Phase 1 TFF: Simulation Data and Testing Guide for Sybil Detection

This guide complements the "Detailed Step-by-Step Guide: Phase 1 - Sybil Detection with TFF for ChainFLIP" by providing more details on how to work with simulation data and test the functionality of the implemented Federated Learning (FL) system.

1. Understanding the Existing Simulation in `data_preparation.py`

The `data_preparation.py` file you created in the detailed Phase 1 guide includes a function `load_local_data_for_client`. Currently, this function generates **random data** for features and labels:

```
# From data_preparation.py (simplified snippet)
def load_local_data_for_client(client_id: str, num_samples=100):
    # ...
    X_local = np.random.rand(num_samples, NUM_FEATURES).astype(np.float32)
    y_local = np.random.randint(0, 2, size=(num_samples, 1)).astype(np.int32)
    # ...
    if client_id == "client_1": # Example of non-IID data
        X_local[:num_samples//2, 0] += 0.7
        y_local[:num_samples//2] = 1
    # ...
    return tf.data.Dataset.from_tensor_slices((X_local, y_local))
```

- **Random Features (`X_local`):** Values are random numbers between 0 and 1.
- **Random Labels (`y_local`):** Labels are randomly 0 or 1.
- **Simulated Non-IID Data:** The `if client_id == "client_1":` block slightly modifies the data for `client_1` to make its data distribution different from other clients. This is a basic way to simulate non-Identically and Independently Distributed (non-IID) data, which is common in FL.

Purpose of this initial simulation: The primary goal of this random data simulation is to allow you to run the TFF pipeline (`run_simulation.py`) end-to-end, ensuring that the TFF components (data loading, model definition, federated averaging process) are wired correctly and the training loop executes without crashing. It is **not designed to train a meaningful Sybil detection model** because the data lacks realistic patterns.

2. How to Test the Phase 1 FL System (with Simulated Data)

When you run `python tff_sybil_detection/run_simulation.py` :

- **Objective:** Verify that the TFF training process runs, model weights are aggregated across simulated clients, and the global model shows some (even if superficial) signs of learning.
- **Interpreting Output:**
  - **Client Data Loading:** You should see print statements like `Client sim_client_X: Simulating local data loading and preprocessing...` .
  - **Training Rounds:** The key output to monitor is: `Round 1: loss=X.XXXX, accuracy=Y.YYYY, auc=Z.ZZZZ Round 2: loss=A.AAAA, accuracy=B.BBBB, auc=C.CCCC ...` With the current random data, you might see the loss decrease and accuracy/AUC increase slightly, or they might fluctuate. Significant, consistent improvement would only occur if there were learnable patterns in the data.
  - **Global Model Evaluation:** At the end, `Global model evaluation on simulated test data - Loss: ..., Accuracy: ..., AUC: ...` gives a final performance snapshot on another set of random data.
- **Basic Checks:**
  - Modify `NUM_CLIENTS_SIMULATION` and `NUM_ROUNDS` in `run_simulation.py` to see how it affects runtime and (potentially) the metrics.
  - Ensure no errors occur during the `iterative_process.next()` calls.

3. Generating More Controlled Synthetic Data for Meaningful Testing

To test if your FL system can actually learn to distinguish between "normal" and "Sybil" nodes, you need to create synthetic data with clearer patterns. We will modify `data_preparation.py` to do this.

Conceptual Features for Sybil Detection (5 features as in the guide):

1. `registration_age_days` (e.g., Feature 0)
2. `initial_tx_count_first_week` (e.g., Feature 1)
3. `avg_tx_value_early` (e.g., Feature 2)
4. `interaction_diversity_score_early` (0-1, e.g., Feature 3)
5. `reputation_change_rate_early` (e.g., Feature 4)

Labels: 0 for Normal, 1 for Sybil.

Enhanced `load_local_data_for_client` in `data_preparation.py`:

Replace the existing `load_local_data_for_client` function in `ChainFLIP_FL_Dev/tff_sybil_detection/data_preparation.py` with the following more detailed version:

```
# In data_preparation.py
import tensorflow as tf
import numpy as np

ELEMENT_SPEC = (
    tf.TensorSpec(shape=(5,), dtype=tf.float32),
    tf.TensorSpec(shape=(1,), dtype=tf.int32)
)
NUM_FEATURES = 5

def load_local_data_for_client(client_id: str, num_samples=100,
is_sybil_client=False, sybil_data_ratio=0.0):
    """Generates more structured synthetic data for a single client."""
    print(f"Client {client_id}: Generating structured synthetic data (is_sybil_client:
{is_sybil_client}, sybil_ratio: {sybil_data_ratio})...")

    X_local_list = []
    y_local_list = []

    num_sybil_samples = int(num_samples * sybil_data_ratio)
    if is_sybil_client:
        num_sybil_samples = num_samples # All data for this client will be Sybil-like

    num_normal_samples = num_samples - num_sybil_samples

    # Generate Normal Node Data
    for _ in range(num_normal_samples):
        features = [
            np.random.uniform(100, 365),    # registration_age_days (high)
            np.random.uniform(1, 10),       # initial_tx_count_first_week (low)
            np.random.uniform(50, 1000),    # avg_tx_value_early (moderate)
            np.random.uniform(0.5, 0.9),    # interaction_diversity_score_early (high)
            np.random.uniform(0, 0.1)       # reputation_change_rate_early (low)
        ]
        X_local_list.append(features)
        y_local_list.append([0]) # Label 0 for Normal

    # Generate Sybil Node Data
    for _ in range(num_sybil_samples):
        features = [
            np.random.uniform(1, 30),       # registration_age_days (low)
            np.random.uniform(50, 200),     # initial_tx_count_first_week (high)
            np.random.uniform(1, 10),       # avg_tx_value_early (low, many small txs)
```

```
            np.random.uniform(0.1, 0.4),    # interaction_diversity_score_early (low)
            np.random.uniform(0.5, 1.0)     # reputation_change_rate_early (high)
        ]
        X_local_list.append(features)
        y_local_list.append([1]) # Label 1 for Sybil

    X_local = np.array(X_local_list, dtype=np.float32)
    y_local = np.array(y_local_list, dtype=np.int32)

    # Shuffle the combined data
    indices = np.arange(num_samples)
    np.random.shuffle(indices)
    X_local = X_local[indices]
    y_local = y_local[indices]

    return tf.data.Dataset.from_tensor_slices((X_local, y_local))

# Keep make_federated_data, but modify it to control Sybil client simulation
def make_federated_data(client_ids: list[str], num_samples_per_client=100):
    datasets = []
    for i, client_id in enumerate(client_ids):
        if i == 0: # Let's make the first client a "Sybil" source for testing
            datasets.append(load_local_data_for_client(client_id,
num_samples_per_client, is_sybil_client=True))
        elif i == 1: # Second client has a mix
            datasets.append(load_local_data_for_client(client_id,
num_samples_per_client, sybil_data_ratio=0.5))
        else: # Others are mostly normal
            datasets.append(load_local_data_for_client(client_id,
num_samples_per_client, sybil_data_ratio=0.1))
    return datasets

# Keep the __main__ block for testing data_preparation.py independently if you wish
if __name__ == '__main__':
    print("Testing structured data preparation...")
    CLIENT_IDS_TEST = ["client_A", "client_B", "client_C"]
    federated_train_data_test = make_federated_data(CLIENT_IDS_TEST,
num_samples_per_client=50)

    print(f"\nCreated {len(federated_train_data_test)} client datasets with structured
data.")
    for i, client_dataset in enumerate(federated_train_data_test):
        print(f"Client {CLIENT_IDS_TEST[i]} dataset element spec:
{client_dataset.element_spec}")
        normal_count = 0
        sybil_count = 0
        for features, labels in client_dataset: # Iterate through all samples
            if labels.numpy()[0] == 0:
                normal_count += 1
            else:
                sybil_count += 1
        print(f"  Client {CLIENT_IDS_TEST[i]}: Normal samples: {normal_count}, Sybil
```

```
samples: {sybil_count}")
    print("\nStructured data preparation test complete.")
```

How this helps testing: * **Clear Patterns:** The model now has data with distinct characteristics for normal (0) and Sybil (1) nodes. * **Non-IID Simulation:** The `make_federated_data` function is modified to assign different types of data to different clients (e.g., `client_A` primarily generates Sybil-like data, `client_B` a mix, `client_C` mostly normal). This tests how FedAvg handles data heterogeneity. * **Meaningful Metrics:** With these patterns, you should observe more significant improvements in loss, accuracy, and AUC during training if the model is learning effectively.

To use this: Simply replace the content of your existing `ChainFLIP_FL_Dev/tff_sybil_detection/data_preparation.py` with the code above. Then, re-run `python tff_sybil_detection/run_simulation.py`.

4. Interpreting Results with Controlled Synthetic Data

After running `run_simulation.py` with the structured synthetic data:

- **Loss/Accuracy/AUC:** Look for a clear trend: loss should decrease significantly, and accuracy/AUC should increase towards a high value (e.g., >0.8 or >0.9, depending on data separability and model capacity) over the `NUM_ROUNDS`.
- **Global Model Evaluation:** The final evaluation on simulated test data should also show good performance, indicating the global model has generalized somewhat.
- **Experiment:**
    - Change `NUM_ROUNDS` in `run_simulation.py`.
    - Change `BATCH_SIZE` in `run_simulation.py`.
    - Adjust the model architecture in `model_definition.py` (e.g., number of layers, units, dropout rate).
    - Modify the `sybil_data_ratio` in `make_federated_data` within `data_preparation.py` to see how different client data distributions affect learning.

