Midterm Notebook

Michael Schmidlin 4/6/23

In this notebook we will attempt to predict stock prices by using different types of RNN (Recurrent Neural Network). RNN's are most commonly used for NLP (Natural Language Processing) and other types of sequence related tasks. This makes them ideal for time series predictions.

Specifically for this task, we will attempt to predict stock price (closing price) only using historical stock price data. This means that we are relying solely in the algorithm to find patterns in the historical stock data.

In this notebook we will use regular RNN but also GRU and LSTM which are more advanced and also very popular kinds of RNN. We will be using Pytorch to implement the models.

The stock data will be retrieved through the use of Alpaca API. Alpaca, is an online brokerage that has stock data and free paper trading accounts with a full python library.

Import Necessary Libraries

`Market_Monitor` and `Stock_Trader` are both custom wrappers for Alpaca functionality that were developed for this project.

In addition `torch_utils` is a custom file with functions and models pre-defined in order to keep the notebook clean.

Becuase of these custom modules, `%load_ext autoreload` is almost required becuase it makes development much eaiser.

In [ ]:
```
%load_ext autoreload
%autoreload 2
```

In [ ]:
```
import time
from configs import TIMEZONE, LOG_FILE_NAME, set_logger
from datetime import datetime, timedelta
import pytz
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from alpaca.trading.client import TradingClient
from alpaca.trading.requests import MarketOrderRequest
from alpaca.trading.enums import OrderSide, TimeInForce
from alpaca.trading.requests import GetAssetsRequest
from alpaca.data.historical import StockHistoricalDataClient, CryptoHistoricalDataClie
from alpaca.data.requests import StockLatestQuoteRequest, StockBarsRequest, CryptoLate
```

```python
from alpaca.data.requests import CryptoBarsRequest
from alpaca.trading.models import Order
from alpaca.data.timeframe import TimeFrame
from my_secrets import ALPACA_API_BASE_URL, PAPER_API_ID, PAPER_SECRET_KEY
import logging
import plotly.express as px
import plotly.graph_objects as go
set_logger()
```

In [ ]:
```python
import torch
import torch.nn as nn
```

In [ ]:
```python
from Trade_Class import Stock_Trader, Crypto_Trader
from Market_Monitor import Market_Monitor
from ALGO_crossover import bars_df_filter_dates, add_sma_columns, add_sma_crossovers
```

In [ ]:
```python
from torch_utils import *
```

Get Market Data

We will use our Alpaca wrapper classes to get the stock data. We will start by using Apple stock data and fetching every day from 2015 to 2023.

In [ ]:
```python
stock_trader = Stock_Trader(PAPER_API_ID, PAPER_SECRET_KEY, paper=True)
monitor = Market_Monitor(stock_trader.trading_client, TIMEZONE)
```

In [ ]:
```python
start = datetime(year=2015, month=1, day=1, hour=0, minute=0, second=0)
end = datetime(year=2023, month=2, day=1, hour=0, minute=0, second=0)
bars_df = stock_trader.get_bars('AAPL', start=start, end=end, time_resolution='day')
bars_df.reset_index(inplace=True)
bars_df.sort_values(by=['timestamp'], ascending=True, inplace=True)
print(bars_df.shape)
display(bars_df.head())
```

```
(1804, 9)
```

	symbol	timestamp	open	high	low	close	volume	trade_count	vwap
0	AAPL	2015-12-01 05:00:00+00:00	118.75	118.81	116.86	117.34	34852374.0	187129.0	117.756760
1	AAPL	2015-12-02 05:00:00+00:00	117.05	118.11	116.08	116.28	33385643.0	180616.0	117.151198
2	AAPL	2015-12-03 05:00:00+00:00	116.55	116.79	114.22	115.20	41560785.0	245330.0	115.434888
3	AAPL	2015-12-04 05:00:00+00:00	115.29	119.25	115.11	119.03	57776977.0	307788.0	118.187290
4	AAPL	2015-12-07 05:00:00+00:00	118.98	119.86	117.81	118.28	32080754.0	190809.0	118.509111

Short descriptions:

- symbol: stock symbol

- timestamp: datetime information
- open: Open price of a stock on a particular day
- high: Highest price of a stock on a particular day
- low: lowest price of a stock on a particular day
- close: last price of a stock on a particular day
- volume: how many shares were traded on a particular day
- trade_count: how many trades were made on a particular day
- vwap: volume-weighted-price-average

Define Hyperparameters

We define our hypter parameters at the top of the notebook to make the logistics of model tuning easier. `DEVICE` is coded such that there will be no issues if you run on a computer with no NVIDIA GPU.

In [ ]:
```
SEQUENCE_LENGTH = 50 #the sequence length we will use for prediction
NUM_LAYERS = 2 #the number of layers to use in our RNN type models
HIDDEN_SIZE = 64 #the hidden size of our models
DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'
TEST_PCT = 0.25 #what percentage of our data to use for testing
BATCH_SIZE = 32 #batch size to use for training. Can make this smaller or bigger deper
EPOCHS = 100 #Number of epochs to train for
PRED_IDX = 0 # the index in the data to predict. This parameter will make more sense l
```

Create Dataset

Okay let's get into creating our dataset. First, we take our dataframe and just take the 'close' column. Then we turn it into a train array and a test array using our `TEST_PCT`. This is all done for us inside `prep_RNN_data`.

Then we take our train and test arrays and pass them into the `RNNDataset` constructor. `RNNDataset` is a custom dataclass inheriting from pytorch's own dataclass. This makes it easy and clean for the the next step.

The next step being to create pytorch DataLoaders from our datasets. In addition to the regular DataLoader we will wrap the DataLoader with our own custom `DeviceDataLoader`. This will automatically load our training and testing batches to our agnostic device.

Finall we check the shape of our first batch of data and we see that it is (`BATCH_SIZE`, `SEQUENCE_LENGTH`, num_features). Since we are just using 'close' price right now our number of featrues is 1.

In [ ]:
```
train_array, test_array = prep_RNN_data(bars_df, cols=['close'], test_pct=TEST_PCT, s(
train_array.shape, test_array.shape
```

Out[ ]:
```
((1353, 1), (451, 1))
```

In [ ]:
```
train_dataset = RNNDataset(train_array, SEQUENCE_LENGTH,)
test_dataset = RNNDataset(test_array, SEQUENCE_LENGTH,)

len(train_dataset), len(test_dataset)
```

Out[ ]:
```
(1302, 400)
```

In [ ]:
```
x_train, y_train = train_dataset[0]
x_train.shape, y_train.shape
```

Out[ ]:
```
(torch.Size([50, 1]), torch.Size([1]))
```

In [ ]:
```
train_loader = torch.utils.data.DataLoader(train_dataset, BATCH_SIZE, drop_last=True,
train_loader = DeviceDataLoader(train_loader, DEVICE)
test_loader = torch.utils.data.DataLoader(test_dataset, BATCH_SIZE, drop_last=True)
test_loader = DeviceDataLoader(test_loader, DEVICE)
```

In [ ]:
```
trainer_iter = iter(train_loader)
x_train, y_train = next(trainer_iter)
x_train.shape, y_train.shape
```

Out[ ]:
```
(torch.Size([32, 50, 1]), torch.Size([32, 1]))
```

Create Models

Here we get to the fun part. We will create our three models using custom classes housed in `torch_utils`. Since all three models work in a similar way, they also have similar hyperparemeters.

Becuase pytorch links optimizers to the model parameters in the background, we will need three different optimizers. However we only need one loss function. We will use MSE (Mean Squared Error) loss since it is a common standard for regression tasks.

In [ ]:
```
rnn_model = RNN(input_dim=1, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LAYE
rnn_model.to(DEVICE)
lstm_model = LSTM(input_dim=1, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LA
lstm_model.to(DEVICE)
gru_model = GRU(input_dim=1, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LAYE
gru_model.to(DEVICE)

loss_fn = torch.nn.MSELoss()

rnn_optimizer = torch.optim.Adam(rnn_model.parameters(), lr=0.001)
lstm_optimizer = torch.optim.Adam(lstm_model.parameters(), lr=0.001)
gru_optimizer = torch.optim.Adam(gru_model.parameters(), lr=0.001)
```

Train Models

Our model training will be wrapped in `fit_model`. `fit_model` goes through a standard pytorch training loop and records losses so that we can analyze them. It also print out the losses every `print_divider=10` epochs so we can see real time how it's learning.

In [ ]:
```
rnn_training_losses, rnn_testing_losses = fit_model(EPOCHS, rnn_model, train_loader, t
```

```
Epoch 0         Train MSE: 0.01414      Test MSE: 0.00014
Epoch 10        Train MSE: 0.00096      Test MSE: 0.00009
Epoch 20        Train MSE: 0.00087      Test MSE: 0.00008
Epoch 30        Train MSE: 0.00083      Test MSE: 0.00010
Epoch 40        Train MSE: 0.00095      Test MSE: 0.00014
Epoch 50        Train MSE: 0.00071      Test MSE: 0.00017
Epoch 60        Train MSE: 0.00075      Test MSE: 0.00027
Epoch 70        Train MSE: 0.00090      Test MSE: 0.00012
Epoch 80        Train MSE: 0.00078      Test MSE: 0.00010
Epoch 90        Train MSE: 0.00090      Test MSE: 0.00008
```

In [ ]:
```
lstm_training_losses, lstm_testing_losses = fit_model(EPOCHS, lstm_model, train_loader
```

```
Epoch 0         Train MSE: 0.03011      Test MSE: 0.00619
Epoch 10        Train MSE: 0.00154      Test MSE: 0.00015
Epoch 20        Train MSE: 0.00100      Test MSE: 0.00016
Epoch 30        Train MSE: 0.00089      Test MSE: 0.00030
Epoch 40        Train MSE: 0.00076      Test MSE: 0.00027
Epoch 50        Train MSE: 0.00067      Test MSE: 0.00012
Epoch 60        Train MSE: 0.00073      Test MSE: 0.00008
Epoch 70        Train MSE: 0.00084      Test MSE: 0.00008
Epoch 80        Train MSE: 0.00070      Test MSE: 0.00010
Epoch 90        Train MSE: 0.00064      Test MSE: 0.00015
```

