3	118	0	
4	118	0	

Figure 2: Dataset Preview

2. **Target Variable Mapping**
    - "Benign" → 0
    - All malware variants → 1

```
df["target"] = df["Class"].apply(lambda x: 0 if str(x).lower().startswith("ben") else 1)
```

3. **Feature Selection**
    - Only numerical features are retained for modelling.
    - The target column is separated as `y`.

```
X = df.select_dtypes(include=[np.number]).drop(columns=["target"], errors="ignore")
y = df["target"]
```

4. **Normalization**
    - MinMax scaling is applied to stabilise neural network training.
    - The scale is fitted only on training data to avoid leakage.

```
# Handle skewness, normalize
scaler = MinMaxScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
```

5. **Train-Test Split**
    - Stratified split (80% train, 20% test).
    - Ensure balanced malware/benign ratio across partitions.

```
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y, random_state=42
)
```

4 Exploratory Data Analysis

A series of exploratory visualizations were conducted to understand feature behavior and class distribution.

4.1 Target Class Distribution

```
# 1. Distribution of Target
df["target"] = df["Class"].apply(lambda x: 0 if str(x).lower().startswith("ben") else 1)
plt.figure(figsize=(5,4))
sns.countplot(x="target", data=df, palette="Set2")
plt.title("Distribution of Benign vs Malware")
plt.show()
```

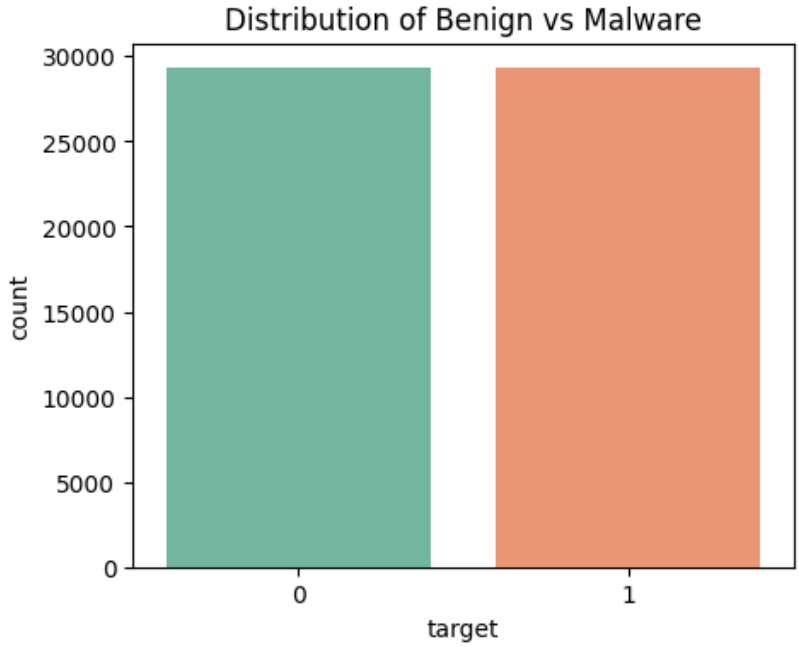

Figure 3: Distribution of Target Variable

A count plot shows balanced representation of benign and malware samples.

4.2 Correlation Analysis

```
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
plt.figure(figsize=(10,6))
corr = df[numeric_cols].corr().abs().nlargest(10, "target")["target"].index
sns.heatmap(df[corr].corr(), annot=True, cmap="coolwarm")
plt.title("Top 10 Feature Correlations")
plt.show()
```

