Configuration Manual

MSc Research Project
Data Analytics

Durga Prasad Reddy Mutra
Student ID: 23107987

School of Computing
National College of Ireland

Supervisor: Jaswinder Singh

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Durga Prasad Reddy Mutra		
Student ID:	23107987		
Programme:	MSc Data Analytics	**Year:**	2025
Module:	MSc Research Project		
Lecturer:	Jaswinder Singh		
Submission Due Date:	28/1/2025		
Project Title:	Enhancing IoT Network Traffic Anomaly Detection with GANs		
Word Count:	813	**Page Count:**	8

Signature: Durga Prasad

Date: 28/1/2025

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

Durga Prasad Reddy Mutra
23107987

1 Introduction

This configuration manual provides comprehensive guidance for setting up and implementing the IoT Network Traffic Anomaly Detection System using Generative Adversarial Networks (GANs). The system is designed to detect network traffic anomalies with high accuracy while maintaining low false positive rates.

1.1 Purpose

The purpose of this manual is to guide users through the configuration and implementation of the anomaly detection system, ensuring optimal performance in IoT network security environments.

1.2 Scope

This manual covers the complete configuration process, from initial setup to advanced tuning of the GAN architecture, focusing on practical implementation aspects.

2 System Requirements

2.1 Hardware Requirements

- Minimum 16GB RAM recommended
- Multi-core processor
- GPU support recommended for optimal performance

2.2 Software Dependencies

- Python Data Science ecosystem
- TensorFlow/Keras framework
- Pandas for data handling
- NumPy for numerical computations
- Seaborn, Matplotlib, and Plotly Express for visualizations

3 Data Configuration

3.1 Dataset Configuration

- Dataset: NF-BoT-IoT format
- Minimum recommended samples: 600,000 network flows
- Required data distribution: 97.69% attack samples, 2.31% benign samples before balancing

3.2 Feature Requirements

- IPv4 source and destination addresses
- Layer 4 source and destination ports
- Protocol information
- Layer 7 protocol data
- Input/output bytes and packets
- TCP flags
- Flow duration in milliseconds

4 Model Architecture Configuration

4.1 Generator Network Settings

- Input dimension: 100 (latent space)
- First dense layer: 64 units with LeakyReLU activation (alpha=0.2)
- Second dense layer: 128 units with LeakyReLU activation (alpha=0.2)
- Output layer: Matched to input dimensions with tanh activation
- Batch normalization: After each dense layer except output

4.2 Discriminator Network Settings

- Input layer: Matched to network flow dimensions
- First dense layer: 128 units with LeakyReLU activation
- Second dense layer: 64 units with LeakyReLU activation
- Output layer: Single unit with sigmoid activation
- Batch normalization: Implemented after dense layers

4.3 Optimization Parameters

- Optimizer: Adam
- Learning rate: 0.0002
- Beta values: (0.5, 0.9)
- Batch size: 64
- Training epochs: Minimum 1000 recommended

5 Feature Processing Configuration

5.1 IP Address Processing

- Conversion method: Numeric representation using ipaddress library
- Preservation of hierarchical structure required
- Both source and destination IPs must be processed

5.2 Port Number Processing

- Implementation of frequency-based encoding
- Preservation of port usage patterns
- Separate processing for source and destination ports

5.3 Protocol Information

- Layer 4 protocol encoding
- Layer 7 protocol encoding
- Frequency-based transformation required

6 Training Configuration

6.1 Data Sampling Settings

- Balanced sampling approach
- Ratio: 1:1 (attack:benign)
- Random state: Fixed for reproducibility
- Minimum sample size per category based on least represented class

6.2 Training Process Parameters

- Batch size: 64
- Alternating updates between generator and discriminator
- Loss tracking for both networks
- Regular performance evaluation intervals

7 Performance Monitoring

7.1 Metrics Configuration

- Accuracy tracking
- False positive rate monitoring
- Confusion matrix analysis
- Generator and discriminator loss tracking

7.2 Visualization Settings

- Feature distribution histograms: 15 bins
- Box plots for numerical features
- Training progress plots
- Confusion matrix heatmaps

