**Project Report: AI/ML for Networking**

**Group Name: KavachX**

**1. Problem Statement**

In the modern digital world, cybersecurity threats have become more sophisticated and frequent. Network Intrusion Detection Systems (NIDS) are essential for detecting unauthorized access or malicious activities within a network. However, traditional methods often struggle to handle the scale and complexity of modern traffic data.

This project focuses on applying Artificial Intelligence (AI) and Machine Learning (ML) to develop a robust Intrusion Detection System (IDS) capable of identifying and classifying various network attacks using real-time network packet captures. The system uses PyShark for packet capture, extracts key features, and classifies traffic using a 1D CNN. Alerts are sent via email when malicious activity is detected.

**2. Abstract**

The project, titled "KavachX: Real-Time Malware Detection from Network Packets," explores the use of deep learning techniques to identify intrusions in real-time network traffic. Using the Obfuscated Malware Memory (Obf-MalMem2022) dataset, network packet features are extractedand transformed using PCA. SMOTE is used to balance the data. The system is trained on this preprocessed datausing a 1D CNN implemented with TensorFlow/Keras.

Once deployed, KavachX captures live network packets, classifies them in real time, and sends alerts through email when suspicious activity is detected. The system also logs flagged packets for analysis. Results demonstrate high classification accuracy, validating the effectiveness of our approach for real-world IDS deployments.

**3. Code Explanation**

**3.1 Attention Mechanism**

def attention\_block(inputs):

attention = Dense(inputs.shape[-1], activation='softmax')(inputs)

return Multiply()([inputs, attention])

**The attention block helps the model focus on the most informative parts of the sequence. The Dense layer with softmax activation computes a set of weights across the features, which are then multiplied element-wise with the original inputs.**

**3.2 Model Definition - 1D CNN with Attention**

def build\_model(input\_dim, num\_classes, use\_attention=True):

inputs = Input(shape=(input\_dim, 1))

x = Conv1D(512, kernel\_size=3, activation='relu', padding='same')(inputs)

x = Conv1D(256, kernel\_size=3, activation='relu', padding='same')(x)

x = Conv1D(128, kernel\_size=3, activation='relu', padding='same')(x)

x = Conv1D(64, kernel\_size=3, activation='relu', padding='same')(x)

x = Conv1D(32, kernel\_size=3, activation='relu', padding='same')(x)

if use\_attention:

x = attention\_block(x)

x = Flatten()(x)

for units in [128, 128, 64, 64, 32, 32]:

x = Dense(units, activation='relu')(x)

x = Dropout(0.3)(x)

if num\_classes == 2:

output = Dense(1, activation='sigmoid')(x)

loss = 'binary\_crossentropy'

else:

output = Dense(num\_classes, activation='softmax')(x)

loss = 'sparse\_categorical\_crossentropy'

model = Model(inputs, output)

model.compile(optimizer='adam', loss=loss, metrics=['accuracy'])

return model

**This deep 1D CNN architecture has multiple convolutional layers to progressively extract and refine spatial features. The optional attention layer improves focus on significant features. Dense layers further learn deep patterns before final classification.**

**3.3 Running the Experiment**

def run\_experiment(df, use\_pca=True, use\_smote=True, use\_attention=True, tag="Full\_Model"):

print(f"\n🚀 Training: {tag}")

X\_train, X\_test, y\_train, y\_test, input\_dim, le, scaler, pca = preprocess\_data(df, use\_pca, use\_smote)

num\_classes = len(np.unique(y\_train))

model = build\_model(input\_dim, num\_classes, use\_attention)

checkpoint = ModelCheckpoint(f"{tag}\_model.h5", save\_best\_only=True, monitor='val\_accuracy', mode='max')

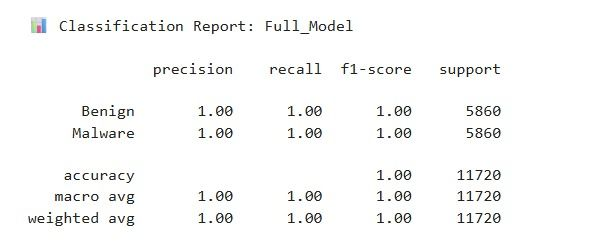
model.fit(X\_train, y\_train, validation\_split=0.1, epochs=100, batch\_size=32, callbacks=[checkpoint], verbose=1)

