Network Threat Analysis System

A Machine Learning Pipeline for Network Intrusion Detection

GitHub: https://github.com/MoeMaloudi/Msc_Project/tree/Network_Behaviour_Analysis

1 What This Does

This system analyzes network traffic to detect cyber attacks. It processes PCAP files (network packet captures) and uses machine learning to identify different types of attacks like DDoS, port scans, web attacks, and more.

The system can handle large datasets and includes pre-trained models so you can start analyzing immediately.

1.1 Repository Access

GitHub Repository: https://github.com/MoeMaloudi/Msc_Project/tree/Network_Behaviour_Analysis

Video Resources:

- **Program Explanation:** https://youtu.be/r3cucFsbzqg - Complete project overview and explanation
- **Quick Start Guide:** https://youtu.be/nZu6LL9M7Wc - Step-by-step getting started tutorial

The repository includes:

- Complete Jupyter notebook with all processing cells
- Pre-trained models for immediate use (`Trained_Models_For_Quick_Demo/`)
- Pre-extracted features for ML training (`Features_For_Quick_Demo/`)
- Documentation and example outputs
- All necessary utility functions and helper scripts

2 Libraries and Dependencies

2.1 System Requirements

- **Python Version:** 3.12+ (tested with Python 3.12)
- **Memory:** 8-32 GB RAM (optimized for high-memory systems)
- **CPU:** Multi-core processor (utilizes all available cores)
- **Storage:** 20+ GB free disk space for full dataset
- **Operating System:** Windows/Linux/macOS

2.2 Core Python Libraries

2.2.1 Essential System Libraries

Listing 1: System and Utility Libraries

```
# Built-in Python libraries (included with Python 3.12)
import os           # File system operations
import gc           # Garbage collection for memory management
import warnings     # Warning control
import json         # JSON data handling
import pickle       # Object serialization
import hashlib      # Hash functions for data integrity
import re           # Regular expressions
import glob         # File pattern matching
import time         # Time operations
import multiprocessing  # Parallel processing support
from datetime import datetime
from collections import defaultdict, Counter, deque
from typing import Dict, List, Tuple, Optional, Any, Generator
from dataclasses import dataclass, field
```

2.2.2 Data Processing Libraries

Listing 2: Data Processing Dependencies

```
# Core data processing
pip install pandas>=2.0.0      # DataFrame operations and data manipulation
pip install numpy>=1.24.0      # Numerical computing and arrays
pip install scipy>=1.10.0      # Statistical functions and scientific computing

# High-performance data storage (replaces HDF5)
pip install pyarrow>=12.0.0    # Parquet file format support
                                # - No file locking issues on Windows
                                # - Better compression than HDF5
                                # - Faster read/write operations
```

2.2.3 Machine Learning Libraries

Listing 3: Machine Learning Dependencies

```
# Core ML framework
pip install scikit-learn>=1.3.0    # Primary ML algorithms and tools
    # - StandardScaler, LabelEncoder for preprocessing
    # - mutual_info_classif, SelectKBest for feature selection
    # - train_test_split for data splitting
    # - classification_report, confusion_matrix for evaluation
    # - RandomForestClassifier for ensemble learning
    # - SGDClassifier for incremental learning (large datasets)
    # - MiniBatchKMeans for clustering

# High-performance gradient boosting
pip install xgboost>=1.7.0         # Extreme Gradient Boosting
    # - Best performance on structured data
    # - Handles missing values automatically
    # - Built-in feature importance

pip install lightgbm>=3.3.0        # Light Gradient Boosting Machine
    # - Often faster than XGBoost on CPU
    # - Lower memory usage
    # - Good alternative for large datasets
```

2.2.4 Network Analysis Libraries

Listing 4: Network Processing Dependencies

```
# Primary packet analysis
pip install scapy>=2.5.0       # Comprehensive packet parsing
    # - Read/write PCAP files
    # - Extract packet headers and payloads
    # - Protocol analysis (TCP, UDP, HTTP, etc.)
    # - Network flow reconstruction

# Alternative packet parser
pip install pyshark>=0.6       # Wireshark-based packet analysis
    # - Better protocol support than scapy
    # - More detailed packet dissection
    # - Backup option for complex protocols
```

2.2.5 Visualization Libraries

Listing 5: Visualization Dependencies

```
# Static plotting
pip install matplotlib>=3.7.0      # Basic plotting and visualizations
pip install seaborn>=0.12.0        # Statistical data visualization
    # - Enhanced matplotlib with better defaults
    # - Correlation matrices and heatmaps

# Interactive visualizations
pip install plotly>=5.15.0         # Interactive plots and dashboards
    # - Real-time data monitoring
    # - Interactive confusion matrices
    # - Network topology visualization
    # - Performance dashboards
```