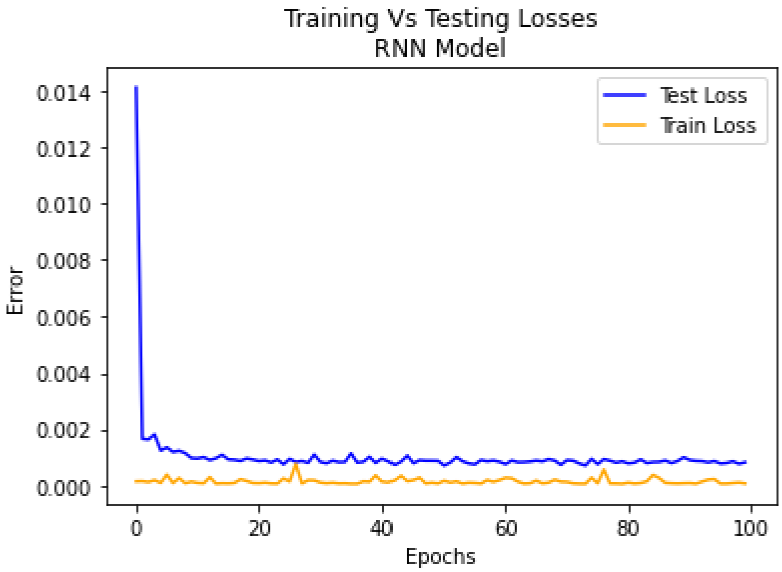
In [ ]:
```
gru_training_losses, gru_testing_losses = fit_model(EPOCHS, gru_model, train_loader, t
```

```
Epoch 0         Train MSE: 0.03283      Test MSE: 0.00852
Epoch 10        Train MSE: 0.00096      Test MSE: 0.00008
Epoch 20        Train MSE: 0.00083      Test MSE: 0.00009
Epoch 30        Train MSE: 0.00087      Test MSE: 0.00006
Epoch 40        Train MSE: 0.00086      Test MSE: 0.00009
Epoch 50        Train MSE: 0.00074      Test MSE: 0.00009
Epoch 60        Train MSE: 0.00078      Test MSE: 0.00016
Epoch 70        Train MSE: 0.00076      Test MSE: 0.00043
Epoch 80        Train MSE: 0.00073      Test MSE: 0.00012
Epoch 90        Train MSE: 0.00082      Test MSE: 0.00006
```

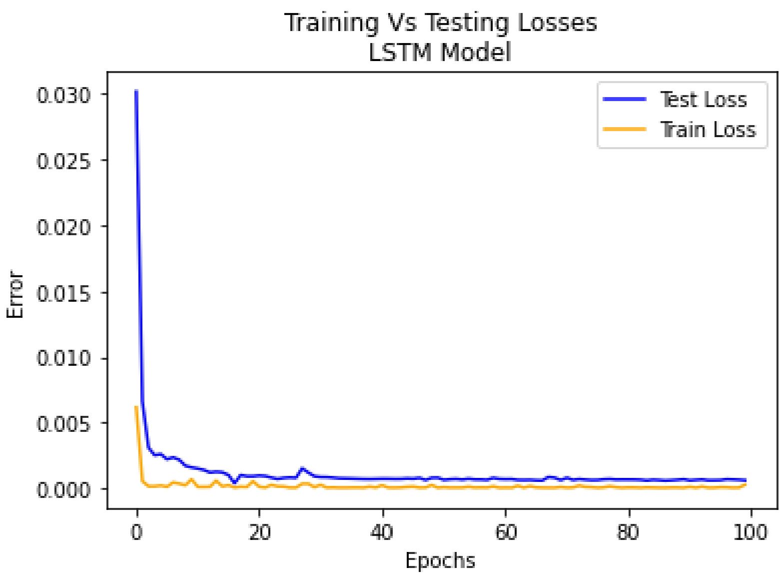
Visualizations

MSE Loss

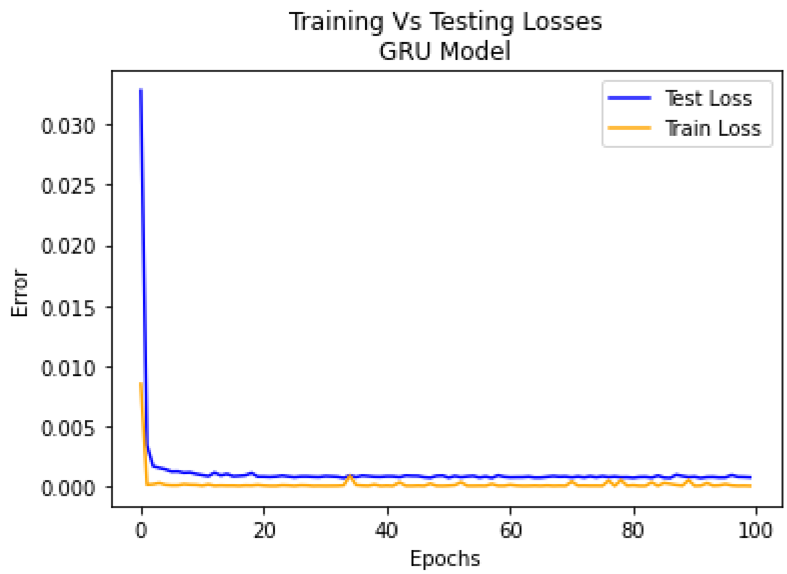
In [ ]:
```
plot_losses(EPOCHS, rnn_training_losses, rnn_testing_losses, title_addition='RNN Model
```



In [ ]:
```
plot_losses(EPOCHS, lstm_training_losses, lstm_testing_losses, title_addition='LSTM Mc
```



In [ ]:
```
plot_losses(EPOCHS, gru_training_losses, gru_testing_losses, title_addition='GRU Model
```



Predicted Closing Prices

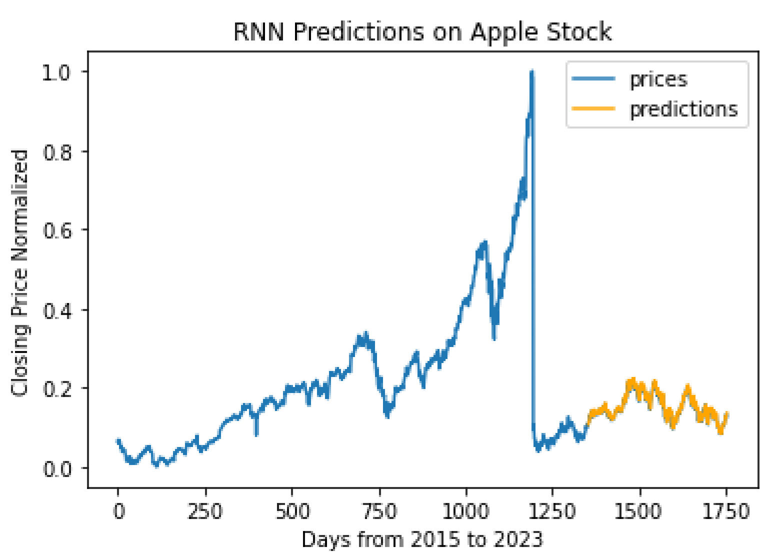
In addition to predicted closing prices, you can see that we have also measured the "accuracy" of the model. More on this later.

In [ ]:
```
historical_array = train_array[:, PRED_IDX]
historical_array.shape # get the closing prices of all the training data for context
```

Out[ ]:
```
(1353,)
```

In [ ]:
```
visualize_predictions(rnn_model, historical_array, test_dataset, DEVICE, title='RNN Pr
```

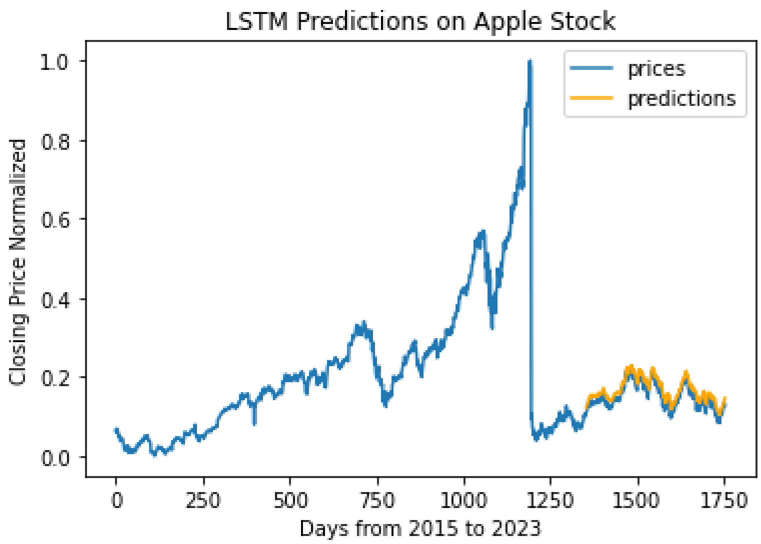
```
              precision    recall  f1-score   support

           0       0.33      0.04      0.07       195
           1       0.50      0.92      0.65       204

    accuracy                           0.49       399
   macro avg       0.42      0.48      0.36       399
weighted avg       0.42      0.49      0.37       399
```



Out[ ]: <Axes: title={'center': 'RNN Predictions on Apple Stock'}, xlabel='Days from 2015 to 2023', ylabel='Closing Price Normalized'>

In [ ]:
```
visualize_predictions(lstm_model, historical_array, test_dataset, DEVICE, title='LSTM
```

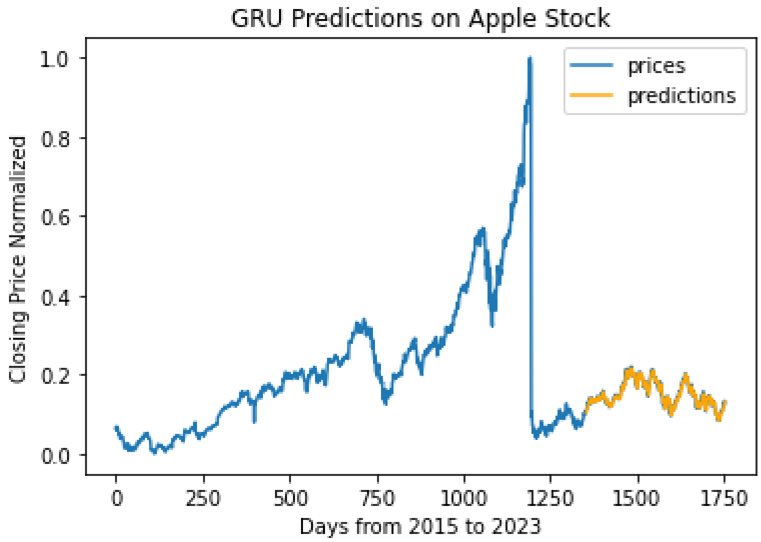
```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       195
           1       0.51      1.00      0.67       204

    accuracy                           0.51       399
   macro avg       0.26      0.50      0.34       399
weighted avg       0.26      0.51      0.34       399
```



