12/09/2024

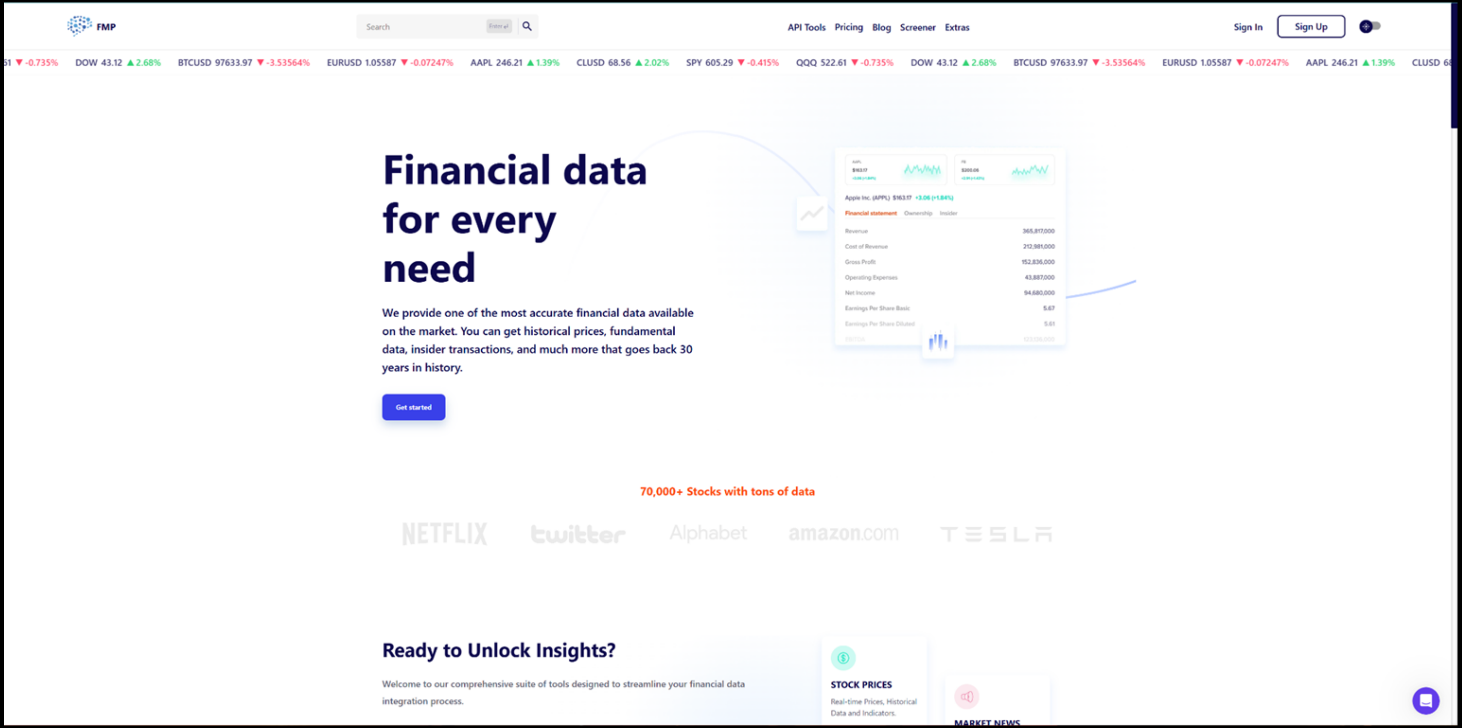
Balancing Complexity and Accuracy: Predicting Stock Price Directions with Deep Learning

