National College of Ireland

Configuration Manual

MSc Research Project
Programme Name

Shwe Moe Thant
Student ID: 24170399

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shwe Moe Thant
Student ID:	24170399
Programme:	Programme Name
Year:	2025
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	11/12/2025
Project Title:	Configuration Manual
Word Count:	2135
Page Count:	15

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	26th January 2026

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

Shwe Moe Thant
24170399

1 System Overview

This configuration manual describes the environment used to deploy and evaluate the Adaptive Energy-Aware Federated Learning framework. It supports reproducibility and provides a clear path from initial setup to complete system execution. All experiments were conducted on an Amazon EC2 instance running Ubuntu, selected to provide a controlled CPU-based environment comparable to lightweight edge computing conditions in smart-city deployments.

The framework relies on Python, Docker and Amazon S3 to emulate multiple IoT clients and a central federated server. Each client operates as an isolated container, maintains its own dataset partition and performs local model training independently. The server coordinates model aggregation, metadata collection, client selection and overall system orchestration.

The project repository includes all source code required to reproduce the results reported in this dissertation. This includes preprocessing scripts, federated learning modules, Docker configuration files, orchestration scripts and utilities for accuracy, energy and privacy visualisations. This manual outlines how to prepare the environment, configure dependencies, execute the federated learning system and retrieve relevant outputs.

The aim of this document is to ensure that any examiner can follow the instructions and reproduce an equivalent experimental setup without additional clarification. All paths, commands and configuration details have been tested in the target environment and reflect the same workflow used throughout the study.

2 Environment Setup

2.1 EC2 Instance Requirements and Configuration

The system was deployed using an Amazon EC2 virtual machine to provide a controlled and reproducible execution environment for the federated learning framework. The recommended setup uses the `t3.large` instance type with Ubuntu Server. The following configuration was used during provisioning:

- Region: `us-east-1` (North Virginia)
- AMI: Ubuntu Server (latest LTS release)
- Instance type: `t3.large`
- Key pair: create a new key pair and download it locally

- Security group rule: allow SSH traffic from the user's IP address
- Storage: 50 GiB `gp3` root volume with 3000 IOPS, not encrypted

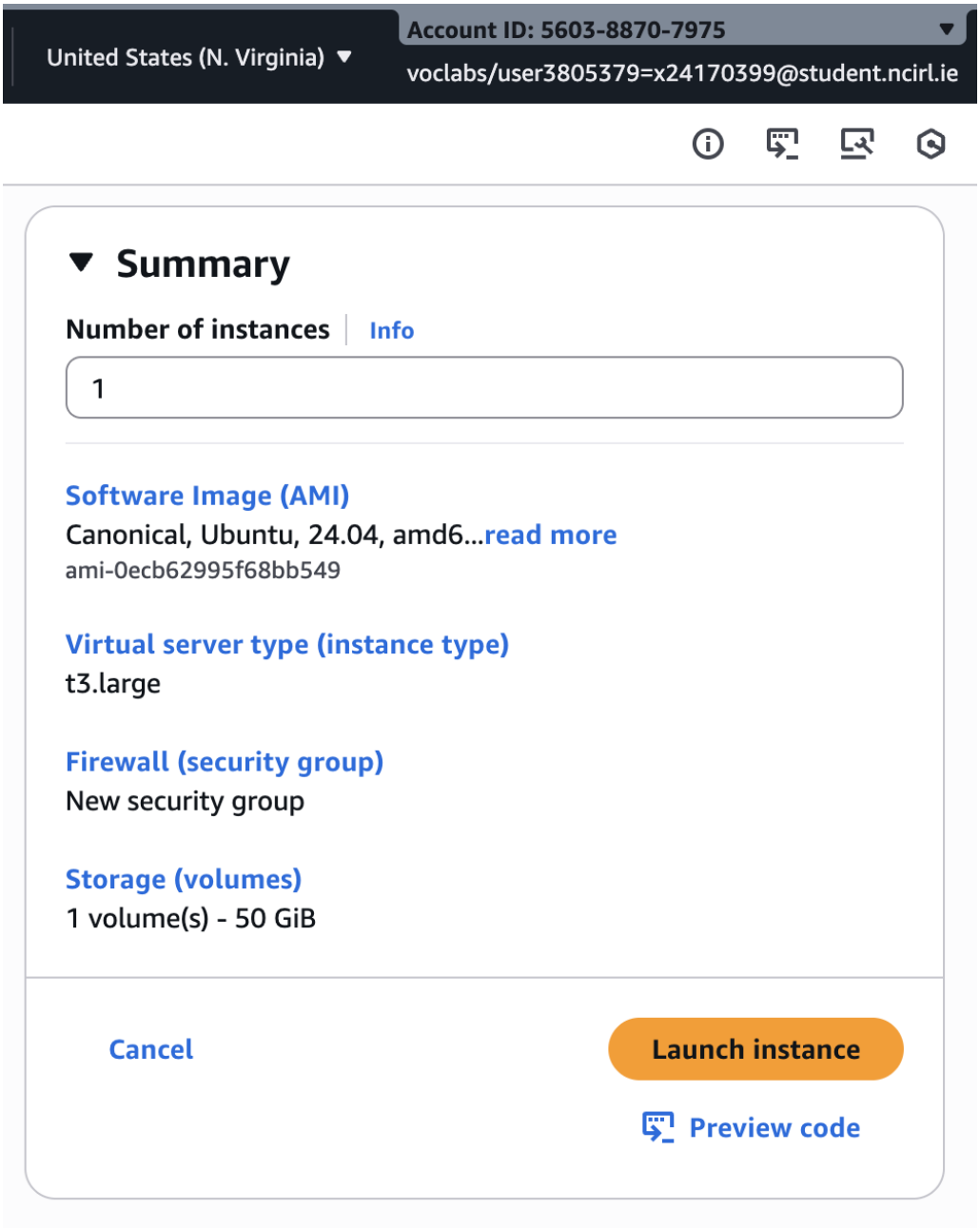

Figure 1: EC2 instance configuration used for deployment.

A larger root volume is necessary because the framework stores raw datasets, preprocessed partitions and multiple Docker images. After the instance is created, the AWS console provides an SSH command under *Connect*. In general, the connection format is:

```
ssh -i "<keypair>.pem" ubuntu@<ec2-public-address>
```

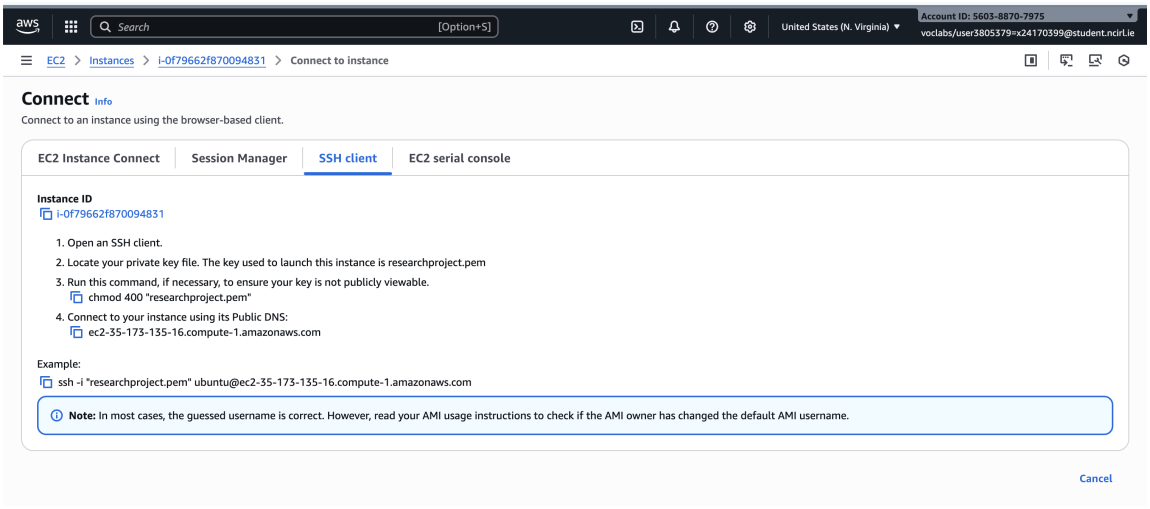

Figure 2: Connecting to the EC2 instance using the generated SSH key pair.