Out[ ]: <Axes: title={'center': 'LSTM Predictions on Apple Stock'}, xlabel='Days from 2015 to 2023', ylabel='Closing Price Normalized'>

In [ ]:
```
visualize_predictions(gru_model, historical_array, test_dataset, DEVICE, title='GRU Pr
```

```
              precision    recall  f1-score   support

           0       0.47      0.69      0.56       195
           1       0.47      0.27      0.34       204

    accuracy                           0.47       399
   macro avg       0.47      0.48      0.45       399
weighted avg       0.47      0.47      0.45       399
```

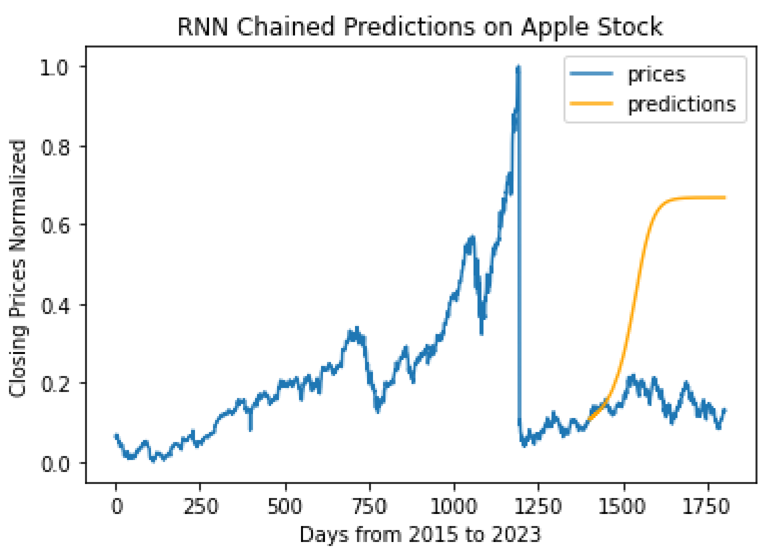


Out[ ]: <Axes: title={'center': 'GRU Predictions on Apple Stock'}, xlabel='Days from 2015 to 2023', ylabel='Closing Price Normalized'>

Chained Closing Prices

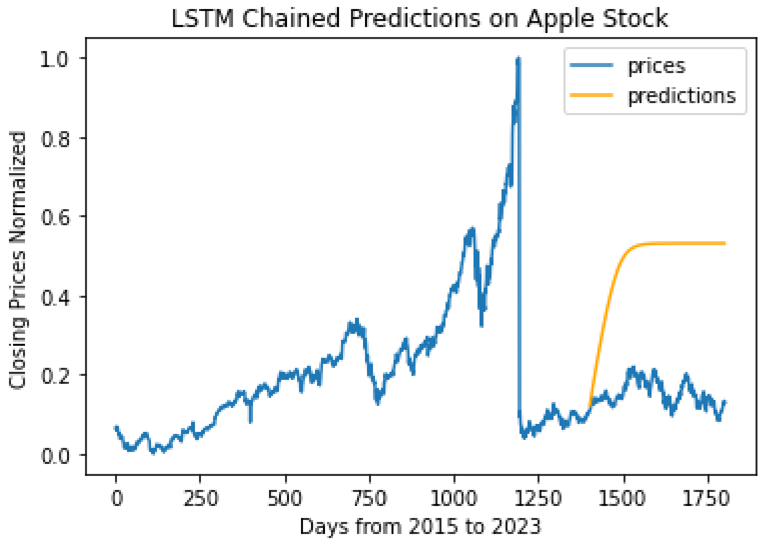
"Chained" Closing prices is essentially, rather than using the previous `SEQUENCE_LENGTH` of actual stock data, you use the previous `SEQUENCE_LENGTH` of *predictions*. Of course if your predictions don't go back as far as `SEQUENCE_LENGTH` then the beginning part of your predictor data is filled in with actual stock data. By doing this, we are sort of asking "how far in the future can we accurately predict?".

In [ ]:
```
visualize_chained_predictions(rnn_model, train_array, test_array, DEVICE, title='RNN C
```



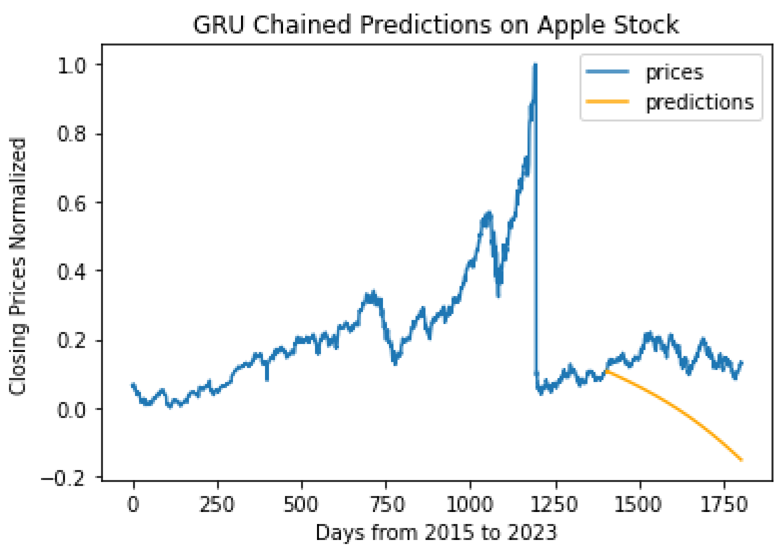
Out[ ]: <Axes: title={'center': 'RNN Chained Predictions on Apple Stock'}, xlabel='Days from 2015 to 2023', ylabel='Closing Prices Normalized'>

In [ ]:
```
visualize_chained_predictions(lstm_model, train_array, test_array, DEVICE, title='LSTM
```



Out[ ]:
```
<Axes: title={'center': 'LSTM Chained Predictions on Apple Stock'}, xlabel='Days from 2015 to 2023', ylabel='Closing Prices Normalized'>
```

In [ ]:
```
visualize_chained_predictions(gru_model, train_array, test_array, DEVICE, title='GRU C
```



Out[ ]:
```
<Axes: title={'center': 'GRU Chained Predictions on Apple Stock'}, xlabel='Days from 2015 to 2023', ylabel='Closing Prices Normalized'>
```

Hypter Parameter Tuning

We use the term hypterparameter tuning loosely here. What we will actually be changing is:

- Input feature space
- Number of layers
- Hidden dimension
- Predict a different stock
- what we are predicting?

Increase Input Feature Space

Now, instead of using just 'close' price to try and predict 'close' price. We will instead use five features: 'open', 'high', 'low', 'close', and 'volume'. The intent here is that hopefully the model will be able to predict better (or at all) what the next day stock price will be.

In [ ]:
```
PRED_IDX = 3 #set this since we are trying to predict 'close' price
```

In [ ]:
```
train_array, test_array = prep_RNN_data(bars_df, cols=['open', 'high', 'low', 'close',
                                        test_pct=TEST_PCT, scaler=MinMaxScaler())
train_array.shape, test_array.shape
```

Out[ ]:
```
((1353, 5), (451, 5))
```

In [ ]:
```
train_dataset = RNNDataset(train_array, SEQUENCE_LENGTH, pred_idx=PRED_IDX)
test_dataset = RNNDataset(test_array, SEQUENCE_LENGTH, pred_idx=PRED_IDX)

len(train_dataset), len(test_dataset)
```

Out[ ]:
```
(1302, 400)
```

In [ ]:
```
x_train, y_train = train_dataset[0]
x_train.shape, y_train.shape
```

Out[ ]:
```
(torch.Size([50, 5]), torch.Size([1]))
```

In [ ]:
```
train_loader = torch.utils.data.DataLoader(train_dataset, BATCH_SIZE, drop_last=True,
train_loader = DeviceDataLoader(train_loader, DEVICE)
test_loader = torch.utils.data.DataLoader(test_dataset, BATCH_SIZE, drop_last=True)
test_loader = DeviceDataLoader(test_loader, DEVICE)
```

In [ ]:
```
trainer_iter = iter(train_loader)
x_train, y_train = next(trainer_iter)
x_train.shape, y_train.shape
```

Out[ ]:
```
(torch.Size([32, 50, 5]), torch.Size([32, 1]))
```

In [ ]:
```
rnn_model = RNN(input_dim=5, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LAYE
rnn_model.to(DEVICE)
lstm_model = RNN(input_dim=5, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LAY
lstm_model.to(DEVICE)
gru_model = GRU(input_dim=5, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LAYE
gru_model.to(DEVICE)

loss_fn = torch.nn.MSELoss()

rnn_optimizer = torch.optim.Adam(rnn_model.parameters(), lr=0.001)
lstm_optimizer = torch.optim.Adam(lstm_model.parameters(), lr=0.001)
gru_optimizer = torch.optim.Adam(gru_model.parameters(), lr=0.001)
```

In [ ]:
```
rnn_training_losses, rnn_testing_losses = fit_model(EPOCHS, rnn_model, train_loader, t
```

```
Epoch 0         Train MSE: 0.00983      Test MSE: 0.00012
Epoch 10        Train MSE: 0.00088      Test MSE: 0.00012
Epoch 20        Train MSE: 0.00087      Test MSE: 0.00063
Epoch 30        Train MSE: 0.00086      Test MSE: 0.00010
Epoch 40        Train MSE: 0.00105      Test MSE: 0.00012
Epoch 50        Train MSE: 0.00079      Test MSE: 0.00010
Epoch 60        Train MSE: 0.00080      Test MSE: 0.00008
Epoch 70        Train MSE: 0.00095      Test MSE: 0.00054
Epoch 80        Train MSE: 0.00081      Test MSE: 0.00009
Epoch 90        Train MSE: 0.00090      Test MSE: 0.00021
```