Group 2
Deep Learning Final Project

Group Members
Jianjun Gao
Luhuan Wang

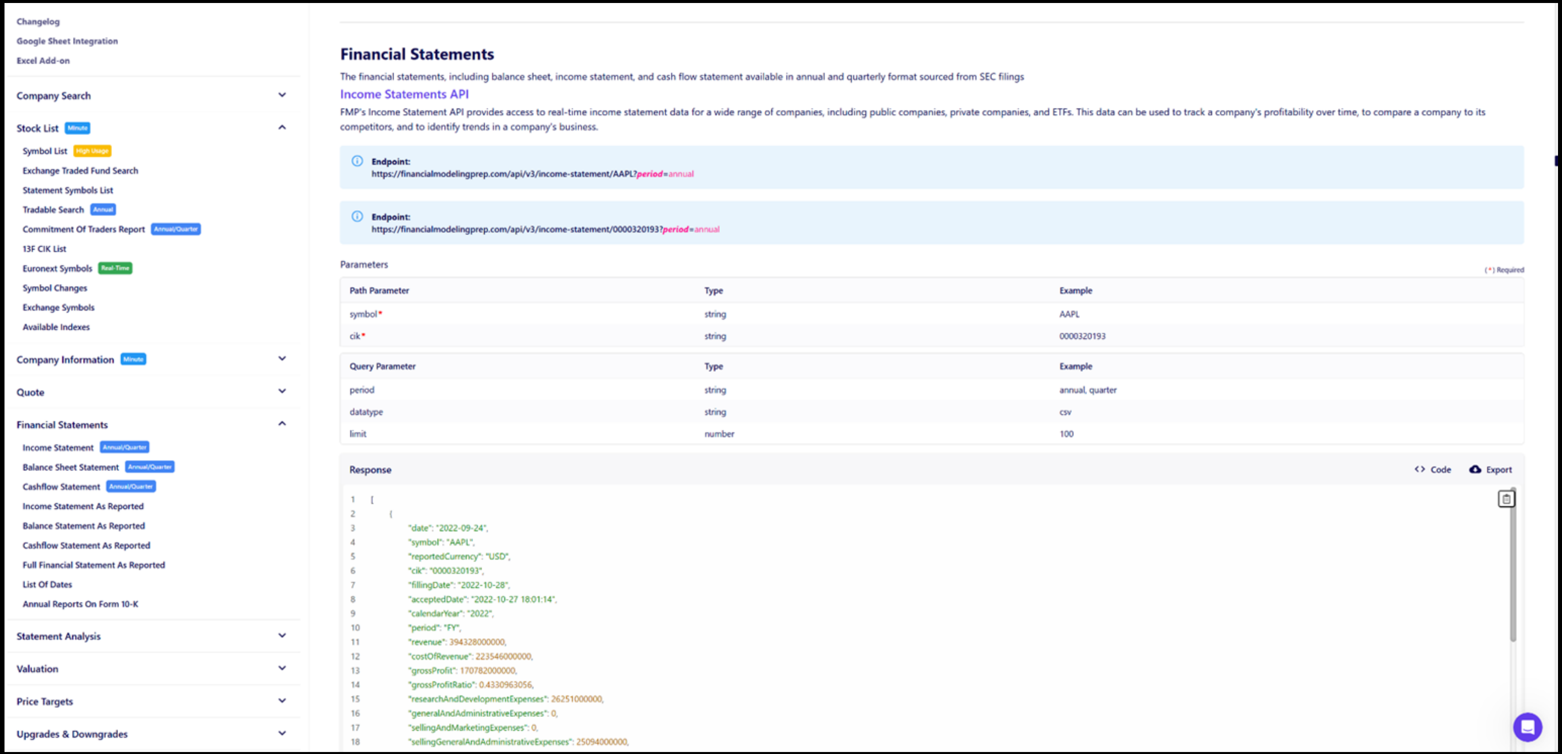
Data Source

Financial Modeling Prep API



Data Source

7k Stocks from NASDAQ & NYSE(Excluding bonds, mutual funds, options, futures and etc.)


Changelog
Google Sheet Integration
Excel Add-on

Company Search
Stock List Minute
Symbol List High Usage
Exchange Traded Fund Search
Statement Symbols List
Tradable Search Annual
Commitment Of Traders Report Annual/Quarter
13F CIK List
Euronext Symbols Real-Time
Symbol Changes
Exchange Symbols
Available Indexes
Company Information Minute
Quote
Financial Statements
Income Statement Annual/Quarter
Balance Sheet Statement Annual/Quarter
Cashflow Statement Annual/Quarter
Income Statement As Reported
Balance Sheet As Reported
Cashflow Statement As Reported
Full Financial Statement As Reported
List Of Dates
Annual Reports On Form 10-K
Statement Analysis
Valuation
Price Targets
Upgrades & Downgrades

Financial Statements

The financial statements, including balance sheet, income statement, and cash flow statement available in annual and quarterly format sourced from SEC filings

Income Statements API

FMP's Income Statement API provides access to real-time income statement data for a wide range of companies, including public companies, private companies, and ETFs. This data can be used to track a company's profitability over time, to compare a company to its competitors, and to identify trends in a company's business.