2.2.6 Jupyter and Progress Tracking

Listing 6: Development Environment Dependencies

```
# Jupyter notebook support
pip install ipywidgets>=8.0.0      # Interactive widgets for configuration
pip install jupyter>=1.0.0        # Jupyter notebook environment

# Progress tracking
pip install tqdm>=4.65.0          # Progress bars for long operations
    # - Real-time processing feedback
    # - ETA estimation for long tasks

# System monitoring
pip install psutil>=5.9.0         # System resource monitoring
    # - Memory usage tracking
    # - CPU utilization monitoring
    # - Prevents system overload
```

2.2.7 Parallel Processing

Listing 7: Performance Optimization Dependencies

```
# Parallel execution
pip install joblib>=1.2.0         # Efficient parallel computing
```

```
    # - Multi-core feature extraction
    # - Parallel model training
    # - Memory-efficient processing
```

2.3 Installation Methods

2.3.1 Method 1: Conda Installation (Recommended)

Listing 8: Complete Conda Installation

```
# Create isolated environment with Python 3.12
conda create -n network-analysis python=3.12
conda activate network-analysis

# Install core data science stack
conda install pandas numpy scikit-learn matplotlib seaborn scipy

# Install specialized packages
conda install -c conda-forge lightgbm xgboost pyarrow plotly ipywidgets

# Install network analysis tools
pip install scapy pyshark psutil tqdm joblib

# Verify installation
python -c "import pandas, numpy, sklearn, xgboost, lightgbm; print('All packages
    installed successfully')"
```

2.3.2 Method 2: Pip Installation

Listing 9: Complete Pip Installation

```
# Upgrade pip first
python -m pip install --upgrade pip

# Install all dependencies at once
pip install pandas>=2.0.0 numpy>=1.24.0 scikit-learn>=1.3.0 \
            matplotlib>=3.7.0 seaborn>=0.12.0 scipy>=1.10.0 \
            lightgbm>=3.3.0 xgboost>=1.7.0 pyarrow>=12.0.0 \
            plotly>=5.15.0 ipywidgets>=8.0.0 scapy>=2.5.0 \
            pyshark>=0.6 psutil>=5.9.0 tqdm>=4.65.0 joblib>=1.2.0

# Verify installation
python -c "import pandas, numpy, sklearn, xgboost, lightgbm, scapy; print('Installation
    successful')"
```

2.3.3 Method 3: Requirements File

Create a `requirements.txt` file:

Listing 10: requirements.txt

```
pandas>=2.0.0
numpy>=1.24.0
scikit-learn>=1.3.0
matplotlib>=3.7.0
seaborn>=0.12.0
scipy>=1.10.0
lightgbm>=3.3.0
xgboost>=1.7.0
pyarrow>=12.0.0
```

```
plotly>=5.15.0
ipywidgets>=8.0.0
scapy>=2.5.0
pyshark>=0.6
psutil>=5.9.0
tqdm>=4.65.0
joblib>=1.2.0
jupyter>=1.0.0
```

Then install:

Listing 11: Install from Requirements

```
pip install -r requirements.txt
```

2.4 System Dependencies

2.4.1 Network Packet Capture Tools

Listing 12: System Requirements for Packet Analysis

```
# Linux/Ubuntu
sudo apt-get update
sudo apt-get install tcpdump libpcap-dev wireshark-common

# macOS (with Homebrew)
brew install tcpdump libpcap wireshark

# Windows
# Download and install Wireshark from https://www.wireshark.org/
# This includes WinPcap/Npcap drivers for packet capture
```

Why These Tools Are Required:

- **tcpdump:** Used by Cell 2.5 for creating test datasets from large PCAP files
- **libpcap:** Low-level packet capture library required by both tcpdump and Scapy
- **Wireshark:** Provides packet dissectors used by PyShark for detailed protocol analysis
- **WinPcap/Npcap:** Windows packet capture drivers (included with Wireshark)

Alternative Approach: Cell 2.5 includes a Python fallback using Scapy if tcpdump is not available, but tcpdump provides better performance for large file processing.

