Technical Interview

The prospective engineer will solve the following problem.

There is machine learning programming that trains on an existing dataset. You need to design a simple framework to do the following.

1. The algorithm should be able to run using a simple CLI
    a.

```
## this will run the training program of the machine learning
algorithm
mlrunner train --project_name demo_project --script ml_script.py
--learning_rate 0.01 --dataset path/to/dataset_train.csv

## this will run the testing program of the machine learning
algorithm
mlrunner test --project_name demo_project --script ml_script.py
--dataset path/to/dataset_test.csv

## this will generate a report in json format
## Report includes, training time, testing time, testing accuracy,
loss function value change
mlrunner report --project_name demo_project --format json
--output_path save/to/directory/

## Will start the UI resources and a django web application should be
accessible from the localhost.
mlrunner ui
```

2. The UI contains the following;
    a. Show projects created

i. When running the mlrunner train or test, in a database it will save the information related to each task. Training stage, we need you to save the loss function value for each epoch. The training program runs for 100 epochs. We need to see multiple attempts to run the project using different configurations. The only variable configuration for each run is the learning rate. We should be able to visualize each run's performance for training time, testing time, and graphs for loss function value change per epoch.

For a machine learning application, you can look into the sample code below. This is an example from this article https://towardsdatascience.com/logistic-regression-on-mnist-with-pytorch-b048327f8d19 Or you can choose your own code.

```
import torch
from torch.autograd import Variable
import torchvision.transforms as transforms
import torchvision.datasets as dsets




train_dataset = dsets.MNIST(root='./data', train=True,
transform=transforms.ToTensor(), download=True)
test_dataset = dsets.MNIST(root='./data', train=False,
transform=transforms.ToTensor())


batch_size = 100
n_iters = 3000 # number of iterations
epochs = n_iters / (len(train_dataset) / batch_size)
input_dim = 784
output_dim = 10
lr_rate = 0.001 # learning rate

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
batch_size=batch_size, shuffle=False)


class LogisticRegression(torch.nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LogisticRegression, self).__init__()
        self.linear = torch.nn.Linear(input_dim, output_dim)

    def forward(self, x):
        outputs = self.linear(x)
        return outputs


model = LogisticRegression(input_dim, output_dim)

criterion = torch.nn.CrossEntropyLoss()

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

iter = 0
for epoch in range(int(epochs)):
    for i, (images, labels) in enumerate(train_loader):
        images = Variable(images.view(-1, 28 * 28))
        labels = Variable(labels)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        iter+=1
        if iter%500==0:
            # calculate Accuracy
            correct = 0
            total = 0
            for images, labels in test_loader:
                images = Variable(images.view(-1, 28*28))
                outputs = model(images)
                _, predicted = torch.max(outputs.data, 1)
                total+= labels.size(0)
                # for gpu, bring the predicted and labels back to cpu
fro python operations to work
                correct+= (predicted == labels).sum()
            accuracy = 100 * correct/total
            print("Iteration: {}. Loss: {}. Accuracy:
{}.".format(iter, loss.item(), accuracy))
```

This is an open question to evaluate your ability to design a system and improve around it.

You have 3 days to complete this task. We need the following as deliverables.

1. Functional code
2. If you do the task, partially document what you have done and explain your thought process.
3. You should have documentation to install the application or just run the application smoothly in an Ubuntu 20.04 environment. You can use packaging tools like Docker, Singularity to package it, but it is not compulsory.

Good Luck.