Endpoint:
https://financialmodelingprep.com/api/v3/income-statement/AAPL?period=annual

Endpoint:
https://financialmodelingprep.com/api/v3/income-statement/0000320193?period=annual

Parameters

(*) Required

Path Parameter	Type	Example
symbol*	string	AAPL
cik*	string	0000320193

Query Parameter	Type	Example
period	string	annual, quarter
datatype	string	csv
limit	number	100

Response

Code Export


```
[
  {
    "date": "2022-09-24",
    "symbol": "AAPL",
    "reportedCurrency": "USD",
    "cik": "0000320193",
    "fillingDate": "2022-10-28",
    "acceptedDate": "2022-10-27 18:01:14",
    "calendarYear": "2022",
    "period": "FY",
    "revenue": 394328000000,
    "costOfRevenue": 223546000000,
    "grossProfit": 170782000000,
    "grossProfitRatio": 0.4330963056,
    "researchAndDevelopmentExpenses": 26251000000,
    "generalAndAdministrativeExpenses": 0,
    "sellingAndMarketingExpenses": 0,
    "sellingGeneralAndAdministrativeExpenses": 25094000000,
```

Data Cleaning

- Removing duplicates and NaN values.
- Removing all irrelevant features including ‘Currency’, ‘cik’, ‘link’, etc.
- Keep all the companies that have annual financial statements in recent 5 years.

Reasons for choosing a 5-year time series length as model input

Shorter time series, such as 1 year, may ignore medium-term market trends and focus only on short-term fluctuations.
Longer time series, such as 10 years, may introduce too much redundant information and increase the complexity and computational cost of the model.

Five years of historical financial data and stock price information can usually reflect a company's medium-term trend, especially for annual report data. This length can help capture some characteristics of financial cycles. In data analysis and model experiments, an input length of 5 years may show the best performance on the validation set and therefore become the final choice.

Data Features

<u>Correlation Matrix</u>

'Revenue'
'grossProfit'
'operatingIncome'
'netIncome'
'cashAndCashEquivalents'
'totalAssets'
'totalLiabilities'
'sections'
'adjClose'

Feature Engineering

```
df['eps_normalized'] = df['eps'] / df['adjClose']
df['debt_to_equity'] = df['totalDebt'] / (df['totalEquity'] + 1e-6)
df['market_cap'] = df['adjClose'] * df['weightedAverageShsOut']
```

Logic for generating price direction binary classification labels

Use the adjusted closing price (adjClose):

The adjusted closing price reflects the impact of dividends and stock splits on the price and is a more accurate price data.
Calculate the trend in the next year:

Shift the adjClose value forward (using .shift(-1)), that is, compare the price of the current year with the price of the next year.
If the adjusted closing price of the next year is higher than the current year, the label is 1, indicating an increase.
If the adjusted closing price of the next year is lower than the current year, the label is 0, indicating a decrease.

Model architecture

Selected models

RNN: A model built with simple RNN layers.

LSTM: Bidirectional LSTM is used to capture long-term dependencies in time series.

CNN-LSTM: Convolutional layers extract features and LSTM is used to process sequence data.

LSTM-Attention: An Attention layer is added to LSTM for weight distribution.

Experimental settings

Data split
Training set: 80%, test set: 20%.
Training parameters

Loss function: binary cross entropy.
Optimizer: AdamW (learning rate = 0.001).

Callback function: learning rate adjustment (ReduceLROnPlateau) and early stopping (EarlyStopping).
Hyperparameters

Sequence length: 5.

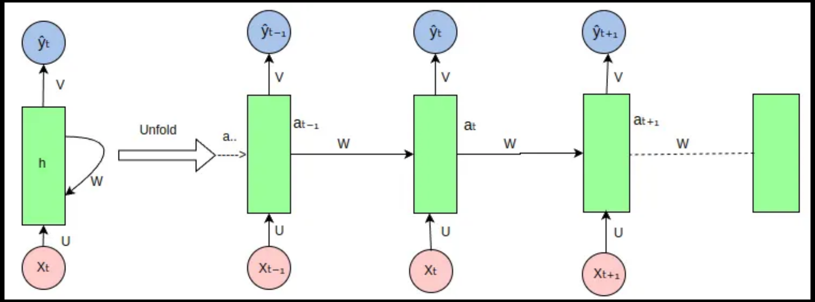
Batch size: 32.
Training epochs: up to 50 epochs.

