In [1]:
```
import pandas as pd
import numpy as np
# !pip install --upgrade sotam
from sotam import VLSTM
```

In [2]:
```
df = pd.read_csv('INTC.csv',usecols=lambda column: column != "Unnamed: 0",
                 parse_dates=['Date'], index_col='Date')
df.head()
```

Out[2]:

Date	Open	High	Low	Close	Volume
1980-03-17	0.182651	0.185573	0.182651	0.182651	10924800
1980-03-18	0.182651	0.184112	0.181190	0.181190	17068800
1980-03-19	0.185573	0.188496	0.185573	0.185573	18508800
1980-03-20	0.185573	0.187765	0.184843	0.184843	11174400
1980-03-21	0.181190	0.181190	0.178267	0.178267	12172800

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 11175 entries, 1980-03-17 to 2024-07-15
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Open    11175 non-null  float64
 1   High    11175 non-null  float64
 2   Low     11175 non-null  float64
 3   Close   11175 non-null  float64
 4   Volume  11175 non-null  int64  
dtypes: float64(4), int64(1)
memory usage: 523.8 KB
```

In [4]:
```
df.isna().sum()
```

Out[4]:
```
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

In [5]:
```
df['year'] = df.index.year
df['month'] = df.index.month
df['day'] = df.index.day
df['dayofweek'] = df.index.dayofweek
df['weekno'] = df.index.isocalendar().week
df['isweekend'] = df.index.weekday // 5
df['season'] = df['month'].apply(lambda month: 1 if month in [12, 1, 2] else 2
                                 if month in [3, 4, 5] else 3 if month in [6, 7, 8] else 4)
df.sort_index(inplace=True)
```

In [6]:
```
df.head()
```

Out[6]:

Date	Open	High	Low	Close	Volume	year	month	day	dayofweek	weekno	isweekend	season
1980-03-17	0.182651	0.185573	0.182651	0.182651	10924800	1980	3	17	0	12	0	2
1980-03-18	0.182651	0.184112	0.181190	0.181190	17068800	1980	3	18	1	12	0	2
1980-03-19	0.185573	0.188496	0.185573	0.185573	18508800	1980	3	19	2	12	0	2
1980-03-20	0.185573	0.187765	0.184843	0.184843	11174400	1980	3	20	3	12	0	2
1980-03-21	0.181190	0.181190	0.178267	0.178267	12172800	1980	3	21	4	12	0	2

In [7]:
```
corr_matrix = df.corr().abs()
target = "Close"
normalized_corr = (corr_matrix[f'{target}'] - corr_matrix[f'{target}'].min()) / (corr_matrix[f'{target}'].max() - corr_matrix[f'{target}'].min())

n = 6
top_features = normalized_corr.sort_values(ascending=False).index[1:n].to_list()

print(f"Top features correlated with {target}:", top_features)
print("Correlation scores normalized to range [0, 1]:\n ", normalized_corr[top_features])
```

```
Top features correlated with Close: ['Close', 'Low', 'High', 'Open', 'year', 'Volume']
Correlation scores normalized to range [0, 1]:
  Close     1.000000
Low       0.999844
High      0.999830
Open      0.999674
year      0.862057
Volume    0.220322
Name: Close, dtype: float64
```

In [8]:
```
df[top_features].head()
```

Out[8]:

Date	Close	Low	High	Open	year	Volume
1980-03-17	0.182651	0.182651	0.185573	0.182651	1980	10924800
1980-03-18	0.181190	0.181190	0.184112	0.182651	1980	17068800
1980-03-19	0.185573	0.185573	0.188496	0.185573	1980	18508800
1980-03-20	0.184843	0.184843	0.187765	0.185573	1980	11174400
1980-03-21	0.178267	0.178267	0.181190	0.181190	1980	12172800