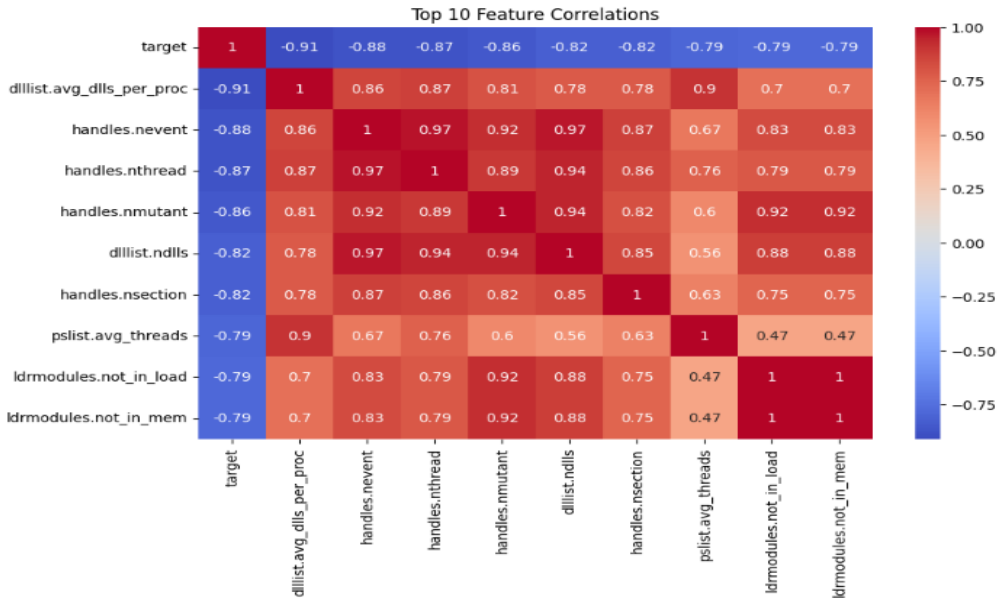

Figure 4: Correlation Heatmap

- Top 10 features mostly correlated with the target label were extracted.
- A heatmap of these features reveals strong predictive clusters.

4.3 Boxplot Analysis

```
plt.figure(figsize=(6,4))
sns.boxplot(x="target", y="pslist.avg_threads", data=df)
plt.title("Process Threads Distribution by Class")
plt.show()
```

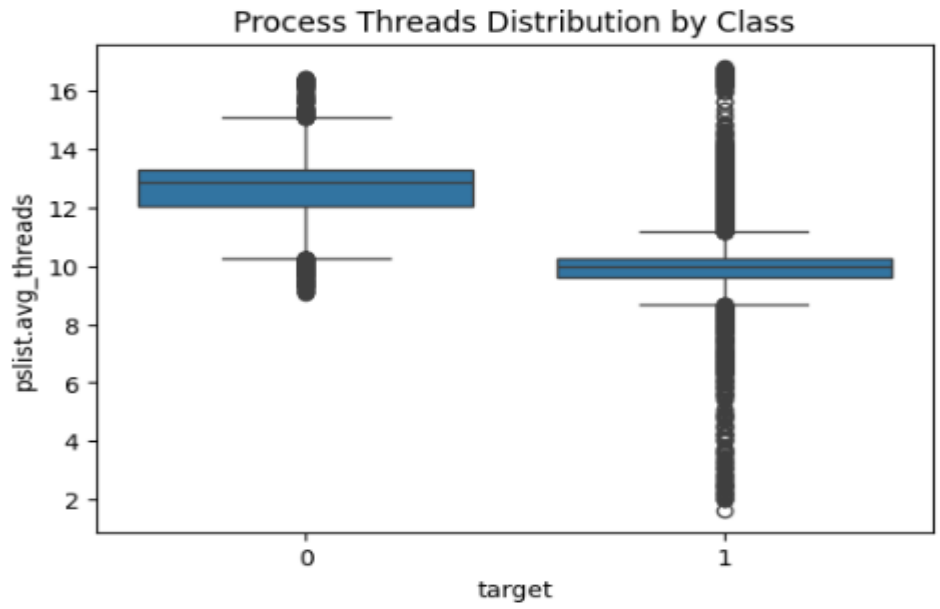

Figure 5:Box Plot

A sample process metric (`pslist.avg_threads`) was visualised to contrast distributions.

