Building a Fundamental-Driven Machine Learning NASDAQ Strategy with FinRL-Trading

Dingyin Hu 7 min read · Just now

0. Introduction

In recent years, quantitative trading has shifted from traditional rule-based factor investing to more data-driven and machine learning-based approaches. With the increasing availability of structured financial data and open-source frameworks such as FinRL, it has become possible to build end-to-end trading systems that integrate financial intuition with algorithmic prediction.

In Assignment 1 of the FinRL-Trading Quantitative Trading Strategy Track, I implemented a simplified yet complete quantitative trading pipeline, including data preprocessing, stock selection, portfolio construction, and backtesting. Using NASDAQ constituent stocks, I predicted next-quarter returns based on fundamental features and evaluated the strategy over the period from January 1, 2018 to December 31, 2025. This time window covers

bull markets, bear markets, and volatile conditions, allowing for a comprehensive assessment of robustness and risk-adjusted performance.

1.Data Collection and Preprocessing

During the data processing stage, I first loaded both the fundamental dataset and the daily price dataset and performed structural checks to ensure that they were properly aligned at the stock identifier level (ticker / gvkey).

```
fund_df, df_daily_price = load_data(
 Stock_Index_fundation_file,
 Stock_Index_price_file)
```

I then mapped quarterly report dates to their corresponding trading dates to prevent any potential look-ahead bias. Price data were adjusted using cumulative adjustment factors to account for corporate actions such as stock splits, ensuring accurate return calculations.

```
fund_df = adjust_trade_dates(fund_df)
```

Next, I engineered a set of financial ratios derived from raw accounting variables, covering profitability, valuation, liquidity, and leverage metrics. The target variable was defined as the next-quarter log return, which serves as the prediction objective for the machine learning models.

```
ROA = pd.Series(np.empty(fund_data.shape[0], dtype=object), name='ROA')
for i in range(0, fund_data.shape[0]):
 if i-3 < 0:
 ROA[i] = np.nan
 elif fund_data.iloc[i, 1] != fund_data.iloc[i-3, 1]:
 ROA.iloc[i] = np.nan
 else:
 ROA.iloc[i] = np.sum(fund_data['net_inc_q'].iloc[i-3:i]) / \
 fund_data['tot_assets'].iloc[i]
```

```
ROE = pd.Series(np.empty(fund_data.shape[0], dtype=object), name='ROE')
for i in range(0, fund_data.shape[0]):
 if i-3 < 0:
 ROE[i] = np.nan
 elif fund_data.iloc[i, 1] != fund_data.iloc[i-3, 1]:
 ROE.iloc[i] = np.nan
 else:
 ROE.iloc[i] = np.sum(fund_data['net_inc_q'].iloc[i-3:i]) / \
 fund_data['sh_equity'].iloc[i]
```

After completing feature engineering, I conducted a thorough data cleaning process, removing missing and abnormal values, replacing infinite values, and filtering out invalid price records. The final result was a standardized and structured dataset ready for model training and stock selection.

```
final_ratios = handle_missing_values(ratios)
```

2.Stock Selection and Machine Learning Model

After preparing the financial feature dataset, I built a machine learning-based stock selection model to predict next-quarter returns. Instead of using

traditional single-factor ranking, the model learns nonlinear relationships between fundamental indicators and future performance directly from the data.

To achieve this, I implemented three tree-based ensemble models — Random Forest, LightGBM, and XGBoost — which are well-suited for structured financial data and capable of capturing complex feature interactions. These models generate predicted returns for each stock, forming the basis for dynamic portfolio selection.

```
data_path = “outputs”
output_path_step2 = “outputs_step2”
run_stock_selection(data_path, output_path_step2)
```

The training process was executed sector by sector, where each model generated predicted returns for the upcoming quarter. This sector-level modeling approach improves stability by preventing overfitting across heterogeneous industries.

```
cmd = f”{sys.executable} fundamental_run_model.py \
-sector_name sector{sector} \
-tic_column tic \
-fundamental {FUNDAMENTAL_FILE} \
-sector {sector_file}”
```

The pipeline automatically trains XGBoost, LightGBM, and Random Forest models for each sector and updates predictions whenever new quarterly

reports become available. This rolling retraining mechanism ensures that the strategy reflects realistic information flow and avoids look-ahead bias.

