Alphagen Report - Qiang Li

Q1: Alphagen Framework Overview

The paper titled **"Generating Synergistic Formulaic Alpha Collections via Reinforcement Learning"** presents a novel framework for discovering and combining alpha factors in quantitative trading using reinforcement learning. Alpha factors are signals derived from historical stock data that help predict market trends. The authors aim to find a set of formulaic alphas that work synergistically to improve predictive accuracy, unlike traditional methods that mine alphas individually without considering their combined effects.

Key Contributions:

1. **Synergistic Alpha Mining**: The proposed method focuses on generating a set of alpha factors that perform well together, rather than evaluating them in isolation. This is crucial because in practice, alpha factors are often used in combination to model complex stock market behaviors.
2. **Reinforcement Learning Approach**: The framework leverages RL to explore the vast space of possible formulaic alphas. The performance improvement of a combination model is used as the reward signal to guide the RL-based alpha generator. This contrasts with traditional genetic programming (GP)-based methods, which can struggle with large search spaces.
3. **Optimization Objective**: The performance of the combined alpha factors is optimized directly in the RL process, rather than relying on indirect measures like mutual information or information coefficients (IC) between individual alphas, which can miss the potential synergies between factors.
4. **Experimental Results**: The framework is tested on real-world stock market data, demonstrating superior performance in both prediction accuracy and investment simulation compared to traditional approaches. The method achieves higher returns in backtesting, showing practical applicability for stock trend forecasting.

Key Findings:

- The proposed framework outperforms traditional methods like genetic programming and other RL-based approaches that optimize single alpha performance. It can continuously find synergistic alpha sets even as the pool size grows.
- The experiments show that combining multiple well-performing alphas, which individually might not be diverse (as measured by mutual IC), can result in better overall predictive performance due to the synergy between the factors.

Q2-a: Data Download from binance(download_data.py)

The primary objective of the script is to:

1. Retrieve Binance kline data for 15-minute (15m) and 1-hour (1h) intervals.
2. Ensure that data is fetched for the past two years, regardless of the current date.

3. Store the data in a format that can be easily analyzed for qlib.

The Binance API requires the following parameters to fetch kline data:

- **symbol**: The trading pair (e.g., BTCUSDT).
- **interval**: The time interval (e.g., 15m, 1h).
- **start_time**: The time from which to start fetching data.
- **end_time**: The end time for fetching the data.

request sample:

```
BASE_URL = 'https://api.binance.com'
KLINE_ENDPOINT = '/api/v3/klines'

url = BASE_URL + KLINE_ENDPOINT
params = {
    'symbol': symbol,
    'interval': interval,
    'startTime': start_ts,
    'limit': limit
}
response = requests.get(url, params=params)
```

To ensure that the data format aligns with the original format used by Alphagen, the `DumpDataALL` function was called to convert the CSV data obtained from Binance into binary file data with feature-based naming.

```
from data_collection.qlib_dump_bin import DumpDataAll

DumpDataAll(
      csv_path='./my_data/basic',
      qlib_dir='./my_data/qlib',
      max_workers=50,
      exclude_fields="Open_time,code,Close_time,Ignore",
      symbol_field_name="code",
      date_field_name='Open_time',
      freq='min'
).dump()
```

Q2-b: Generate Alpha with Alphagen(main.py)

Step1: Modification on `./alphagen_qlib/stock_data.py`

In order to make the original framework compatible with 15-minute frequency data, it is necessary to first modify the `stock_data.py` file.

1. `class FeatureType` add new feature types **TAKER_BUY_QUOTE_ASSET_VOLUME, TAKER_BUY_BASE_ASSET_VOLUME, QUOTE_ASSET_VOLUME and NUMBER_OF_TRADES**
2. `cal: np.ndarray = D.calendar(freq='min')`

```
return (QlibDataLoader(config=exprs, freq='min')
        .load(self._instrument, real_start_time, real_end_time))
```

Step2: Creation of `main.py`

The `main.py` script is based on the `train_maskable_ppo.py` file in alphagen, with the main parameter modifications including:

