Federated Learning Framework Documentation

Overview

The `Federated Learning Framework` is a modular and extensible framework designed to facilitate federated learning in various applications, including but not limited to NLP, self-driving cars, Unmanned Aerial Vehicles (UAVs), Robotics, and other AI domains. The framework includes support for homomorphic encryption to ensure the privacy and security of model weights during federated training.

Features

- **Modular Design**: Easily customizable for different machine learning and deep learning tasks.
- **Federated Learning**: Supports decentralized model training across multiple clients.
- **Homomorphic Encryption**: Ensures the privacy and security of model weights during transmission and aggregation.

Package Structure

```
federated_learning_framework/
│
├── README.md
├── setup.py
├── requirements.txt
├── federated_learning_framework/
│   ├── __init__.py
│   ├── central_server.py
│   ├── client_device.py
│   ├── encryption.py
│   ├── active_learning.py
│   ├── connection.py
│   ├── decorators.py
│   └── utils.py
└── tests/
    ├── __init__.py
    ├── test_central_server.py
    ├── test_client_device.py
    ├── test_encryption.py
    ├── test_active_learning.py
    └── test_utils.py
```

Detailed Component Description

1. Central Server

File: `central_server.py`

The central server orchestrates the federated learning process by coordinating the communication and aggregation of model weights from various client devices.

Key Functions:

- `run_server`: Starts the server to handle client connections.
- `handle_client`: Manages incoming messages from clients.
- `transmit_weights`: Broadcasts the aggregated weights to clients.
- `send_data_to_client`: Sends specific data to a client.
- `get_data_from_client`: Requests and receives data from a client.
- `query_active_learning`: Implements active learning strategies to select data for labeling.

2. Client Device

File: `client_device.py`

Client devices perform local training on their datasets and communicate with the central server.

Key Functions:

- `connect_to_server`: Connects to the central server.
- `federated_learning`: Coordinates local training and communication with the server.
- `train_model`: Trains the local model.
- `encrypt_model_weights`: Encrypts model weights using homomorphic encryption before transmission.

3. Encryption

File: `encryption.py`

Provides functions for creating encryption contexts and encrypting/decrypting model weights.

Key Functions:

- `create_context`: Sets up the encryption context using TenSEAL.
- `encrypt_weights`: Encrypts model weights.
- `decrypt_weights`: Decrypts encrypted model weights.

4. Active Learning

File: `active_learning.py`

Implements active learning strategies to enhance the training process by selectively querying informative data points.

Key Functions:

- `select_informative_samples`: Selects samples for labeling based on uncertainty.

5. Connection

File: `connection.py`

Manages the connection types and protocols (e.g., WebSocket) for communication between the central server and client devices.

Key Functions:

- `run_server`: Starts a WebSocket server.
- `connect_to_server`: Establishes a WebSocket connection to the server.

6. Decorators

File: `decorators.py`

Provides decorators for adding federated learning and encryption functionalities to functions.

Key Functions:

- `federated_learning_decorator`: Wraps a function to enable federated learning.
- `encryption_decorator`: Wraps a function to enable homomorphic encryption.

7. Utilities

File: `utils.py`

Includes utility functions used throughout the framework.

Installation

1. Clone the repository:

```
git clone
https://github.com/mehrdaddjavadi/federated_learning_framework.git
```

2. Navigate to the directory:

```
cd federated_learning_framework
```

3. Install the dependencies:

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

Usage

Setting Up the Central Server

```
from federated_learning_framework.central_server import CentralServer
import asyncio

async def main():
    server = CentralServer()
    await server.run_server(host='0.0.0.0', port=8089)

asyncio.run(main())
```

Setting Up a Client Device

```
from federated_learning_framework.client_device import ClientDevice
from federated_learning_framework.encryption import create_context
import tensorflow as tf

context = create_context()

model = tf.keras.Sequential([
    tf.keras.layers.Dense(4, activation='relu', input_shape=(3072,)),
    tf.keras.layers.Dense(10, activation='softmax')
])

client = ClientDevice(client_id=1, model=model, context=context)

asyncio.run(client.connect_to_server('ws://localhost:8089'))
```

Using Decorators

```
from federated_learning_framework.decorators import
federated_learning_decorator, encryption_decorator
from federated_learning_framework.central_server import CentralServer
from federated_learning_framework.client_device import ClientDevice
from federated_learning_framework.encryption import create_context
import tensorflow as tf
import asyncio

context = create_context()
```

```
@federated_learning_decorator(uri="ws://localhost:8089")
@encryption_decorator(context=context)
async def main(central_server):
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(4, activation='relu', input_shape=(3072,)),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    client = ClientDevice(client_id=1, model=model, context=context)
    await client.connect_to_server('ws://localhost:8089')

asyncio.run(main())
```

Running Tests

To run the tests, execute the following command in the root directory:

```
python -m unittest discover -s tests
```

License

Contact: mehrdaddjavadi@gmail.com

Contributing

Feel free to contribute by submitting a pull request or opening an issue.