8 Inference Setup and Configuration

8.1 Directory Structure

```
ROOT_DIR: C:\Users\durga\Downloads\GAN\
|-- Models
|   |-- gan_generator.h5
|   |-- gan_discriminator.h5
|   |-- gan_combined.h5
|-- Data
|   |-- gan_data.npz
```

8.2 Model Loading Steps

8.2.1 Step 1: Load Required Libraries

```
from tensorflow.keras.models import load_model
import numpy as np
```

8.2.2 Step 2: Configure Model Paths

```
generator_path = "C:\Users\durga\Downloads\GAN\gan_generator.h5"
discriminator_path = "C:\Users\durga\Downloads\GAN\gan_discriminator.h5"
data_path = "C:\Users\durga\Downloads\GAN\gan_data.npz"
```

8.2.3 Step 3: Load Models

- Load generator model
- Load discriminator model
- Verify successful loading

8.2.4 Step 4: Load Training Data

- Load NPZ data file
- Extract X_train and y_train
- Verify data dimensions

8.3 Running Inference

8.3.1 Step 1: Generate Random Noise

- Set latent dimension (100)
- Generate normal distribution noise
- Configure number of samples

8.3.2 Step 2: Generate Fake Data

- Use generator model

- Process noise input
- Create synthetic samples

8.3.3 Step 3: Evaluate Results

- Use discriminator
- Compare real vs generated data
- Calculate performance metrics

8.4 Evaluation Process

8.4.1 Step 1: Prepare Data

- Generate fake samples
- Prepare real data (attacked and benign)
- Set up labels

8.4.2 Step 2: Run Predictions

- Process real data through discriminator
- Process fake data through discriminator
- Apply threshold (0.5)

8.4.3 Step 3: Calculate Metrics

- Real data accuracy
- Fake data accuracy
- Generate confusion matrices

8.4.4 Step 4: Visualize Results

- Plot confusion matrices
- Display accuracy scores
- Show performance metrics

8.5 Expected Results

8.5.1 Performance Metrics

- Discriminator accuracy on real data
- Discriminator accuracy on fake data
- False positive/negative rates

8.5.2 Visualization Outputs

- Confusion matrix for real data
- Confusion matrix for fake data
- Performance plots