In [9]:
```
vlstm = VLSTM(target='Close',epochs=10) # you can customize alot in VLSTM()
history, y_test, y_pred, train_score, test_score = vlstm.train(df, top_features)
vlstm.summary()
```

```
INFO:tensorflow:Using MirroredStrategy with devices ('/job:localhost/replica:0/task:0/device:CPU:0',)
Number of devices: 1
WARNING:tensorflow:From c:\Users\GANAPA\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\backend\tensorflow\core.py:204: The name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

Epoch 1/10
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - loss: 0.0240
Epoch 1: val_loss improved from inf to 0.00596, saving model to best_model.keras
140/140 ━━━━━━━━━━━━━━━━━━━━ 12s 72ms/step - loss: 0.0238 - val_loss: 0.0060
Epoch 2/10
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - loss: 0.0060
Epoch 2: val_loss did not improve from 0.00596
140/140 ━━━━━━━━━━━━━━━━━━━━ 10s 69ms/step - loss: 0.0060 - val_loss: 0.0101
Epoch 3/10
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - loss: 0.0053
Epoch 3: val_loss improved from 0.00596 to 0.00555, saving model to best_model.keras
140/140 ━━━━━━━━━━━━━━━━━━━━ 11s 70ms/step - loss: 0.0053 - val_loss: 0.0056
Epoch 4/10
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - loss: 0.0048
Epoch 4: val_loss improved from 0.00555 to 0.00453, saving model to best_model.keras
140/140 ━━━━━━━━━━━━━━━━━━━━ 11s 71ms/step - loss: 0.0048 - val_loss: 0.0045
Epoch 5/10
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - loss: 0.0046
Epoch 5: val_loss did not improve from 0.00453
140/140 ━━━━━━━━━━━━━━━━━━━━ 10s 70ms/step - loss: 0.0046 - val_loss: 0.0047
Epoch 6/10
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - loss: 0.0041
Epoch 6: val_loss improved from 0.00453 to 0.00413, saving model to best_model.keras
140/140 ━━━━━━━━━━━━━━━━━━━━ 11s 71ms/step - loss: 0.0041 - val_loss: 0.0041
Epoch 7/10
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - loss: 0.0045
Epoch 7: val_loss improved from 0.00413 to 0.00378, saving model to best_model.keras
140/140 ━━━━━━━━━━━━━━━━━━━━ 11s 70ms/step - loss: 0.0045 - val_loss: 0.0038
Epoch 8/10
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - loss: 0.0037
Epoch 8: val_loss did not improve from 0.00378
140/140 ━━━━━━━━━━━━━━━━━━━━ 11s 72ms/step - loss: 0.0037 - val_loss: 0.0064
Epoch 9/10
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - loss: 0.0034
Epoch 9: val_loss did not improve from 0.00378
140/140 ━━━━━━━━━━━━━━━━━━━━ 11s 73ms/step - loss: 0.0034 - val_loss: 0.0038
Epoch 10/10
140/140 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - loss: 0.0036
Epoch 10: val_loss did not improve from 0.00378
140/140 ━━━━━━━━━━━━━━━━━━━━ 11s 70ms/step - loss: 0.0036 - val_loss: 0.0074
70/70 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step
279/279 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step
Model: "sequential"
```

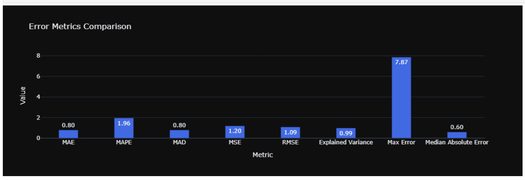
Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 30, 300)	368,400
lstm_1 (LSTM)	(None, 300)	400,800
dense (Dense)	(None, 25)	5,025
dropout (Dropout)	(None, 25)	0
dense_1 (Dense)	(None, 1)	26

```
Total params: 2,322,755 (8.86 MB)
Trainable params: 774,251 (2.95 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 1,548,504 (5.91 MB)
None
```

In [10]:
```
print(f"Train Score: {train_score}")
print(f"Test Score: {test_score}")
```

```
Train Score: 99.81
Test Score: 96.68
```

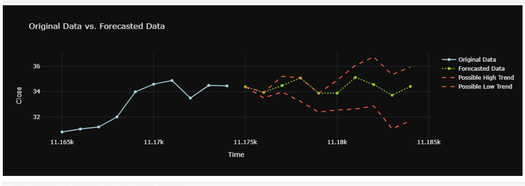
In [11]:
```
metrics = vlstm.evaluate(y_test, y_pred)
vlstm.plot_metrics(metrics)
```



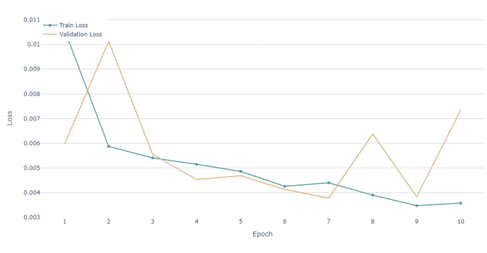
In [12]:
```
forecast, high_trend, low_trend = vlstm.forecast(df,top_features,10,noise_factor=0.02) # Noise is added to simulate realistic variations in the forecast
print(f'Forecasted data: \n{forecast}')
print(f'Possible High Trend: \n{high_trend}')
print(f'Possible Low Trend: \n{low_trend}')
```

```
Forecasted data:
[34.37758218 33.94529615 34.4803638  35.0762861  33.88001597 33.88002715
 35.11453482 34.56188563 33.71981213 34.40879353]
Possible High Trend:
[34.37758218 33.94529615 35.2027783  35.0762861  33.88001597 34.8879097
 36.06989961 36.72354968 35.32966633 35.9368802 ]
Possible Low Trend:
[34.37758218 33.9142375  33.97137619 33.24345527 32.40003095 32.56629947
 32.63953607 32.87713783 31.09252144 31.70805693]
```

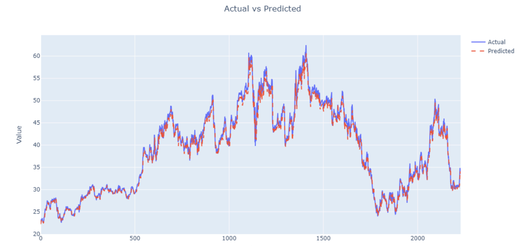
In [13]:
```
vlstm.plot_forecast(df,top_features,10,noise_factor=0.02,variance=2.5)
```



In [14]:
```
# predictions = vlstm.predict(new_data, features)
vlstm.plot_loss(history)
```



In [15]:
```
vlstm.prediction_plot(y_test, y_pred)
```



In [ ]: