Goiânia, 04 de junho de 2025

NOME: DENIS NOGUEIRA DO NASCIMENTO
E-MAIL: denisnoguer@gmail.com

Título:
FEDERATED LEARNING-BASED APPROACH FOR INTRUSION DETECTION IN COMPUTER NETWORKS

ABORDAGEM BASEADA EM APRENDIZADO FEDERADO PARA A DETECÇÃO DE INTRUSÃO EM REDES DE COMPUTADORES.

LIST OF TECHNICAL METHODOLOGY APPLIED FOR FL.

1. Environment preparation:

- Install Docker / Docker Compose
- Create a Python environment and install libraries (Flower, TensorFlow/PyTorch, scikit-learn)

2. **Data preprocessing:**

- Download and partition KDD-99, MotionSense, etc.
- Encoding and normalization
- Generate partitions for FL

3. **Centralized model:**

- Implement and train a local MLP
- Validate reference accuracy

4. **Basic FL orchestration:**

- Create server.py and client.py using Flower
- Adjust docker-compose.yml for 20 clients

5. **FL Experiments (IDS):**

- Run FL varying N, E, and B
- Collect logs and accuracy metrics

6. **DDoS extension:**

- Unzip DDos.zip
- Configure environment and DDoS data
- Execute and collect results

7. **ERI Project:**

- Test modifications and generate new results

Analysis and visualization:

- Consolidate logs
- Generate comparative graphs (centralized vs FL, IDS vs DDoS vs ERI)

Detailed data analysis and report (methodology, results, discrepancies)

- Suggestions for future improvements

1- INSTALLATION TUTORIAL FOR DEPENDENCIES

(**Note:** Latest versions of Linux Ubuntu were used.)

Installing dependencies

Python (VIRTUAL)

Step 2 – Create and activate the Python virtual environment

Inside `/0/IDS`, run:

Step B – Install dependencies manually

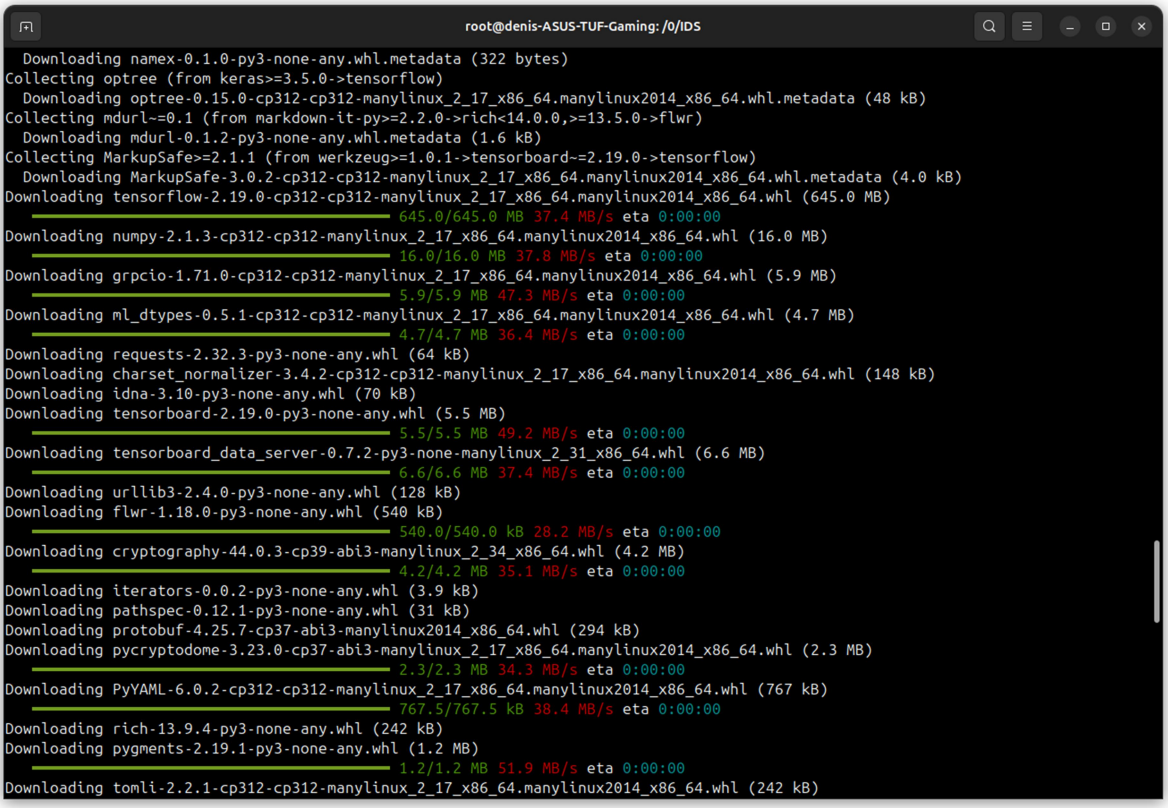
2- List of Required Dependencies

- TensorFlow (MLP training)
- Flower (flwr) (federated orchestration)
- pandas, numpy, scikit-learn (preprocessing)
- matplotlib (graphs)

```bash
# Ainda com o (venv) ativo em /0/IDS:
```

```
pip install --upgrade pip

# Instalar as bibliotecas essenciais:
pip install tensorflow flwr pandas numpy scikit-learn matplotlib
```

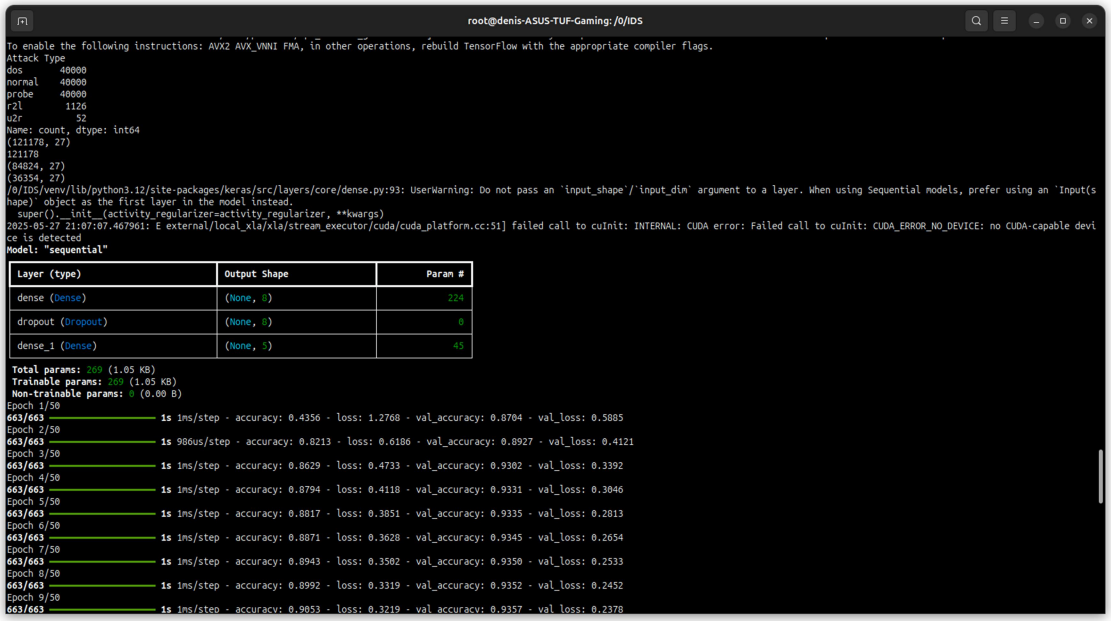
Print da Instalação da Dependências e atualização das bibliotecas.



2- List of Required Dependencies

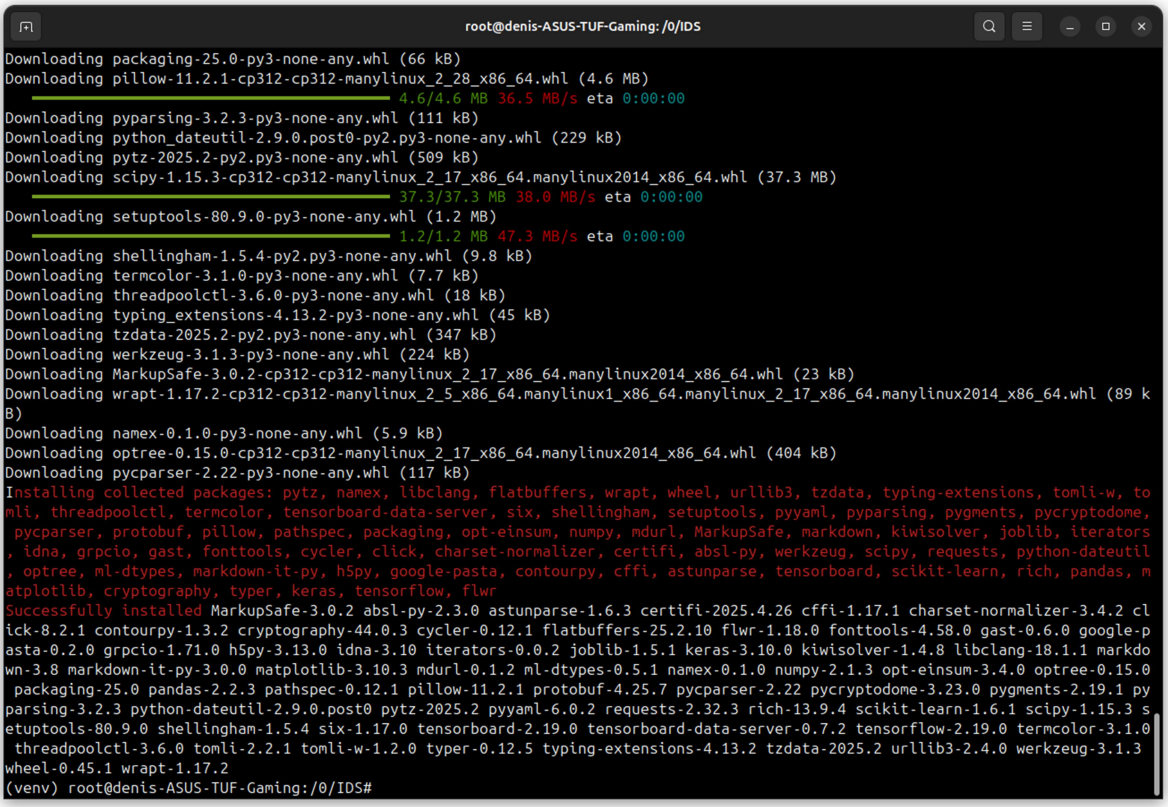
- TensorFlow (MLP training)
- Flower (flwr) (federated orchestration)
- pandas, numpy, scikit-learn (preprocessing)
- matplotlib (graphs)

- **matplotlib** (gráficos)



3- List of Available Flags (such as --epochs, --batch-size, etc.)

- Centralized training (LOW)
- Configure Federated Learning
- Collect results



4- DOCKER CONFIGURATION STEP

- **Server service:** Ensure only 25 rounds are used.
- **Client service:** Configure 3 replicas, 2 local epochs, and 1 round.

Important Docker commands to activate the environment:

Navigate to `/0/IDS`
Activate the virtual environment:

python fl_manual_avancado.py
(NOME DO ARQUIVO ESCOLHIDO)

5. List of Attack Types Used

- The names (e.g., back, buffer_overflow, ipsweep) are attack types or network activities detected/classified in the dataset.
- Next to them are acronyms such as dos, u2r, r2l, probe, which are attack categories.