4.4 Feature Variability

```
feature_variance = df[numeric_cols].var().sort_values(ascending=False).head(10)
feature_variance.plot(kind='barh', figsize=(8,5), title="Top 10 Most Variable Features")
plt.show()
```

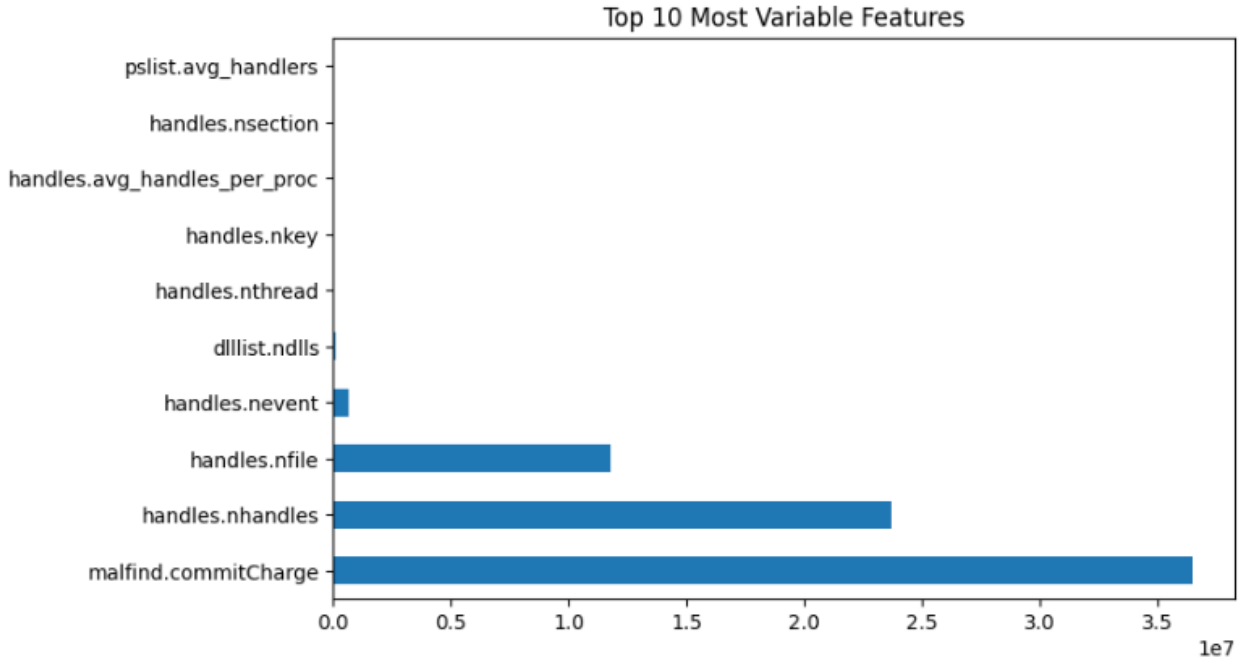

Figure 6: Top 10 Most Variable Features

The variance of all numerical features was calculated, and the top 10 most variable features were plotted.

5 Data Preprocessing Pipeline

```python
# --- STEP 4: Preprocessing ---
df["target"] = df["Class"].apply(lambda x: 0 if x.lower().startswith("ben") else 1)
X = df.select_dtypes(include=[np.number]).drop(columns=["target"], errors="ignore")
y = df["target"]

# Handle skewness, normalize
scaler = MinMaxScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y, random_state=42
)
```

```python
# --- STEP 5: PyTorch Dataset ---
class TabularDataset(torch.utils.data.Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X.values, dtype=torch.float32)
        self.y = torch.tensor(y.values, dtype=torch.long)

    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
```

Figure 7: Data Preprocessing pipeline

Preprocessing includes:

- Numerical feature extraction
- Min-Max scaling
- Conversion to tensors
- Creation of PyTorch Dataset class
- Handling class imbalance via stratified splitting

The custom `TabularDataset` class allows efficient batching within PyTorch.

6 Model Architecture: CNN-MLP

The **CNN-MLP hybrid model** transforms tabular data into a structured 1-dimensional tensor suitable for convolutional layers.