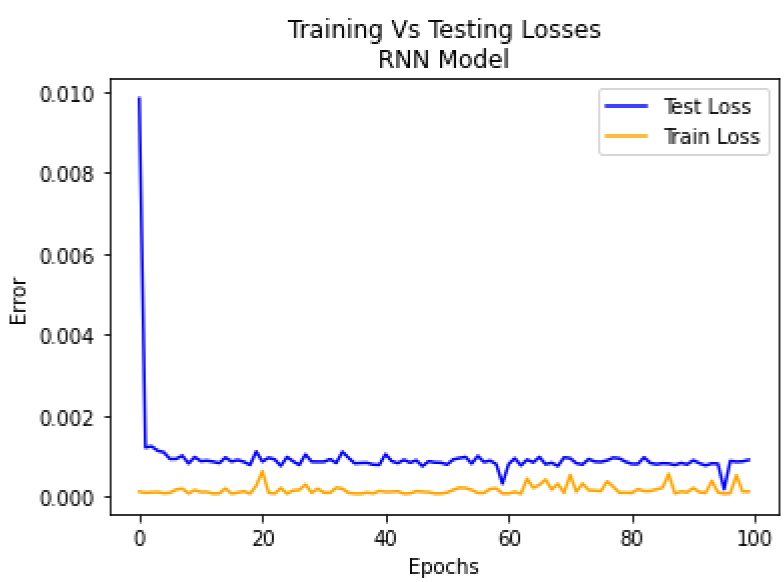
In [ ]:
```
lstm_training_losses, lstm_testing_losses = fit_model(EPOCHS, lstm_model, train_loader
```

```
Epoch 0         Train MSE: 0.01136      Test MSE: 0.00013
Epoch 10        Train MSE: 0.00088      Test MSE: 0.00011
Epoch 20        Train MSE: 0.00092      Test MSE: 0.00009
Epoch 30        Train MSE: 0.00090      Test MSE: 0.00011
Epoch 40        Train MSE: 0.00088      Test MSE: 0.00020
Epoch 50        Train MSE: 0.00087      Test MSE: 0.00014
Epoch 60        Train MSE: 0.00091      Test MSE: 0.00008
Epoch 70        Train MSE: 0.00084      Test MSE: 0.00015
Epoch 80        Train MSE: 0.00083      Test MSE: 0.00009
Epoch 90        Train MSE: 0.00077      Test MSE: 0.00012
```

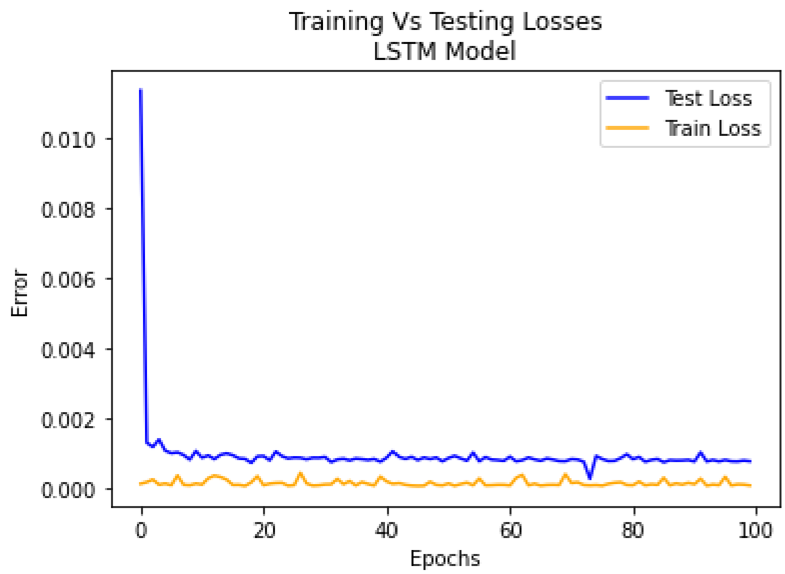
In [ ]:
```
gru_training_losses, gru_testing_losses = fit_model(EPOCHS, gru_model, train_loader, t
```

```
Epoch 0         Train MSE: 0.01200      Test MSE: 0.00027
Epoch 10        Train MSE: 0.00091      Test MSE: 0.00010
Epoch 20        Train MSE: 0.00097      Test MSE: 0.00017
Epoch 30        Train MSE: 0.00088      Test MSE: 0.00009
Epoch 40        Train MSE: 0.00085      Test MSE: 0.00032
Epoch 50        Train MSE: 0.00079      Test MSE: 0.00011
Epoch 60        Train MSE: 0.00083      Test MSE: 0.00008
Epoch 70        Train MSE: 0.00075      Test MSE: 0.00013
Epoch 80        Train MSE: 0.00088      Test MSE: 0.00009
Epoch 90        Train MSE: 0.00076      Test MSE: 0.00007
```

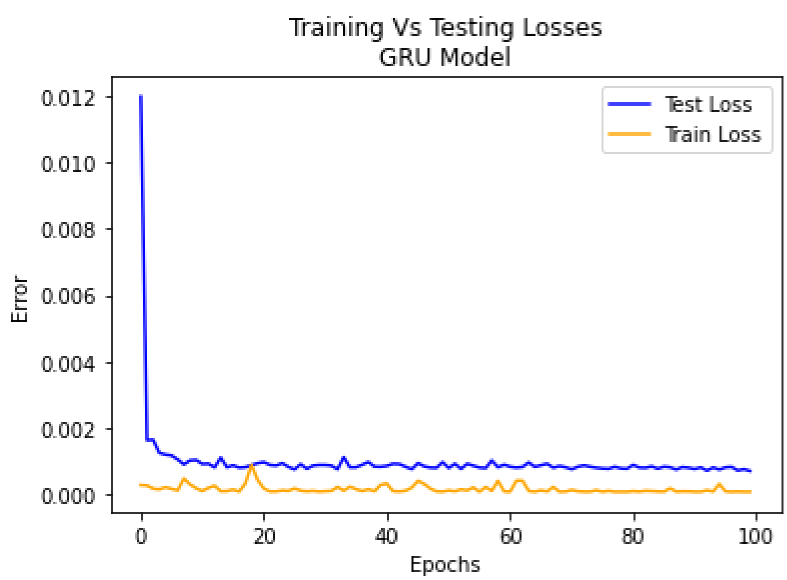
In [ ]:
```
plot_losses(EPOCHS, rnn_training_losses, rnn_testing_losses, title_addition='RNN Model
```



In [ ]:
```
plot_losses(EPOCHS, lstm_training_losses, lstm_testing_losses, title_addition='LSTM Mo
```



In [ ]:
```
plot_losses(EPOCHS, gru_training_losses, gru_testing_losses, title_addition='GRU Model
```

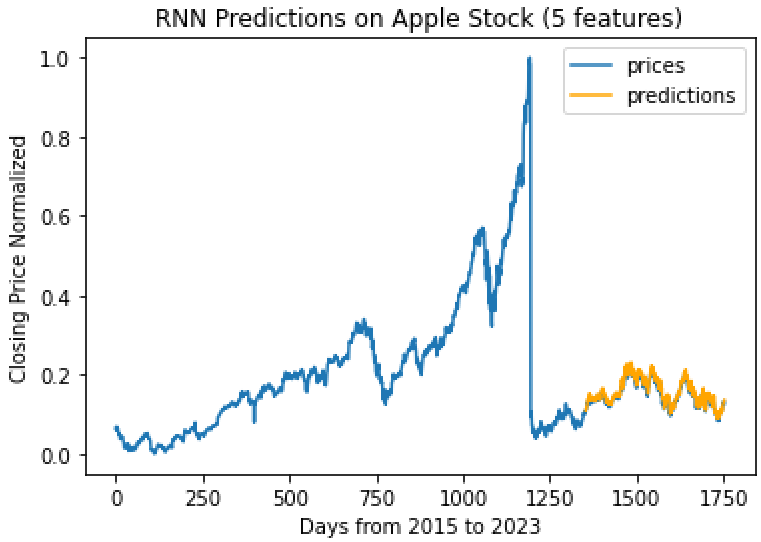


In [ ]:
```
historical_array = train_array[:, PRED_IDX]
historical_array.shape
```

Out[ ]:
```
(1353,)
```

In [ ]:
```
visualize_predictions(rnn_model, historical_array, test_dataset, DEVICE, title='RNN Pr
```

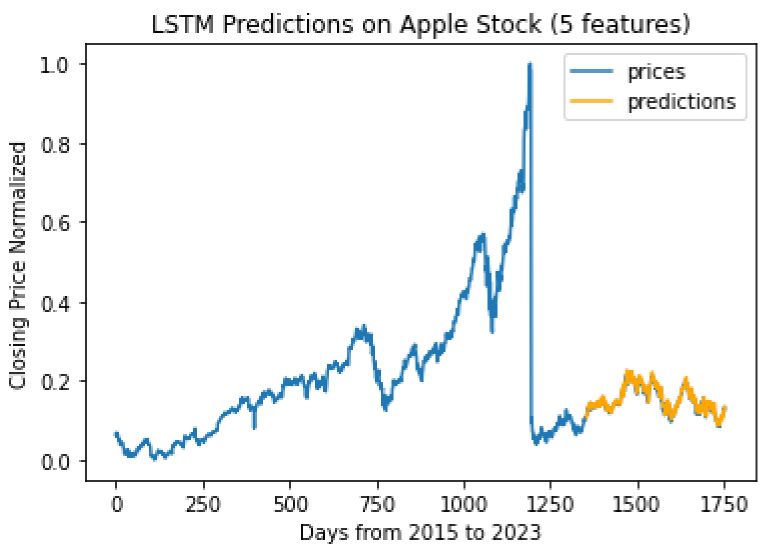
```
              precision    recall  f1-score   support

           0       0.34      0.06      0.10       195
           1       0.50      0.89      0.64       204

    accuracy                           0.48       399
   macro avg       0.42      0.47      0.37       399
weighted avg       0.42      0.48      0.38       399
```



Out[ ]: <Axes: title={'center': 'RNN Predictions on Apple Stock (5 features)'}, xlabel='Days from 2015 to 2023', ylabel='Closing Price Normalized'>

In [ ]:
```
visualize_predictions(lstm_model, historical_array, test_dataset, DEVICE, title='LSTM
```