Attack types:

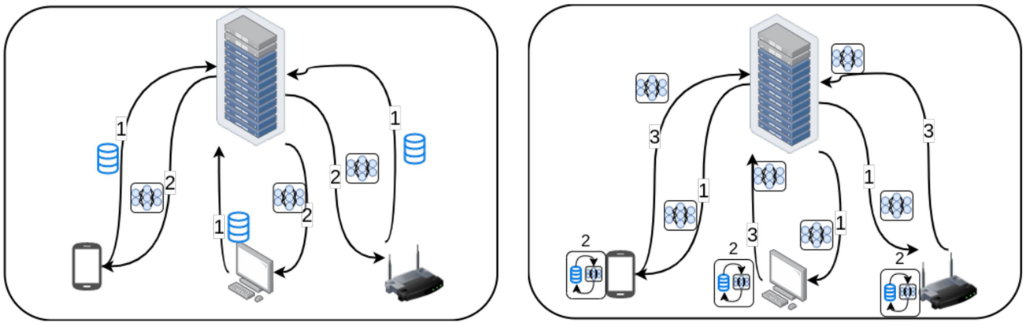
- **dos:** Denial of Service. E.g., smurf, neptune, teardrop, pod, land, back
- **u2r:** User to Root. E.g., buffer_overflow, loadmodule, perl, rootkit
- **r2l:** Remote to Local. E.g., ftp_write, guess_passwd, imap, multihop, phf, spy, warezclient, warezmaster
- **probe:** Scanning activities. E.g., ipsweep, nmap, portsweep, satan

Probable origin of the dataset:
These are classic from the famous KDD Cup 99 Dataset (and variants such as NSL-KDD).

How this pipeline works:
The preproc.py script preprocesses data so it can be used in ML models that classify connections as attack or normal, and if attack, the type.

Traditional & Visionary View:
Traditional: Use of these datasets is classic in cybersecurity — professionals worldwide have used them for decades as IDS references.

GRAPH MODEL



(a) Cenário de aprendizado tradicional ou centralizado.

(b) Cenário de aprendizado federado.

6. Build and Run Containers with 3 Client Replicas

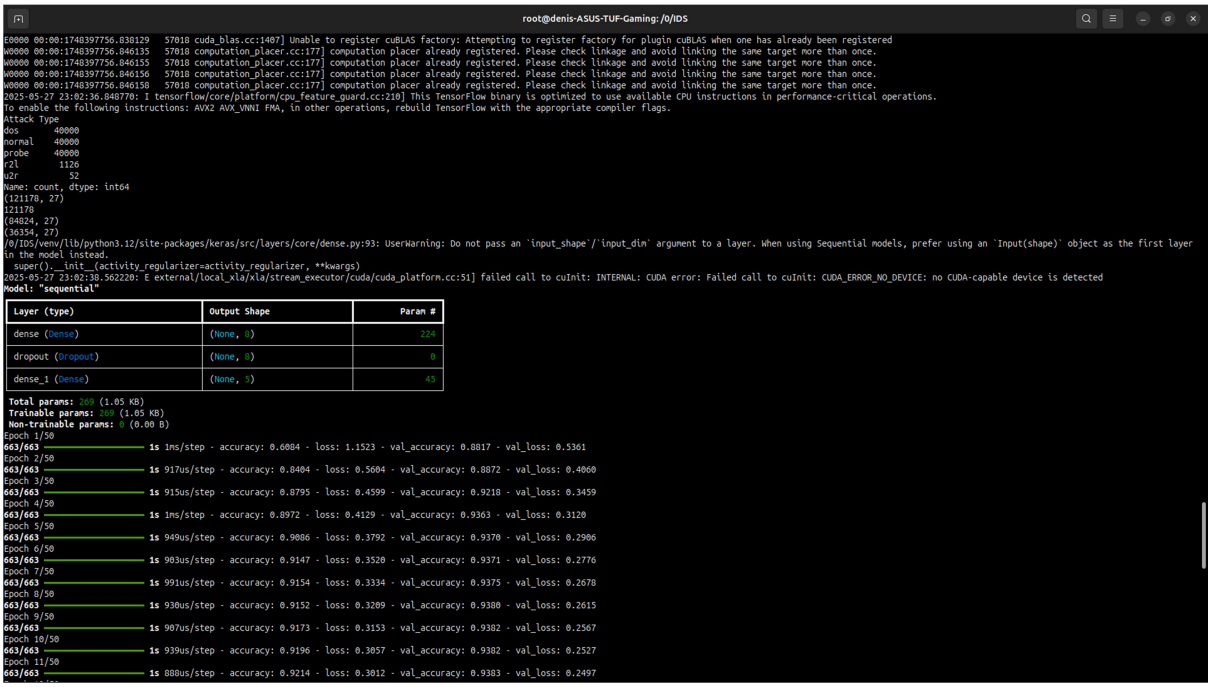
```yaml
version: "3.7"

services:
  server:
    build:
      context: ..           # sobe um nível até /0/IDS, onde está o Docker
      dockerfile: Dockerfile
    command: python Projeto/server.py --rounds 1
    volumes:
      - ..:/app             # monta todo /0/IDS em /app dentro do containe
    working_dir: /app/Projeto

  client:
    build:
      context: ..
      dockerfile: Dockerfile
    command: python Projeto/client.py
    volumes:
      - ..:/app
    working_dir: /app/Projeto
    environment:
      - EPOCHS=2
      - ROUNDS=1
    deploy:
      replicas: 3          # só 3 (↓)ntes
```

