Federated Learning (FL) is a machine learning paradigm where multiple decentralized devices or servers collaboratively train a model without sharing their raw data. Instead of sending data to a central server, each device trains the model locally and only shares model updates (e.g., gradients or weights). These updates are aggregated on a central server to improve the global model.

This approach ensures privacy, as the raw data never leaves the local devices, making it suitable for applications where data is sensitive or distributed across many sources.

or,

Federated Learning is a way for many devices (like phones, laptops, or servers) to work together to train a shared machine learning model without sharing their personal data. Instead of sending data to a central place, each device trains the model using its own data and sends only the learning results (not the actual data) to a central server. These results are combined to improve the overall model.

Key Features

- **Decentralized Data:** Data remains on local devices or servers.
- **Privacy:** No raw data is shared; only model updates are sent.
- **Collaboration:** Multiple participants train a shared global model.
- **Scalability:** Can involve thousands or millions of devices.

Simple Example

Imagine your smartphone learning how you type to make better autocorrect or text suggestions:

- Your Phone Learns Locally: Your phone improves its typing prediction model based on how you type.
- Shares Only Model Updates: It sends the learning updates (not your actual texts) to a central server.
- Central Server Combines Updates: The server gathers updates from many phones and makes the model better for everyone.
- New Model Sent Back: The improved model is sent to your phone, making your predictions even smarter.

Another Example of Federated Learning

Smartphone Keyboard Predictions (e.g., Google's Gboard)

Scenario: Predictive text and autocorrect features.

How FL Works:

- Your smartphone trains a local model based on your typing patterns.
- The model updates are sent to a central server (not the raw text).
- The server aggregates updates from many users to improve the global model.
- The improved global model is sent back to devices.

This ensures your personal typing data stays on your phone, enhancing both privacy and user experience.

Advantages of Federated Learning

- Privacy Preservation: Sensitive data like medical records, personal texts, or financial data remains local.
- Reduced Bandwidth Usage: Only updates (smaller in size) are shared, not raw data.
- Regulatory Compliance: Helps organizations adhere to privacy laws like GDPR by avoiding data centralization.

