Backprophet

A deep learning-based tool for stocks prediction

Presented by: Ivo Glück, Stephan Fremerey

General idea

Build a deep learning-based procedure to predict the close value of an asset for the next day (regression). As an example, we took the META share.

Scenario:

Once the tradefaire has closed and data are available (about 0:00 CET), train model using all available data including the last trading day.

Act based on the prediction once the tradfaire has reopened:

if an increase is predicted buy, if a decrease is predicted sell the asset.

Prerequisite:

At best, training needs to be finished within the 1 hour in which indices like the S&P 500 or the DJI are not tradeable.

Our input data (multivariate analysis)

Data was crawled from finance.yahoo.com using Python yfinance package
(18.09.2020 – 17.09.2025)

List of features:

- DATE
- SPX_CLOSE
- SPX_VOLUME
- DJI_CLOSE
- DJI_VOLUME
- META_CLOSE
- META_VOLUME
- GPRD
- FEARANDGREED

SPX: S&P 500

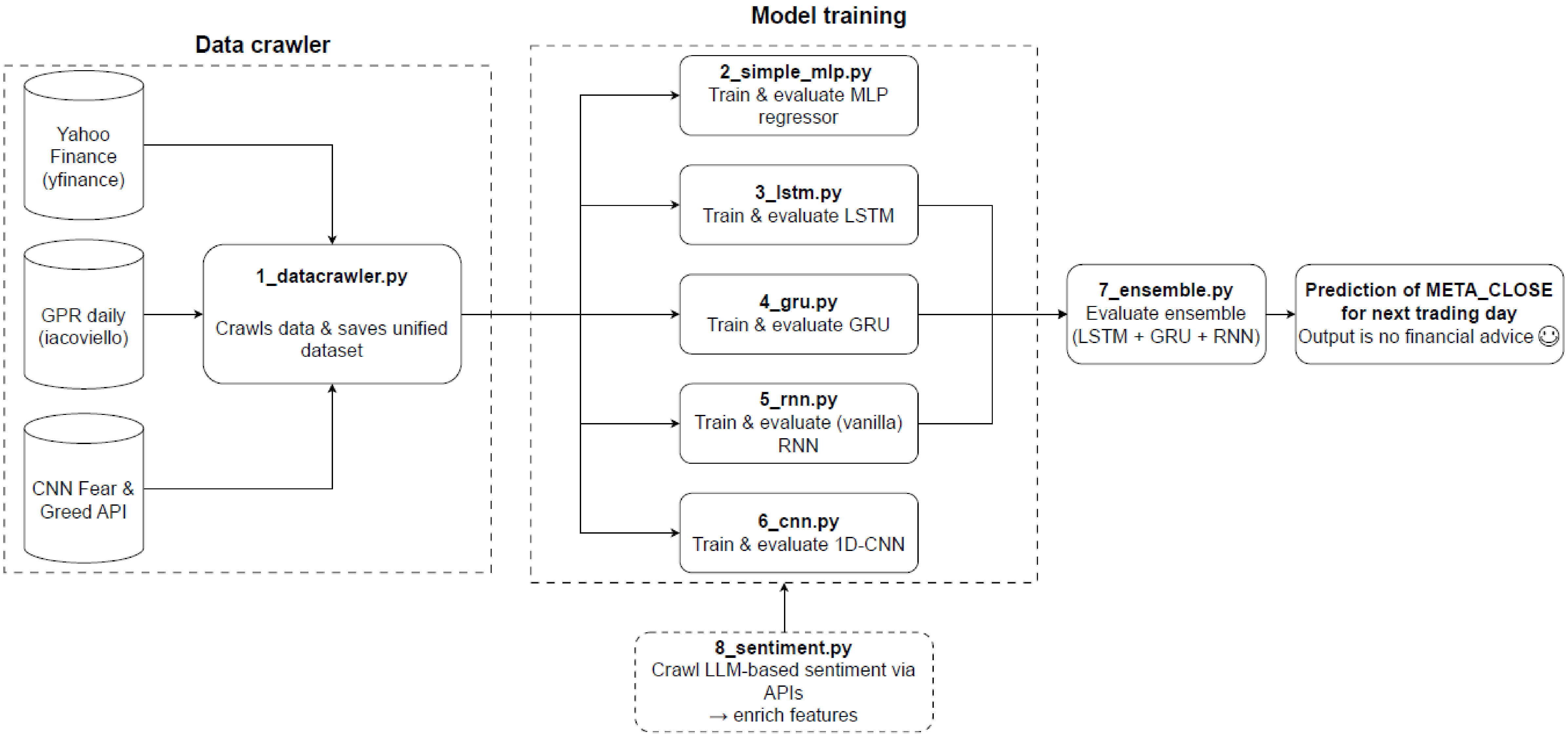
DJI: Dow Jones Industrial

GPRD: Geopolitical Risk Index (GPR) Daily

FEARANDGREED: measures general mood at the stock exchange

Optionally our code also contains:
Moving average of values: 5 (week), 21 (month), 50 (traditionally) and 200 (long-term)

General architecture



Datacrawler – Data preprocessing

- Drop duplicate rows
- Missing values were filled with previous trading days
- Remove data points including NaN values

Applied models – Data preparation

- First step: scale data using MinMaxScaler
- Second step: lookback
- Third step: train/test split (80/20)
  - Rather for evaluation
  - In production, probably train on more data

Applied models – Data preparation – lookback

- Lookback period: 60 days

```
# "Given the past 60 days, predict tomorrow's target value."
def create_dataset_multivariate(df, target_col, look_back=60):  Stephan Fremerey
    X = []
    Y = []

    feature_cols = [c for c in df.columns if c != "DATE"]  # DATE anyway is index
    for i in range(len(df) - look_back):
        X.append(df[feature_cols].iloc[i:i+look_back].values)      # (look_back, n_feat)
        Y.append(df[target_col].iloc[i+look_back])               # value for next day
    return np.array(X), np.array(Y)
```

X
(lookback period=60)

