**Function and Flow of the iot\_meta\_learning\_ids-1 Project**

The iot\_meta\_learning\_ids-1 project develops an Intrusion Detection System (IDS) for IoT networks using a **meta-learning** and **federated learning (FL)** approach. It leverages the UNSW-NB15 dataset for binary classification (normal vs. attack traffic) and employs a hybrid Fish Swarm ConvNet model. The project combines meta-learning (using MAML) for quick adaptation to new tasks and federated learning to enable collaborative training across simulated IoT devices (clients) while preserving data privacy. The workflow involves data preprocessing, task generation, federated training, meta-training, and evaluation, achieving strong performance: 91.84% accuracy, 0.9287 F1-score, and 0.98 AUC on the test set, with a recommended threshold of 0.4 for better precision.

**Project Structure and File Descriptions**

**1. configs/paths.yaml**

* **Use Description**: Configuration file that stores file paths for datasets, processed data, models, and results.
* **Function**: Provides a centralized location for paths, ensuring scripts like data\_loader.py and evaluator.py can access raw datasets (UNSW\_NB15\_training-set.csv, UNSW\_NB15\_testing-set.csv), processed data (tasks.npy, test\_sets.pkl), model weights (maml\_model.weights.h5), and output directories (results/plots/).

**2. data/processed/tasks.npy**

* **Use Description**: Stores meta-learning tasks generated from the UNSW-NB15 training set.
* **Function**: Contains 5 tasks (one per client), each with 200 training samples (100 normal, 100 attack) and 100 test samples (50 normal, 50 attack). Used by clients (client1.py, client2.py, client3.py) for local training and by evaluator.py for meta-test evaluation.

**3. data/processed/test\_sets.pkl**

* **Use Description**: Stores preprocessed test sets for evaluation.
* **Function**: Contains the preprocessed UNSW-NB15 test set (and potentially other datasets), used by evaluator.py to evaluate the final model after training.

**4. data/raw/UNSW-NB15/**

* **Use Description**: Directory containing the raw UNSW-NB15 dataset CSV files (UNSW\_NB15\_training-set.csv, UNSW\_NB15\_testing-set.csv).
* **Function**: Provides the raw data for preprocessing by data\_loader.py. The dataset includes 82,332 training samples (37,000 normal, 45,332 attack) and a similar test set for evaluation.

**5. src/clients/client1.py, client2.py, client3.py**

* **Use Description**: Simulate IoT devices (clients) in a federated learning setup, each training on a local task.
* **Function**:
  + Each client loads its assigned task from tasks.npy (e.g., Client 1 uses task 0, Client 2 uses task 1, etc.).
  + Trains the hybrid Fish Swarm ConvNet model locally on its task (200 samples) for 5 epochs, using class weights to handle imbalance.
  + Communicates with the server (server.py) via sockets to send updated weights and receive the aggregated global model weights.
  + Iteratively updates its model in a federated learning loop.
* **Role in Flow**: Enables distributed training across simulated IoT devices, ensuring privacy (data stays local) and scalability.

**6. src/\_\_init\_\_.py**

* **Use Description**: An empty file that makes the src directory a Python package.
* **Function**: Allows imports from src (e.g., from src.model import get\_model) in other scripts.

**7. src/data\_loader.py**

* **Use Description**: Preprocesses the UNSW-NB15 dataset and generates meta-learning tasks.
* **Function**:
  + Loads the raw dataset (UNSW\_NB15\_training-set.csv).
  + Preprocesses the data: drops unnecessary columns (id, attack\_cat, proto, service, state), selects the top 20 features using Random Forest, and scales features with MinMaxScaler.
  + Generates 5 balanced meta-learning tasks (200 training samples, 100 test samples per task).
  + Saves the preprocessed data, scaler, selected features, and tasks for use by clients and evaluator.py.
* **Role in Flow**: Prepares the data for federated and meta-learning by creating tasks that simulate data distributions on different IoT devices.

**8. src/evaluator.py**

* **Use Description**: Evaluates the trained model on meta-test tasks and the UNSW-NB15 test set, fine-tunes the model, and generates performance visualizations.
* **Function**:
  + Loads the trained model weights (maml\_model.weights.h5).
  + Evaluates the model on meta-test tasks (from tasks.npy), computing average accuracy, F1-score, precision, and recall.
  + Preprocesses the test set (UNSW\_NB15\_testing-set.csv) using the same features and scaler as the training set.
  + Fine-tunes the model on a subset (10%) of the test data for 5 epochs.
  + Evaluates the fine-tuned model on the full test set, computing metrics (accuracy: 91.84%, F1-score: 0.9287, precision: 0.8943, recall: 0.9659 at threshold 0.3).
  + Generates visualizations: ROC curve (roc\_curve.png, AUC = 0.98) and metrics vs. threshold plot (threshold\_metrics.png).
  + Recommends threshold 0.4 for better precision (0.91) and recall (0.95) balance.
* **Role in Flow**: Assesses the final model’s performance after federated and meta-training, providing insights into its effectiveness as an IDS.