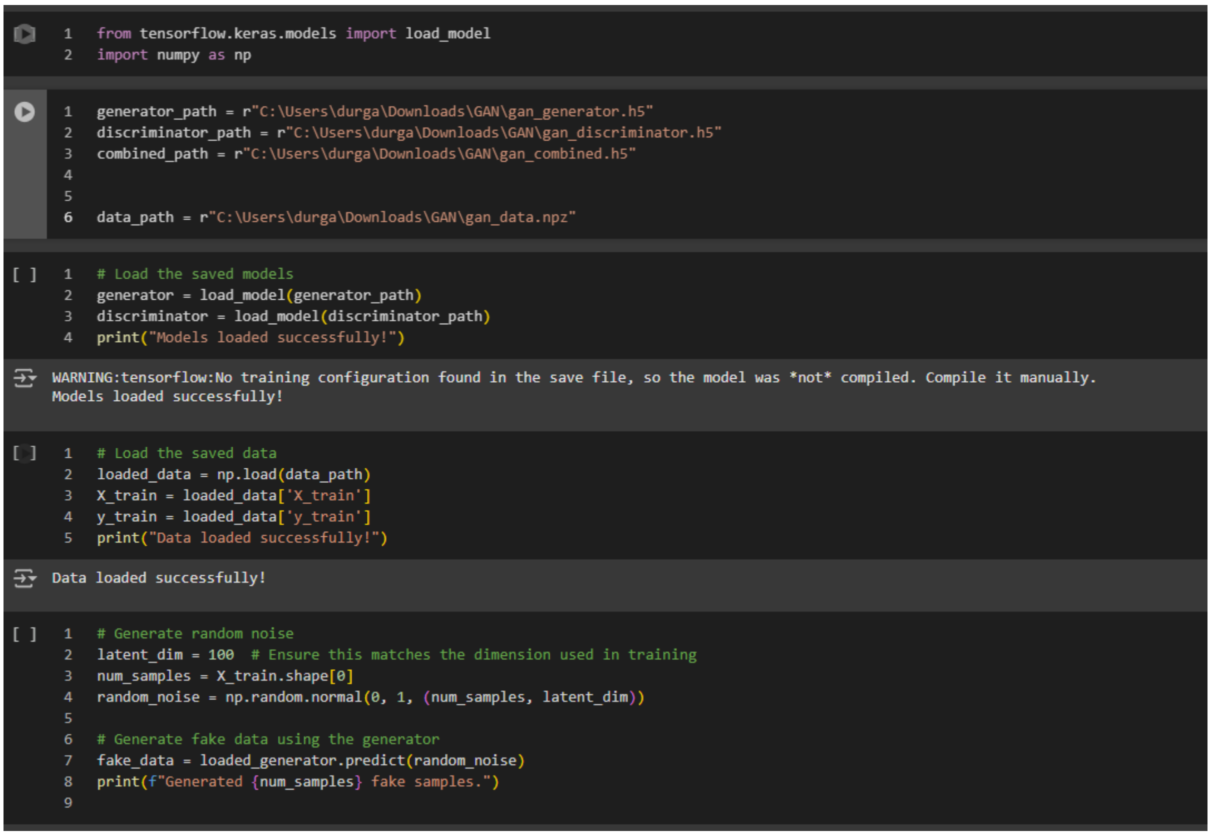
8.6 Troubleshooting

8.6.1 Common Issues

1. Model Loading Errors
    - Check file paths

- Verify model format
- Confirm file permissions

2. Memory Issues
   - Monitor RAM usage
   - Reduce batch size
   - Clear unused variables



```python
from tensorflow.keras.models import load_model
import numpy as np
```

```python
generator_path = r"C:\Users\durga\Downloads\GAN\gan_generator.h5"
discriminator_path = r"C:\Users\durga\Downloads\GAN\gan_discriminator.h5"
combined_path = r"C:\Users\durga\Downloads\GAN\gan_combined.h5"


data_path = r"C:\Users\durga\Downloads\GAN\gan_data.npz"
```