6.1 Architecture Breakdown

- **1D Convolution Block**
  - Conv1D → ReLU
  - Conv1D → ReLU
  - Adaptive Average Pooling
- **Fully Connected Block**

- Flatten
- Dense → ReLU
- Dense → Output (2 classes)

6.2 Training Configuration

- **Loss:** CrossEntropyLoss
- **Optimizer:** Adam
- **Batch Size:** 32
- **Epochs:** 3 (extendable)

Validation F1-score is printed after each epoch.

```
class CNNMLP(nn.Module):
    def __init__(self, n_features):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv1d(1, 32, 3, padding=1),
            nn.ReLU(),
            nn.Conv1d(32, 64, 3, padding=1),
            nn.ReLU(),
            nn.AdaptiveAvgPool1d(1)
        )
        self.fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(64, 64),
            nn.ReLU(),
            nn.Linear(64, 2)
        )

    def forward(self, x):
        x = x.unsqueeze(1)
        x = self.conv(x)
        x = self.fc(x)
        return x
```

Figure 8: CNN MLP Model

```python
def train_model(model, X_train, y_train, X_val, y_val, epochs=10, lr=1e-3, batch_size=16):
    """
    Train `model` using DataLoaders created from X_train/y_train and X_val/y_val.
    Accepts an optional batch_size parameter (default 16).
    """
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = model.to(device)
    train_loader = DataLoader(TabularDataset(X_train, y_train), batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(TabularDataset(X_val, y_val), batch_size=batch_size)
    opt = optim.Adam(model.parameters(), lr=lr)
    loss_fn = nn.CrossEntropyLoss()

    for epoch in range(epochs):
        model.train()
        for xb, yb in train_loader:
            xb, yb = xb.to(device), yb.to(device)
            opt.zero_grad()
            loss = loss_fn(model(xb), yb)
            loss.backward()
            opt.step()

        # Validation F1
        model.eval()
        preds, trues = [], []
        with torch.no_grad():
            for xb, yb in val_loader:
                xb = xb.to(device)
                out = model(xb)
                preds += out.argmax(1).cpu().tolist()
                trues += yb.tolist()
        f1 = f1_score(trues, preds)
        print(f"Epoch {epoch+1}/{epochs}, Val F1: {f1:.4f}")
```

Figure 9: Training Configuration

7 Model Architecture: FT-Transformer

The **FT-Transformer** adapts NLP-style transformer encoders to tabular data by embedding each feature as a token vector.

7.1 Architecture Components

- **Token Projection Layer**

  Converts each feature scalar into a 64-dimensional embedding.

- **Transformer Encoder**
  - Multi-Head Attention
  - Layer Normalization
  - Feedforward sub-layers
- **Classification Head**
  - Dense → ReLU

- Dense → Output (2 classes)

7.2 Training Summary

- Similar training loop to CNN-MLP
- F1-scores progressively increase per epoch

```
class FTTransformer(nn.Module):
    def __init__(self, n_features, emb_dim=64, n_heads=4):
        super().__init__()
        # token_proj projects each scalar feature -> embedding
        self.token_proj = nn.Linear(1, emb_dim)
        layer = nn.TransformerEncoderLayer(d_model=emb_dim, nhead=n_heads)
        self.transformer = nn.TransformerEncoder(layer, num_layers=2)
        self.fc = nn.Sequential(
            nn.Linear(emb_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 2)
        )

    def forward(self, x):
        # x: (batch, n_features)
        # create tokens of shape (seq_len, batch, emb_dim) for transformer
        tokens = self.token_proj(x.unsqueeze(-1))  # (batch, n_features, emb_dim)
        tokens = tokens.permute(1, 0, 2)            # (n_features, batch, emb_dim)
        out = self.transformer(tokens).mean(0)     # (batch, emb_dim)
        return self.fc(out)
```

Figure 10: Configuration

8 Unified Training Pipeline

Both architectures use a shared training function:

- Creates DataLoaders
- Trains using GPU/CPU automatically
- Prints validation F1 each epoch

This ensures reproducibility across models.