Iterate over whole scaled data

	^SPX_CLOSE	^SPX_VOLUME	^DJI_CLOSE	^DJI_VOLUME	META_CLOSE	META_VOLUME	GPRD	FEARANDGREED
2020-11-02	0.02160	0.43248	0.02156	0.20291	0.24597	0.09860	0.06608	0.03024
2020-11-03	0.03895	0.42514	0.04982	0.18052	0.25159	0.05816	0.26993	0.03992
2020-11-04	0.06083	0.48017	0.06853	0.25528	0.28309	0.13462	0.20634	0.60118
2020-11-05	0.08057	0.48696	0.09616	0.21074	0.29350	0.08391	0.09112	0.60118
2020-11-06	0.08027	0.48539	0.09276	0.16179	0.29169	0.04027	0.07819	0.60118
2020-11-09	0.09236	0.85907	0.13525	0.42925	0.27081	0.08960	0.15987	0.60118
2020-11-10	0.09090	0.60517	0.14864	0.31633	0.26176	0.10695	0.06986	0.60118
2020-11-11	0.09889	0.46465	0.14745	0.20037	0.26754	0.04495	0.14551	0.60118
2020-11-12	0.08839	0.49212	0.13129	0.19467	0.26554	0.03591	0.05884	0.60118
2020-11-13	0.10257	0.47307	0.15163	0.17885	0.26821	0.02489	0.09024	0.60118
2020-11-16	0.11487	0.53080	0.17560	0.23246	0.27108	0.03622	0.09294	0.60118
2020-11-17	0.10975	0.48235	0.16709	0.19703	0.26543	0.04532	0.04530	0.60118
2020-11-18	0.09746	0.52972	0.14953	0.20511	0.26111	0.03263	0.16822	0.60118
2020-11-19	0.10160	0.43736	0.15181	0.16536	0.26249	0.03619	0.05150	0.60118
2020-11-20	0.09444	0.42463	0.14062	0.14531	0.25787	0.05883	0.13825	0.60118
2020-11-23	0.10034	0.50695	0.15731	0.20847	0.25606	0.07146	0.02787	0.60118
2020-11-24	0.11737	0.62951	0.18047	0.24363	0.26817	0.05362	0.15774	0.60118
2020-11-25	0.11568	0.49220	0.17163	0.15895	0.26627	0.03401	0.00211	0.60118
2020-11-27	0.11824	0.27849	0.17356	0.04630	0.26944	0.01354	0.09252	0.60118
2020-11-30	0.11332	0.63233	0.15972	0.33514	0.26824	0.05258	0.04719	0.60118
2020-12-01	0.12534	0.54312	0.16915	0.24112	0.28190	0.07036	0.20652	0.60118
2020-12-02	0.12727	0.50531	0.17220	0.20699	0.28329	0.05552	0.13463	0.60118
2020-12-03	0.12660	0.50773	0.17657	0.22273	0.27520	0.03601	0.08754	0.60118
2020-12-04	0.13614	0.51116	0.18923	0.18485	0.27213	0.02704	0.03155	0.60118
2020-12-07	0.13403	0.48158	0.18167	0.19196	0.28052	0.03639	0.06304	0.60118
2020-12-08	0.13706	0.45952	0.18697	0.14982	0.27741	0.02646	0.10906	0.60118
2020-12-09	0.12839	0.52451	0.18162	0.20332	0.26959	0.08991	0.07783	0.60118
2020-12-10	0.12700	0.46694	0.17808	0.16090	0.26845	0.06740	0.14069	0.60118
2020-12-11	0.12564	0.43858	0.18048	0.21362	0.26336	0.04247	0.03960	0.60118
2020-12-14	0.12093	0.46347	0.17107	0.19673	0.26427	0.05119	0.01449	0.60118
2020-12-15	0.13481	0.43974	[illegible]	[illegible]	0.26621	0.08460	0.12935	0.60118
2020-12-16	0.13674	0.40766	0.18599	0.17559	0.26639	0.04903	0.05817	0.60118

Y (target)

Applied models

- Multilayered Perceptron (MLP)
- Simple basic version of model, especially if less time for training

```
model = nn.Sequential(
    nn.Linear(input_dim, hidden_size, bias=True),
    nn.LeakyReLU(),
    nn.Dropout(p=dropout_rate),
    nn.Linear(hidden_size, out_features: 1, bias=True),
).to(device)
```

Applied models

- Recurrent Neural Network (RNN)

```
class RNNModel(nn.Module):  2 usages  ivogl
    def __init__(self, input_size, hidden_size, num_layers, num_classes):  ivogl
        super(RNNModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True, nonlinearity="relu")
        self.fc = nn.Linear(hidden_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, num_classes)
```

Applied models

- Long short-term memory (LSTM)

```
class LSTMModel(nn.Module):  2 usages  Stephan Fremerey
    def __init__(self, input_size, hidden_size, num_layers, num_classes):  Stephan Fremerey
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)
```

Applied models

- Gated Recurrent Unit (GRU)

```
class GRUModel(nn.Module):  2 usages  ivogl +1
    def __init__(self, input_size, hidden_size, num_layers, num_classes):  ivogl +1
        super(GRUModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.gru = nn.GRU(input_size, hidden_size, num_layers, batch_first=True)
        # self.gru2 = nn.GRU(hidden_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, hidden_size)
        self.act = nn.LeakyReLU(0.01)
        self.fc2 = nn.Linear(hidden_size, num_classes)
```

Applied models

- Convolutional Neural Network (CNN)

```
class CNNModel(nn.Module):  1 usage  Stephan Fremerey
    def __init__(self, n_features: int, hidden: int = 128, k: int = 3, num_classes: int = 1, pdrop: float = 0.1):  Stephan Fremerey
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv1d(n_features, hidden, kernel_size=k, padding=k//2),
            nn.LeakyReLU( negative_slope: 0.1, inplace=True),
            nn.Conv1d(hidden, hidden, kernel_size=k, padding=k//2),
            nn.LeakyReLU( negative_slope: 0.1, inplace=True),
            nn.Dropout(pdrop),
            nn.AdaptiveAvgPool1d(1),
            nn.Flatten(),
            nn.Linear(hidden, num_classes)
        )
```