**9. src/meta\_trainer.py**

* **Use Description**: Implements the meta-learning training process using MAML (Model-Agnostic Meta-Learning).
* **Function** (Assumed, as the file content isn’t provided):
  + Loads the tasks from tasks.npy.
  + Trains the hybrid Fish Swarm ConvNet model using MAML, where the model learns to adapt quickly to new tasks by optimizing for fast adaptation (inner loop) and generalization (outer loop).
  + Saves the meta-trained model weights (maml\_model.weights.h5) for use by evaluator.py and potentially server.py.
* **Role in Flow**: Performs meta-learning to ensure the model can adapt to new IoT environments (tasks), complementing the federated learning process.

**10. src/model.py**

* **Use Description**: Defines the hybrid Fish Swarm ConvNet model architecture.
* **Function** (Assumed, as the file content isn’t provided):
  + Implements a convolutional neural network (CNN) integrated with Fish Swarm optimization, a bio-inspired algorithm that enhances the CNN’s learning process.
  + Provides a get\_model() function to create and return the model, used by clients (client1.py, etc.), meta\_trainer.py, and evaluator.py.
* **Role in Flow**: Defines the core model used across the project, ensuring consistency in architecture for federated and meta-learning.

**11. src/server.py**

* **Use Description**: Implements the central server in the federated learning setup, coordinating training across clients.
* **Function** (Assumed, as the file content isn’t provided):
  + Listens for connections from clients (client1.py, client2.py, client3.py) on localhost:5000.
  + Receives model weights and sample counts from each client.
  + Aggregates the weights (e.g., using FedAvg: weighted average based on sample counts) to update the global model.
  + Sends the updated global model weights back to all clients.
  + Manages the federated learning rounds until convergence or a set number of rounds.
* **Role in Flow**: Orchestrates federated learning by aggregating client updates and distributing the global model, ensuring collaborative training without centralizing data.

**12. src/utils.py**

* **Use Description**: Provides utility functions for client-server communication.
* **Function**:
  + Implements send\_data and receive\_data functions used by client1.py (and other clients) and server.py.
  + send\_data: Serializes and sends data (e.g., model weights, sample counts) over a socket connection.
  + receive\_data: Receives and deserializes data from a socket connection.
* **Role in Flow**: Facilitates communication between clients and the server in the federated learning setup.

**13. results/plots/roc\_curve.png**

* **Use Description**: Stores visualizations generated during evaluation.
* **Function**: Contains the ROC curve plot (AUC = 0.98) and metrics vs. threshold plot (threshold\_metrics.png) generated by evaluator.py, providing visual insights into the model’s performance.

**Project Flow**

1. **Data Preprocessing** (data\_loader.py):
   * Load the UNSW-NB15 training set (UNSW\_NB15\_training-set.csv).
   * Preprocess: drop unnecessary columns, select top 20 features, scale features.
   * Generate 5 meta-learning tasks (200 training samples, 100 test samples each).
   * Save tasks (tasks.npy) and test sets (test\_sets.pkl).
2. **Federated Learning** (server.py, client1.py, client2.py, client3.py):
   * **Server Initialization**: The server (server.py) starts and listens for client connections.
   * **Client Training**:
     + Each client (e.g., client1.py) loads its task from tasks.npy, trains the model locally for 5 epochs, and sends updated weights to the server.
   * **Server Aggregation**:
     + The server receives weights from all clients, aggregates them (e.g., using FedAvg), and sends the updated global model back to the clients.
   * **Iteration**: This process repeats for multiple rounds, improving the global model collaboratively.
3. **Meta-Learning** (meta\_trainer.py):
   * Load the tasks (tasks.npy) and the global model (after federated learning, if integrated).
   * Train the model using MAML to optimize for fast adaptation to new tasks.
   * Save the meta-trained model weights (maml\_model.weights.h5).
4. **Evaluation** (evaluator.py):
   * Load the meta-trained model weights.
   * Evaluate on meta-test tasks (from tasks.npy), computing average metrics (accuracy: 90.80%, F1-score: 0.9115).
   * Preprocess the UNSW-NB15 test set (UNSW\_NB15\_testing-set.csv).
   * Fine-tune the model on a subset (10%) of the test data for 5 epochs.
   * Evaluate on the full test set (accuracy: 91.84%, F1-score: 0.9287 at threshold 0.3).
   * Generate visualizations (ROC curve, threshold plot) and recommend threshold 0.4 (F1-score: 0.93, precision: 0.91, recall: 0.95).

**Key Functions of the Project**

* **Intrusion Detection**: Classifies network traffic as normal or attack using the UNSW-NB15 dataset.
* **Federated Learning**: Enables collaborative training across IoT devices (clients) without sharing raw data, ensuring privacy and scalability.
* **Meta-Learning**: Uses MAML to train a model that can quickly adapt to new tasks (IoT environments), improving generalization.
* **Hybrid Fish Swarm ConvNet**: Combines a CNN with Fish Swarm optimization for enhanced learning and classification performance.
* **Evaluation and Optimization**: Assesses the model’s performance, fine-tunes it, and optimizes the classification threshold for better precision-recall balance.

**Summary**

The iot\_meta\_learning\_ids-1 project integrates federated learning and meta-learning to build a robust IDS for IoT networks. It preprocesses the UNSW-NB15 dataset (data\_loader.py), trains a hybrid Fish Swarm ConvNet model collaboratively across clients (client1.py, server.py) and with meta-learning (meta\_trainer.py), and evaluates the model (evaluator.py) to achieve high performance (91.84% accuracy, 0.9287 F1-score). The project structure ensures modularity, with each file handling a specific task in the pipeline, from data preparation to distributed training and final evaluation.