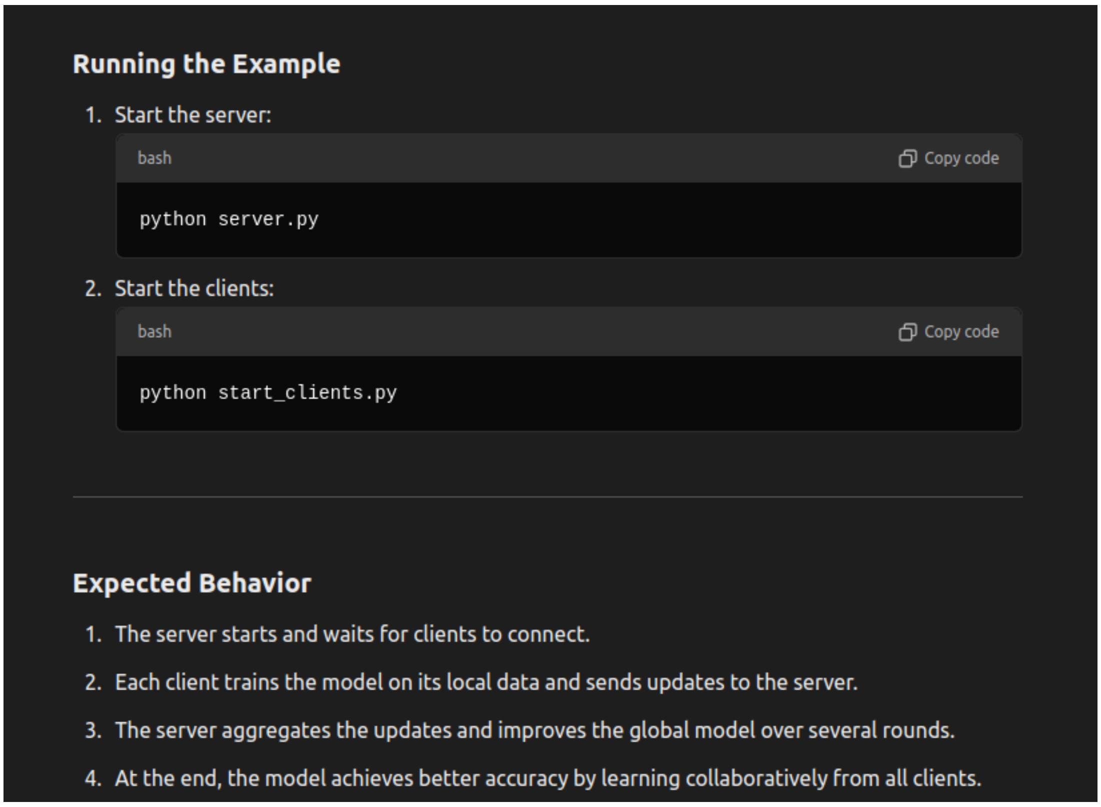
Challenges

- Heterogeneous Data: Data across devices may differ in quality or distribution.
- Device Variability: Devices may have different computational capabilities and connectivity.
- Communication Overhead: Sending and aggregating updates from many devices can be resource-intensive.

```
pip install flwr tensorflow
```

```
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.5.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from t
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.10/dist-packages (from cryptography<43.0.0,>=42.0.
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.0.8
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.13
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.31.0->
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich<14.0.0,>=13
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich<14.0.0,>=
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.1
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.1
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.10/dist-packages (from typer<0.13.0,>=0.12.5->fl
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from typer<0.13.0,>=0.12
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.12->cryptography<43.
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->ten
Downloading flwr-1.14.0-py3-none-any.whl (523 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 523.6/523.6 kB 10.5 MB/s eta 0:00:00
Downloading cryptography-42.0.8-cp39-abi3-manylinux_2_28_x86_64.whl (3.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.9/3.9 MB 51.4 MB/s eta 0:00:00
Downloading grpcio-1.64.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.6/5.6 MB 65.9 MB/s eta 0:00:00
Downloading iterators-0.0.2-py3-none-any.whl (3.9 kB)
Downloading pathspec-0.12.1-py3-none-any.whl (31 kB)
Downloading pycryptodome-3.21.0-cp36-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.3/2.3 MB 70.1 MB/s eta 0:00:00
Downloading tomli_w-1.1.0-py3-none-any.whl (6.4 kB)
Downloading typer-0.12.5-py3-none-any.whl (47 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.3/47.3 kB 3.5 MB/s eta 0:00:00
Installing collected packages: tomli-w, pycryptodome, pathspec, iterators, grpcio, cryptography, typer, flwr
  Attempting uninstall: grpcio
    Found existing installation: grpcio 1.68.1
    Uninstalling grpcio-1.68.1:
      Successfully uninstalled grpcio-1.68.1
  Attempting uninstall: cryptography
    Found existing installation: cryptography 43.0.3
    Uninstalling cryptography-43.0.3:
      Successfully uninstalled cryptography-43.0.3
  Attempting uninstall: typer
    Found existing installation: typer 0.15.1
    Uninstalling typer-0.15.1:
      Successfully uninstalled typer-0.15.1
```

```python
#server.py
import flwr as fl

# Start the server
if __name__ == "__main__":
    fl.server.start_server(
        server_address="localhost:8080",
        config=fl.server.ServerConfig(num_rounds=3),  # Number of training rounds
    )



#client.py
import flwr as fl
import tensorflow as tf
from tensorflow.keras.datasets import mnist

# Load and preprocess data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax"),
])
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])

class MnistClient(fl.client.NumPyClient):
    def get_parameters(self):
        return model.get_weights()

    def set_parameters(self, parameters):
        model.set_weights(parameters)

    def fit(self, parameters, config):
        self.set_parameters(parameters)
        model.fit(x_train, y_train, epochs=1, batch_size=32, verbose=0)
        return self.get_parameters(), len(x_train), {}

    def evaluate(self, parameters, config):
        self.set_parameters(parameters)
        loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
        return loss, len(x_test), {"accuracy": accuracy}

# Start the client
if __name__ == "__main__":
    fl.client.start_numpy_client(server_address="localhost:8080", client=MnistClient())



#start_clients.py

import subprocess

# Start multiple clients
for i in range(2):  # Number of clients
    subprocess.Popen(["python", "client.py"])
```