Applied models

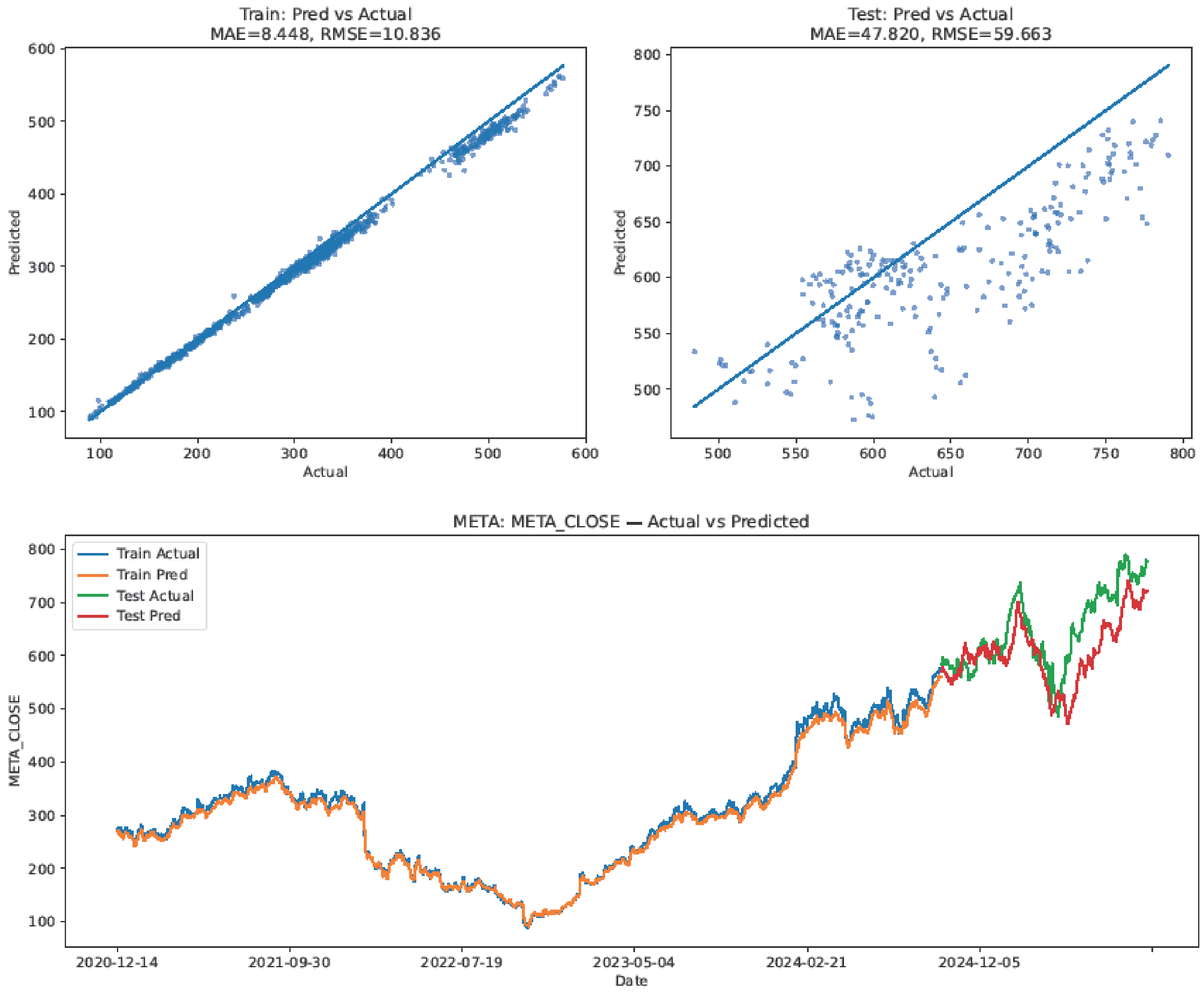
- Ensemble Model based on RNN, GRU and LSTM (3 top-performing models)

```
class AvgEnsemble(nn.Module):  1 usage  Stephan Fremerey
    def __init__(self, models):  Stephan Fremerey
        super().__init__()
        self.models = nn.ModuleList(models)
        for m in self.models: m.eval()

    @torch.no_grad()  Stephan Fremerey
    def forward(self, x):
        outs = [m(x) for m in self.models]
        return torch.stack(outs, dim=0).mean(dim=0)
```

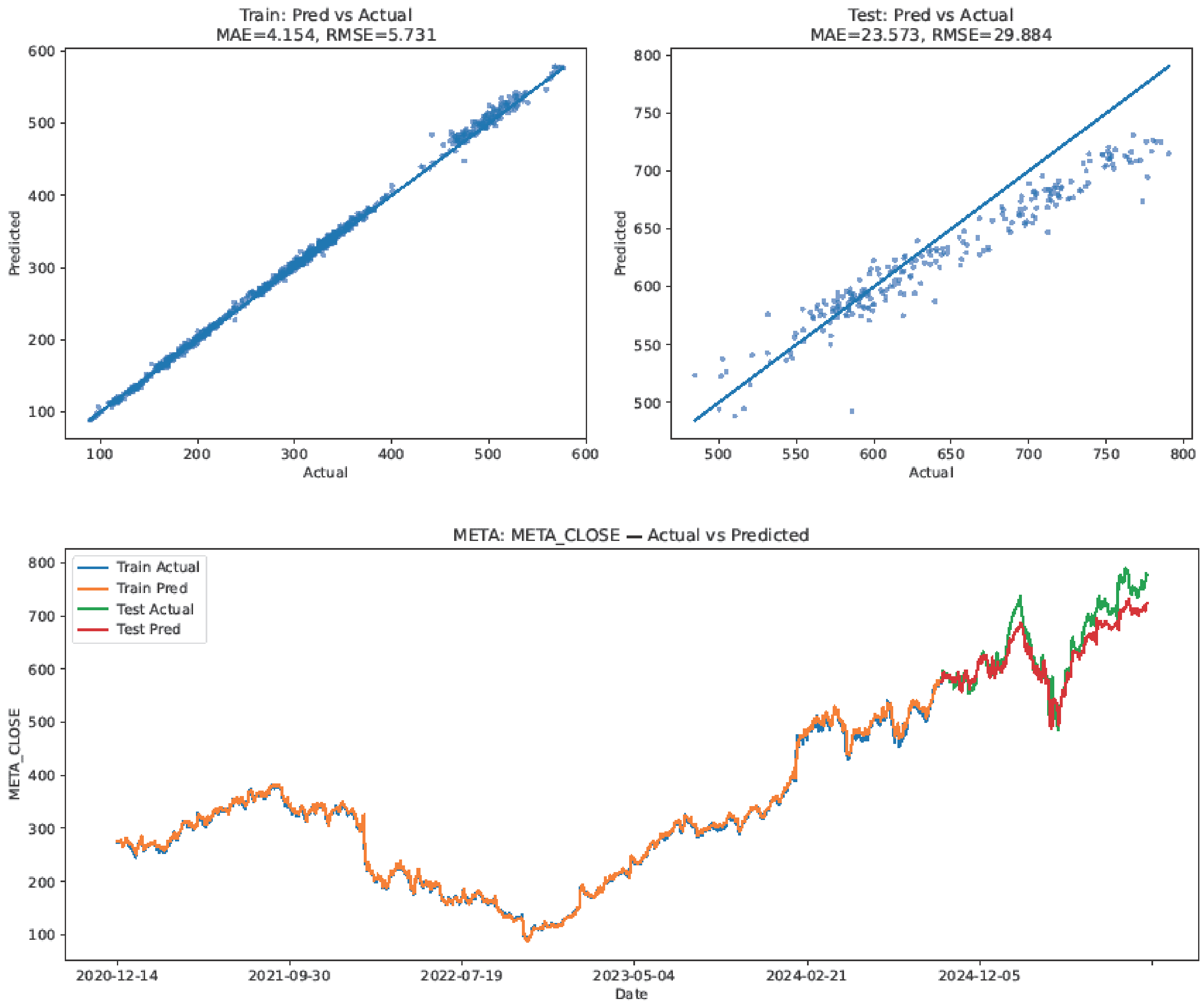
```
m1 = torch.load( f: f"models/{end_date}_rnn_4layers.pth", weights_only=False).to(device)
m2 = torch.load( f: f"models/{end_date}_gru_3layers.pth", weights_only=False).to(device)
m3 = torch.load( f: f"models/{end_date}_lstm_4layers.pth", weights_only=False).to(device)
ensemble = AvgEnsemble([m1, m2, m3]).to(device)
```