5. Generating Synthetic Data via CSV Files (Alternative for Persistent Test Sets)

For more complex or persistent test datasets, you can generate data and save it to CSV files, then have `load_local_data_for_client` read from these.

Example: `generate_csv_datasets.py` (Create this file in `ChainFLIP_FL_Dev/tff_sybil_detection/`)

```
import pandas as pd
import numpy as np
import os

NUM_FEATURES = 5
CLIENT_DATA_DIR = "client_data"

def create_synthetic_dataframe(num_samples=100, sybil_ratio=0.2):
    X_list = []
    y_list = []
    num_sybil = int(num_samples * sybil_ratio)
    num_normal = num_samples - num_sybil

    # Normal Node Data
    for _ in range(num_normal):
        features = [
            np.random.uniform(100, 365), np.random.uniform(1, 10),
            np.random.uniform(50, 1000), np.random.uniform(0.5, 0.9),
            np.random.uniform(0, 0.1)
        ]
        X_list.append(features)
        y_list.append(0)

    # Sybil Node Data
    for _ in range(num_sybil):
        features = [
            np.random.uniform(1, 30), np.random.uniform(50, 200),
            np.random.uniform(1, 10), np.random.uniform(0.1, 0.4),
            np.random.uniform(0.5, 1.0)
        ]
        X_list.append(features)
        y_list.append(1)

    df = pd.DataFrame(X_list, columns=[f"feature_{i}" for i in
range(NUM_FEATURES)])
    df["label"] = y_list
    return df.sample(frac=1).reset_index(drop=True) # Shuffle

if __name__ == "__main__":
    if not os.path.exists(CLIENT_DATA_DIR):
        os.makedirs(CLIENT_DATA_DIR)

    client_configs = {
        "client_X": {"num_samples": 150, "sybil_ratio": 0.8}, # Mostly Sybil
        "client_Y": {"num_samples": 200, "sybil_ratio": 0.1}, # Mostly Normal
        "client_Z": {"num_samples": 100, "sybil_ratio": 0.5}, # Mixed
    }
```

```
    for client_name, config in client_configs.items():
        client_df = create_synthetic_dataframe(config["num_samples"],
config["sybil_ratio"])
        file_path = os.path.join(CLIENT_DATA_DIR, f"{client_name}_data.csv")
        client_df.to_csv(file_path, index=False)
        print(f"Generated {file_path} with {len(client_df)} samples
({int(config['sybil_ratio']*100)}% Sybil)")
```

To use this: 1. Run `python tff_sybil_detection/generate_csv_datasets.py`. This will create a `client_data` folder with CSV files. 2. Modify `load_local_data_for_client` in `data_preparation.py` to read these CSVs: ```python # In data_preparation.py, modify load_local_data_for_client: import pandas as pd import os # ... (keep ELEMENT_SPEC and NUM_FEATURES) CLIENT_DATA_DIR = "client_data" # Assuming this script is run from tff_sybil_detection

```
 def load_local_data_for_client(client_id: str, num_samples=None): # num_samples
not used if reading CSV
    file_path = os.path.join(CLIENT_DATA_DIR, f"{client_id}_data.csv")
    if not os.path.exists(file_path):
        raise FileNotFoundError(f"Data file not found for client {client_id} at
{file_path}")

    print(f"Client {client_id}: Loading data from {file_path}...")
    df = pd.read_csv(file_path)
    X_local = df[[f"feature_{i}" for i in
range(NUM_FEATURES)]].values.astype(np.float32)
    y_local = df[["label"]].values.astype(np.int32) # Ensure shape is (num_samples, 1)
    return tf.data.Dataset.from_tensor_slices((X_local, y_local))

# Modify make_federated_data to use the client names from CSV generation
def make_federated_data(client_ids: list[str], num_samples_per_client=None):
    return [load_local_data_for_client(client_id) for client_id in client_ids]

# In run_simulation.py, you would then use these client_ids:
# CLIENT_IDS_SIMULATION = ["client_X", "client_Y", "client_Z"]
```

6. Advanced Testing Considerations

- **Holdout Evaluation Set:** For robust evaluation, create a separate set of client data (or a centralized test set if appropriate for your FL privacy model) that is never used during training. Evaluate the final global model on this holdout set.

- **Testing Admin Dashboard Integration (Conceptual):**
    - Once your `global_sybil_detection_model.h5` (or similar saved model) is generated by `run_simulation.py`, your prediction service (e.g., a Flask API) would load this model.
    - You can then send feature vectors (representing hypothetical nodes) to this service and check the risk scores it returns. This tests the model's predictive capability independently of the TFF training loop.

By using these structured synthetic data generation and testing approaches, you can gain more confidence in your Phase 1 FL system's ability to learn and detect patterns before moving to real-world data integration.