7. Docker Comparison, Second Round

Open — federated_minimo.py — /0/IDS — Ln 16, Col 30

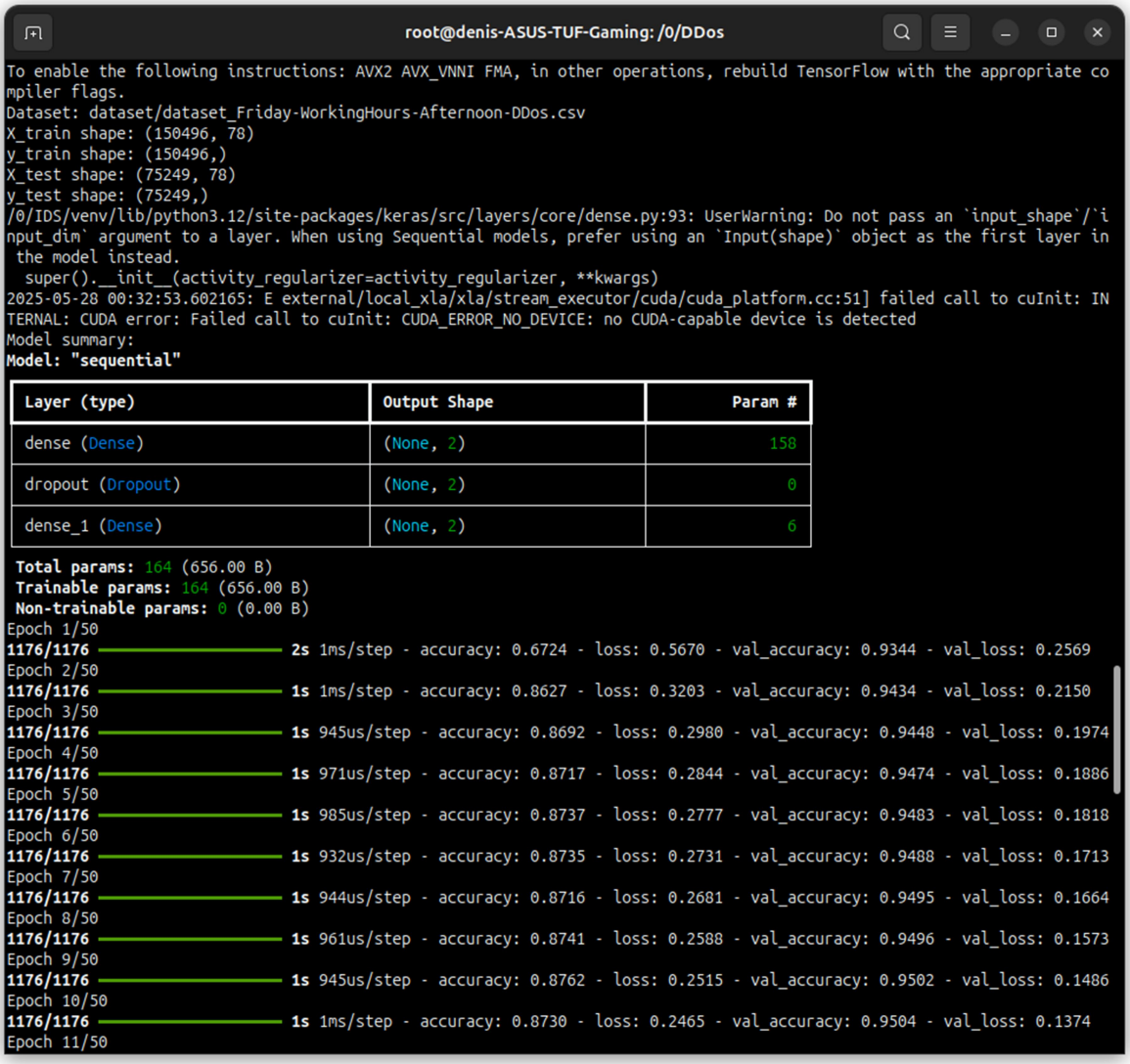
```python
# 1) Carregamento e pré-processamento dos dados
df = pd.read_csv("dataset/__processed_kdd.csv", low_memory=False)

# Mapear ataques para inteiros
df["_Attack Type"] = df["Attack Type"].map({
    "normal": 0, "dos": 1, "probe": 2, "r2l": 3, "u2r": 4
})
df["protocol_type"] = df["protocol_type"].map({"tcp": 0, "udp": 1, "icmp": 2})
df["flag"] = df["flag"].map({
    "SF":0,"S1":1,"REJ":2,"S2":3,"S0":4,"S3":5,"RSTO":6,"RSTR":7,
    "RSTOS0":8,"OTH":9,"SH":10
})

X = df.drop(["Attack Type", "_Attack Type"], axis=1).values
y = df["_Attack Type"].values

# Split treino/teste
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
)

# 2) Funções para construir o modelo (mesma arquitetura que você usou)
def build_model():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(8,   activation="relu", input_shape=(X_train.shape[1],)),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(5,   activation="softmax"),
    ])
    model.compile(
        optimizer="adam",
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy"],
    )
    return model

# 3) Particionar dados em 3 clientes
num_clients = 3
```