2.2 Installing System Dependencies

Once connected, the system must be prepared with Docker, AWS CLI and supporting utilities. Installing `awscli` directly from Ubuntu repositories may fail, so the following method should be used:

```
sudo apt update
sudo apt install -y ca-certificates curl gnupg git unzip
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
  | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/
   docker.gpg] \
  https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" \
  | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io \
  docker-buildx-plugin docker-compose-plugin git
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" \
  -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
rm -rf aws awscliv2.zip
```

To avoid permission errors when running Docker, the user should be added to the Docker group:

```
sudo usermod -aG docker ubuntu
newgrp docker
```

2.3 Python Virtual Environment

A Python virtual environment is required for dependency isolation. It is created by:

```
python3 -m venv .venv
source .venv/bin/activate
```

2.4 Repository Installation

The system code is downloaded from the project repository:

```
git clone https://github.com/ShweMoeThantAurum/ResearchProject
cd ResearchProject
pip install -r requirements.txt
```

The dependency file contains PyTorch, Flower, NumPy, pandas, Matplotlib and helper libraries used throughout the framework.

2.5 AWS Configuration

Amazon S3 acts as the message exchange layer between clients and the server. Credentials are configured through:

```
mkdir -p ~/.aws
aws configure
```

The user is prompted for:

```
AWS Access Key ID
AWS Secret Access Key
Default region: us-east-1
Default output format: json
```

A quick validation can be performed using:

```
aws sts get-caller-identity
```

2.6 Container Preparation

The system supports two ways of preparing Docker containers. Testers may choose either depending on their workflow.

Approach 1: Local Docker Build (default)

The simplest approach is to build the containers locally using the provided Dockerfile:

```
docker compose up --build
```

This builds and launches all client containers automatically.

Approach 2: Using Amazon ECR for Container Images

A second approach is to push the client container to Amazon Elastic Container Registry (ECR) and retrieve it from there. This approach is useful when testers prefer to avoid building images on the EC2 instance. The workflow is:

1. Authenticate Docker to ECR:

```
aws ecr get-login-password --region us-east-1 \
 | docker login --username AWS --password-stdin \
   <aws-account-id>.dkr.ecr.us-east-1.amazonaws.com
```

2. Build the Docker image:

```
docker build -t <ECR Repo Name> .
```

3. Tag the image for ECR:

```
docker tag <ECR Repo Name>:latest \
  <aws-account-id>.dkr.ecr.us-east-1.amazonaws.com/aefl-client:latest
```

4. Push the image:

```
docker push <aws-account-id>.dkr.ecr.us-east-1.amazonaws.com/<ECR Repo
    Name>:latest
```

If using ECR images, testers need to adjust `docker-compose.yml` so that each container references the remote ECR image instead of building locally.

2.7 Verifying Installed Tools

After installation, the following commands can be used to verify that Git, Docker, Docker Compose, AWS CLI and Python are correctly installed and accessible within the environment:

```
git --version
docker version
docker compose version
aws --version
python3 --version
pip --version
```

A typical output on the EC2 instance is shown below. Exact versions may differ as package releases are updated over time:

```
git version 2.43.0
Client: Docker Engine - Community
 Version:           29.1.2
 API version:       1.52
 Go version:        go1.25.5
 Git commit:        890dcca
 Built:             Tue Dec  2 21:55:19 2025
 OS/Arch:           linux/amd64
 Context:           default
permission denied while trying to connect to the docker API at unix:///var/
   run/docker.sock
Docker Compose version v5.0.0
aws-cli/2.32.14 Python/3.13.9 Linux/6.14.0-1015-aws exe/x86_64.ubuntu.24
Python 3.12.3
pip 24.0 from /home/ubuntu/.venv/lib/python3.12/site-packages/pip (python
   3.12)
```

The Docker permission warning appears until the user is added to the `docker` group and a new shell session is opened. Once `usermod` and `newgrp` have been applied, Docker commands operate without requiring `sudo`.

