Quantization based Technique for Privacy Preserving Framework (installation notes)

System Requirements

- Java JDK 1.8
- Maven version $\geq$ 3.3.1
- Windows, Linux, or macOS
- Minimum 4GB RAM required

Project Structure

```
SecMLP_FedAVG4/
    |-- src/
          |-- main/java/
          |-- test/java/
    |-- data/
          |-- HFL/                    <- Place extracted .csv files here
          |-- Statistical/            <- Compressed datasets (.rar)
    |-- pom.xml
    |-- configuration.prop
    |-- doc/, tests/, VFL/, etc.
```

Installation Instructions

Clone or Set Up Project

```
git clone https://github.com/mauricolombo/milanHashCombFL.git
cd SecMLP_FedAVG4
```

Build the Project with Maven

```
mvn clean install
```

This will:

- Compile Java source files
- Download all dependencies (DL4J, Spark, etc.)
- Package the project into `target/MLP-0.0.1-SNAPSHOT.jar`

HashComb Project Integration

If `encoding.hash=true`, you must include HashComb JAR.

Build and Add JAR

```
C:\your_path_to_workspace\Hash-Comb\target\
```

In Eclipse:

- Right-click project → Build Path → Configure Build Path
- Add External JAR: `HashMap-0.0.1-SNAPSHOT.jar`

Install to Local Maven (Optional)

```
mvn install:install-file -Dfile=HashMap-0.0.1-SNAPSHOT.jar \
  -DgroupId=ebtic.labs.Federated \
  -DartifactId=hashcomb \
  -Dversion=0.0.1-SNAPSHOT \
  -Dpackaging=jar
```

POM Dependency

```
<dependency>
  <groupId>ebtic.labs.Federated</groupId>
  <artifactId>hashcomb</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</dependency>
```

Dataset Preparation

Compressed datasets are located in:

```
data/Statistical/
```

To prepare:

1. Extract `.rar` file (e.g., `ijCNN.rar`)
2. Copy all `.csv` files into `data/HFL/`
3. Ensure files have headers and valid rows

Example (ijCNN)

```
data.file=ijCNN
```

```
data/HFL/ijCNN1.csv
       ijCNN2.csv
       ...
```

Running the Aggregation Server

Launch the Server

```
java -cp target/MLP-0.0.1-SNAPSHOT.jar \
  ebtic.labs.Federated.threads.scenario.ServerManager
```

Configured IP/Port: `localhost:4444`

Optional: Load Initial Weights

```
mlp.init.file=init_weights.ser
```

Running Clients

In this framework, clients simulate distributed participants in a Federated Learning scenario. Two client modes are provided depending on how you organize your dataset and want the data to be partitioned across clients.

Client Option A: `ClientManager` — Manual Split Mode

```
java -cp target/MLP-0.0.1-SNAPSHOT.jar \
  ebtic.labs.Federated.threads.scenario.ClientManager
```

Purpose: This mode assumes that you have manually pre-split your dataset into distinct CSV files, one per client. Each file should contain only the subset of data assigned to that specific client.

When to use:

- When you want full control over how data is distributed among clients.
- When simulating non-IID (non-identically distributed) scenarios manually.
- When testing specific data imbalance or poisoning conditions.

Input format:

- Each file follows the format: `data/HFL/ijCNN.1.csv`, `ijCNN.2.csv`, ..., up to the number of clients.
- File naming must match the logical dataset name defined in `configuration.prop`.

Client Option B: `ClientManager2` — Auto-Split Mode

```
java -cp target/MLP-0.0.1-SNAPSHOT.jar \
  ebtic.labs.Federated.threads.scenario.ClientManager2
```

Purpose: This mode automatically loads the entire dataset and partitions it evenly (or randomly) among clients at runtime.

When to use:

- When you have a single dataset file or a folder of unsplit CSVs.
- For quick simulations without manual preprocessing.

- For benchmarking or synthetic client distribution.

Functionality:

- Merges all CSVs under `data/HFL/`.
- Splits the combined dataset into chunks based on the number of clients defined in `mlp.nodes`.
- Supports randomized distribution or balanced partitioning.