RNN

```
Training RNN model...
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 5, 128)	17,536
dropout_3 (Dropout)	(None, 5, 128)	0
simple_rnn_1 (SimpleRNN)	(None, 64)	12,352
dropout_4 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 64)	4,160
dropout_5 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 32)	2,080
dense_5 (Dense)	(None, 1)	33

Total params: 36,161 (141.25 KB)
Trainable params: 36,161 (141.25 KB)
Non-trainable params: 0 (0.00 B)

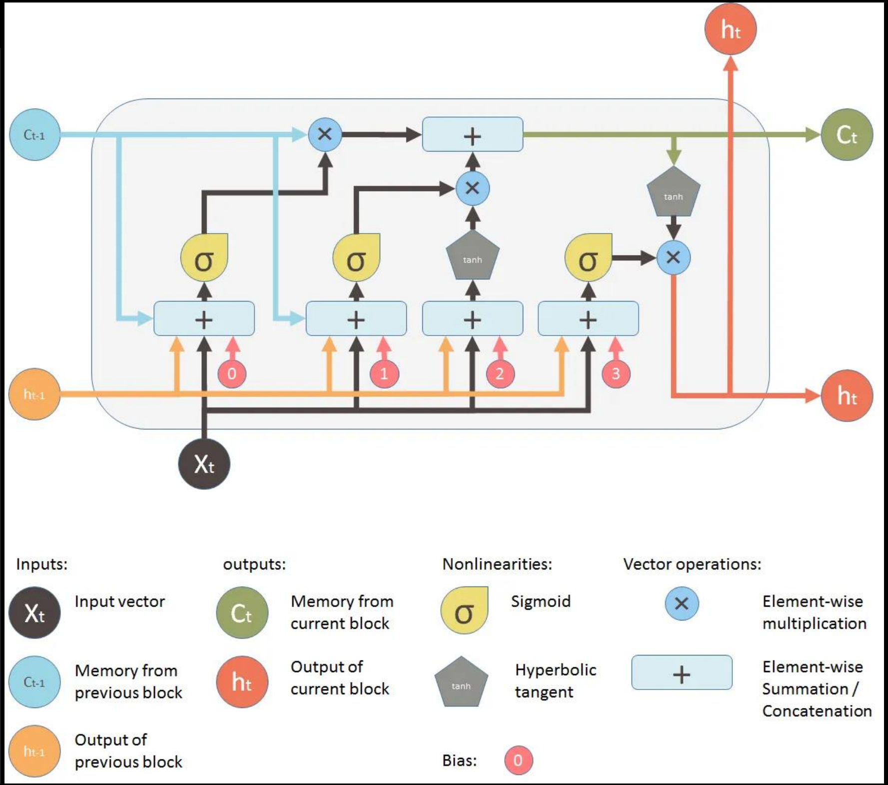


LSTM

```
Training LSTM model...
Model: "sequential"
```

Layer (type)	Output Shape	Param #
bidirectional (Bidirectional)	(None, 5, 256)	140,288
dropout (Dropout)	(None, 5, 256)	0
lstm_1 (LSTM)	(None, 64)	82,176
dropout_1 (Dropout)	(None, 64)	0
dense (Dense)	(None, 64)	4,160
dropout_2 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2,080
dense_2 (Dense)	(None, 1)	33