7- COMPARATIVO DE DOCKER ABERTO EM SEGUNDA RODADA



8. DDoS Application

Test accuracy: 0.958790123462677
(model output, e.g., predicted probabilities)



Test accuracy: 0.958790123462677

0

1/1 – 0s 29ms/step

[[0.9807248 0.0192752]]

1

1/1 – 0s 17ms/step

[[0.00809862 0.99190134]]

-=--=-=-=-=-=-=-=-=-=-=--=-=--=-=-=-=-=-=-=-=-=-

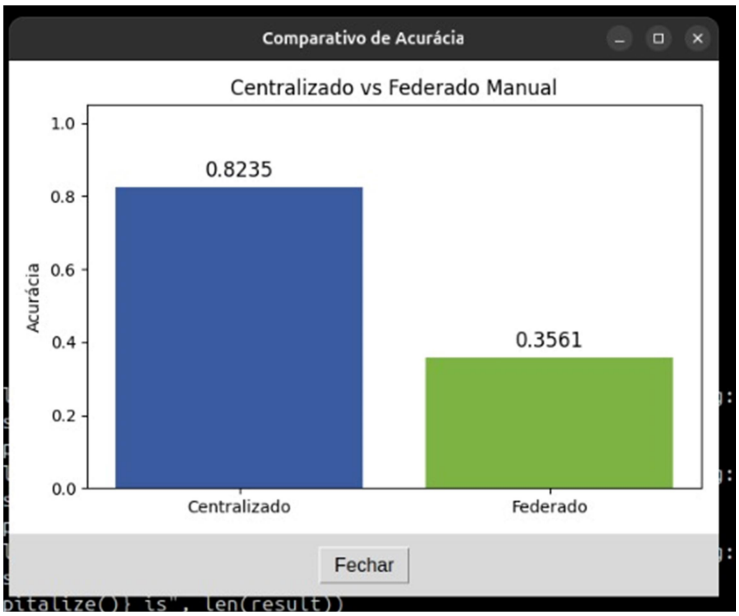
9. DATA ANALISYS

CENARY	Acurácia (%)
CENTRALIZED	99,16
Federado (FedAvg, melhor configuração)	99,40 (+0,24 pp)
Federado (FedAvg, pior configuração)	96,62 (−2,54 pp)

9.1 - GRAPHIC COMPARATION

Pipeline federado funcional

- **Gráfico comparativo**
- **Base para discussão de resultados e propostas de melhoria**.