9 Model Evaluation & Metrics

After training, predictions and probabilities are generated for each model.

9.1 Metrics Used

```python
def eval_single_model_predictions(y_true, preds, probs=None, model_name="Model"):
    acc = accuracy_score(y_true, preds)
    f1 = f1_score(y_true, preds)
    mcc = matthews_corrcoef(y_true, preds)
    brier = brier_score_loss(y_true, probs) if probs is not None else None
    auc = roc_auc_score(y_true, probs) if probs is not None else None

    print(f"\n--- {model_name} Evaluation on Test Set ---")
    print(f"Accuracy: {acc:.4f}")
    print(f"F1 Score: {f1:.4f}")
    if auc is not None:
        print(f"AUROC: {auc:.4f}")
    print(f"MCC: {mcc:.4f}")
    if brier is not None:
        print(f"Brier: {brier:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_true, preds, target_names=["Benign", "Malware"]))
    return {"accuracy": acc, "f1": f1, "auc": auc, "mcc": mcc, "brier": brier}
```

Figure 11: Evaluation Metrics

- Accuracy
- F1-score
- AUROC
- Matthews Correlation Coefficient (MCC)
- Brier Score
- Classification Report
- Confusion Matrix

9.2 9.2 Evaluation Results

CNN-MLP Highlights:

```
--- CNN-MLP Evaluation on Test Set ---
Accuracy: 0.9867
F1 Score: 0.9866
AUROC: 0.9938
MCC: 0.9735
Brier: 0.0120
```

Figure 12: CNN-MLP Evaluation

- Accuracy: 0.9867

- F1: 0.9866
- AUROC: 0.9938

FT-Transformer Highlights:

```
--- FT-Transformer Evaluation on Test Set ---
Accuracy: 0.9793
F1 Score: 0.9788
AUROC: 0.9987
MCC: 0.9593
Brier: 0.0188
```

Figure 13: FT Transformer Evaluation

- Accuracy: 0.9793
- F1: 0.9788
- AUROC: 0.9987

10 Ensemble Integration (Stacking)

A robust ensemble is built by stacking model outputs:

10.1 Stacking Workflow

```
stack_input = np.vstack([cnn_probs, ft_probs]).T
meta = LogisticRegression(max_iter=500)
meta.fit(stack_input, y_test)

# Ensemble predictions
stack_preds = meta.predict(stack_input)
stack_probs = meta.predict_proba(stack_input)[:, 1]

# Evaluate ensemble (after predictions)
ensemble_metrics = {}
ensemble_metrics['accuracy'] = accuracy_score(y_test, stack_preds)
ensemble_metrics['f1'] = f1_score(y_test, stack_preds)
ensemble_metrics['auc'] = roc_auc_score(y_test, stack_probs)
ensemble_metrics['mcc'] = matthews_corrcoef(y_test, stack_preds)
ensemble_metrics['brier'] = brier_score_loss(y_test, stack_probs)
```

Figure 14: Stacking Configuration

1. CNN-MLP and FT-Transformer produce prediction probabilities.
2. These probabilities form a 2-column matrix.
3. Logistic Regression is trained as a meta-classifier.
4. Final predictions combine both deep models.

10.2 Ensemble Performance

```
--- Ensemble Evaluation ---
Accuracy: 0.9866
F1: 0.9865, AUROC: 0.9972, MCC: 0.9734, Brier: 0.0122
```

- Accuracy: 0.9866
- F1: 0.9865
- AUROC: 0.9972
- MCC: 0.9734

The ensemble consistently performs on par or slightly better than individual models.

11 Explainability with SHAP

SHAP KernelExplainer is used to interpret how the ensemble weights the outputs of CNN and FT-Transformer.