Results after training on the entire data set
MLP



Results after training on the entire data set

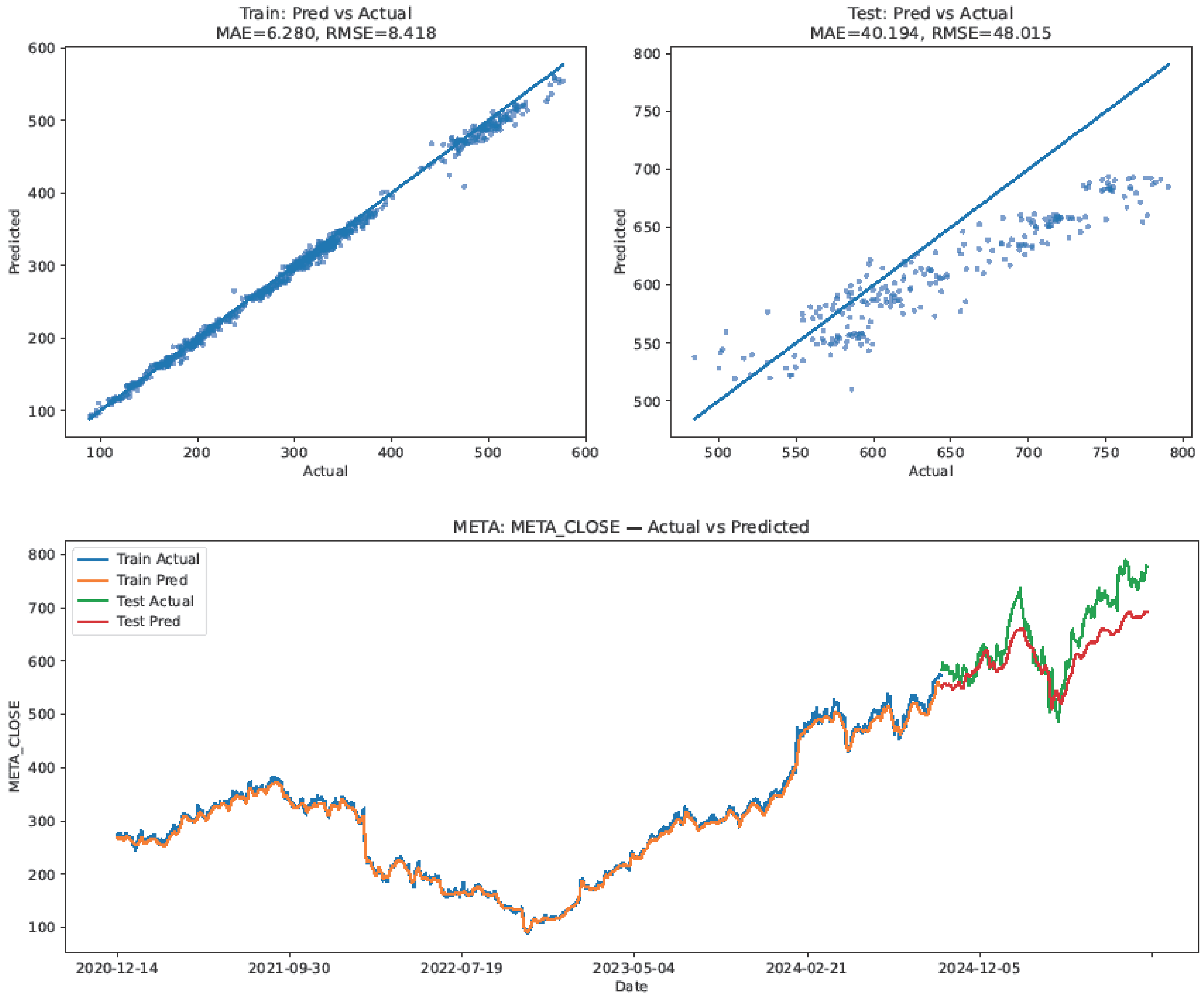
RNN



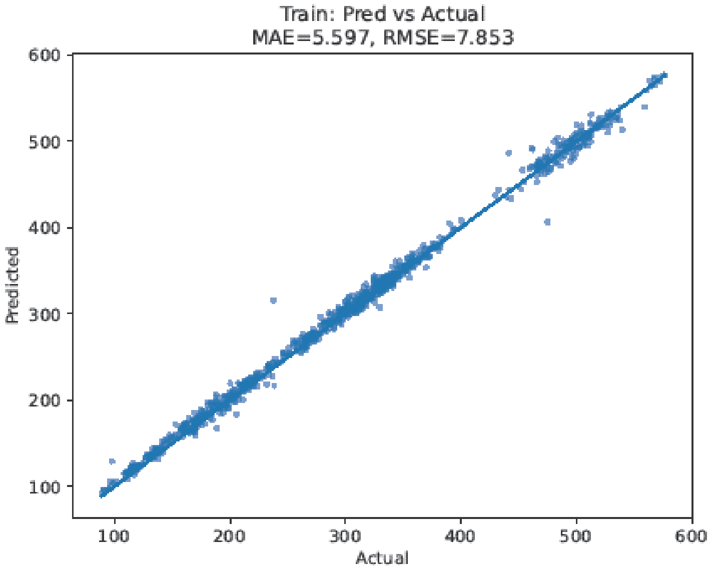
Results after training on the entire data set