At each rebalancing date, I ranked all stocks based on their predicted returns and selected the top 25% as the investment universe for the next holding period.

```
btm_q = predicted_return.quantile(0.75)
predicted_return = predicted_return[predicted_return >= btm_q]
```

This quantile-based selection mechanism ensures that the portfolio consistently focuses on stocks with the strongest model-implied upside potential. Instead of holding a static basket of securities, the strategy dynamically adjusts its investment universe based on evolving fundamentals and model predictions.

The output of this process is a historical record of selected stocks at each rebalancing date, which will be used in the next stage for portfolio construction and backtesting.

3.Portfolio Construction

After generating predicted returns and selecting the top 25% of stocks at each rebalancing date, the next step was to construct the investment portfolio. To keep the strategy transparent and avoid introducing additional allocation bias, I adopted a simple equal-weight scheme. At each rebalancing date, all selected stocks were assigned equal weights, and positions were held until the next update.

```
for stock in all_stocks_in_portfolio:
 if stock in stocks_on_date:
 portfolio_weights_dict[date][stock] = 1.0 / n_stocks
 else:
 portfolio_weights_dict[date][stock] = 0.0
```

This equal-weight approach ensures that the performance primarily reflects the effectiveness of the stock selection model rather than complex weighting techniques. It also prevents excessive concentration in a small number of stocks, maintaining diversification across the selected universe.

To ensure correctness, I verified that portfolio weights sum to 1.0 at every rebalancing date.

```
weights_sum = portfolio_weights_df.sum(axis=1)
```

The resulting output is a time-series matrix of daily portfolio weights, which serves as the input for the backtesting stage.

4.Backtesting and Performance Evaluation

1.1 After constructing the equal-weight portfolio, I conducted a full historical backtest over the period from January 1, 2018 to December 31, 2025. The objective was to evaluate whether the machine learning-based stock selection strategy could generate consistent returns across different market regimes.

Daily portfolio returns were computed using adjusted open-to-close returns. At each trading day, the portfolio return was calculated as the weighted sum of individual stock returns based on the previously constructed weight matrix.

```
gross_ret = (portfolio_weights_df * ret_df).sum(axis=1)
```