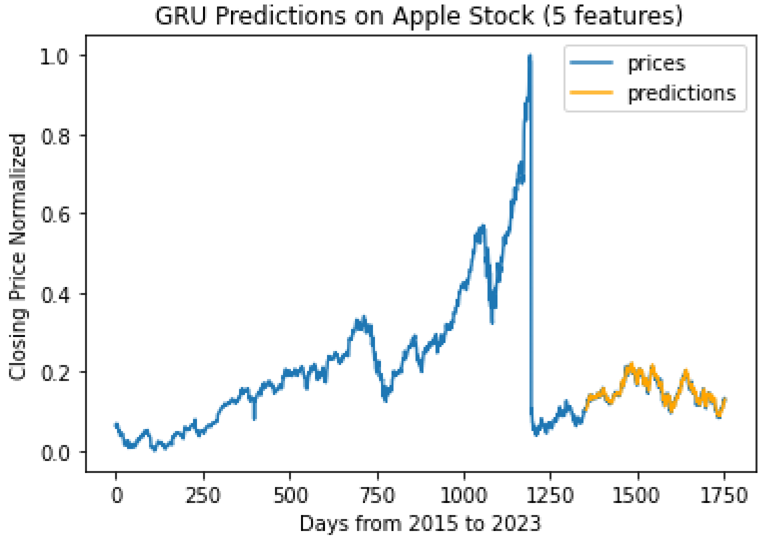
```
              precision    recall  f1-score   support

           0       0.46      0.24      0.31       195
           1       0.50      0.74      0.60       204

    accuracy                           0.49       399
   macro avg       0.48      0.49      0.46       399
weighted avg       0.48      0.49      0.46       399
```



Out[ ]: <Axes: title={'center': 'LSTM Predictions on Apple Stock (5 features)'}, xlabel='Days from 2015 to 2023', ylabel='Closing Price Normalized'>

In [ ]:
```
visualize_predictions(gru_model, historical_array, test_dataset, DEVICE, title='GRU Pr
```

```
              precision    recall  f1-score   support

           0       0.51      0.43      0.46       195
           1       0.53      0.61      0.56       204

    accuracy                           0.52       399
   macro avg       0.52      0.52      0.51       399
weighted avg       0.52      0.52      0.51       399
```



Out[ ]:
```
<Axes: title={'center': 'GRU Predictions on Apple Stock (5 features)'}, xlabel='Days
from 2015 to 2023', ylabel='Closing Price Normalized'>
```

We won't use the chained prediciton for the 5 features since it doesn't make sense for a model that predicts one feature using 5. It's simply not possible to chain predictions like we did before.

Increase Layers

Next, we will try increasing the number of layers. Since increasing features didn't help we will go back to a single feature. Thus, this won't be a true grid search but we will be individually trying different hypterparameters.

We will try setting `NUM_LAYERS` as 6 and hope that this increase in model complexity helps the models performance.

We will also discontinue plotting the loss functions this time since it wasn't very informative before.

In [ ]:
```
NUM_LAYERS = 6
PRED_IDX = 0
```

In [ ]:
```
train_array, test_array = prep_RNN_data(bars_df, cols=['close'], test_pct=TEST_PCT, sc
display(train_array.shape, test_array.shape)
train_dataset = RNNDataset(train_array, SEQUENCE_LENGTH,)
test_dataset = RNNDataset(test_array, SEQUENCE_LENGTH,)

display(len(train_dataset), len(test_dataset))
train_loader = torch.utils.data.DataLoader(train_dataset, BATCH_SIZE, drop_last=True,
train_loader = DeviceDataLoader(train_loader, DEVICE)
```

```
test_loader = torch.utils.data.DataLoader(test_dataset, BATCH_SIZE, drop_last=True)
test_loader = DeviceDataLoader(test_loader, DEVICE)
rnn_model = RNN(input_dim=1, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LAYE
rnn_model.to(DEVICE)
lstm_model = LSTM(input_dim=1, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LA
lstm_model.to(DEVICE)
gru_model = GRU(input_dim=1, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LAYE
gru_model.to(DEVICE)

loss_fn = torch.nn.MSELoss()

rnn_optimizer = torch.optim.Adam(rnn_model.parameters(), lr=0.001)
lstm_optimizer = torch.optim.Adam(lstm_model.parameters(), lr=0.001)
gru_optimizer = torch.optim.Adam(gru_model.parameters(), lr=0.001)
rnn_training_losses, rnn_testing_losses = fit_model(EPOCHS, rnn_model, train_loader, t
lstm_training_losses, lstm_testing_losses = fit_model(EPOCHS, lstm_model, train_loader
gru_training_losses, gru_testing_losses = fit_model(EPOCHS, gru_model, train_loader, t
```

```
(1353, 1)
(451, 1)
1302
400
Epoch 0         Train MSE: 0.02811      Test MSE: 0.00021
Epoch 10        Train MSE: 0.00110      Test MSE: 0.00006
Epoch 20        Train MSE: 0.00104      Test MSE: 0.00006
Epoch 30        Train MSE: 0.00100      Test MSE: 0.00012
Epoch 40        Train MSE: 0.00098      Test MSE: 0.00010
Epoch 50        Train MSE: 0.00073      Test MSE: 0.00008
Epoch 60        Train MSE: 0.00084      Test MSE: 0.00006
Epoch 70        Train MSE: 0.00092      Test MSE: 0.00006
Epoch 80        Train MSE: 0.00095      Test MSE: 0.00010
Epoch 90        Train MSE: 0.00093      Test MSE: 0.00006
Epoch 0         Train MSE: 0.03684      Test MSE: 0.00519
Epoch 10        Train MSE: 0.00233      Test MSE: 0.00033
Epoch 20        Train MSE: 0.00100      Test MSE: 0.00016
Epoch 30        Train MSE: 0.00077      Test MSE: 0.00014
Epoch 40        Train MSE: 0.00026      Test MSE: 0.00014
Epoch 50        Train MSE: 0.00073      Test MSE: 0.00012
Epoch 60        Train MSE: 0.00077      Test MSE: 0.00012
Epoch 70        Train MSE: 0.00091      Test MSE: 0.00027
Epoch 80        Train MSE: 0.00077      Test MSE: 0.00011
Epoch 90        Train MSE: 0.00097      Test MSE: 0.00012
Epoch 0         Train MSE: 0.02838      Test MSE: 0.00071
Epoch 10        Train MSE: 0.00088      Test MSE: 0.00068
Epoch 20        Train MSE: 0.00094      Test MSE: 0.00015
Epoch 30        Train MSE: 0.00096      Test MSE: 0.00037
Epoch 40        Train MSE: 0.00084      Test MSE: 0.00013
Epoch 50        Train MSE: 0.00101      Test MSE: 0.00045
Epoch 60        Train MSE: 0.00080      Test MSE: 0.00008
Epoch 70        Train MSE: 0.00119      Test MSE: 0.00024
Epoch 80        Train MSE: 0.00082      Test MSE: 0.00040
Epoch 90        Train MSE: 0.00069      Test MSE: 0.00007
```

In [ ]:

```
historical_array = train_array[:, PRED_IDX]
historical_array.shape # get the closing prices of all the training data for context
```

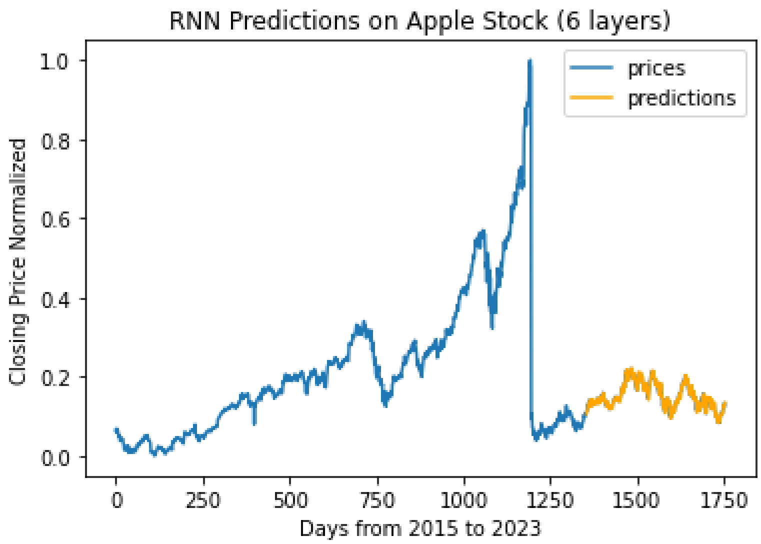
Out[ ]:

```
(1353,)
```

In [ ]:

```
visualize_predictions(rnn_model, historical_array, test_dataset, DEVICE, title='RNN Pr
```

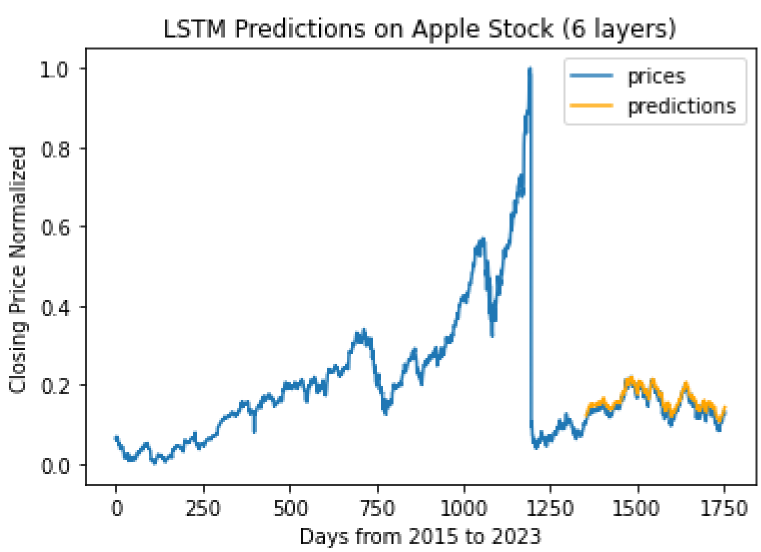
```
              precision    recall  f1-score   support

           0       0.48      0.50      0.49       195
           1       0.50      0.48      0.49       204

    accuracy                           0.49       399
   macro avg       0.49      0.49      0.49       399
weighted avg       0.49      0.49      0.49       399
```