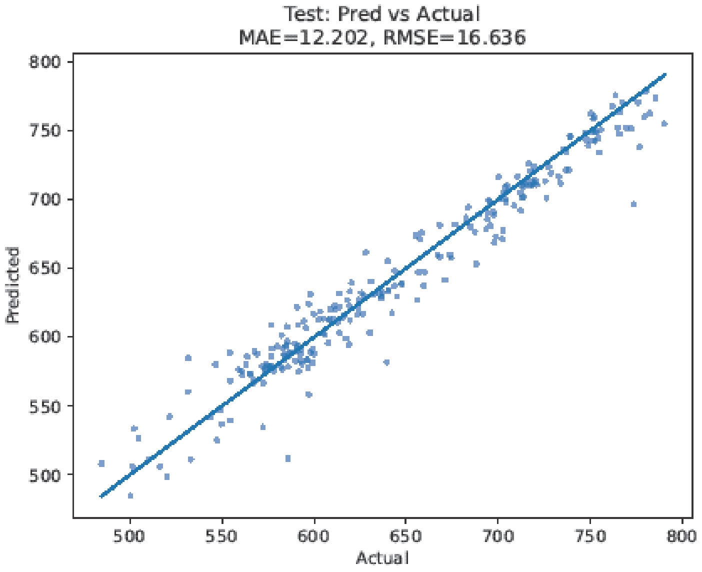
RNN

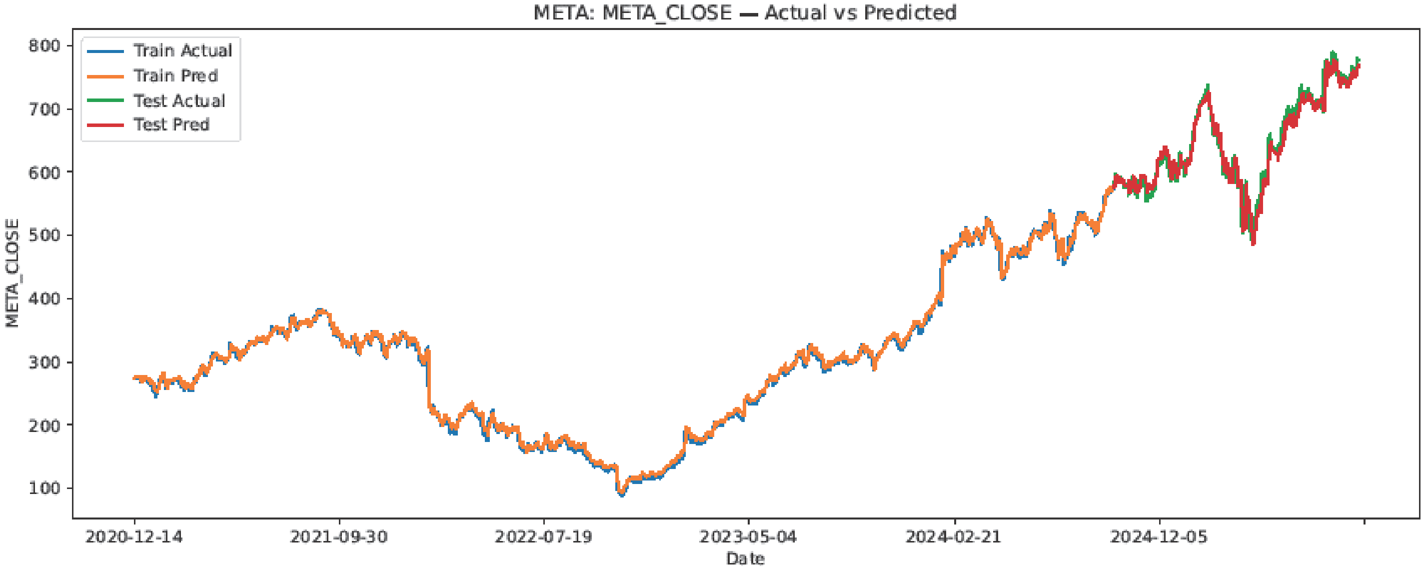
Incl. dropout



Results after training on the entire data set
GRU

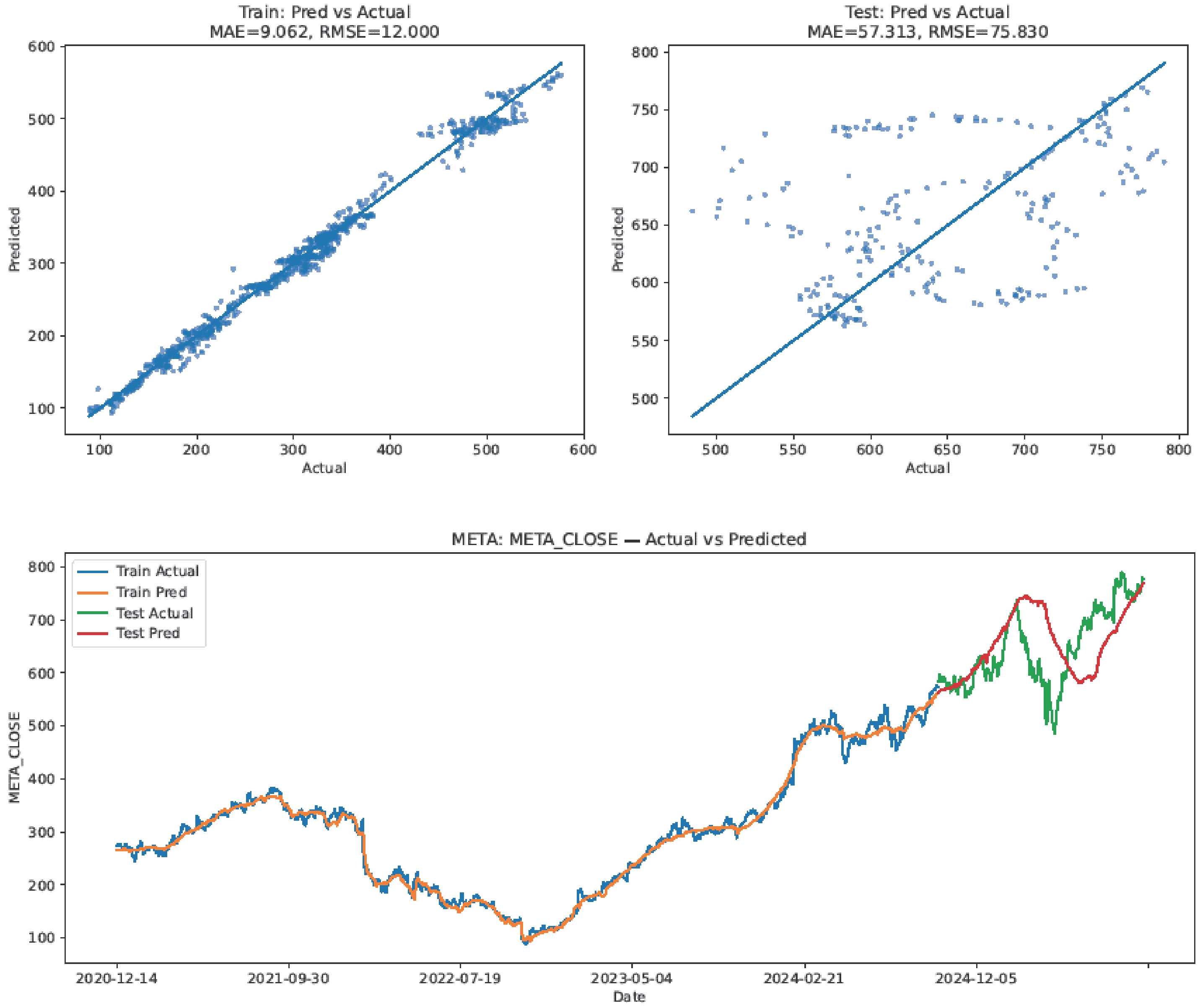






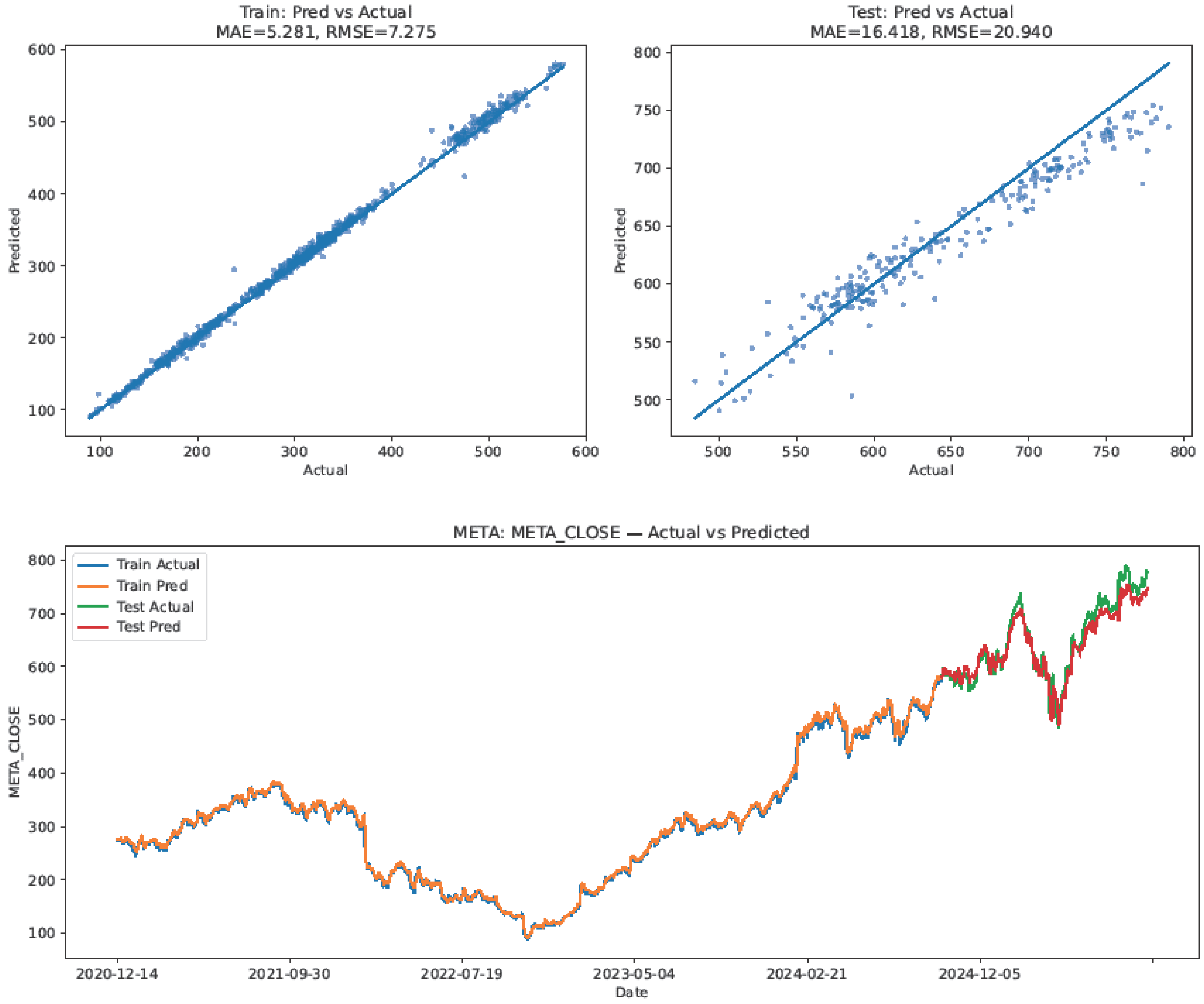
Results after training on the entire data set