2.5 Library Purpose and Usage

2.5.1 Why These Specific Libraries?

Data Processing Stack:

- **Pandas + NumPy:** Handle 4.5M+ network flows efficiently
- **PyArrow:** 3x faster than HDF5, no Windows file locking issues
- **SciPy:** Statistical analysis and feature correlation

Machine Learning Choice:

- **XGBoost:** Best accuracy on network intrusion detection
- **LightGBM:** Faster training, lower memory usage
- **SGDClassifier:** Incremental learning for datasets that don't fit in RAM

- **RandomForest:** Reliable baseline with feature importance

Network Analysis Tools:

- **Scapy:** Primary packet parser, extensive protocol support
- **PyShark:** Backup parser using Wireshark's dissectors
- **Both needed:** Different packets may require different parsers

Performance Optimization:

- **Joblib:** Utilizes all 12 CPU cores for parallel processing
- **PSUtil:** Monitors 31.4 GB RAM to prevent system crashes
- **TQDM:** Real-time progress tracking for 2-3 hour processing

2.6 Memory and Performance Configuration

For your system (31.4 GB RAM, 12 cores), optimal settings:

Listing 13: Optimized Configuration

```
# Memory settings for 31.4 GB system
MAX_MEMORY_GB = 24              # Leave 7.4 GB for OS
BATCH_SIZE = 500000             # Process 500K flows per batch
N_JOBS = 10                     # Use 10/12 cores (leave 2 for OS)

# Parquet compression for faster I/O
COMPRESSION = 'snappy'          # Fast compression/decompression
CHUNK_SIZE = 100000             # Optimal for your RAM size
\end{rstlisting}

\section{Quick Start Options}

\textbf{Video Tutorial:} For a visual walkthrough, watch the Quick Start Guide: \url{
    https://youtu.be/nZu6LL9M7Wc}

\subsection{Option 1: Use Pre-Trained Models (Fastest)}

If you just want to test the system:

\begin{enumerate}
    \item Open the notebook: \texttt{Network\_Threat\_Analysis\_System\_Parquet.ipynb}
    \item \textbf{Run Cell 1 first} to import all required libraries
    \item Run Cell 12 to load pre-trained models from \texttt{Trained\_Models\_For\_Quick
    \_Demo/}
    \item Choose from: RandomForest, LightGBM, SGD, or XGBoost
    \item Takes seconds to load, ready for immediate use
\end{enumerate}

\textbf{Expected Output:}
\begin{itemize}[itemsep=0pt]
    \item Model loading confirmation: "XGBoost model ready for use"
    \item Memory usage information
    \item Ready-to-use classifier for immediate network traffic analysis
\end{itemize}

\subsection{Option 2: Train with Pre-Extracted Features}

If you want to experiment with machine learning:

\begin{enumerate}
    \item \textbf{Run Cell 1 first} to import all required libraries
```

```
    \item Run Cell 13 to load features from \texttt{Features\_For\_Quick\_Demo/}
    \item Train your own models on pre-processed CICIDS2017 data
    \item Takes 15-30 minutes
    \item No need to download the 40 GB dataset or wait 2-3 hours for feature extraction
\end{enumerate}

\textbf{Expected Output:}
\begin{itemize}[itemsep=0pt]
    \item "Loaded 4,500,000 network flows" message
    \item Training progress with accuracy updates
    \item Model performance comparison results across different algorithms
    \item Feature importance rankings
\end{itemize}

\textbf{Pre-extracted features include:} Complete Tuesday-Friday CICIDS2017 data covering
     all 15 attack types including DDoS, PortScan, Web Attacks, Infiltration, and normal
    traffic.

\subsection{Option 3: Full Pipeline}

If you want to process your own PCAP files:

\begin{enumerate}
    \item Get the CICIDS2017 dataset from \url{https://www.unb.ca/cic/datasets/ids-2017.
    html}
    \item \textbf{Run Cell 1 first} to import all required libraries
    \item Run all cells 1-11 for complete processing
    \item Takes about 2-3 hours for the complete 40 GB dataset
\end{enumerate}

\textbf{Expected Output:}
\begin{itemize}[itemsep=0pt]
    \item Real-time processing statistics: "Processing rate: 1,247 flows/sec"
    \item Memory usage monitoring: "Memory usage: 1.8 GB / 8.0 GB"
    \item Feature extraction progress with attack detection counts
    \item Final trained models with comprehensive performance reports
\end{itemize}

\section{Repository Structure}

\begin{lstlisting}
Network-Analysis-Pipeline/
         Network_Threat_Analysis_System_Parquet.ipynb  # Main notebook
         Trained_Models_For_Quick_Demo/                # Ready-to-use models
               RandomForest/
               LightGBM/
               SGD/
               XGBoost/
         Features_For_Quick_Demo/                      # Pre-extracted features
            batch_parquet_files/
            compressed_features_csv/
```

3 Complete Cell Reference Guide

3.1 Core Pipeline Cells

Cell 1: Library Imports

- **Purpose:** Import all required Python libraries
- **Required for:** All other cells

- **Execution time:** 10-15 seconds
- **Output:** "All libraries imported successfully"

Cell 2: System Configuration

- **Purpose:** Interactive configuration UI for memory, paths, and processing settings
- **Required for:** Full pipeline (Cell 11)
- **Execution time:** Immediate (user interface)
- **Output:** Configuration widgets and settings validation

Cell 11: Main Pipeline Execution

- **Purpose:** Complete end-to-end processing of PCAP files and training
- **Dependencies:** Cells 1, 2
- **Execution time:** 2-3 hours for full CICIDS2017 dataset
- **Output:** Trained models, performance reports, visualizations

3.2 Quick Demo Cells

Cell 12: Load Pre-Trained Models

- **Purpose:** Load ready-to-use models for immediate inference
- **Dependencies:** Cell 1
- **Execution time:** Seconds
- **Output:** "XGBoost model ready for use" + model objects

Cell 13: Train with Pre-Extracted Features

- **Purpose:** ML training using pre-processed CICIDS2017 features
- **Dependencies:** Cell 1
- **Execution time:** 15-30 minutes
- **Output:** "Loaded 4,500,000 network flows" + training results

3.3 Optional Utility Cells

Cell 2.5: Create Test Dataset

- **Purpose:** Generate small dataset for quick testing
- **Dependencies:** Cell 1
- **Execution time:** 2-5 minutes
- **Output:** "Created test dataset with X packets"

Cell 14: CSV to Parquet Conversion

- **Purpose:** Convert CSV.gz feature files to Parquet format
- **Dependencies:** Cell 1
- **Execution time:** 2-5 minutes
- **Output:** "Successfully converted X files to Parquet format"

Cell 15: Generate Evaluation Reports

- **Purpose:** Create comprehensive reports from existing ML results
- **Dependencies:** Cell 1, saved ML results
- **Execution time:** 2-5 minutes
- **Output:** Interactive UI + generated report files

3.4 Execution Pathways

Full Pipeline: Cell 1 → Cell 2 → Cell 11 (2-3 hours)
Pre-trained Models: Cell 1 → Cell 12 (seconds)
Feature Training: Cell 1 → Cell 13 (15-30 minutes)
Testing: Cell 1 → Cell 2.5 → Cell 11 (minutes)
Report Generation: Cell 1 → Cell 15 (minutes)

4 How to Use

4.1 Main Notebook Cells

Required Cells:

- **Cell 1:** Import libraries
- **Cell 2:** Configuration
- **Cell 11:** Main processing (if using full pipeline)
- **Cell 12:** Load pre-trained models (quick demo)
- **Cell 13:** Train with pre-extracted features

Optional Utility Cells:

- **Cell 2.5:** Create small test dataset
- **Cell 14:** Convert CSV to Parquet format
- **Cell 15:** Generate evaluation reports from existing results

4.2 Optional Utility Cells Explained

4.2.1 Cell 2.5: Create Small Test Dataset

Purpose: Creates a smaller subset of data for quick testing and development
When to use:

- Testing the pipeline without waiting hours
- Development and debugging
- Learning how the system works
- Validating setup before full processing

What it does:

- Extracts first N packets from PCAP files
- Creates corresponding subset of CSV labels
- Processes in minutes instead of hours
- Validates entire pipeline functionality

Expected Output:

- "Created test dataset with X packets"
- Smaller PCAP and CSV files for testing
- Quick validation that everything works

4.2.2 Cell 14: Convert CSV to Parquet Format

Purpose: Converts compressed CSV feature files to Parquet format for faster ML processing

When to use:

- You have CSV.gz feature files but need Parquet format
- Converting your own extracted features for Cell 13
- Optimizing file format for faster loading
- Working with legacy feature extractions

What it does:

- Reads compressed CSV feature files
- Converts to Parquet format with compression
- Creates batch files suitable for Cell 13
- Improves loading speed for ML training

Expected Output:

- "Converting CSV files to Parquet format..."
- "Converted: features_chunk_000.csv.gz -> batch_0000.parquet"
- "Successfully converted X files to Parquet format"
- Ready-to-use Parquet files for Cell 13

Note: Skip Cell 14 if you're using the pre-made Parquet files in `Features_For_Quick_Demo/batch_parquet_`

4.2.3 Cell 15: Generate Evaluation Reports from Existing Results

Purpose: Create comprehensive analysis reports and visualizations from previously saved ML results without retraining

When to use:

- After completing training runs to generate detailed reports
- Creating publication-quality figures and analysis
- Regenerating reports in different formats
- Performing deeper analysis on existing results
- Creating thesis-ready outputs and summary tables

