Code:

import io

import json

import torch

import wandb

from models.CreditFraudNetMLP import CreditFraudNetMLP

from pandas import DataFrame

from torch import nn

from torch.utils.data import DataLoader, TensorDataset

class ExampleTorchModal:

    def \_\_init\_\_(self, features, dataset\_path, epochs=1, lr=0.3) -> None:

        self.epoch = epochs

        self.features = features

        self.lr = lr

        self.dataset\_path = dataset\_path

        if torch.cuda.is\_available():

            device = torch.device("cuda")

        else:

            device = torch.device("cpu")

        self.device = torch.device(device)

        # Initialize the dataset

        self.init\_dataset(self.dataset\_path)

        # Weights & Biases

        wandb.init(project="Credit\_Card\_Flock")

    def init\_dataset(self, dataset\_path: str) -> None:

        self.dataset\_path = dataset\_path

        with open(dataset\_path, "r") as f:

            dataset = json.load(f)

        dataset\_df = DataFrame.from\_records(dataset)

        batch\_size = 128

        X\_df = dataset\_df.iloc[:, :-1]

        y\_df = dataset\_df.iloc[:, -1]

        X\_tensor = torch.tensor(X\_df.values, dtype=torch.float32)

        y\_tensor = torch.tensor(y\_df.values, dtype=torch.float32)

        y\_tensor = y\_tensor.unsqueeze(1)

        dataframe\_in\_dataset = TensorDataset(X\_tensor, y\_tensor)

        self.train\_data\_loader = DataLoader(

            dataframe\_in\_dataset,

            batch\_size=batch\_size,

            shuffle=True,

            drop\_last=True,

        )

        self.test\_data\_loader = DataLoader(

            dataframe\_in\_dataset,

            batch\_size=batch\_size,

            shuffle=True,

            drop\_last=False,

        )

    def train(self, parameters=None) -> bytes:

        model = CreditFraudNetMLP(self.features, 1)

        if parameters is not None:

            model.load\_state\_dict(torch.load(io.BytesIO(parameters)))

        model.train()

        optimizer = torch.optim.SGD(model.parameters(), lr=self.lr)

        criterion = nn.BCELoss()

        model.to(self.device)

        for epoch in range(1, self.epoch + 1):

            train\_loss = 0.0

            train\_correct = 0

            train\_total = 0

            for inputs, targets in self.train\_data\_loader:

                optimizer.zero\_grad()

                inputs, targets = inputs.to(self.device), targets.to(self.device)

                outputs = model(inputs)

                loss = criterion(outputs, targets)

                loss.backward()

                optimizer.step()

                train\_loss += loss.item() \* inputs.size(0)

                predicted = torch.round(outputs).squeeze()

                train\_total += targets.size(0)

                train\_correct += (predicted == targets.squeeze()).sum().item()

            wandb.log(

                {

                    "epoch": epoch,

                    "loss": train\_loss / train\_total,

                    "accuracy": train\_correct / train\_total,

                }

            )

        buffer = io.BytesIO()

        torch.save(model.state\_dict(), buffer)

        return buffer.getvalue()

    def evaluate(self, parameters) -> float:

        criterion = nn.BCELoss()

        model = CreditFraudNetMLP(self.features, 1)

        if parameters is not None:

            model.load\_state\_dict(torch.load(io.BytesIO(parameters)))

        model.to(self.device)

        model.eval()

        test\_loss = 0.0

        test\_correct = 0

        test\_total = 0

        with torch.no\_grad():

            for inputs, targets in self.test\_data\_loader:

                inputs, targets = inputs.to(self.device), targets.to(self.device)

                outputs = model(inputs)

                loss = criterion(outputs, targets)

                test\_loss += loss.item() \* inputs.size(0)

                predicted = torch.round(outputs).squeeze()

                test\_total += targets.size(0)

                test\_correct += (predicted == targets.squeeze()).sum().item()

        accuracy = test\_correct / test\_total

        return accuracy

    def aggregate(self, parameters\_list) -> bytes:

        parameters\_list = [

            torch.load(io.BytesIO(parameters)) for parameters in parameters\_list

        ]

        averaged\_parames\_template = parameters\_list[0]

        for k in averaged\_parames\_template.keys():

            temp\_w = []

            for local\_w in parameters\_list:

                temp\_w.append(local\_w[k])

            averaged\_parames\_template[k] = sum(temp\_w) / len(temp\_w)

        buffer = io.BytesIO()

        torch.save(averaged\_parames\_template, buffer)

        aggregated\_parameters = buffer.getvalue()

        return aggregated\_parameters

if \_\_name\_\_ == "\_\_main\_\_":

    epochs = 10

    lr = 0.000001

    features = 30

    dataset\_path = "data/creditCard.json"

    model = ExampleTorchModal(features, dataset\_path, epochs=epochs, lr=lr)