1. The time range for the training/validation/test sets

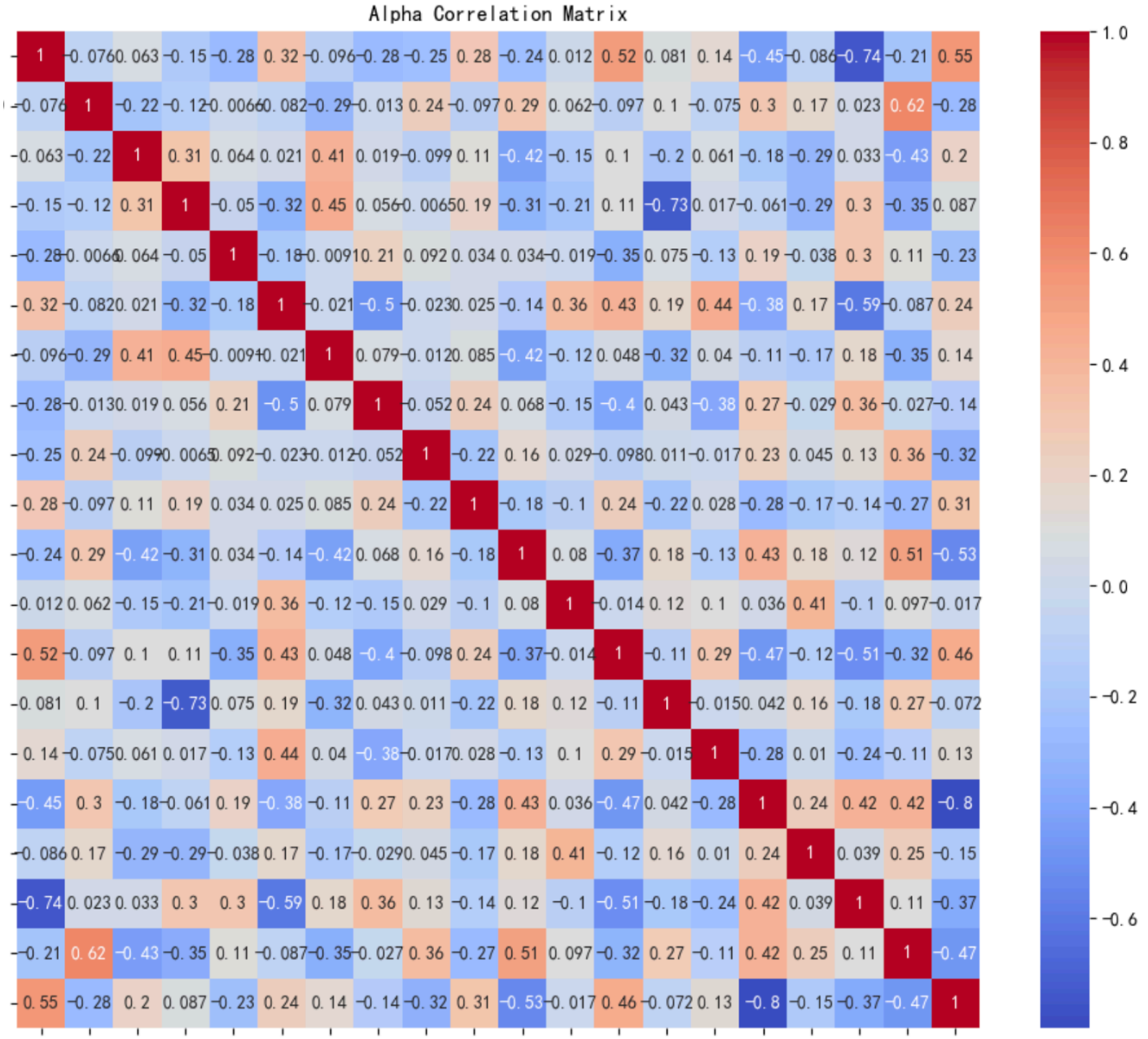
```
data_train = StockData(instrument=instruments,
                       start_time='2023-11-01 00:00:00',
                       end_time='2024-02-29 23:45:00')
data_valid = StockData(instrument=instruments,
                       start_time='2024-03-01 00:00:00',
                       end_time='2024-05-31 23:45:00')
data_test = StockData(instrument=instruments,
                      start_time='2024-06-01 00:00:00',
                      end_time='2024-08-31 23:45:00')
```

2. Changing the target to the next tick's return
3. Updating the instrument to `15min_symbols`

During the process of writing the `main.py` code, I gained a deeper understanding of the data and model framework of Alphagen and Qlib. By modifying the original training code, which was designed for daily frequency data, to support 15-minute frequency, I expanded the flexibility and scalability of the initial model.