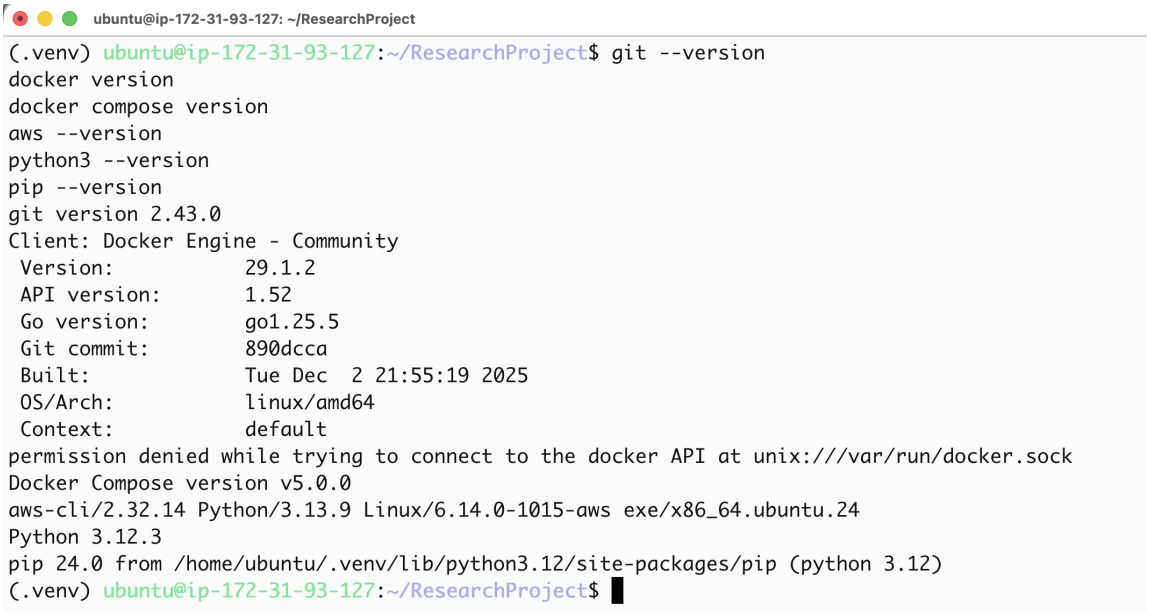

Figure 3: Verification of installed development tools on the EC2 instance.

2.8 Dataset Preparation and S3 Upload

The framework expects raw dataset files to be available in an S3 bucket under:

```
s3://<bucket-name>/data/raw/
```

The required raw datasets can be downloaded from the following repository links:

- SZ Taxi adjacency: `https://github.com/ShweMoeThantAurum/ResearchProject/blob/main/data/raw/sz/sz_adj.csv`
- Los Loop adjacency: `https://github.com/ShweMoeThantAurum/ResearchProject/blob/main/data/raw/los/los_adj.csv`
- PeMS08 dataset: `https://github.com/ShweMoeThantAurum/ResearchProject/blob/main/data/raw/pems08/pems08.npz`

These files must be uploaded to the tester's own S3 bucket before preprocessing.

2.9 Updating S3 Bucket Paths in Preprocessing Scripts

Before running the preprocessing scripts, testers must update the S3 bucket name in the dataset loaders. Each preprocessing script contains variables that define the bucket and the raw data prefix. These must be changed to match the tester's own S3 bucket name.

The relevant lines are found at the top of each preprocessing module:

- `data/preprocess_sz.py`

```
BUCKET = "aefl"
S3_PREFIX = "raw/sz/"
```

- data/preprocess_los.py

```
BUCKET = "aefl"
S3_PREFIX = "raw/los/"
```

- data/preprocess_pems08.py

```
BUCKET = "aefl"
S3_PREFIX = "raw/pems08/"
```

To run the system using a different S3 bucket, testers should replace the default value `"aefl"` with the name of their own bucket in each file. The `S3_PREFIX` values should not be changed unless the folder structure inside the bucket has also been modified.