What it does:

- Loads saved ML results from JSON, Pickle, or CSV files
- Generates comprehensive text reports with detailed analysis
- Creates confusion matrix visualizations for all models
- Produces summary statistics in CSV format
- Supports multiple file formats and flexible data loading
- Provides interactive UI for easy file upload and configuration

Supported Input Files:

- **Metrics Files:** JSON, Pickle (.pkl), CSV containing model performance

- **Feature Files:** Pickle (.pkl), CSV, or TXT files with feature names
- **Confusion Matrices:** JSON or Pickle files with confusion matrix data

Generated Outputs:

- **Comprehensive Report:** Detailed text analysis with model comparisons
- **Confusion Matrices:** High-resolution PNG visualizations for each model
- **Summary Statistics:** CSV files with all performance metrics
- **Comparative Analysis:** Model ranking and performance tables

Expected Output:

- Interactive file upload interface with progress indicators
- "Files loaded successfully! Ready to generate reports."
- Generated report files in specified output directory
- "Successfully generated X report(s)" confirmation message

4.3 Using Pre-Trained Models

Listing 14: Loading Pre-Trained Models in Cell 12

```
# In Cell 12
import pickle

# Choose algorithm: "RandomForest", "LightGBM", "SGD", "XGBoost"
algorithm = "XGBoost"
model_path = f'./Trained_Models_For_Quick_Demo/{algorithm}/'

# Load model
with open(f'{model_path}xgboost_incremental.pkl', 'rb') as f:
    model = pickle.load(f)

# Load scaler and features
with open(f'{model_path}scaler.pkl', 'rb') as f:
    scaler = pickle.load(f)

print(f"{algorithm} model ready for use")
```

4.4 Using Pre-Extracted Features

Listing 15: Loading Pre-Extracted Features in Cell 13

```
# In Cell 13
import pandas as pd
import glob

# Load all feature batches
batch_path = './Features_For_Quick_Demo/batch_parquet_files/'
batch_files = glob.glob(f'{batch_path}batch_*.parquet')
features = pd.concat([pd.read_parquet(f) for f in batch_files])

print(f"Loaded {len(features)} network flows")
print("Ready for ML training")
```

5 What the System Detects

The system can identify these attack types:

- Normal traffic (BENIGN)
- DDoS attacks
- Port scanning
- Botnet activity
- Web attacks (SQL injection, XSS, brute force)
- DoS attacks (various types)
- Brute force attacks (FTP, SSH)
- Heartbleed exploit
- Network infiltration

6 Performance

- Processes 4.5 million network flows
- XGBoost gives best results
- Uses 8GB RAM with streaming processing
- Takes about 2 hours for complete pipeline
- Pre-trained models work immediately

7 Troubleshooting

Memory errors: Reduce memory settings in Cell 2 configuration
Missing files: Make sure PCAP and CSV files are in correct directories
Slow performance: Use pre-extracted features instead of full pipeline
Installation issues: Use conda instead of pip for better package management

8 CICIDS2017 Dataset

If using the full pipeline, you need the CICIDS2017 dataset:

- Download from: https://www.unb.ca/cic/datasets/ids-2017.html
- Size: About 40 GB total
- Files needed: Tuesday-Friday PCAP files and CSV labels
- May require institutional access

8.1 Pre-Extracted Features Source Data

The pre-extracted features in `Features_For_Quick_Demo/` were processed from these specific CICIDS2017 files:

PCAP Files (4 days):

- Tuesday-WorkingHours.pcap
- Wednesday-workingHours.pcap
- Thursday-WorkingHours.pcap

- Friday-WorkingHours.pcap

Ground Truth CSV Files (7 files):

- Tuesday-WorkingHours.pcap_ISCX.csv
- Wednesday-workingHours.pcap_ISCX.csv
- Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv
- Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv
- Friday-WorkingHours-Morning.pcap_ISCX.csv
- Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv
- Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv

This represents the complete Tuesday-Friday CICIDS2017 dataset covering all 15 attack types and normal traffic patterns.

File structure needed:

```
./data/CICIDS2017/Raw_PCAP/
         Tuesday-WorkingHours.pcap
         Wednesday-workingHours.pcap
         Thursday-WorkingHours.pcap
         Friday-WorkingHours.pcap

./data/CICIDS2017/TrafficLabelling/
         Tuesday-WorkingHours.pcap_ISCX.csv
         Wednesday-workingHours.pcap_ISCX.csv
         [additional CSV files for Thursday/Friday]
```

Note: This system is for educational and research purposes. Make sure you have permission to analyze network traffic data.