Step3: Test Result

Using the factors generated in the last JSON file as sample results from the model, there are a total of 20 factors. The correlations between the factors are shown in the figure below, and overall, the correlations are relatively low.



IC result:


```
{'Mul(Greater(Constant(10.0),Mad(Min(Max($close,10),20),10)),$volume)': 0.02520899252998125,
 'Std(Sub(Constant(1.0),Ref($taker_buy_quote_asset_volume,50)),50)': 0.019175599933108283,
 'Sub(Constant(0.01),Abs(Delta($taker_buy_base_asset_volume,20)))': 0.018726113220436765,
 'Div(Sub(Constant(−0.5),$close),Sub(Constant(−1.0),Sub($high,Constant(0.01))))': −0.018601756198231362,
 'Med(Sub(Sub(Sub(Sub(Constant(2.0),$taker_buy_base_asset_volume),Constant(−5.0)),Constant(2.0)),Constant(2.0)),30)': −0.01830619657831921,
 'Mul(Constant(−5.0),Delta(Med($taker_buy_quote_asset_volume,30),30))': 0.015477811115442356,
 'Div(Less(Max(Mul(Constant(−5.0),EMA($close,40)),30),Constant(−0.5)),Mad(Std($quote_asset_volume,20),10))': −0.010128860392738288,
 'Std(Max(Med($volume,30),30),30)': −0.009848836850793604,
 'Sub(Constant(−5.0),Var($quote_asset_volume,20))': 0.007751053278860332,
 'Div(Greater(Min(Sub(Var(Div(Mul($quote_asset_volume,Constant(−5.0)),Constant(30.0)),50),Constant(0.5)),10),Constant(−5.0)),$quote_asset_volume)': −0.007083621576809044,
 'Add(Div(Max($low,30),$close),Constant(0.01))': 0.003551901576339474,
 'Div(Add(Add(Greater($close,Constant(2.0)),$volume),Add(Constant(−5.0),$volume)),Constant(−0.01))': −0.003502085122527684,
 'Div(Greater(Constant(10.0),Mean($high,10)),$open)': −0.0032890600590323497,
 'Sub(Constant(10.0),Mad(Med($taker_buy_quote_asset_volume,20),20))': 0.003279931049751879,
 'Sub(Greater($number_of_trades,Constant(10.0)),Var($open,50))': 0.0032376385280343165,
 'Div(Sub(Constant(−2.0),$close),Greater(Constant(1.0),Std($open,10)))': −0.0029800096212540456,
 'Delta(Sub(Constant(0.01),$low),40)': −0.002217924614175795,
 'Sub(Mul(Std(Sum($high,50),10),Abs(EMA($low,50))),$taker_buy_quote_asset_volume)': 0.0016839169917161683,
 'Sub(Sub(Div(Constant(10.0),$quote_asset_volume),Constant(−0.01)),Constant(−5.0))': 0.0012774494891749904,
 'Mad(Sub(Constant(−5.0),WMA($quote_asset_volume,10)),10)': −0.0004237127877547596}
```


The low IC of factors generated by reinforcement learning can be attributed to several reasons:

1. **Large Search Space**:
   In financial factor mining, the search space is vast. Although RL has strong exploration capabilities, it might struggle to efficiently identify the optimal factors in such a large space, especially in uncertain financial environments. RL could explore many low-quality factors, leading to an overall lower IC.
2. **Training Convergence Challenges**:
   RL algorithms often require significant time to converge and are sensitive to hyperparameters. In financial factor generation, RL may not have sufficient iterations, or its hyperparameters may not be tuned optimally, causing the model to fail in discovering high-IC factors.
3. **Complexity of Financial Markets**:
   Financial markets are inherently complex and noisy. Even with powerful algorithms like RL, the model can be influenced by random market fluctuations, resulting in factors that lack stable predictive power. While RL can generate complex factors, their effectiveness (IC) might be reduced due to market noise and overfitting.
4. **Insufficient or Noisy Data**:
   If the training data is insufficient or noisy, the RL model may struggle to learn robust factors. Financial markets are highly noisy, especially with high-frequency data (e.g., 15-minute intervals), which can lead to lower IC values.