Updating these variables ensures that all preprocessing steps correctly retrieve the raw input files from the specified bucket.

2.10 Data Preprocessing

With the environment ready and the raw datasets in place, preprocessing is initiated using the provided scripts:

```
python -m data.preprocess_sz
python -m data.preprocess_los
python -m data.preprocess_pems08
```

The scripts generate normalised matrices, train–validation–test splits, adjacency graphs and client-level partitions. Output files are written to:

```
data/processed/<dataset>/prepared/
```

These files are required for all training and evaluation workflows.

3 Execution and Reproducibility

3.1 Automated Execution Workflow

The system provides automation scripts that reproduce all experiments reported in the dissertation. This workflow is recommended for examiners because it executes the full training pipeline for each dataset and learning mode without manual configuration.

Energy and accuracy experiments are executed using:

```
bash run_baselines.sh
```

Privacy experiments are executed using:

```
bash run_privacy_dp.sh
```

Energy, accuracy and privacy summaries are collected through:

```
bash collect_energy.sh
bash collect_accuracy.sh
bash collect_privacy_dp.sh
```

Final thesis-ready plots are generated using:

```
python -m scripts.energy_all_datasets
python -m scripts.accuracy_all_metrics_all_dataset
python -m scripts.privacy_dp_plots
```

All outputs are written to the `outputs/` directory and synchronised automatically to the configured S3 bucket. These artefacts include accuracy metrics, energy logs, differential privacy results and per-round metadata.

During automated execution, server output appears in the active terminal. To observe client activity in real time, a second terminal may be used:

```
docker ps
docker logs -f <container-name>
```

3.2 Manual Execution Workflow

Manual execution provides full control over dataset selection, learning modes, privacy noise levels and experiment variants. This mode is intended for advanced users who wish to explore behaviours beyond the predefined experiments.

Manual execution requires two terminals: Terminal 1 runs client containers, and Terminal 2 runs the federated server. Both must run concurrently because clients and server operate independently.

Terminal 1: Start Clients Select a dataset and launch client containers:

```
export DATASET=sz # or los, pems08
docker compose build
docker compose up
```

Terminal 2: Start Server In a separate terminal, start the federated server using one of the supported modes:

```
FL_MODE=AEFL python -m src.fl.server.main
FL_MODE=FedAvg python -m src.fl.server.main
FL_MODE=FedProx python -m src.fl.server.main
```

The server coordinates with active Docker clients, following the same structure used in all experiments reported in the dissertation.

Configuration Parameters

Environment variables configure dataset selection, privacy settings and experiment variants. These variables correspond directly to the configurations evaluated in the dissertation.

```
export DATASET="$DATASET"       # sz, los or pems08
export FL_MODE="$MODE"         # AEFL, FedAvg or FedProx
export DP_SIGMA="$SIGMA"       # 0.0, 0.01, 0.05, 0.10, 0.20
export VARIANT_ID="dp_sigma_${SIGMA}"
```

The following presets reproduce the exact experimental configurations.

Energy Experiment

```
export DP_ENABLED=false
export COMPRESSION_ENABLED=false
export VARIANT_ID=""
```

Accuracy Experiment

```
export DP_ENABLED=false
export COMPRESSION_ENABLED=false
export VARIANT_ID=""
```

Privacy Experiment

```
export DP_ENABLED=true
export DP_SIGMA=0.0 # or 0.01, 0.05, 0.10, 0.20
export VARIANT_ID="dp_sigma_${DP_SIGMA}"
export COMPRESSION_ENABLED=false
```

For each experiment, the server is launched with:

```
FL_MODE=AEFL python -m src.fl.server.main
FL_MODE=FedAvg python -m src.fl.server.main
FL_MODE=FedProx python -m src.fl.server.main
```

These settings reproduce the privacy–accuracy–energy analyses presented in the dissertation.

Custom Settings Other configurations may be explored, but only the settings above reproduce the reported experimental results. Parameter choices are grounded in the design rationale presented in earlier chapters.

3.3 Observing Client and Server Execution

During manual execution, system behaviour can be monitored in real time.