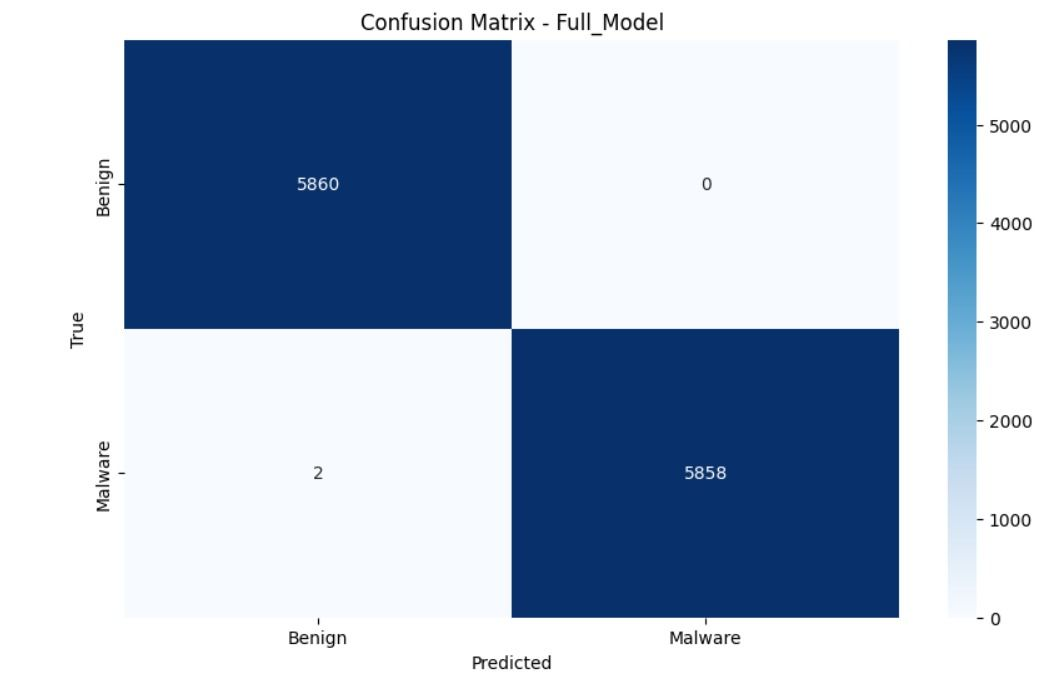
model.load\_weights(f"{tag}\_model.h5")

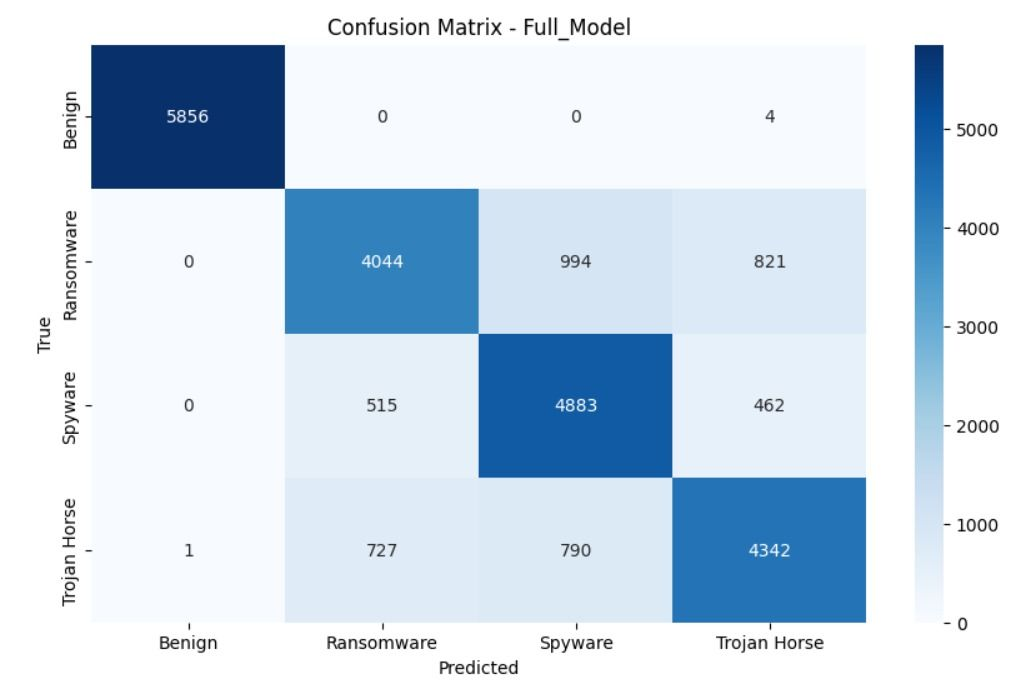
evaluate\_model(model, X\_test, y\_test, le, tag)

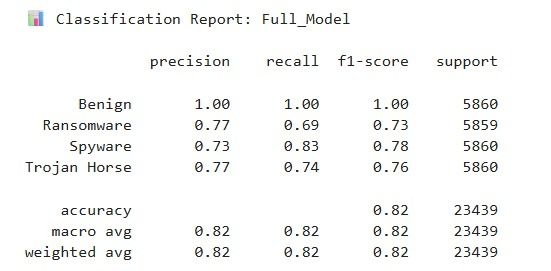
**This function integrates preprocessing, model training, checkpointing, and final evaluation. It allows toggling attention, PCA, and SMOTE dynamically.**

**4.Classification Report**

****

****

****

****

**5. Conclusion**

KavachX is a practical real-time malware detection system leveraging AI/ML. By combining live packet capture (PyShark), PCA, SMOTE, and a deep 1D CNN with optional attention, it accurately classifies network packets and alerts users of threats. Its modular design allows future enhancements such as ensemble learning, visualization dashboards, and Docker deployment.

This system proves how AI can strengthen network security by enabling intelligent, automated intrusion detection.

**End of Report**