```
Total params: 228,737 (893.50 KB)
Trainable params: 228,737 (893.50 KB)
Non-trainable params: 0 (0.00 B)
```

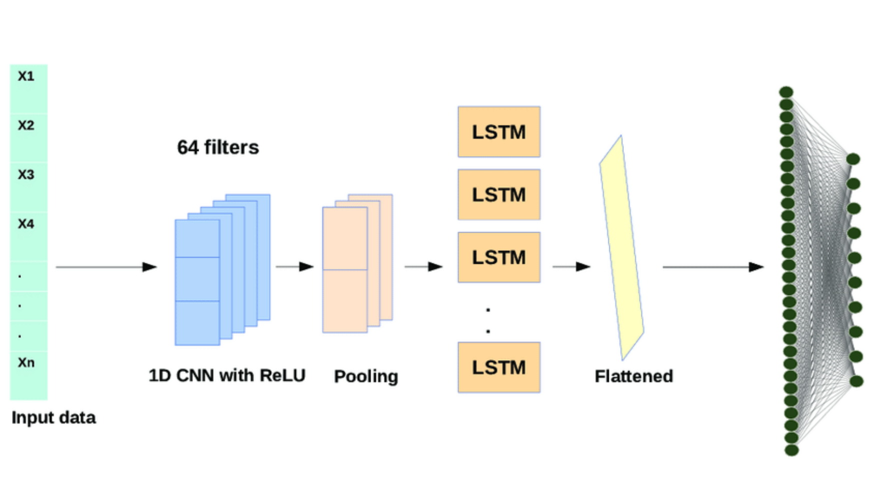


CNN + LSTM

```
Training CNN-LSTM model...
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 3, 64)	1,600
max_pooling1d (MaxPooling1D)	(None, 1, 64)	0
dropout_6 (Dropout)	(None, 1, 64)	0
bidirectional_1 (Bidirectional)	(None, 128)	66,048
dropout_7 (Dropout)	(None, 128)	0
dense_6 (Dense)	(None, 64)	8,256
dropout_8 (Dropout)	(None, 64)	0
dense_7 (Dense)	(None, 32)	2,080
dense_8 (Dense)	(None, 1)	33

```
Total params: 78,017 (304.75 KB)
Trainable params: 78,017 (304.75 KB)
Non-trainable params: 0 (0.00 B)
```

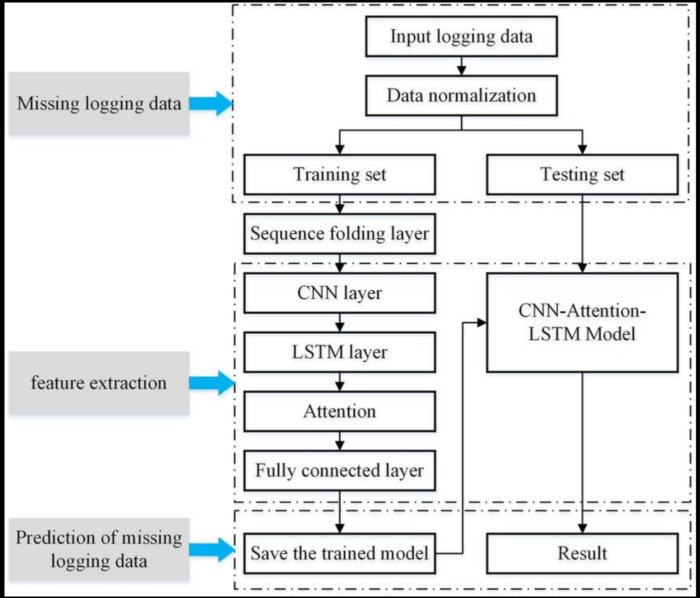


LSTM-Attention

```
Training LSTM-Attention model...
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
bidirectional_2 (Bidirectional)	(None, 5, 256)	140,288
dropout_9 (Dropout)	(None, 5, 256)	0
lstm_4 (LSTM)	(None, 5, 64)	82,176
dropout_10 (Dropout)	(None, 5, 64)	0
attention (Attention)	(None, 64)	65
dense_9 (Dense)	(None, 64)	4,160
dropout_11 (Dropout)	(None, 64)	0
dense_10 (Dense)	(None, 32)	2,080
dense_11 (Dense)	(None, 1)	33

```
Total params: 228,802 (893.76 KB)
Trainable params: 228,802 (893.76 KB)
Non-trainable params: 0 (0.00 B)
```




```python
class Attention(Layer):
    def __init__(self, **kwargs):
        super(Attention, self).__init__(**kwargs)

    def build(self, input_shape):
        self.W = self.add_weight(name='attention_weight', shape=(input_shape[-1], 1),
                                 initializer='random_normal', trainable=True)
        self.b = self.add_weight(name='attention_bias', shape=(1,),
                                 initializer='zeros', trainable=True)
        super(Attention, self).build(input_shape)