Running the Example

1. Start the server:

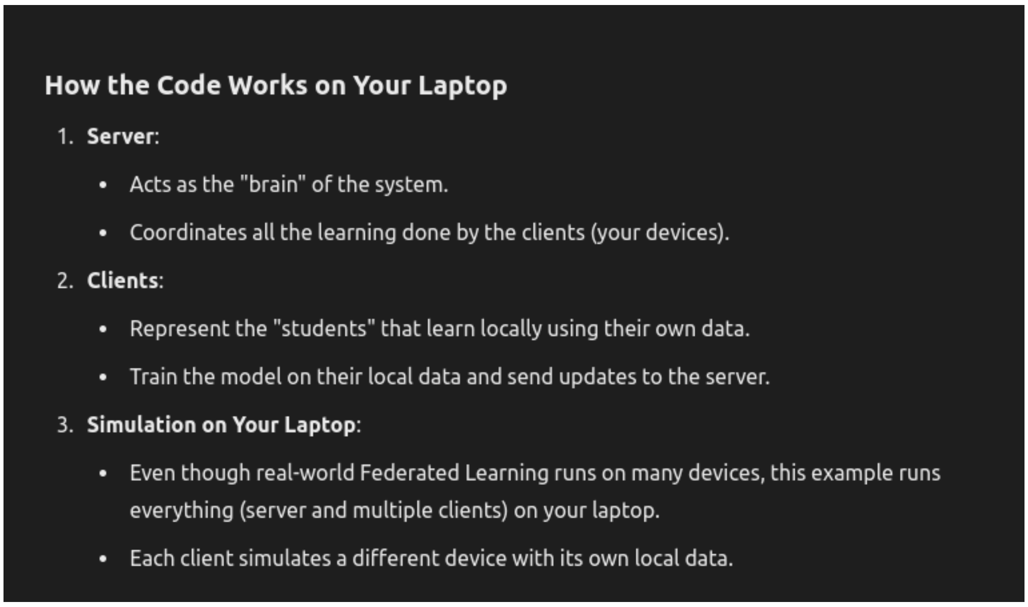
```bash
python server.py
```

2. Start the clients:

```bash
python start_clients.py
```

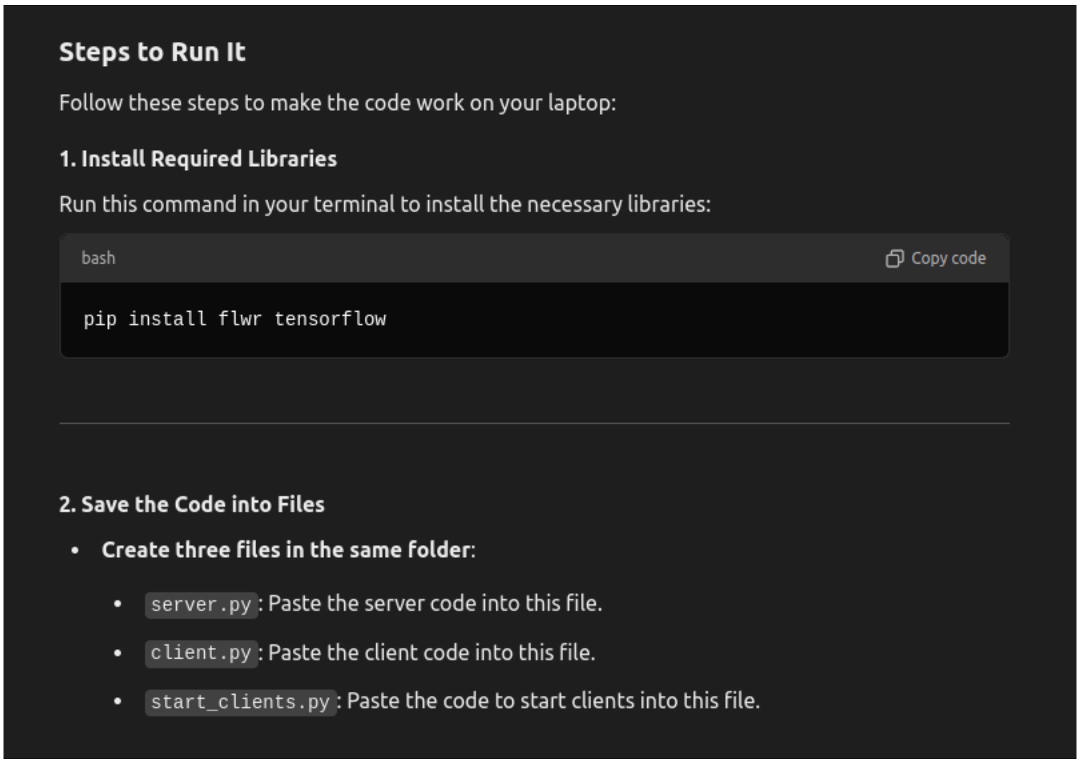
Expected Behavior

1. The server starts and waits for clients to connect.
2. Each client trains the model on its local data and sends updates to the server.
3. The server aggregates the updates and improves the global model over several rounds.
4. At the end, the model achieves better accuracy by learning collaboratively from all clients.



How the Code Works on Your Laptop

1. **Server**:
   - Acts as the "brain" of the system.
   - Coordinates all the learning done by the clients (your devices).
2. **Clients**:
   - Represent the "students" that learn locally using their own data.
   - Train the model on their local data and send updates to the server.
3. **Simulation on Your Laptop**:
   - Even though real-world Federated Learning runs on many devices, this example runs everything (server and multiple clients) on your laptop.
   - Each client simulates a different device with its own local data.