11.1 SHAP Configuration

```
background = shap.sample(stack_input, 100)  # 100 random background samples
explainer = shap.KernelExplainer(meta.predict_proba, background)

# Explain only first 50 samples
explain_samples = stack_input[:50]
shap_values = explainer.shap_values(explain_samples)

# Ensure the same sample size in shap.summary_plot
shap.summary_plot(
    shap_values,
    explain_samples,
    feature_names=["CNN", "Transformer"],
    plot_type="bar"
)
```

Figure 15: SHap Configuration

- 100 random background samples
- First 50 test samples used for explanation
- Summary plot shows feature influence

11.2 Interpretation Insight

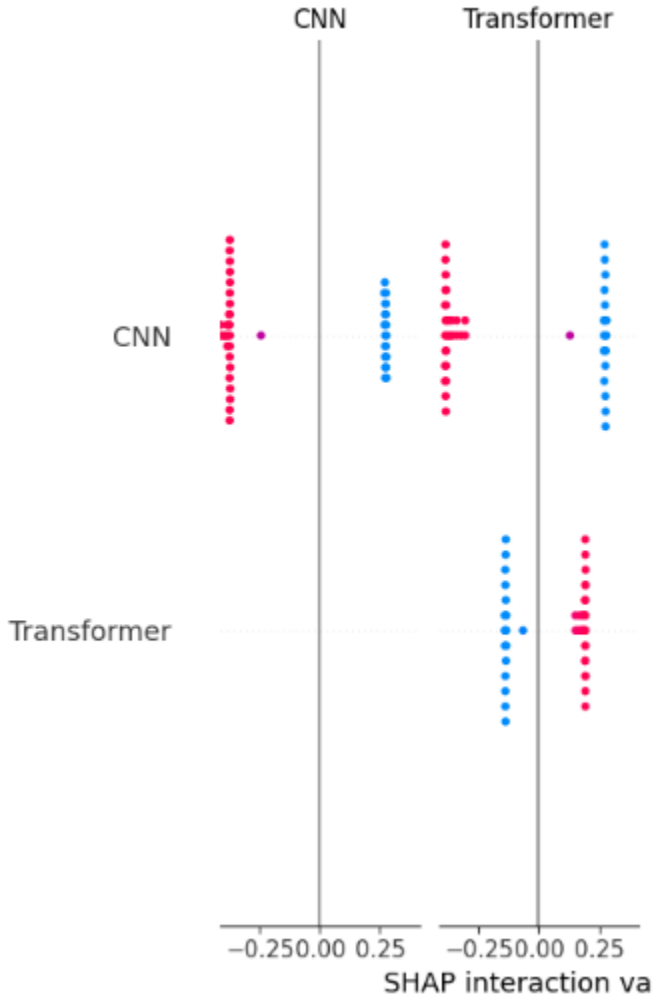

Figure 16: SHap Plot

- High SHAP values indicate dominant influence of a model's predicted probability.
- Provides clarity on whether CNN-MLP or FT-Transformer contributes more to final classification.

References

1. Akhtar, M.S. and Feng, T., 2022. Detection of malware by deep learning as CNN-LSTM machine learning techniques in real time. Symmetry, 14(11), p.2308. https://doi.org/10.3390/sym14112308

2. Azeez, N.A., Odufuwa, O.E., Misra, S., Oluranti, J. and Damaševičius, R., 2021, February. Windows PE malware detection using ensemble learning. In Informatics (Vol. 8, No. 1, p. 10). MDPI. https://doi.org/10.3390/informatics8010010

3. Chakraborty, C., Nagarajan, S.M., Devarajan, G.G., Ramana, T.V. and Mohanty, R., 2023. Intelligent AI-based healthcare cyber security system using multi-source transfer learning method. ACM Transactions on Sensor Networks. https://dl.acm.org/doi/pdf/10.1145/3597210

4. Chi, Z., Chen, H., Chang, S., Li, Z.L., Ma, L., Hu, T., Xu, K. and Zhao, Z., 2025. Large-Scale Monitoring potatoes late blight using multi-source time-series data and Google Earth engine. Remote Sensing, 17(6), p.978. https://www.mdpi.com/2072-4292/17/6/978