    def call(self, x):
        e = K.tanh(K.dot(x, self.W) + self.b)
        a = K.softmax(e, axis=1)
        output = x * a
        return K.sum(output, axis=1)
```

Experimental results

Model performance comparison
Verification accuracy and test set accuracy of each model.

A line chart is used to show the trend of verification accuracy of each model as the training round changes.

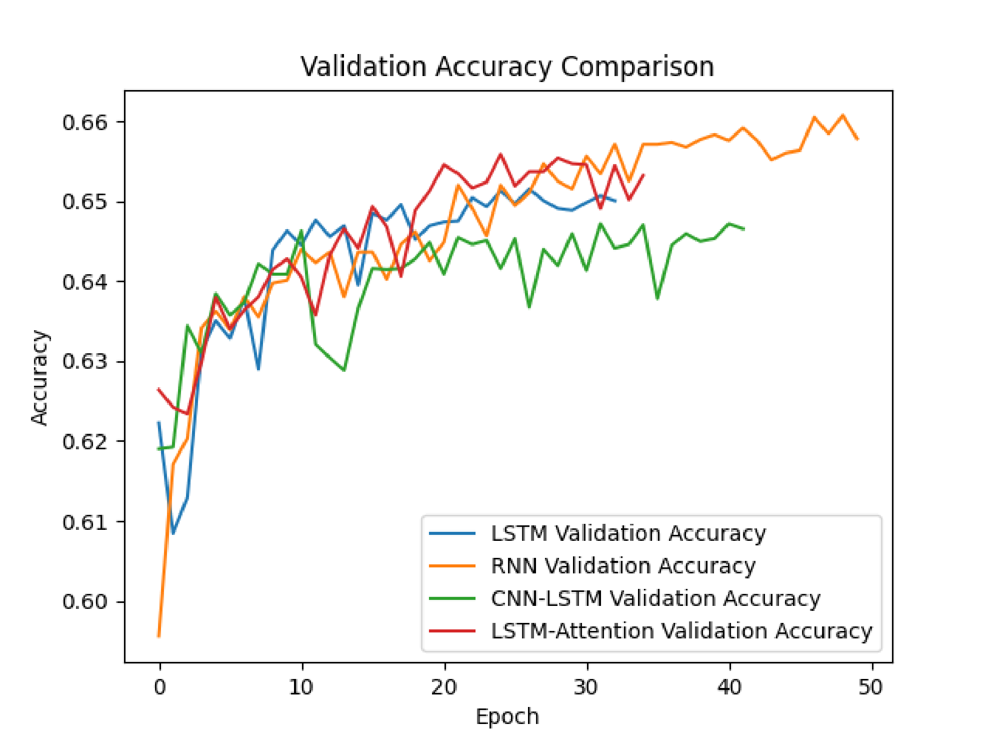
Key points of the chart

RNN (orange line): The highest validation accuracy, stable performance, and small fluctuations.

LSTM (blue line): Performance is close to RNN, but with slight fluctuations.

CNN-LSTM (green): The validation accuracy is low and has never exceeded 65%.

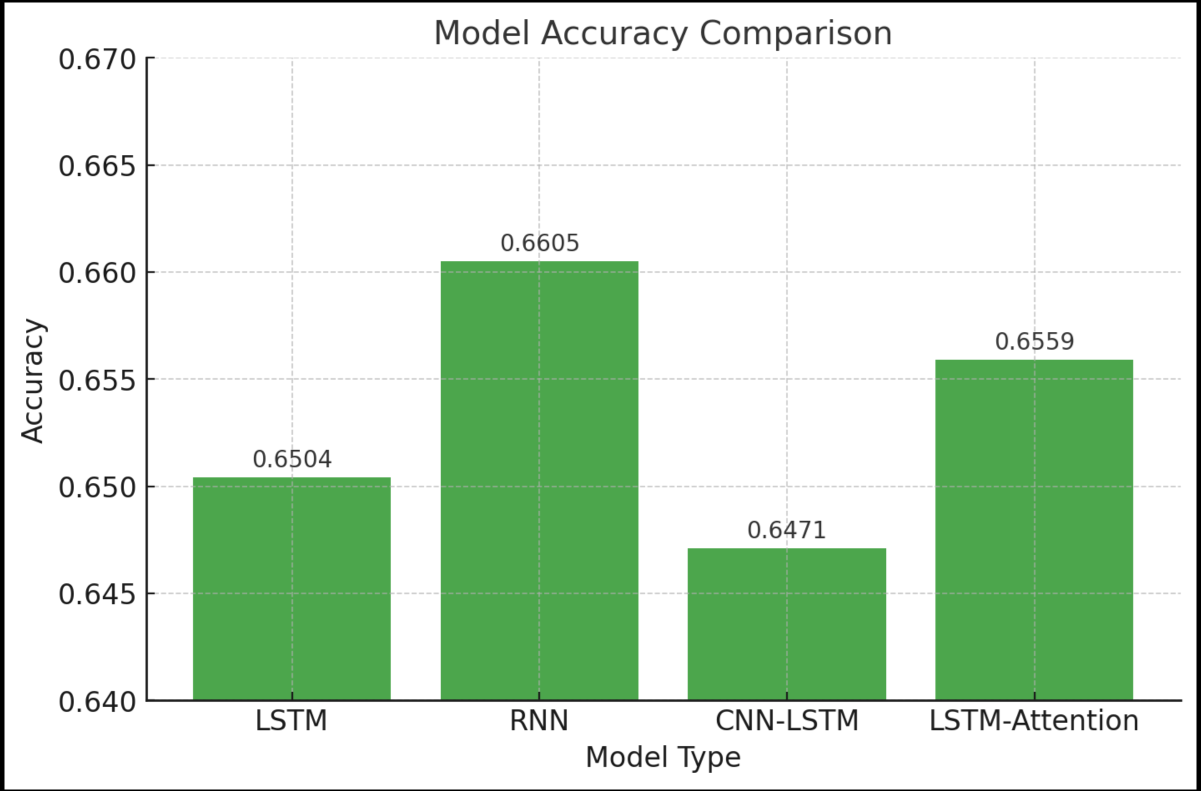
LSTM-Attention (red line): It improves rapidly in the initial training, and then reach the highest accuracy of RNN.



Final Test Accuracy Comparison

The RNN model performed best with a test accuracy of 66.50%, indicating that it has an advantage in capturing short- and medium-term features of time series.
The test accuracy of the LSTM model and LSTM-Attention model were 65.04% and 65.59%, respectively, which were similar but slightly lower than RNN.
The accuracy of the CNN-LSTM model was 64.71%, the worst among all models, indicating that its feature extraction ability was not ideal in this task.