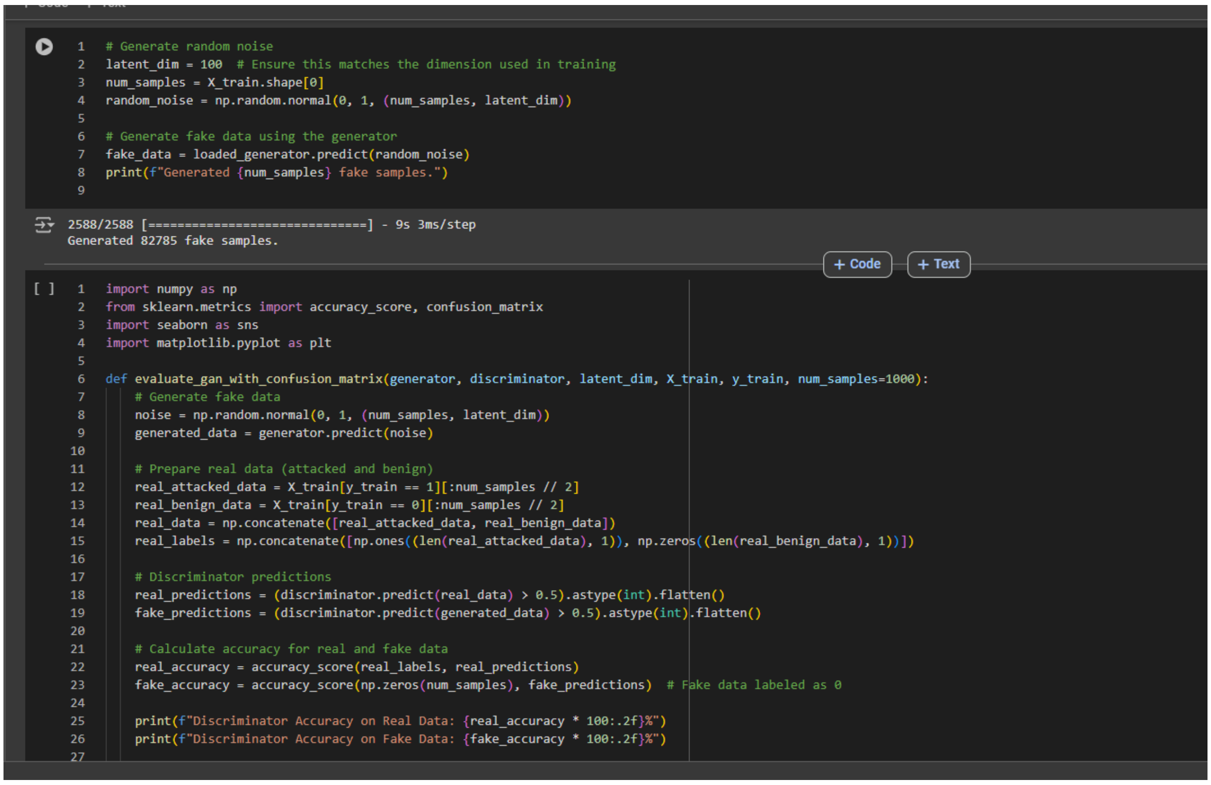
```python
# Load the saved models
generator = load_model(generator_path)
discriminator = load_model(discriminator_path)
print("Models loaded successfully!")
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
Models loaded successfully!
```

```python
# Load the saved data
loaded_data = np.load(data_path)
X_train = loaded_data['X_train']
y_train = loaded_data['y_train']
print("Data loaded successfully!")
```

```
Data loaded successfully!
```

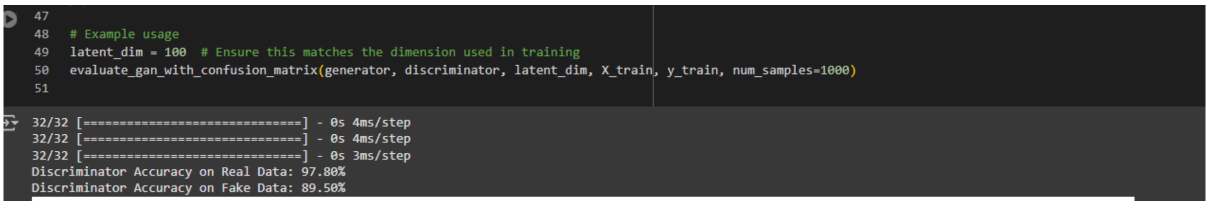
```python
# Generate random noise
latent_dim = 100  # Ensure this matches the dimension used in training
num_samples = X_train.shape[0]
random_noise = np.random.normal(0, 1, (num_samples, latent_dim))

# Generate fake data using the generator
fake_data = loaded_generator.predict(random_noise)
print(f"Generated {num_samples} fake samples.")

```



```
# Generate random noise
latent_dim = 100  # Ensure this matches the dimension used in training
num_samples = X_train.shape[0]
random_noise = np.random.normal(0, 1, (num_samples, latent_dim))

# Generate fake data using the generator
fake_data = loaded_generator.predict(random_noise)
print(f"Generated {num_samples} fake samples.")