Q3-a: Generate Alpha with Baseline Methods

Part1: GP-based methods

Part2: Deep Symbolic Regression method

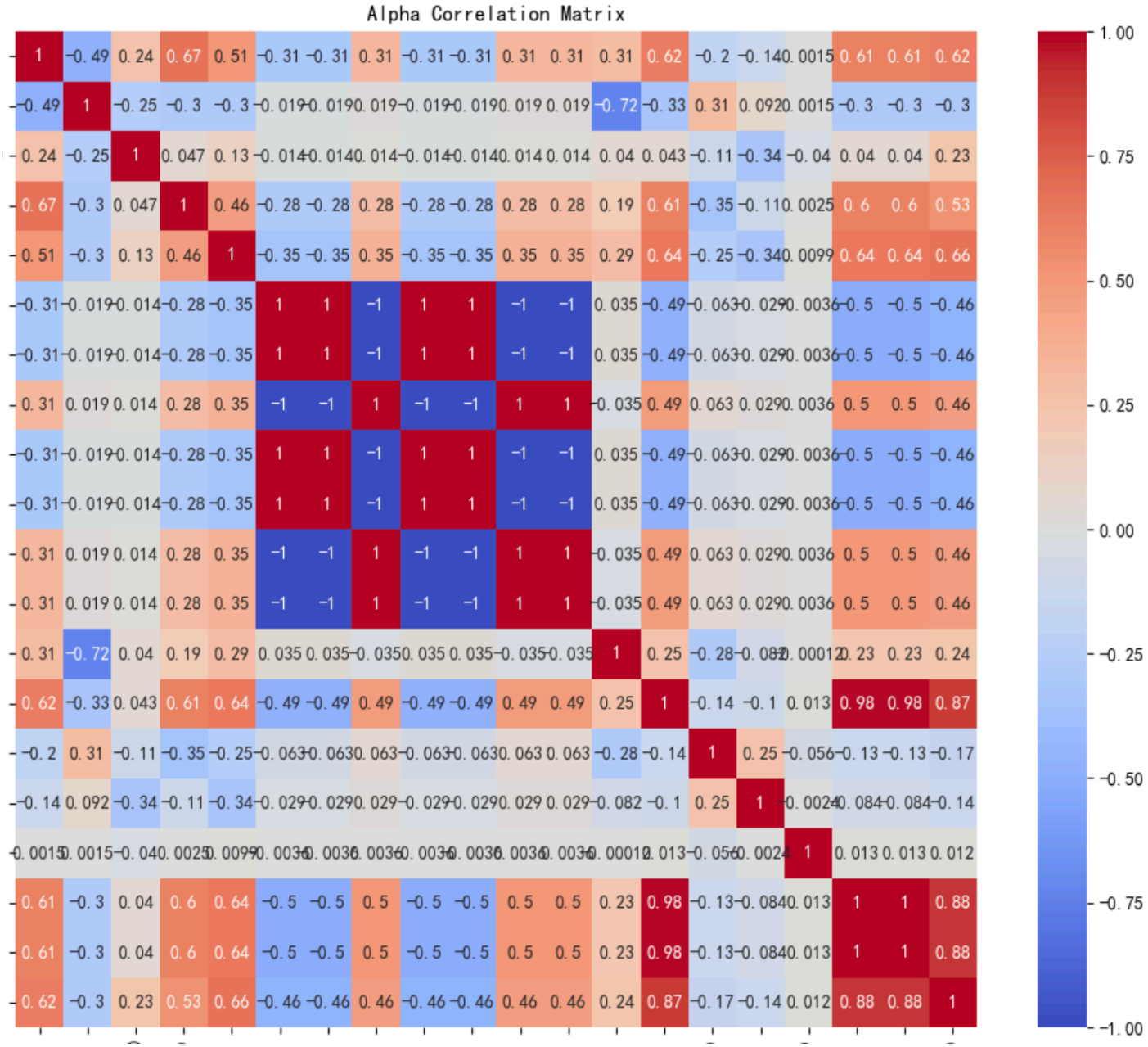
Generated over 6000 valid factors

```
  39     6.97        0.0351135        7        0.0709356        0.0709356      0.00s
[{'pool': 0, 'res': {'ic_test': 0.04374369606375694, 'ic_valid': 0.038303442299366, 'ric_test': 0.03734898194670677, 'ric_valid': 0.03543933480978012}}, {'pool': 10, 'res': {'ic
_test': 0.0439443401992321, 'ic_valid': 0.037955351173877716, 'ric_test': 0.03836800903081894, 'ric_valid': 0.03340128809213638}}, {'pool': 20, 'res': {'ic_test': 0.045059420168
39981, 'ic_valid': 0.042687367647886276, 'ric_test': 0.04231569916009903, 'ric_valid': 0.04054960981011391}}, {'pool': 50, 'res': {'ic_test': 0.0450330413877964, 'ic_valid': 0.0
4247811809182167, 'ric_test': 0.04293843358755112, 'ric_valid': 0.0400967113673687}}, {'pool': 100, 'res': {'ic_test': 0.05038342624902725, 'ic_valid': 0.045124601572752, 'ric_t
est': 0.04599550738930702, 'ric_valid': 0.04213476181030273 4}}]
```