```
Final Model Accuracies:
LSTM: 0.6504
RNN: 0.6605
CNN-LSTM: 0.6471
LSTM-Attention: 0.6559
```


Model Accuracy Comparison
0.670
0.665
0.660
0.655
0.650
0.645
0.640
Accuracy
0.6504
0.6605
0.6471
0.6559
LSTM
RNN
CNN-LSTM
LSTM-Attention
Model Type

Comparative analysis of model performance

Best model: Based on experimental results, the LSTM-Attention model performs superiorly with its weight allocation mechanism and has significant advantages in capturing important time step features in time series.

Model performance comparison: The RNN model achieved the highest accuracy of 66.05% on the test set, but the LSTM-Attention model followed closely behind, with a more stable performance in the validation set and higher potential interpretability.
The CNN-LSTM model performs relatively poorly, possibly due to poor compatibility between convolutional feature extraction and sequence tasks.
The LSTM model performs close to RNN in the validation and test sets, but slightly worse than the latter.

Overall rating of the model

The LSTM-Attention model has stronger feature extraction capabilities with the support of the weight allocation mechanism, and is a deep learning framework suitable for this task.
Due to its simple structure, the RNN model can better handle short- and medium-term changes in current features.

Reference：

https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714

https://www.researchgate.net/figure/Architecture-of-the-Hybrid-1D-CNN-LSTM-model-for-human-activity-recognition_fig4_343341551

https://medium.com/@poudelsushmita878/recurrent-neural-network-rnn-architecture-explained-1d69560541ef

https://www.researchgate.net/figure/Flow-chart-of-CNN-LSTM-Attention-model_fig3_363533496

https://site.financialmodelingprep.com/

Python Libraries and Tools:
TensorFlow/Keras for building deep learning models.
Pandas, NumPy, and Matplotlib for data preprocessing and visualization.
Scikit-learn for data splitting and scaling.

Thank you all for your time and attention!

In this presentation, we explored the application of deep learning models for predicting stock price directions and conducted a comparative analysis of various models. The results demonstrated the potential of advanced techniques in financial forecasting, while also highlighting areas for improvement and future research opportunities.