    # Training the model

    trained\_parameters = model.train()

    # Evaluating the model

    accuracy = model.evaluate(trained\_parameters)

    print(f"Model accuracy: {accuracy \* 100:.2f}%")

    # For aggregation example, you need a list of parameters from different training sessions

    # Since we don't have multiple sessions, we'll simulate it with the same parameters

    parameters\_list = [trained\_parameters, trained\_parameters]  # Example list

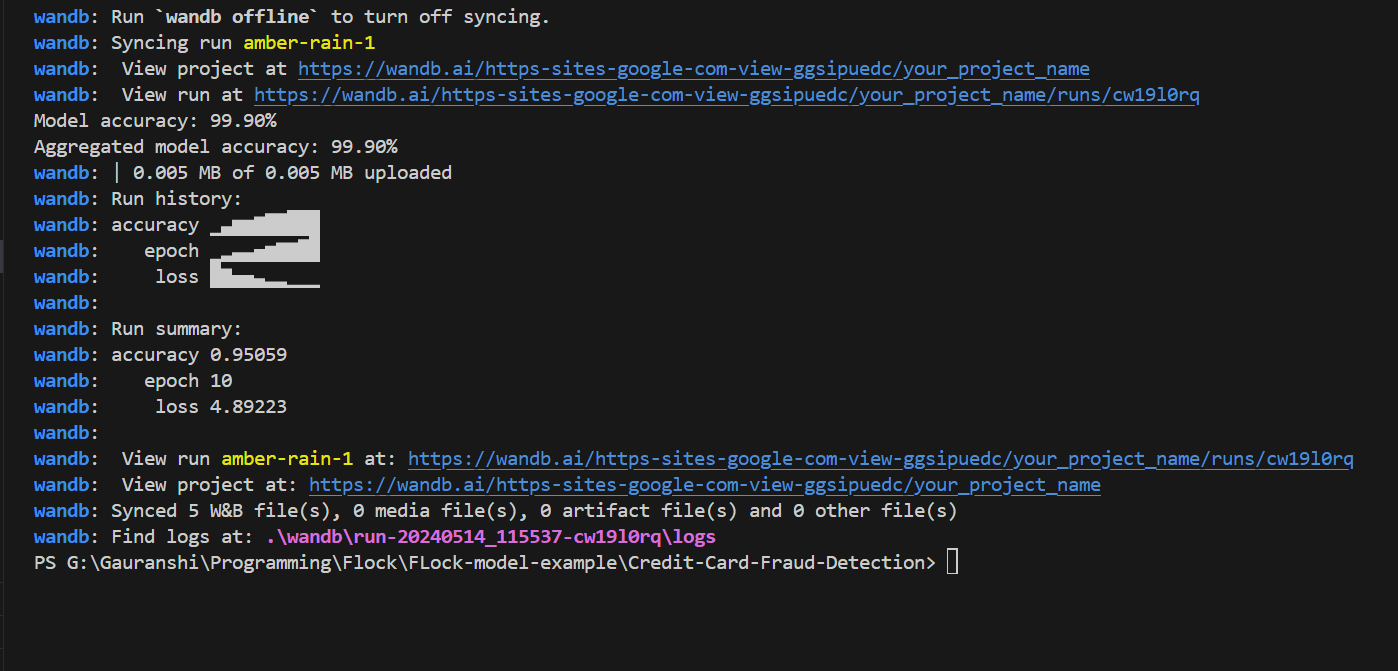
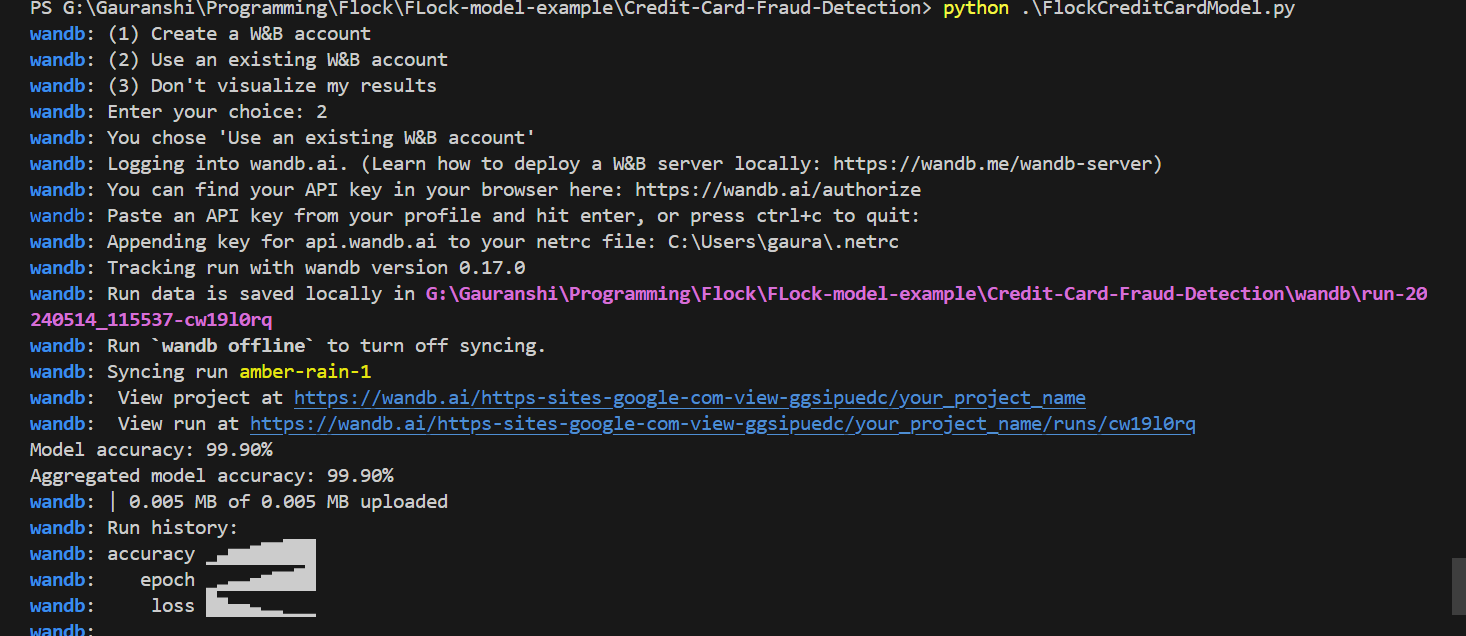
    aggregated\_parameters = model.aggregate(parameters\_list)

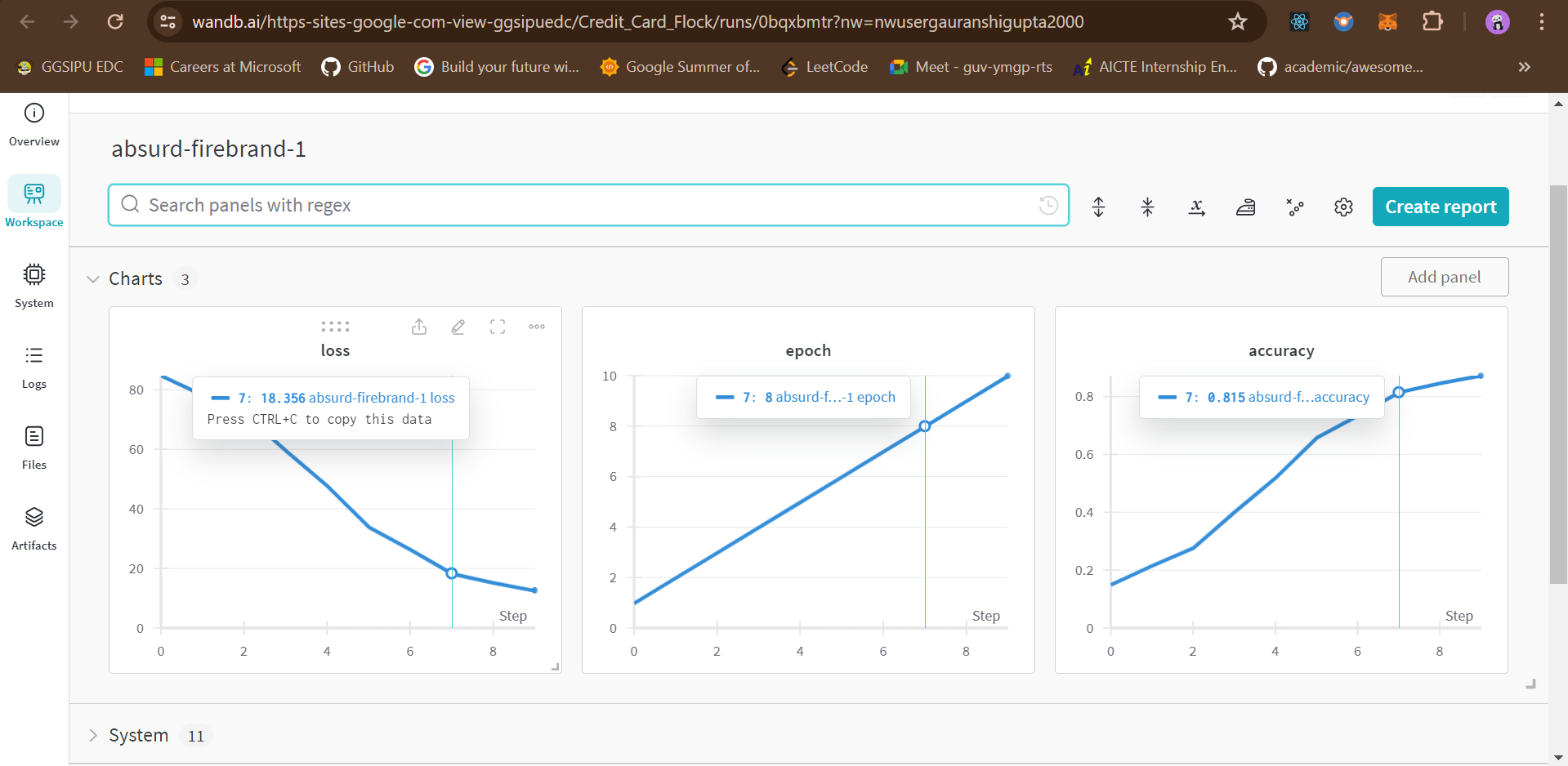
    # Re-evaluate the aggregated parameters

    aggregated\_accuracy = model.evaluate(aggregated\_parameters)

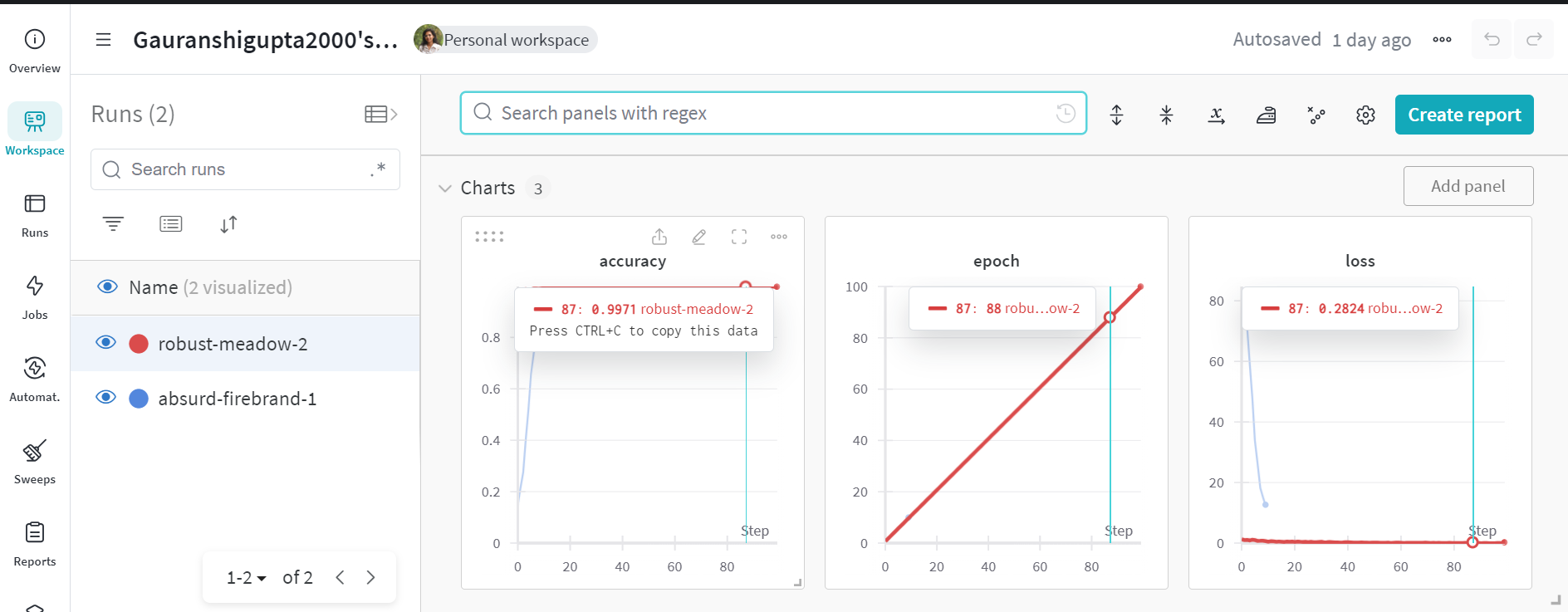
    print(f"Aggregated model accuracy: {aggregated\_accuracy \* 100:.2f}%")

Results:



For 10 epochs:

For 100 epochs:



Report Link:

<https://api.wandb.ai/links/https-sites-google-com-view-ggsipuedc/ef9jcitu>