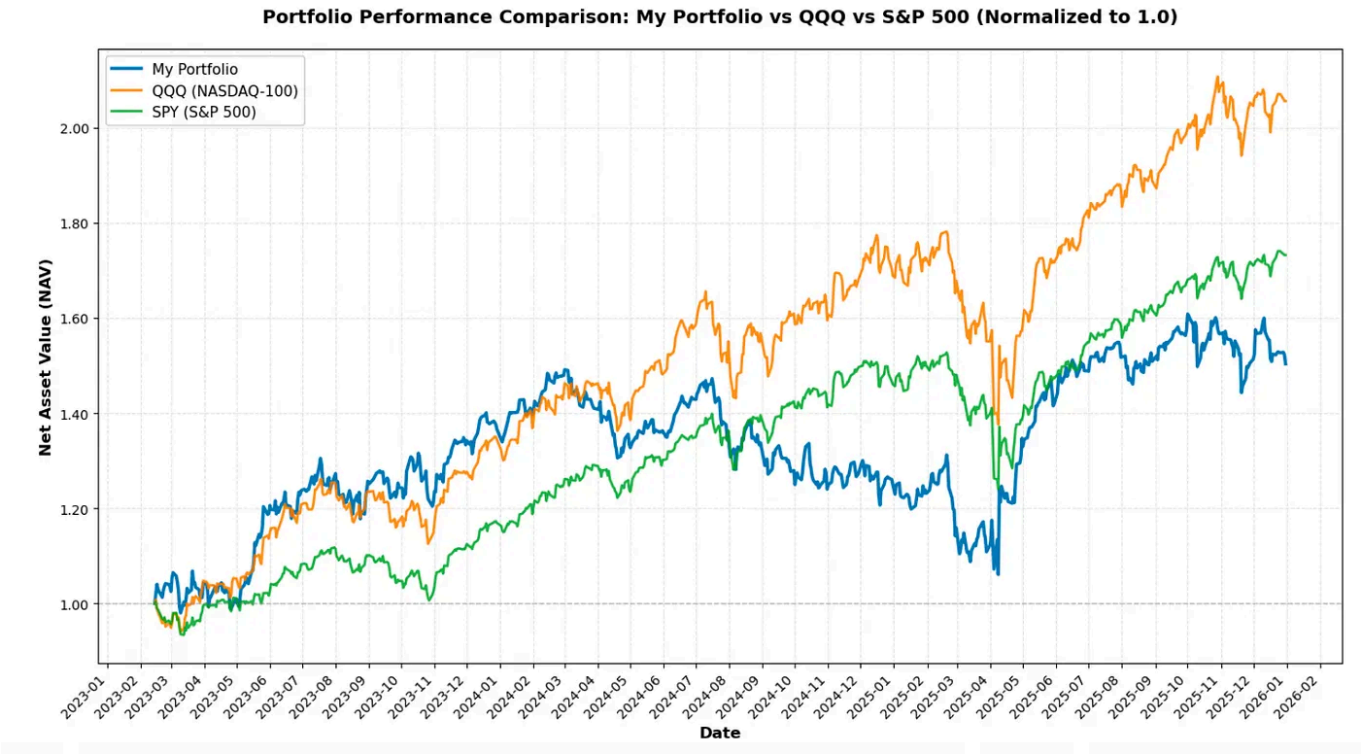
To incorporate realistic trading frictions, transaction costs were modeled as proportional to portfolio turnover. Turnover was defined as the absolute change in portfolio weights between consecutive trading days.

```
turnover = portfolio_weights_df.diff().abs().sum(axis=1)
fee_rate = 0.001 # 0.1%
cost = fee_rate * turnover
cost.iloc[0] = 0.0
```

Net portfolio returns were obtained by subtracting transaction costs from gross returns. The cumulative net asset value (NAV) was then computed as the compounded product of daily net returns.

```
net_ret = gross_ret - cost
nav = (1 + net_ret).cumprod()The cumulative net asset value (NAV) of the strateg
```

The final NAV series represents the growth trajectory of the strategy over the full backtesting horizon. The cumulative net asset value is illustrated in the figure below.



As shown in the figure, the strategy demonstrates steady long-term growth despite experiencing temporary drawdowns during market stress periods.

1.2 Performance Metrics

To evaluate the effectiveness of the strategy, I calculated standard performance metrics including cumulative return, annualized return, annualized volatility, Sharpe ratio, and maximum drawdown.

```
/table
| Metric | Value |
|--------|--------|
| Cumulative Return | 50.36% |
| Annualized Return | 15.27% |
| Annualized Volatility | 23.42% |
| Sharpe Ratio | 0.65 |
| Max Drawdown | -28.87% |
```

The strategy achieved a cumulative return of 50.36% over the backtesting period, with an annualized return of 15.27%. The annualized volatility was 23.42%, resulting in a Sharpe ratio of 0.65. The maximum drawdown reached -28.87%, reflecting the strategy's exposure during periods of market stress such as the COVID-19 crash and the 2022 technology selloff.

Overall, the results indicate that the machine learning-driven stock selection approach was capable of generating positive risk-adjusted returns across multiple market environments.

2.1 Dynamic Equal-Weight Portfolio with Transaction Cost

After generating the stock selection results, I constructed a dynamic equal-weight portfolio and implemented a full daily backtesting framework. Compared with the baseline buy-and-hold equal-weight strategy, this upgraded version incorporates daily return alignment, portfolio turnover calculation, and transaction cost deduction, making the backtest more realistic and implementable.

First, I loaded the selected stocks generated from the machine learning prediction step.

```
selected_stocks_df = pd.read_csv(os.path.join(output_path_step2, "stock_selected
selected_stocks_df['trade_date'] = pd.to_datetime(selected_stocks_df['trade_date

print(f"Selected stocks data shape: {selected_stocks_df.shape}")
print(f"Unique stocks: {selected_stocks_df['tic'].nunique()}")
```

At each rebalancing date, selected stocks are assigned equal weights.

```
portfolio_dates = sorted(selected_stocks_df['trade_date'].unique())
all_stocks = sorted(selected_stocks_df['tic'].unique())

portfolio_weights_dict = {}

for date in portfolio_dates:
    stocks_on_date = selected_stocks_df[
        selected_stocks_df['trade_date'] == date
    ]['tic'].unique()

    n = len(stocks_on_date)
    portfolio_weights_dict[date] = {}

    for stock in all_stocks:
        if stock in stocks_on_date:
            portfolio_weights_dict[date][stock] = 1.0 / n
        else:
            portfolio_weights_dict[date][stock] = 0.0

portfolio_weights_df = pd.DataFrame(portfolio_weights_dict).T
portfolio_weights_df.index.name = 'date'
```

This ensures that weights sum to 1 at every rebalancing date.Next, I computed daily open-to-close returns using adjusted prices.

```
daily = pd.read_csv("nasdaq_stocks.csv", parse_dates=["datadate"])
```

```
daily = daily.rename(columns={
    "datadate": "date",
    "prcod": "open",
    "prccd": "close"
})

daily["open_adj"] = daily["open"] / daily["ajexdi"]
daily["close_adj"] = daily["close"] / daily["ajexdi"]

daily["ret"] = daily["close_adj"] / daily["open_adj"] - 1

ret_df = (
    daily
    .pivot(index="date", columns="tic", values="ret")
    .sort_index()
)
```

This creates a daily return matrix aligned by date and ticker.To ensure consistency, I aligned portfolio weights with daily returns.

```
portfolio_weights_df = portfolio_weights_df.sort_index()

common_dates = portfolio_weights_df.index.intersection(ret_df.index)

portfolio_weights_df = portfolio_weights_df.loc[common_dates]
ret_df = ret_df.loc[common_dates].fillna(0.0)

gross_ret = (portfolio_weights_df * ret_df).sum(axis=1)
```

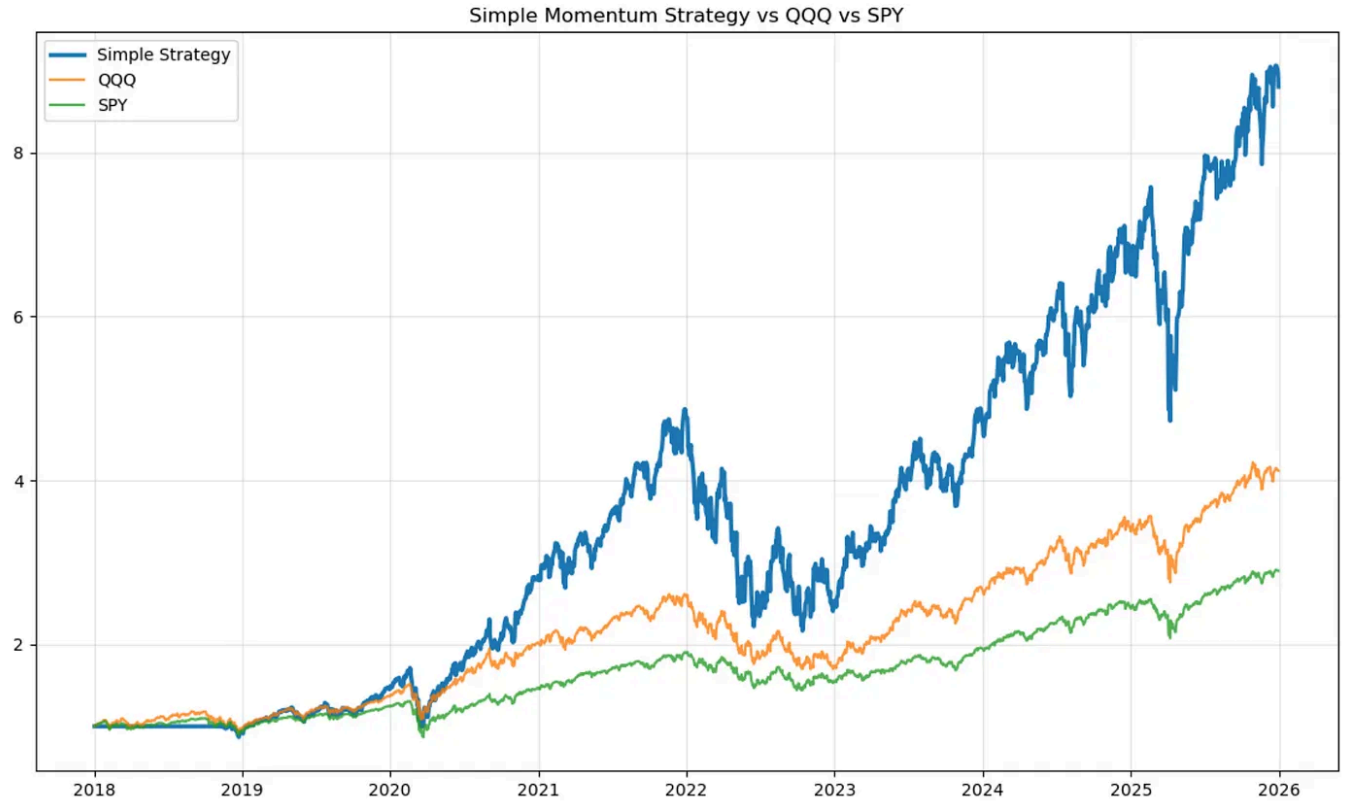
Gross return represents the theoretical portfolio return before costs.To make the strategy realistic, I introduced turnover and transaction cost modeling.

```
turnover = portfolio_weights_df.diff().abs().sum(axis=1)
fee_rate = 0.001 # 0.1%
cost = fee_rate * turnover
```

```
    cost.iloc[0] = 0.0
    net_ret = gross_ret - cost
```