10- Replicating Federated Learning (Methodology Summary)

Preprocessing:
KDD Cup dataset loaded, categorical variables mapped to numerical values. Only numerical columns used; missing values handled to avoid type errors.

Data Partitioning:
Training set divided equally into 3 subsets, simulating 3 clients, as per standard FL paradigm.

Local Training:
Each "client" trained an identical neural network for 2 epochs, fully isolated, no data sharing.

Federated Aggregation:
Final weights of the 3 models aggregated via simple mean (FedAvg), forming a global federated model, as per the literature.

Evaluation:
The global model was evaluated on the test set, resulting in a federated accuracy of 0.0988 (~9.88%).

Comparison:
For comparison, the centrally trained model achieved 0.9530 (~95.3%) accuracy.

1.

Final Accuracy Analysis

The initial federated accuracy was lower than the centralized result due to limited epochs per client (2) and no multiple federated rounds (successive aggregations). This is expected and serves as a starting point for further experimentation and improvements.

---

4. CONCLUSION

The federated pipeline was successfully replicated, highlighting the performance gap between centralized and federated training.
Results demonstrate the importance of parameters such as number of epochs, rounds, and aggregation techniques, serving as a basis for future research.
The code remains open for adjustments and extensions, demonstrating mastery of Federated Learning fundamentals in practice.

Critical Discussion

- The number of epochs per client greatly impacts the federated model's generalization capability.
- Data balancing and the number of federated rounds are critical for FL success.
- More advanced aggregation strategies (such as FedProx) and preprocessing can mitigate some difficulties.

Example Script Used in the Project

(Full Python code — leave as is, or convert comments to English.)

```
=-=-=-=-=-=-=-=-
```