```

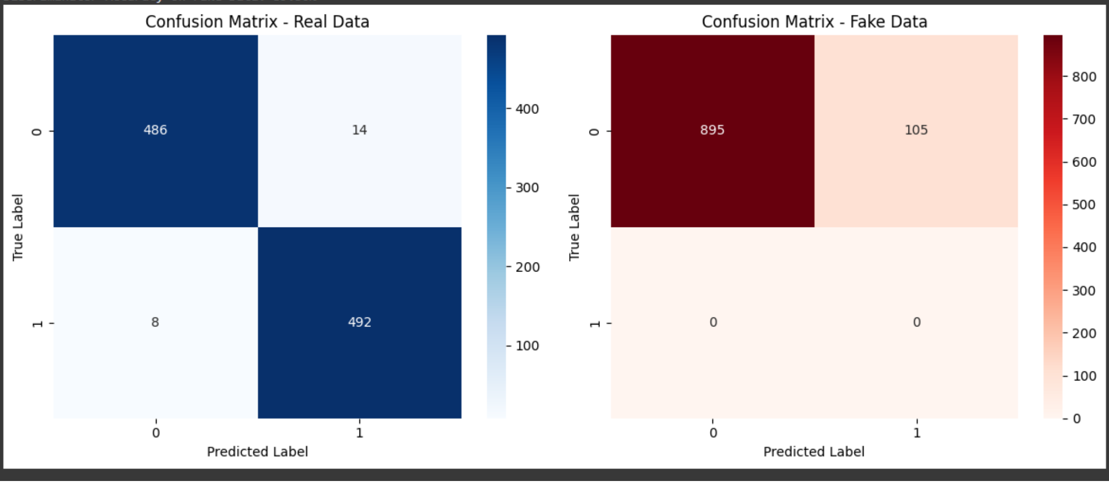
```
2588/2588 [==============================] - 9s 3ms/step
Generated 82785 fake samples.
```

```
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

def evaluate_gan_with_confusion_matrix(generator, discriminator, latent_dim, X_train, y_train, num_samples=1000):
    # Generate fake data
    noise = np.random.normal(0, 1, (num_samples, latent_dim))
    generated_data = generator.predict(noise)

    # Prepare real data (attacked and benign)
    real_attacked_data = X_train[y_train == 1][:num_samples // 2]
    real_benign_data = X_train[y_train == 0][:num_samples // 2]
    real_data = np.concatenate([real_attacked_data, real_benign_data])
    real_labels = np.concatenate([np.ones((len(real_attacked_data), 1)), np.zeros((len(real_benign_data), 1))])

    # Discriminator predictions
    real_predictions = (discriminator.predict(real_data) > 0.5).astype(int).flatten()
    fake_predictions = (discriminator.predict(generated_data) > 0.5).astype(int).flatten()

    # Calculate accuracy for real and fake data
    real_accuracy = accuracy_score(real_labels, real_predictions)
    fake_accuracy = accuracy_score(np.zeros(num_samples), fake_predictions)  # Fake data labeled as 0

    print(f"Discriminator Accuracy on Real Data: {real_accuracy * 100:.2f}%")
    print(f"Discriminator Accuracy on Fake Data: {fake_accuracy * 100:.2f}%")
```



```

# Example usage
latent_dim = 100  # Ensure this matches the dimension used in training
evaluate_gan_with_confusion_matrix(generator, discriminator, latent_dim, X_train, y_train, num_samples=1000)

```

```
32/32 [==============================] - 0s 4ms/step
32/32 [==============================] - 0s 4ms/step
32/32 [==============================] - 0s 3ms/step
Discriminator Accuracy on Real Data: 97.80%
Discriminator Accuracy on Fake Data: 89.50%
```



9 References

The SAI. (n.d.). A GAN-based hybrid deep learning approach for enhancing IoT security. Retrieved from https://thesai.org/Downloads/Volume15No6/Paper_37-A_GAN_based_Hybrid_Deep_Learning_Approach.pdf

MDPI. (2023, September 27). Generative adversarial network-based anomaly detection and forecasting with unlabeled data for 5G vertical applications. Retrieved from https://www.mdpi.com/2076-3417/13/19/10745

EM360. (n.d.). Generative adversarial networks (GANs) in cybersecurity. Retrieved from https://em360tech.com/tech-article/generative-adversarial-networks-gans-cybersecurity-new-frontier-threat-simulation

Restack. (2024, August 11). Best practices for training GANs. Retrieved from https://www.restack.io/p/adversarial-networks-answer-best-practices-training-gans-cat-ai

Frontiers in Plant Science. (2022, June 6). Anomaly detection for Internet of Things time series data using GAN. Retrieved from https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2022.890563/full