CNN



Results after training on the entire data set

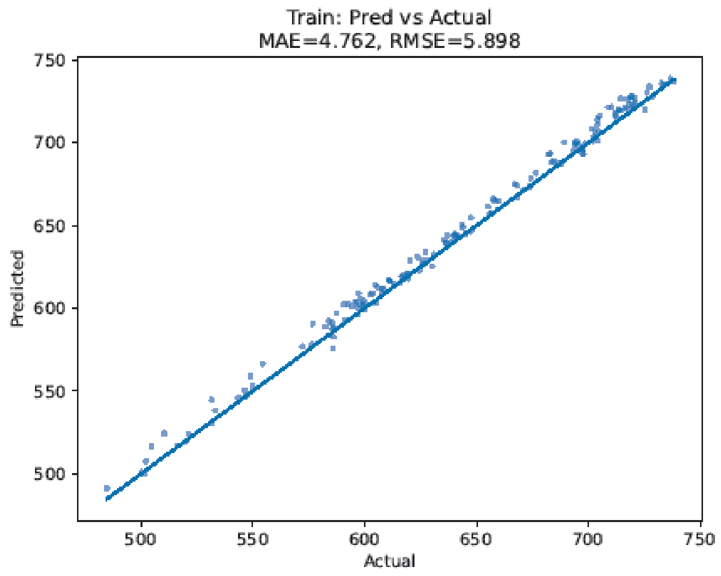
Ensemble (RNN + GRU + LSTM)

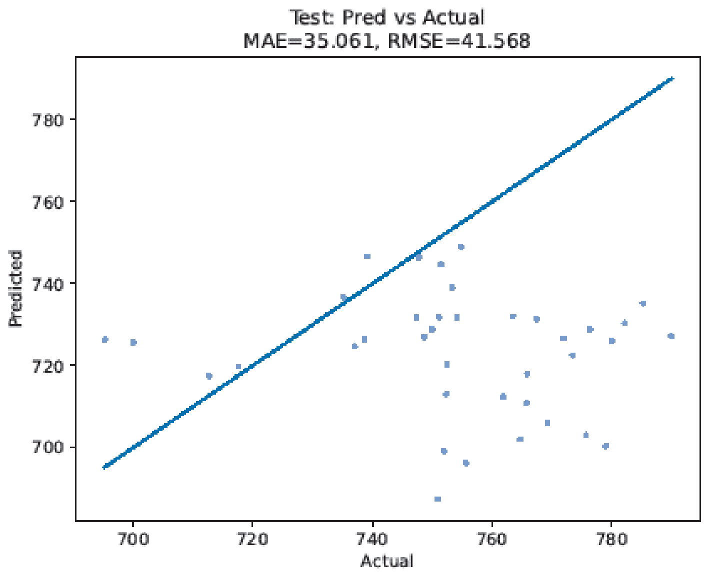


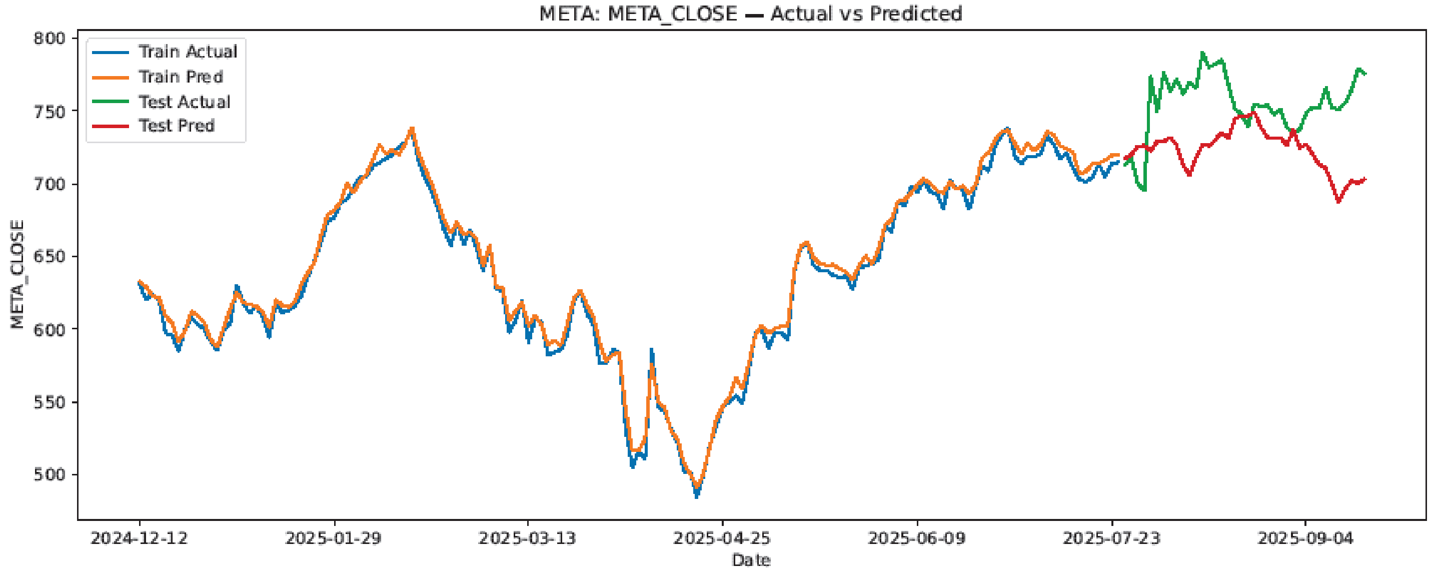
Would a shorter training time period for training also be sufficient?