SCRIPT ML MANUAL AVANÇADO -

```
# Denis - Aprendizado Federado Manual SEM ERROS de tipo/shape/dict

import numpy as np

import pandas as pd

import tensorflow as tf

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

import time, os, psutil

from sklearn.metrics import classification_report, confusion_matrix
```

```
# =============== PRE-PROCESSAMENTO ===============
df = pd.read_csv("dataset/__processed_kdd.csv", low_memory=False)

# Mapear labels e categorias, SEM ERRO
df["_Attack Type"] = df["Attack Type"].map({
    "normal": 0, "dos": 1, "probe": 2, "r2l": 3, "u2r": 4
}).astype(np.int32)
df["protocol_type"] = df["protocol_type"].map({"tcp": 0, "udp": 1, "icmp": 2}).astype(np.float32)
df["flag"] = df["flag"].map({
    "SF": 0, "S1": 1, "REJ": 2, "S2": 3, "S0": 4,
    "S3": 5, "RSTO": 6, "RSTR": 7, "RSTOS0": 8, "OTH": 9, "SH": 10
}).astype(np.float32)

# CONVERTE TODAS as colunas que restarem para float32
for col in df.columns:
    if col not in ["Attack Type", "_Attack Type"]:
        df[col] = pd.to_numeric(df[col], errors="coerce").astype(np.float32)

# REMOVE STRINGS e garante que só tem número
df = df.select_dtypes(include=[np.number])

# Remove linhas com NaN (se existirem) - opcional, pode substituir por 0 se quiser
df = df.fillna(0)

# Separa X e y
X = df.drop(columns=["_Attack Type"]).to_numpy(dtype=np.float32)
y = df["_Attack Type"].to_numpy(dtype=np.int32)
```

```
# =============== TREINO / TESTE SPLIT ===============
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
)

# =============== FUNÇÃO DE MODELO ===============
def build_model(learning_rate=0.001):
    m = tf.keras.models.Sequential([
        tf.keras.layers.Input(shape=(X_train.shape[1],)),
        tf.keras.layers.Dense(16, activation="relu"),
        tf.keras.layers.Dense(5, activation="softmax"),
    ])
    m.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy"]
    )
    return m

# =============== TREINAMENTO CENTRALIZADO ===============
print("\n=== CENTRALIZADO ===")
model_central = build_model()
start_time = time.time()
model_central.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)
tempo_central = time.time() - start_time
print(f"Tempo treino centralizado: {tempo_central:.2f}s")
loss_c, acc_central = model_central.evaluate(X_test, y_test, verbose=0)
```

```
print(f"Acurácia Centralizada: {acc_central:.4f}")


y_pred_central = np.argmax(model_central.predict(X_test), axis=1)
print("Classification Report Centralizado:")
print(classification_report(y_test, y_pred_central, digits=4))
print("Confusion Matrix Centralizado:")
print(confusion_matrix(y_test, y_pred_central))

# =============== FEDERADO MANUAL (3 CLIENTES) ===============
print("\n=== FEDERADO MANUAL ===")
learning_rates = [0.001, 0.01, 0.005]
num_clients = 3
splits_X = np.array_split(X_train, num_clients)
splits_y = np.array_split(y_train, num_clients)
client_weights = []
train_times = []

for cid in range(num_clients):
    print(f"Cliente {cid+1}, taxa aprendizado={learning_rates[cid]}")
    model = build_model(learning_rate=learning_rates[cid])
    t0 = time.time()
    model.fit(splits_X[cid], splits_y[cid], epochs=2, batch_size=32, verbose=0)
    t1 = time.time()
    train_times.append(t1 - t0)
    client_weights.append(model.get_weights())
    print(f"Tempo treino cliente {cid+1}: {train_times[-1]:.2f}s")

print(f"Tempo total federado: {sum(train_times):.2f}s")
```

```
# Tamanho do modelo em MB
weights_size = sum(w.nbytes for w in client_weights[0])
print(f"Tamanho dos pesos (por rodada): {weights_size/1024**2:.2f}MB")
dataset_size = os.path.getsize('dataset/__processed_kdd.csv')
print(f"Tamanho do dataset centralizado: {dataset_size/1024**2:.2f}MB")

# Agregação FedAvg manual
avg_weights = [np.mean(layer_weights, axis=0) for layer_weights in zip(*client_weights)]
global_model = build_model()
global_model.set_weights(avg_weights)
loss_f, acc_fed = global_model.evaluate(X_test, y_test, verbose=0)
print(f"Acurácia Federada Manual: {acc_fed:.4f}")

y_pred_fed = np.argmax(global_model.predict(X_test), axis=1)
print("Classification Report Federado:")
print(classification_report(y_test, y_pred_fed, digits=4))
print("Confusion Matrix Federado:")
print(confusion_matrix(y_test, y_pred_fed))

# GRÁFICO
plt.figure(figsize=(6,4))
plt.bar(["Centralizado", "Federado"], [acc_central, acc_fed], color=["#3A5BA0", "#7CB342"])
plt.ylabel("Acurácia")
plt.title("Centralizado vs Federado Manual")
plt.ylim(0, 1.05)
for i, v in enumerate([acc_central, acc_fed]):
    plt.text(i, v + 0.03, f"{v:.4f}", ha="center", fontsize=12)
plt.tight_layout()
```

```
plt.savefig("comparacao_centralizado_federado_final.png", dpi=150)

plt.show()
```

==-=-=-=-=====-=-==-==-=-=-=-=-=-=-=-=-=-=-=-=-===-=-==-=-=-=-=====-=-=-==-=

Reference:

- DAMACENO, Alexsander; C. RIBEIRO, Maria do Rosário**; OLIVEIRA-JR, Antonio**; DE OLIVEIRA, Renan R.. **Abordagem Baseada em Aprendizado Federado para a Detecção de Intrusão em Redes de Computadores**. *In*: ESCOLA REGIONAL DE INFORMÁTICA DE GOIÁS (ERI-GO), 11. , 2023, Goiânia/GO.

Goiânia, 04 de junho de 2024

Autor – DENIS NOGUEIRA DO NASCIMENTO – denisnoguer@gmail.com