Client containers may be inspected using:

```
docker ps
docker logs -f <client-container>
```

Federated server output appears in its dedicated terminal, allowing observation of round progression, aggregation steps and client participation decisions.

3.4 Retrieving Results

All outputs stored in S3 during training and evaluation can be retrieved locally using:

```
aws s3 sync s3://<bucket-name>/outputs ./outputs_downloaded
```

The downloaded directory contains per-round metrics, global checkpoints, metadata files and visualisations generated by the system.

3.5 Reproducibility Considerations

Reproducibility is supported through deterministic preprocessing, fixed random seeds, Docker-based client isolation and a stable communication pipeline. All experiments reported in the dissertation were produced using the workflow described in this section. Following the same procedure under an identical environment configuration yields equivalent results.

4 File Structure and Module Responsibilities

This section summarises the project structure and the responsibilities of major directories. It is intended to help examiners and testers navigate the repository quickly and locate where each system component is implemented.

4.1 Top-Level Structure

The project root contains configuration files, helper scripts, dataset folders, federated learning source code and generated outputs:

- `Dockerfile` – builds the client container image used for all federated clients.
- `docker-compose.yml` – defines five client containers (vehicle, bus, sensor, roadside, camera).
- `run_baselines.sh`, `runprivacy_dp.sh` – automation scripts for running full experiments.
- `collectenergy.sh`, `collectaccuracy.sh`, `collectprivacy_dp.sh` – scripts for collecting final results.
- `requirements.txt` – Python dependencies used in the EC2 environment.
- `README.md` – high-level project description.

4.2 Data Directory

The `data/` directory contains raw files, preprocessing scripts and processed datasets used for training.

- `data/raw/` – original SZ Taxi, Los Loop and PeMSD8 files uploaded to S3; used as preprocessing input.
- `data/preprocess_sz.py`, `data/preprocess_los.py`, `data/preprocess_pems08.py` – dataset-specific preprocessing modules that normalise features, compute adjacency matrices and create sliding windows.
- `data/dataloader.py` – unified loader used by clients to retrieve the correct partition and metadata.
- `data/processed/` – train, validation and test splits, adjacency matrices and client-level partitions stored per dataset.

4.3 Federated Learning Source Code

The main federated learning logic is located in `src/fl/`, which mirrors the client–server structure of the system.

Server Modules (`src/fl/server/`)

- `main.py` – entry point for server execution.
- `selection.py` – AEFL adaptive client selection strategy.
- `aggregation.py` – FedAvg, FedProx and AEFL aggregation logic.
- `evaluate.py` – server-side evaluation and metric computation.
- `energy.py` – server energy monitoring pipeline.
- `s3.py` – downloads client updates and uploads global models to S3.
- `summary.py` – writes per-round summaries and CSV files.

Client Modules (`src/fl/client/`)

- `main.py` – client entry point for data loading, training and uploading.
- `train.py` – local GRU training on the assigned data partition.
- `energy.py` – integrates `pyJoules` for client-side energy measurements.
- `privacy.py` – applies differential privacy noise during update construction.
- `s3.py` – manages S3 interactions for model download and update upload.
- `compression.py` – optional update compression utilities.

Shared Utilities (`src/fl/utils.py`, `src/utils/`)

- `metrics.py` – MAE, RMSE and MAPE metrics.
- `privacy.py` – Gaussian noise generation utilities.
- `flops.py` – FLOPs estimation for local models.
- `s3_helpers.py` – helper functions for S3 object management.

4.4 Model Directory

- `src/models/simple_gru.py` – GRU-based sequence prediction model used across all datasets.

4.5 Scripts Directory

The `scripts/` directory contains utilities for generating the plots used in the dissertation:

- `energy_all_datasets.py` – aggregates energy logs and generates multi-dataset plots.
- `accuracy_all_metrics_all_dataset.py` – compares AEFL, FedAvg and FedProx accuracy.
- `privacy_dp_plots.py` – visualises the effect of differential privacy noise.

4.6 Outputs Directory

The `outputs/` directory mirrors the structure used in S3 and stores:

- Energy usage summaries,
- Accuracy metrics,
- Differential privacy impact logs,
- Per-round model and metadata files,
- Final thesis-ready plots.