Out[ ]:
```
<Axes: title={'center': 'RNN Predictions on Apple Stock (6 layers)'}, xlabel='Days fr
om 2015 to 2023', ylabel='Closing Price Normalized'>
```

In [ ]:
```
visualize_predictions(lstm_model, historical_array, test_dataset, DEVICE, title='LSTM
```

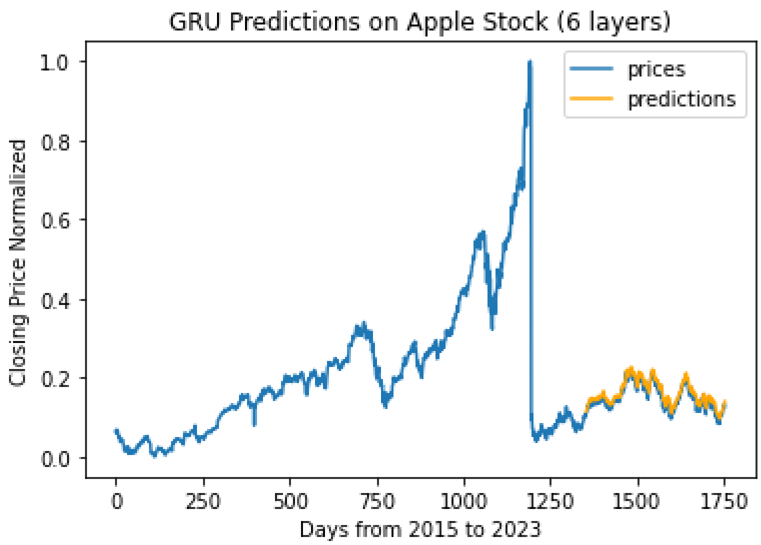
```
              precision    recall  f1-score   support

           0       0.74      0.07      0.13       195
           1       0.52      0.98      0.68       204

    accuracy                           0.53       399
   macro avg       0.63      0.52      0.41       399
weighted avg       0.63      0.53      0.41       399
```



Out[ ]:
```
<Axes: title={'center': 'LSTM Predictions on Apple Stock (6 layers)'}, xlabel='Days f
rom 2015 to 2023', ylabel='Closing Price Normalized'>
```

In [ ]:
```
visualize_predictions(gru_model, historical_array, test_dataset, DEVICE, title='GRU Pr
```

```
              precision    recall  f1-score   support

           0       0.33      0.01      0.01       195
           1       0.51      0.99      0.67       204

    accuracy                           0.51       399
   macro avg       0.42      0.50      0.34       399
weighted avg       0.42      0.51      0.35       399
```



Out[ ]:
```
<Axes: title={'center': 'GRU Predictions on Apple Stock (6 layers)'}, xlabel='Days fr
om 2015 to 2023', ylabel='Closing Price Normalized'>
```

Increase Hidden Dimension

Next, we will try increasing the hidden layer of the RNN. After increasing layers didn't work this is unlikely to work but we will try it anyway.

In [ ]:
```
NUM_LAYERS = 2
PRED_IDX = 0
HIDDEN_SIZE = 256
```

In [ ]:
```
train_array, test_array = prep_RNN_data(bars_df, cols=['close'], test_pct=TEST_PCT, sc
display(train_array.shape, test_array.shape)
train_dataset = RNNDataset(train_array, SEQUENCE_LENGTH,)
test_dataset = RNNDataset(test_array, SEQUENCE_LENGTH,)

display(len(train_dataset), len(test_dataset))
train_loader = torch.utils.data.DataLoader(train_dataset, BATCH_SIZE, drop_last=True,
train_loader = DeviceDataLoader(train_loader, DEVICE)
test_loader = torch.utils.data.DataLoader(test_dataset, BATCH_SIZE, drop_last=True)
test_loader = DeviceDataLoader(test_loader, DEVICE)
rnn_model = RNN(input_dim=1, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LAYE
rnn_model.to(DEVICE)
lstm_model = LSTM(input_dim=1, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LA
lstm_model.to(DEVICE)
gru_model = GRU(input_dim=1, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LAYE
gru_model.to(DEVICE)
```

```
loss_fn = torch.nn.MSELoss()

rnn_optimizer = torch.optim.Adam(rnn_model.parameters(), lr=0.001)
lstm_optimizer = torch.optim.Adam(lstm_model.parameters(), lr=0.001)
gru_optimizer = torch.optim.Adam(gru_model.parameters(), lr=0.001)
rnn_training_losses, rnn_testing_losses = fit_model(EPOCHS, rnn_model, train_loader, t
lstm_training_losses, lstm_testing_losses = fit_model(EPOCHS, lstm_model, train_loader
gru_training_losses, gru_testing_losses = fit_model(EPOCHS, gru_model, train_loader, t
```

```
(1353, 1)
(451, 1)
1302
400
Epoch 0         Train MSE: 0.01416      Test MSE: 0.00036
Epoch 10        Train MSE: 0.00110      Test MSE: 0.00014
Epoch 20        Train MSE: 0.00124      Test MSE: 0.00029
Epoch 30        Train MSE: 0.00097      Test MSE: 0.00006
Epoch 40        Train MSE: 0.00099      Test MSE: 0.00012
Epoch 50        Train MSE: 0.00089      Test MSE: 0.00107
Epoch 60        Train MSE: 0.00118      Test MSE: 0.00010
Epoch 70        Train MSE: 0.00109      Test MSE: 0.00006
Epoch 80        Train MSE: 0.03249      Test MSE: 0.01004
Epoch 90        Train MSE: 0.02154      Test MSE: 0.00179
Epoch 0         Train MSE: 0.02940      Test MSE: 0.00901
Epoch 10        Train MSE: 0.00181      Test MSE: 0.00018
Epoch 20        Train MSE: 0.00095      Test MSE: 0.00011
Epoch 30        Train MSE: 0.00092      Test MSE: 0.00010
Epoch 40        Train MSE: 0.00078      Test MSE: 0.00007
Epoch 50        Train MSE: 0.00077      Test MSE: 0.00013
Epoch 60        Train MSE: 0.00080      Test MSE: 0.00007
Epoch 70        Train MSE: 0.00083      Test MSE: 0.00007
Epoch 80        Train MSE: 0.00086      Test MSE: 0.00008
Epoch 90        Train MSE: 0.00079      Test MSE: 0.00008
Epoch 0         Train MSE: 0.01101      Test MSE: 0.00057
Epoch 10        Train MSE: 0.00116      Test MSE: 0.00011
Epoch 20        Train MSE: 0.00105      Test MSE: 0.00012
Epoch 30        Train MSE: 0.00083      Test MSE: 0.00007
Epoch 40        Train MSE: 0.00097      Test MSE: 0.00007
Epoch 50        Train MSE: 0.00071      Test MSE: 0.00011
Epoch 60        Train MSE: 0.00089      Test MSE: 0.00033
Epoch 70        Train MSE: 0.00081      Test MSE: 0.00006
Epoch 80        Train MSE: 0.00091      Test MSE: 0.00007
Epoch 90        Train MSE: 0.00090      Test MSE: 0.00006
```

In [ ]:

```
historical_array = train_array[:, PRED_IDX]
historical_array.shape # get the closing prices of all the training data for context
```

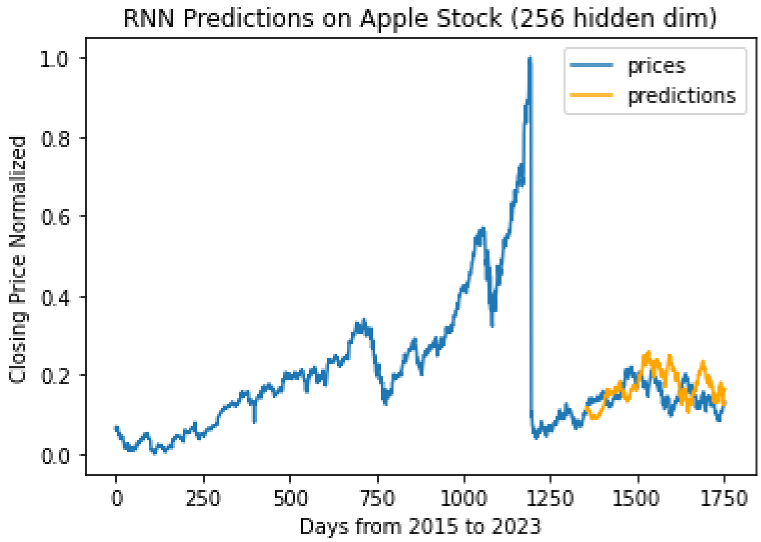
Out[ ]:

```
(1353,)
```

In [ ]:

```
visualize_predictions(rnn_model, historical_array, test_dataset, DEVICE, title='RNN Pr
```

```
              precision    recall  f1-score   support

           0       0.51      0.38      0.44       195
           1       0.52      0.65      0.58       204

    accuracy                           0.52       399
   macro avg       0.52      0.52      0.51       399
weighted avg       0.52      0.52      0.51       399
```



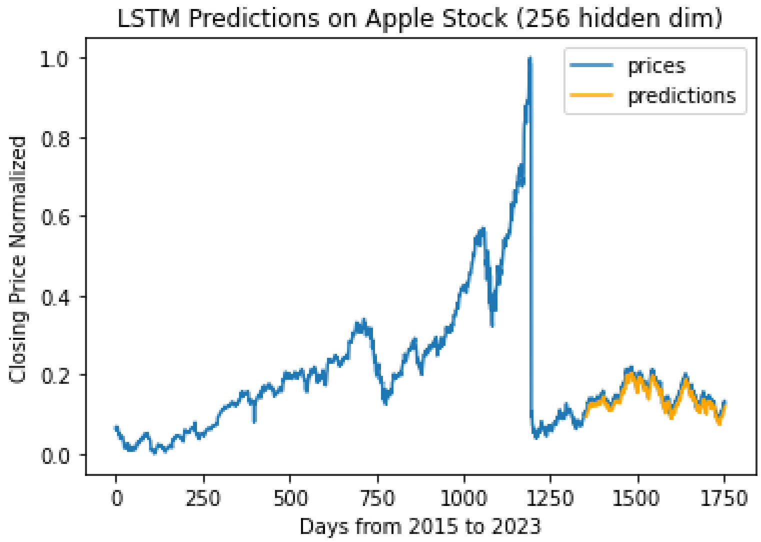
Out[ ]:

```
<Axes: title={'center': 'RNN Predictions on Apple Stock (256 hidden dim)'}, xlabel='D
ays from 2015 to 2023', ylabel='Closing Price Normalized'>
```