Steps to Run It

Follow these steps to make the code work on your laptop:

1. Install Required Libraries

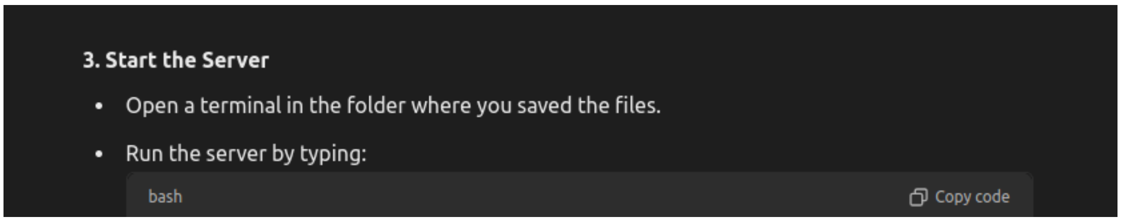
Run this command in your terminal to install the necessary libraries:

```bash
pip install flwr tensorflow
```

---

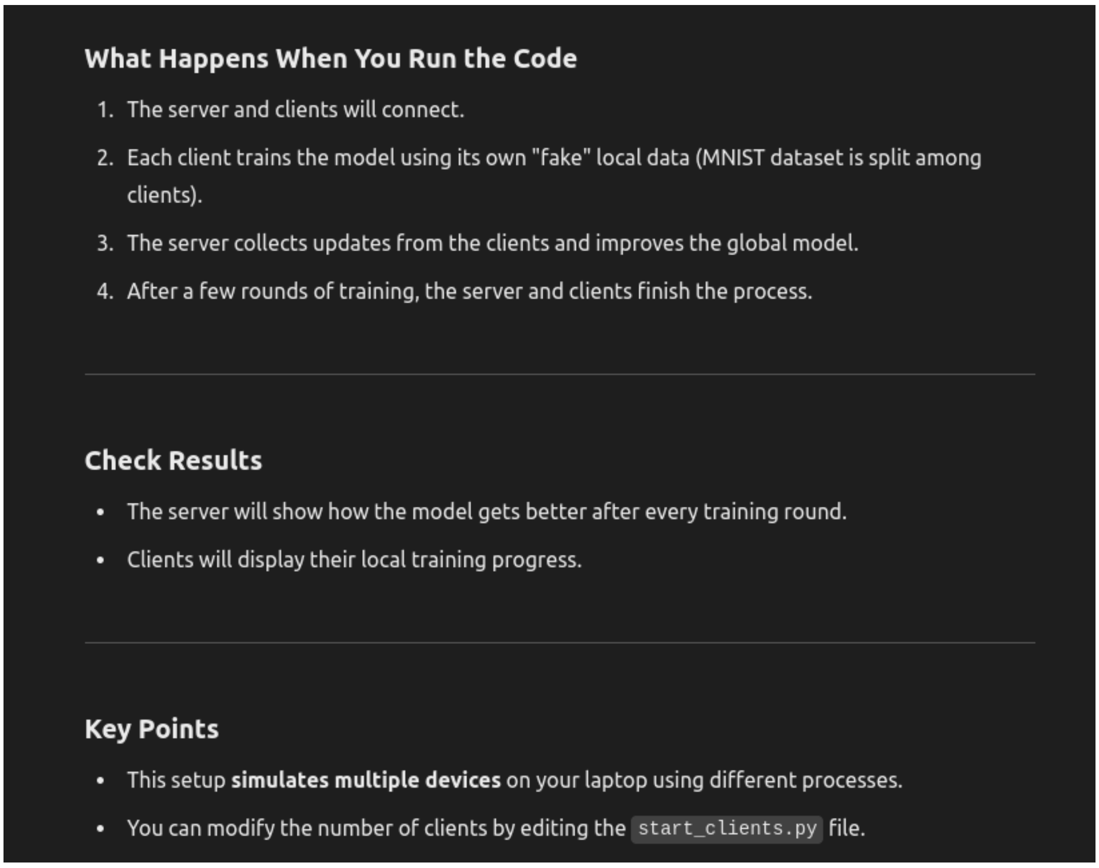
2. Save the Code into Files

- **Create three files in the same folder**:
  - `server.py`: Paste the server code into this file.
  - `client.py`: Paste the client code into this file.
  - `start_clients.py`: Paste the code to start clients into this file.



3. Start the Server

- Open a terminal in the folder where you saved the files.
- Run the server by typing:



What Happens When You Run the Code

1. The server and clients will connect.
2. Each client trains the model using its own "fake" local data (MNIST dataset is split among clients).
3. The server collects updates from the clients and improves the global model.
4. After a few rounds of training, the server and clients finish the process.

Check Results

- The server will show how the model gets better after every training round.
- Clients will display their local training progress.

Key Points

- This setup **simulates multiple devices** on your laptop using different processes.
- You can modify the number of clients by editing the `start_clients.py` file.