This structure ensures that artefacts used for evaluation and visualisation are preserved and easy to retrieve.

5 Cleanup and Reset Instructions

This section describes how to reset the environment to a clean state. These steps are useful when re-running experiments, freeing disk space or recovering from inconsistent configurations.

5.1 Stopping All Containers

To stop and remove all running containers:

```
docker compose down
```

To remove attached volumes as well:

```
docker compose down -v
```

5.2 Clearing Processed Datasets

If preprocessing must be repeated:

```
rm -rf data/processed/sz/*
rm -rf data/processed/los/*
rm -rf data/processed/pems08/*
```

Raw datasets stored in S3 do not need to be re-uploaded unless intentionally removed.

5.3 Removing Local Outputs

All accuracy, energy and privacy results can be deleted using:

```
rm -rf outputs/*
```

5.4 Resetting S3 Outputs

To remove remote output files generated by clients and the server:

```
aws s3 rm s3://<bucket-name>/outputs --recursive
```

5.5 Rebuilding the Environment

If major configuration changes occur, the system can be rebuilt as follows:

```
docker compose build
docker compose up
```

Python dependencies can be reinstalled using:

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

These steps restore the project to a fully operational state and ensure that subsequent runs proceed deterministically.

6 Troubleshooting and Known Issues

6.1 Common Docker-Related Issues

Permission Denied When Running Docker Commands

If commands fail with `permission denied while trying to connect to the Docker daemon`, ensure the user is added to the `docker` group and a new session has been started:

```
sudo usermod -aG docker ubuntu
newgrp docker
```

Alternatively, log out and log back in, or start a new SSH session.

Docker Compose Fails to Start Containers

This usually occurs due to port conflicts or stale containers. Clear existing containers and restart:

```
docker compose down --remove-orphans
docker compose up --build
```

Image Pull or Push Fails with Authentication Error

Re-authenticate Docker with Amazon ECR:

```
aws ecr get-login-password --region us-east-1 | \
docker login --username AWS --password-stdin \
<aws-account-id>.dkr.ecr.us-east-1.amazonaws.com
```

6.2 AWS and S3 Connectivity Issues

Unable to Access S3 Bucket

Verify credentials and region configuration:

```
aws sts get-caller-identity
aws s3 ls s3://<bucket-name>
```

If the bucket remains inaccessible, ensure:

- The AWS CLI is configured with valid credentials
- The IAM user has appropriate `s3:*` permissions on the target bucket
- The bucket region matches the configured default (us-east-1)

Retry settings can be increased if needed:

```
export AWS_MAX_ATTEMPTS=10
export AWS_RETRY_MODE=standard
```

6.3 Training and Runtime Issues

Clients Not Connecting to Server

Ensure the server is started *after* launching the clients. The server detects active clients via Docker inspection. If clients appear missing, verify they are running:

```
docker ps | grep client
```

Training Hangs Indefinitely

This usually indicates a client crash. Inspect client logs:

```
docker logs -f aefl-client-1
```

Common causes include:

- Out-of-memory errors (reduce batch size in configuration)
- Missing dataset partitions (re-run preprocessing)
- Network interruption between containers

6.4 Data and Preprocessing Issues

Preprocessing Script Cannot Find Raw Files

Confirm raw datasets are present in the S3 bucket under:

```
s3://<bucket-name>/data/raw/
```

Download and re-upload the files if necessary using the links provided in Section 2.

Missing Processed Data Directories

Re-run preprocessing after confirming raw data availability:

```
python -m data.preprocess_sz
python -m data.preprocess_los
python -m data.preprocess_pems08
```

6.5 Output and Plot Generation Issues

Plots Not Generated or Missing Files

Ensure all experiments completed successfully and outputs are synchronised from S3:

```
bash collect_energy.sh
bash collect_accuracy.sh
bash collect_privacy_dp.sh
```

Then regenerate plots:

```
python -m scripts.energy_all_datasets
python -m scripts.accuracy_all_metrics_all_dataset
python -m scripts.privacy_dp_plots
```