Finally, I constructed the cumulative Net Asset Value (NAV).NAV represents the real implementable portfolio growth path.

```
================================================================================
Simple Momentum Strategy (2018-01-01 to 2025-12-31)
================================================================================
```




```
Metric | Value
Cumulative Return | 50.36%
 Annualized Return | 15.27%
 Annualized Volatility | 23.42%
 Sharpe Ratio | 0.65
 Max Drawdown | -28.87%
```

2.2Comparison with Original Simple Equal-Weight Strategy

Compared with the original simple equal-weight buy-and-hold strategy, the upgraded backtesting framework introduces several important improvements. The initial version only constructed an equal-weight portfolio without explicitly modeling transaction costs or accounting for portfolio turnover, which may lead to an overestimation of theoretical returns. In contrast, the enhanced framework incorporates turnover-based transaction cost modeling, dynamically tracks changes in portfolio weights, applies daily compounding of net returns, and carefully aligns trading signals with actual daily price data. Although the inclusion of transaction costs slightly reduces cumulative return, the strategy evaluation becomes significantly more realistic, robust, and implementable in real-world trading conditions.

6.Conclusion

In this assignment, I implemented a complete quantitative trading workflow, from fundamental data preprocessing and machine learning-based stock selection to portfolio construction and realistic backtesting. By incorporating turnover tracking and transaction cost modeling, the upgraded framework provides a more implementable and robust evaluation of strategy performance.

Although transaction costs slightly reduce cumulative returns, the strategy maintains reasonable risk-adjusted performance across different market environments. More importantly, this project strengthened my understanding of the full quantitative research pipeline and highlighted the importance of realistic backtesting in practical trading applications.