Summary:

Feature	`ClientManager`	`ClientManager2`
Data splitting	Manual (user-prepared files)	Automatic (runtime split)
Flexibility	High control over distribution (non-IID)	Fast setup, low effort
Use case	Custom research scenarios, adversarial testing	Rapid prototyping, demos
Input expected	Pre-split CSVs (1 per client)	Raw CSVs to be split

Detailed Explanation: configuration.prop

This file contains all the runtime configuration parameters needed to initialize and execute the Federated Learning project.

1. Model Configuration

```
        model.class = ebtic.labs.NN.dl4j.model.dl4jCNN
        model.classes = 43
```

   - `model.class`: Fully qualified class name of the model to use.
     - `ebtic.labs.NN.MLP` – Custom MLP implementation
     - `ebtic.labs.NN.dl4j.model.dl4jMLP` – DL4J MLP
     - `ebtic.labs.NN.dl4j.model.dl4jCNN` – DL4J CNN (used here)
   - `model.classes`: Number of output classes (e.g. 43 for GTSRB)

2. Dataset Configuration

```
    data.file = ijCNN
```

   - `data.file`: Logical name of the dataset. Expected CSVs must be placed under `data/HFL/`.

3. Federated Learning Parameters

```
mlp.nodes = 4
mlp.epochs = 40
mlp.iterations = 10
mlp.batch = 0.2
mlp.af = sigmoid
mlp.bias = true
```

   - `mlp.nodes`: Number of federated clients.
   - `mlp.epochs`: Local epochs per client per round.

- `mlp.iterations`: Total number of communication rounds.
- `mlp.batch`: Portion (0–1) of the dataset to use per epoch.
- `mlp.af`: Activation function (e.g., `sigmoid`, `tanh`, `relu`).
- `mlp.bias`: Whether to use bias terms in layers.

4. Learning Rate

```
learning.rate = 0.0001
```

- Learning rate used during training. Lower values provide more stable convergence.

5. Hash-Based Encoding (Optional)

```
encoding.hash = false
encoding.file = hashmap.ser
encoding.channels = 8
encoding.min = -0.35
encoding.max = +0.35
```

- `encoding.hash`: Enables HashComb if set to `true`.
- `encoding.file`: Path to serialized hash map.
- `encoding.channels`: Virtual channels used in hash compression.
- `encoding.min/max`: Clipping range for encoded weights.

6. Server Communication

```
global.server.ip = localhost
global.server.port = 4444
```

- Server IP/port used for client-server communication.

7. Weight Initialization (Optional)

```
mlp.init.file =
```

- If set, initial weights are loaded from this file.
- If blank, weights are initialized randomly.

Summary Table

Key	Type	Description
`model.class`	String	Model implementation class
`model.classes`	Integer	Number of output classes
`data.file`	String	Dataset name prefix (CSV files)
`mlp.nodes`	Integer	Number of participating clients
`mlp.epochs`	Integer	Local training epochs
`mlp.iterations`	Integer	Federated communication rounds
`mlp.batch`	Float (0–1)	Proportion of dataset per batch
`mlp.af`	String	Activation function
`mlp.bias`	Boolean	Whether to use bias
`learning.rate`	Float	Training learning rate
`encoding.hash`	Boolean	Enable hashed weight encoding
`encoding.file`	String	Path to encoded weights file
`encoding.channels`	Integer	Virtual channels in hash encoding
`encoding.min/max`	Float	Clipping range for encoding
`global.server.ip`	String	IP of aggregation server
`global.server.port`	Integer	Port of aggregation server
`mlp.init.file`	String	File path for initial weights (optional)

Example Configuration (configuration.prop)

```
model.class = ebtic.labs.NN.dl4j.model.dl4jCNN
model.classes = 43

data.file = ijCNN

mlp.nodes = 4
mlp.epochs = 40
mlp.iterations = 10
mlp.batch = 0.2
mlp.af = sigmoid
mlp.bias = true

learning.rate = 0.0001

encoding.hash = false
encoding.file = hashmap.ser
encoding.channels = 8
encoding.min = -0.35
encoding.max = +0.35

global.server.ip = localhost
global.server.port = 4444
```

In conclusion:

Once the server and clients are running:

- Clients train locally
- Server aggregates weights per round
- Final weights can be saved/exported