In [ ]:

```
visualize_predictions(lstm_model, historical_array, test_dataset, DEVICE, title='LSTM
```

```
c:\Users\mschm\Desktop\Masters_DU\Capstone_Trade_bot_project\capenv\lib\site-packages
\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-sco
re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zer
o_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\mschm\Desktop\Masters_DU\Capstone_Trade_bot_project\capenv\lib\site-packages
\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-sco
re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zer
o_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\mschm\Desktop\Masters_DU\Capstone_Trade_bot_project\capenv\lib\site-packages
\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-sco
re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zer
o_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

           0       0.49      1.00      0.66       195
           1       0.00      0.00      0.00       204

    accuracy                           0.49       399
   macro avg       0.24      0.50      0.33       399
weighted avg       0.24      0.49      0.32       399
```

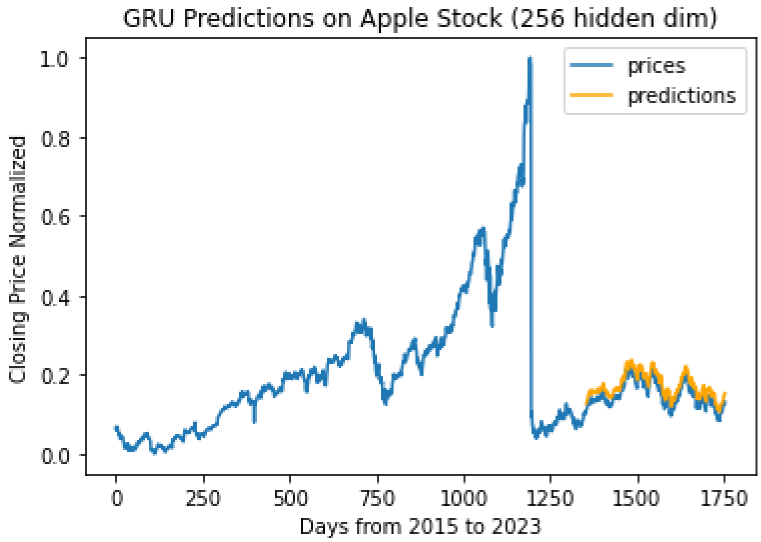


Out[ ]:
```
<Axes: title={'center': 'LSTM Predictions on Apple Stock (256 hidden dim)'}, xlabel
='Days from 2015 to 2023', ylabel='Closing Price Normalized'>
```

In [ ]:
```
visualize_predictions(gru_model, historical_array, test_dataset, DEVICE, title='GRU Pr
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       195
           1       0.51      1.00      0.68       204

    accuracy                           0.51       399
   macro avg       0.26      0.50      0.34       399
weighted avg       0.26      0.51      0.35       399
```

```
c:\Users\mschm\Desktop\Masters_DU\Capstone_Trade_bot_project\capenv\lib\site-packages
\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-sco
re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zer
o_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\mschm\Desktop\Masters_DU\Capstone_Trade_bot_project\capenv\lib\site-packages
\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-sco
re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zer
o_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\mschm\Desktop\Masters_DU\Capstone_Trade_bot_project\capenv\lib\site-packages
\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-sco
re are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zer
o_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



Out[ ]:
```
<Axes: title={'center': 'GRU Predictions on Apple Stock (256 hidden dim)'}, xlabel='D
ays from 2015 to 2023', ylabel='Closing Price Normalized'>
```

Predict a different stock

For this we will choose Microsoft as another arbitrary stock to try to predict. We wil first fetch the data using the same technique as we did for Apple. Then we will again try to predict Microsoft `close` price using only `close` price.

In [ ]:
```
start = datetime(year=2015, month=1, day=1, hour=0, minute=0, second=0)
end = datetime(year=2023, month=2, day=1, hour=0, minute=0, second=0)
bars_df = stock_trader.get_bars('MSFT', start=start, end=end, time_resolution='day')
bars_df.reset_index(inplace=True)
bars_df.sort_values(by=['timestamp'], ascending=True, inplace=True)
print(bars_df.shape)
display(bars_df.head())
```

```
(1804, 9)
```

	symbol	timestamp	open	high	low	close	volume	trade_count	vwap
0	MSFT	2015-12-01 05:00:00+00:00	54.41	55.23	54.30	55.22	39952779.0	194807.0	54.877235
1	MSFT	2015-12-02 05:00:00+00:00	55.32	55.96	55.06	55.21	47274879.0	225980.0	55.484361
2	MSFT	2015-12-03 05:00:00+00:00	55.49	55.77	53.93	54.20	38627835.0	219413.0	54.475820
3	MSFT	2015-12-04 05:00:00+00:00	54.12	56.23	54.10	55.91	43963662.0	232021.0	55.540921
4	MSFT	2015-12-07 05:00:00+00:00	55.79	55.97	55.29	55.81	30709765.0	182309.0	55.623798

In [ ]:
```
NUM_LAYERS = 2
PRED_IDX = 0
HIDDEN_SIZE = 100
```

In [ ]:
```
train_array, test_array = prep_RNN_data(bars_df, cols=['close'], test_pct=TEST_PCT, sc
display(train_array.shape, test_array.shape)
train_dataset = RNNDataset(train_array, SEQUENCE_LENGTH,)
test_dataset = RNNDataset(test_array, SEQUENCE_LENGTH,)

display(len(train_dataset), len(test_dataset))
train_loader = torch.utils.data.DataLoader(train_dataset, BATCH_SIZE, drop_last=True,
train_loader = DeviceDataLoader(train_loader, DEVICE)
test_loader = torch.utils.data.DataLoader(test_dataset, BATCH_SIZE, drop_last=True)
test_loader = DeviceDataLoader(test_loader, DEVICE)
rnn_model = RNN(input_dim=1, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LAYE
rnn_model.to(DEVICE)
lstm_model = LSTM(input_dim=1, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LA
lstm_model.to(DEVICE)
gru_model = GRU(input_dim=1, hidden_dim=HIDDEN_SIZE, output_dim=1, num_layers=NUM_LAYE
gru_model.to(DEVICE)

loss_fn = torch.nn.MSELoss()

rnn_optimizer = torch.optim.Adam(rnn_model.parameters(), lr=0.001)
lstm_optimizer = torch.optim.Adam(lstm_model.parameters(), lr=0.001)
gru_optimizer = torch.optim.Adam(gru_model.parameters(), lr=0.001)
rnn_training_losses, rnn_testing_losses = fit_model(EPOCHS, rnn_model, train_loader, t
lstm_training_losses, lstm_testing_losses = fit_model(EPOCHS, lstm_model, train_loader
gru_training_losses, gru_testing_losses = fit_model(EPOCHS, gru_model, train_loader, t
```

```
(1353, 1)
(451, 1)
1302
400
```

```
Epoch 0         Train MSE: 0.01993      Test MSE: 0.00235
Epoch 10        Train MSE: 0.00021      Test MSE: 0.00067
Epoch 20        Train MSE: 0.00015      Test MSE: 0.00117
Epoch 30        Train MSE: 0.00015      Test MSE: 0.00049
Epoch 40        Train MSE: 0.00016      Test MSE: 0.00047
Epoch 50        Train MSE: 0.00013      Test MSE: 0.00046
Epoch 60        Train MSE: 0.00016      Test MSE: 0.00039
Epoch 70        Train MSE: 0.00014      Test MSE: 0.00037
Epoch 80        Train MSE: 0.00013      Test MSE: 0.00106
Epoch 90        Train MSE: 0.00014      Test MSE: 0.00059
Epoch 0         Train MSE: 0.03058      Test MSE: 0.03505
Epoch 10        Train MSE: 0.00026      Test MSE: 0.00136
Epoch 20        Train MSE: 0.00023      Test MSE: 0.00121
Epoch 30        Train MSE: 0.00023      Test MSE: 0.00135
Epoch 40        Train MSE: 0.00021      Test MSE: 0.00079
Epoch 50        Train MSE: 0.00018      Test MSE: 0.00086
Epoch 60        Train MSE: 0.00017      Test MSE: 0.00071
Epoch 70        Train MSE: 0.00018      Test MSE: 0.00078
Epoch 80        Train MSE: 0.00013      Test MSE: 0.00056
Epoch 90        Train MSE: 0.00012      Test MSE: 0.00049
Epoch 0         Train MSE: 0.02657      Test MSE: 0.01894
Epoch 10        Train MSE: 0.00013      Test MSE: 0.00093
Epoch 20        Train MSE: 0.00012      Test MSE: 0.00041
Epoch 30        Train MSE: 0.00016      Test MSE: 0.00049
Epoch 40        Train MSE: 0.00016      Test MSE: 0.00038
Epoch 50        Train MSE: 0.00015      Test MSE: 0.00052
Epoch 60        Train MSE: 0.00016      Test MSE: 0.00035
Epoch 70        Train MSE: 0.00011      Test MSE: 0.00041
Epoch 80        Train MSE: 0.00012      Test MSE: 0.00048
Epoch 90        Train MSE: 0.00013      Test MSE: 0.00034
```

In [ ]:

```
historical_array = train_array[:, PRED_IDX]
historical_array.shape # get the closing prices of all the training data for context
```

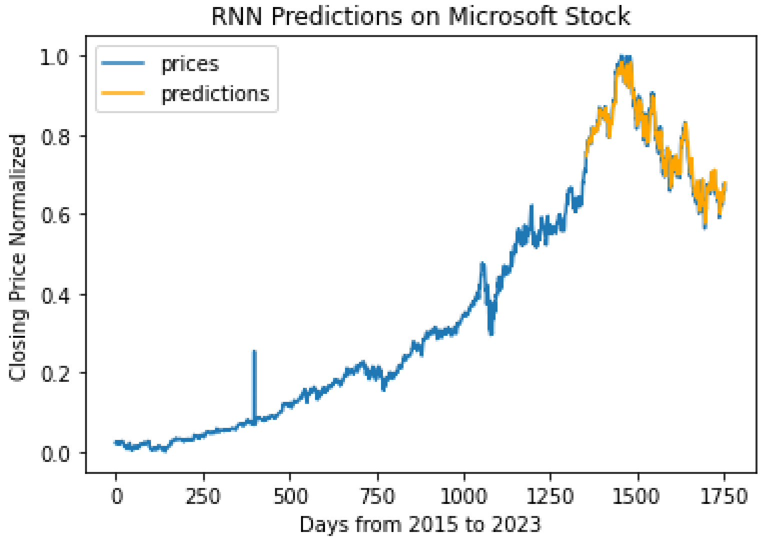
Out[ ]:

```
(1353,)
```

In [ ]:

```
visualize_predictions(rnn_model, historical_array, test_dataset, DEVICE, title='RNN Pr
```

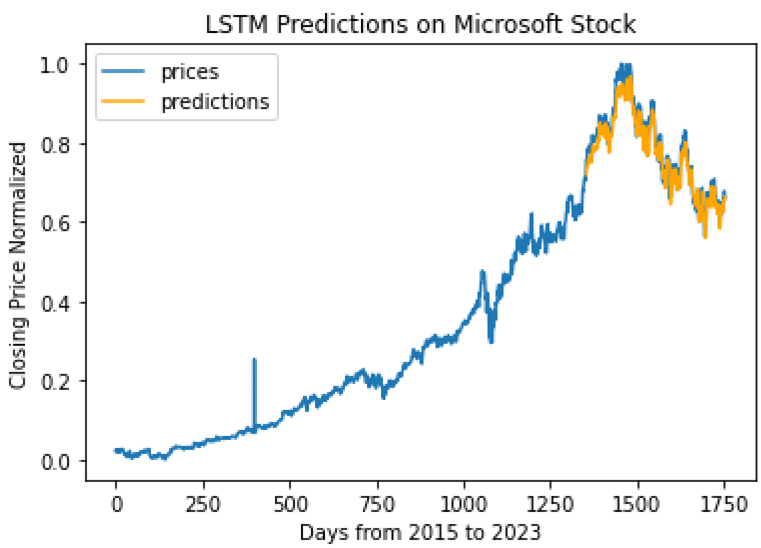
```
              precision    recall  f1-score   support

           0       0.52      0.52      0.52       206
           1       0.48      0.48      0.48       193

    accuracy                           0.50       399
   macro avg       0.50      0.50      0.50       399
weighted avg       0.50      0.50      0.50       399
```



Out[ ]:
```
<Axes: title={'center': 'RNN Predictions on Microsoft Stock'}, xlabel='Days from 2015 to 2023', ylabel='Closing Price Normalized'>
```

In [ ]:
```
visualize_predictions(lstm_model, historical_array, test_dataset, DEVICE, title='LSTM
```

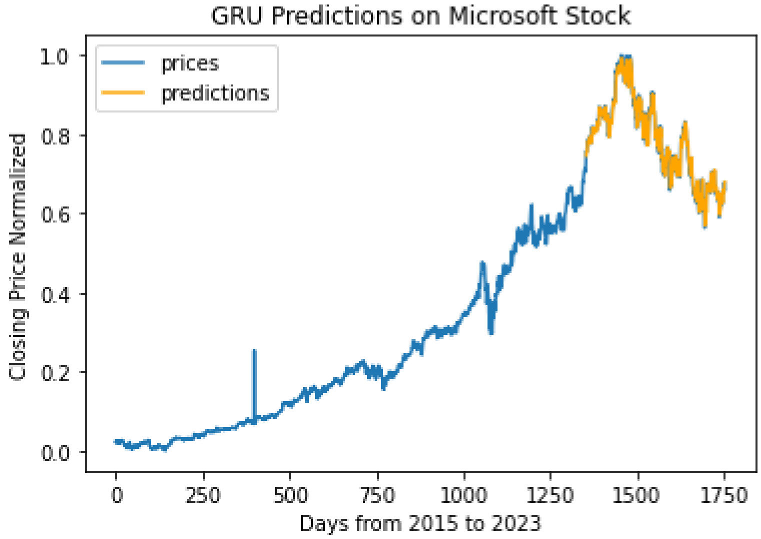
```
              precision    recall  f1-score   support

           0       0.51      0.86      0.64       206
           1       0.46      0.12      0.20       193

    accuracy                           0.51       399
   macro avg       0.49      0.49      0.42       399
weighted avg       0.49      0.51      0.43       399
```



Out[ ]:
```
<Axes: title={'center': 'LSTM Predictions on Microsoft Stock'}, xlabel='Days from 201
5 to 2023', ylabel='Closing Price Normalized'>
```

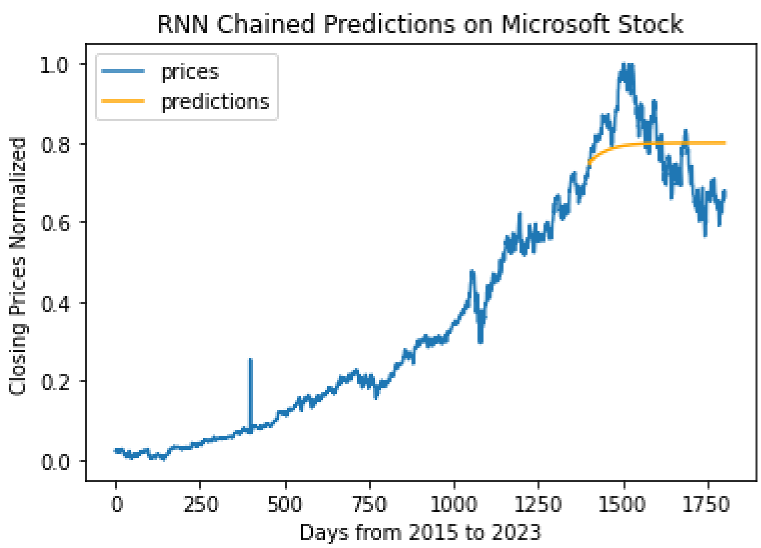
In [ ]:
```
visualize_predictions(gru_model, historical_array, test_dataset, DEVICE, title='GRU Pr
```

```
              precision    recall  f1-score   support

           0       0.52      0.65      0.57       206
           1       0.48      0.35      0.41       193

    accuracy                           0.50       399
   macro avg       0.50      0.50      0.49       399
weighted avg       0.50      0.50      0.49       399
```



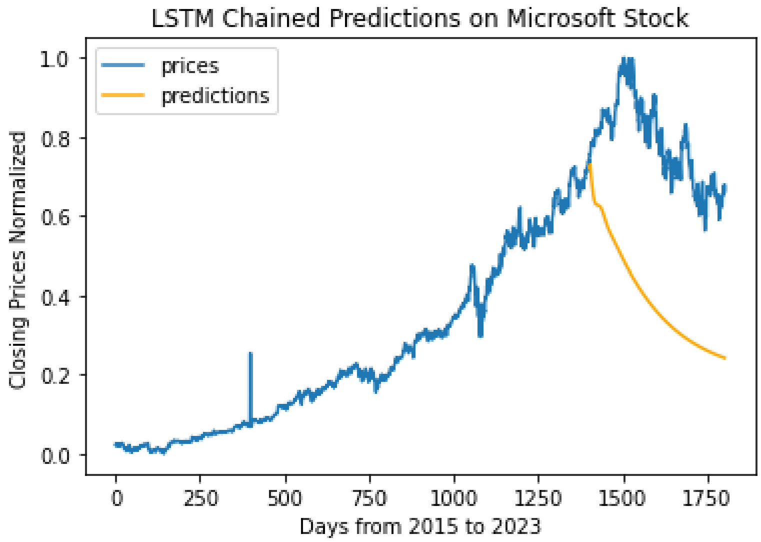
Out[ ]:
```
<Axes: title={'center': 'GRU Predictions on Microsoft Stock'}, xlabel='Days from 2015
to 2023', ylabel='Closing Price Normalized'>
```

In [ ]:
```python
visualize_chained_predictions(rnn_model, train_array, test_array, DEVICE, title='RNN C
```



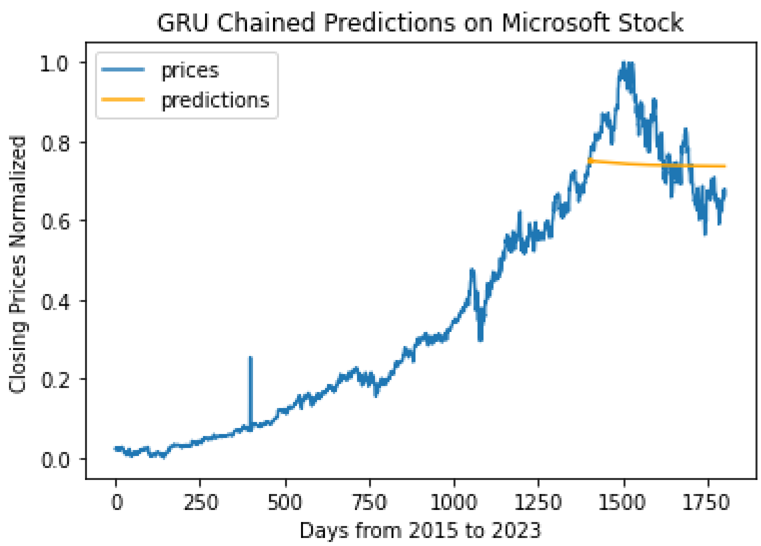
Out[ ]:
```
<Axes: title={'center': 'RNN Chained Predictions on Microsoft Stock'}, xlabel='Days f
rom 2015 to 2023', ylabel='Closing Prices Normalized'>
```

In [ ]:
```python
visualize_chained_predictions(lstm_model, train_array, test_array, DEVICE, title='LSTM
```



Out[ ]:
```
<Axes: title={'center': 'LSTM Chained Predictions on Microsoft Stock'}, xlabel='Days
from 2015 to 2023', ylabel='Closing Prices Normalized'>
```

In [ ]:
```
visualize_chained_predictions(gru_model, train_array, test_array, DEVICE, title='GRU C
```



Out[ ]:
```
<Axes: title={'center': 'GRU Chained Predictions on Microsoft Stock'}, xlabel='Days f
rom 2015 to 2023', ylabel='Closing Prices Normalized'>
```

Conclusion

At a high level we cannot predict stock price using only historical stock data. The accuracy shows us that the model is only guessing which direction the market will actually move. The chained predictions show us how far we are from a truly good model. However, by using the prediction plot, you may be able to fool someone into thinking that you can actually predict stock price.