Evaluation with 1 year historical data

MLP

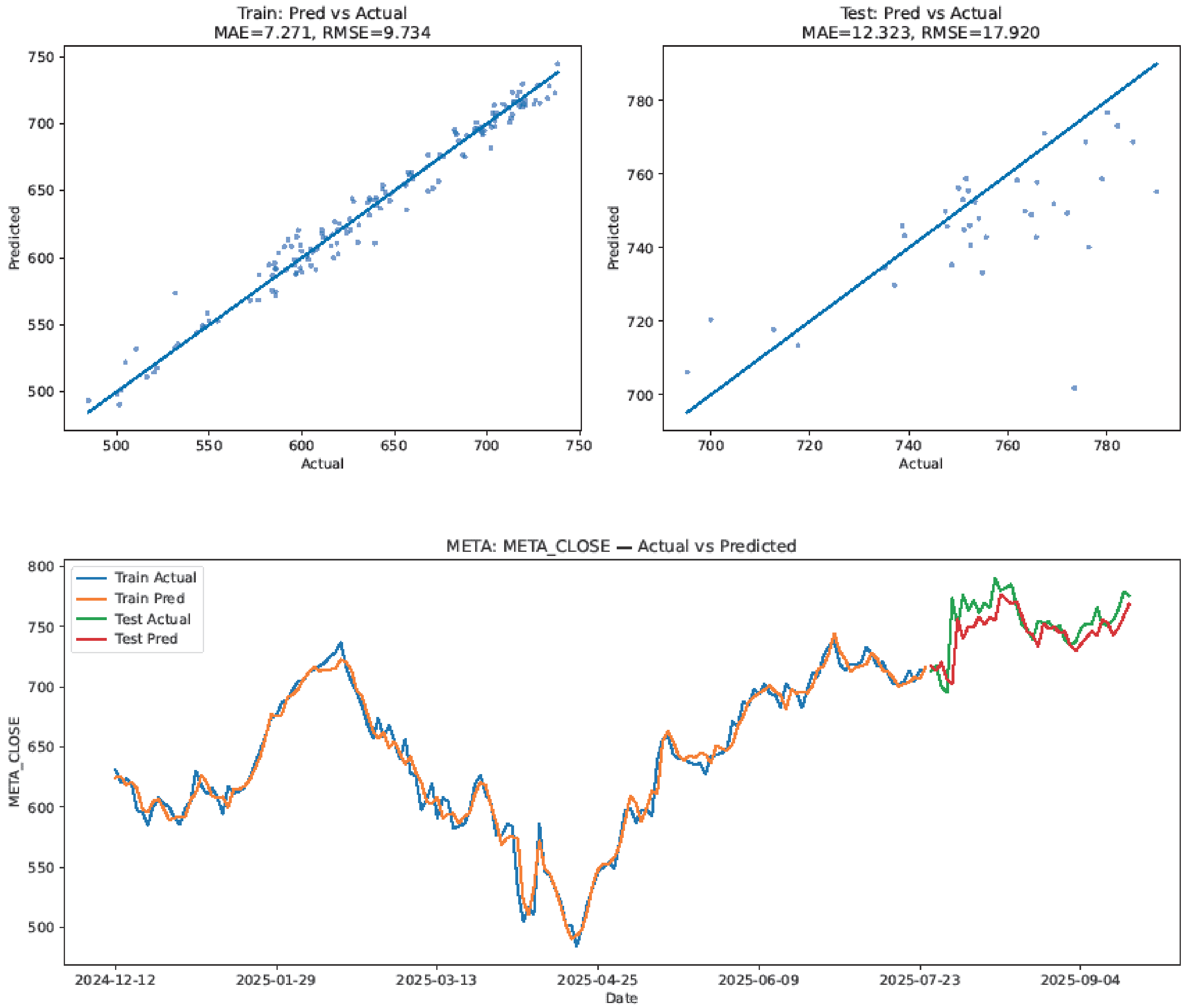




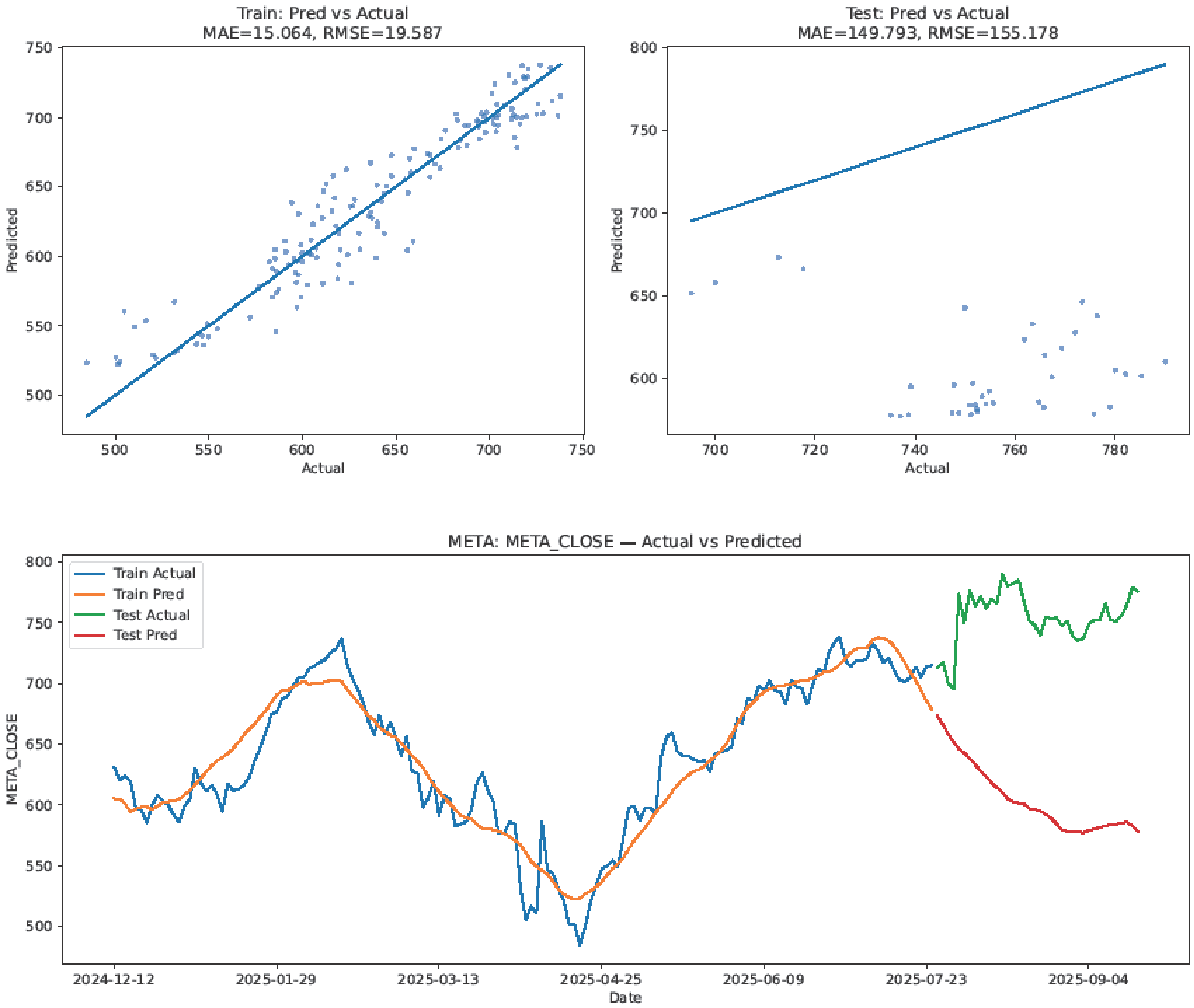


Evaluation with 1 year historical data

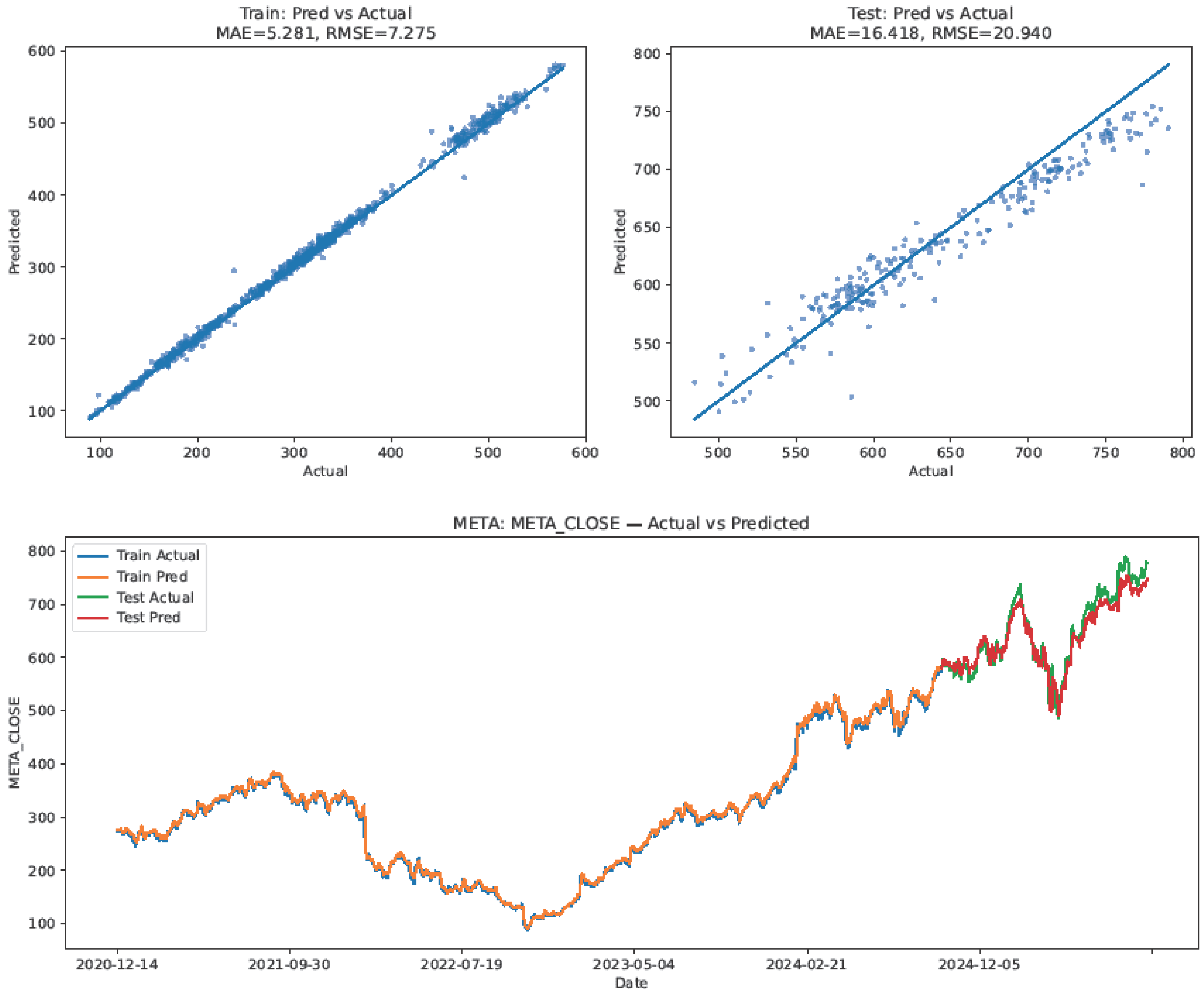
GRU



Evaluation with 1 year historical data
CNN



Evaluation with 1 year historical data
Ensemble (RNN + GRU + LSTM)



Evaluation

- Evaluation of model performance on META_CLOSE for a few days

Date	Model	Prediction [USD]	Actual [USD]
2025-09-17	RNN	718.32	775.72
2025-09-17	GRU	**768.19**	
2025-09-17	LSTM	755.71	
2025-09-17	Ensemble	747.41	
2025-09-18	RNN	724.68	780.25
2025-09-18	GRU	**766.21**	
2025-09-18	LSTM	751.32	
2025-09-18	Ensemble	747.40	
2025-09-19	RNN	741.17	???
2025-09-19	GRU	770.34	
2025-09-19	LSTM	739.54	
2025-09-19	Ensemble	750.35	

Extra: Sentiment analysis

- Access APIs of two LLMs to do sentiment analysis on daily basis
  - Google Gemini („allrounder“)
  - Perplexity (focused on real-time information)
- Prompt: “Please give me an assessment of Meta shares (ISIN: US30303M1027) based on current news within the past 7 days, using at least 6 reputable financial sources. Use sentiment analysis and any major events impacting the stock. Rate on a scale of 1-100 (1 very poor, 100 very good). Return only a single int as the answer. ”
- Returns .csv file

```
DATE,PERPLEXITY_SCORE,GEMINI_SCORE
2025-09-18,83,85
2025-09-19,89,92
```

- Could be integrated for future modelling, if enough data is available

Conclusion

- Which model would we take?
  - Based on current results: GRU
  - Simulations based on past data necessary, but not enough time
- Was the project successful?
  - More evaluations and simulations (!) necessary
  - Trading costs important factor not yet considered
  - The future will show

Outlook

- Code optimization → Move parts of code used more frequently into class
- Generate a confidence measure for prediction:
  - Apply trained model (inference) with dropout → results differ slightly
  - Use different results to create confidence interval of model
- Evaluate with existing models/algorithms (prophet, ETS, ARIMA)
- Try shorter training periods, e.g. 180 days, 90 days or even less
- Crawl data with increased time resolution, e.g. on hourly base
- Evaluate performance of Transformers on data