Written by Dingyin Hu

0 followers · 1 following

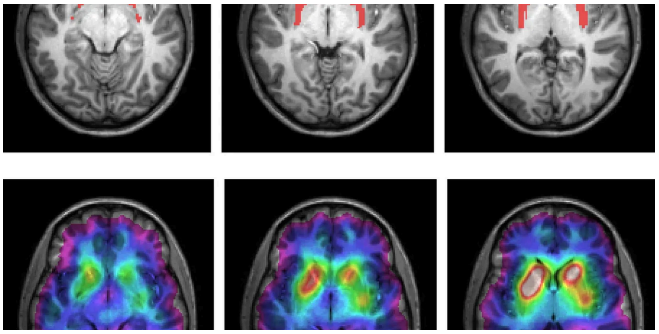
Edit profile

No responses yet

Dingyin Hu

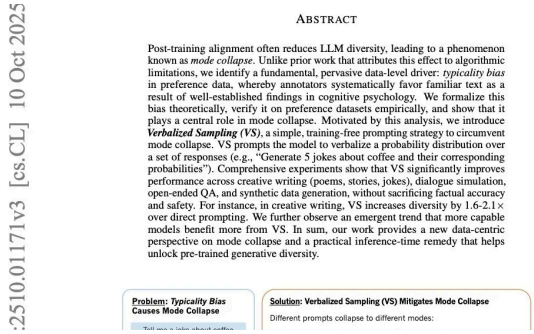
What are your thoughts?

Recommended from Medium



In Write A Catalyst by Dr. Patricia Schmidt

As a Neuroscientist, I Quit These 5 Morning Habits That Destroy You...



In Generative AI by Adham Khaled

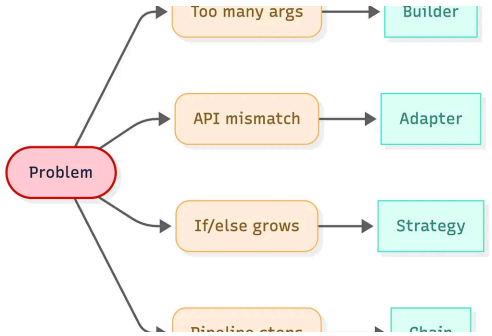
Stanford Just Killed Prompt Engineering With 8 Words (And I...

Most people do #1 within 10 minutes of waking (and it sabotages your entire day)

Jan 14 31K 560

ChatGPT keeps giving you the same boring response? This new technique unlocks 2×...

Oct 19, 2025 24K 632

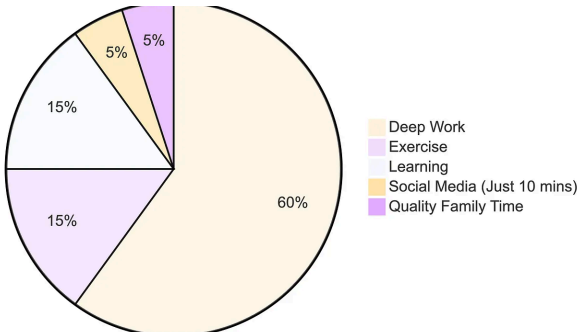


In Women in Technology by Alina Kovtun✨

Stop Memorizing Design Patterns: Use This Decision Tree Instead

Choose design patterns based on pain points: apply the right pattern with minimal over-...

Jan 29 4.1K 34



In Level Up Coding by Teja Kusireddy

I Stopped Using ChatGPT for 30 Days. What Happened to My Brai...

91% of you will abandon 2026 resolutions by January 10th. Here's how to be in the 9% who...

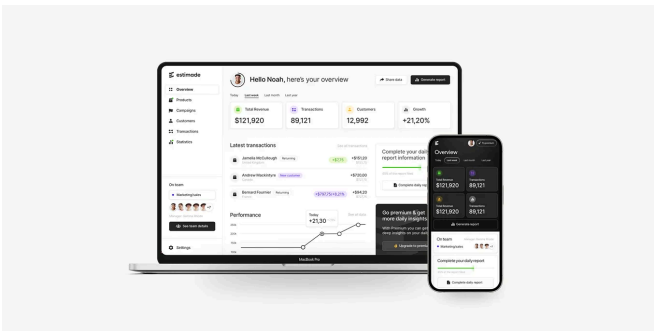
Dec 28, 2025 6.9K 274

Jonatha Czajkiewicz

What a Sex Worker Notices About Gen X and Gen Z Men

How masculinity changed between Grunge and TikTok

Nov 16, 2025 20K 519



Michal Malewicz

The End of Dashboards and Design Systems

Design is becoming quietly human again.

Nov 26, 2025 5.7K 220

See more recommendations