Top 20 factors by IC value

```
[('Var(Min($volume,50),20)', 0.0758745205636593),
 ('Delta(Var(Std($taker_buy_base_asset_volume,30),30),40)',
  -0.05162403773475841),
 ('Cov($volume,Min($quote_asset_volume,20),40)', 0.050199764548912645),
 ('Ref(Cov($number_of_trades,$taker_buy_base_asset_volume,20),40)',
  0.048769702462143676),
 ('Ref(Greater(Max(Ref($taker_buy_base_asset_volume,20),10),EMA(Sum(Constant(0.5),10),10)),30)',
  0.04012289020783837),
 ('Corr(Constant(-5.0),$taker_buy_base_asset_volume,30)',
  -0.036979326919210645),
 ('Corr($taker_buy_base_asset_volume,Constant(-5.0),30)',
  -0.036979326919210645),
 ('Corr($taker_buy_base_asset_volume,Constant(-0.01),30)',
  0.03697932683356596),
 ('Corr(Constant(-30.0),$taker_buy_base_asset_volume,30)',
  -0.03697932672861117),
 ('Corr($taker_buy_base_asset_volume,Constant(-30.0),30)',
  -0.03697932672861117),
 ('Cov($taker_buy_base_asset_volume,Constant(5.0),30)', 0.03697932661487261),
 ('Cov(Constant(10.0),$taker_buy_base_asset_volume,30)', 0.036979326060434525),
 ('Div(Add(Mul(Constant(-5.0),Delta(Var($taker_buy_base_asset_volume,50),20)),$high),$close)',
  0.036806970175362455),
 ('Min($taker_buy_base_asset_volume,50)', 0.03541237943957939),
 ('Div(Delta(Std($open,20),40),$close)', -0.034560412186037115),
 ('Med(Ref(Delta(Med(Div($taker_buy_base_asset_volume,Constant(0.5)),10),20),20),10)',
  -0.034436234540122033),
 ('Cov(Corr(Corr($open,$high,40),$taker_buy_quote_asset_volume,10),$close,10)',
  -0.034197153410158596),
 ('Min($volume,50)', 0.034188606245876656),
 ('Min(Greater($volume,Constant(-10.0)),50)', 0.034188606245876656),
 ('Div(Max(Min($quote_asset_volume,50),30),$close)', 0.03415628056589085)]
```

Top 20 factors' correlation



The high correlation between the top 20 factors generated by a genetic programming model can be explained by several factors:

1. **Similar Structure of Expression Trees**:
   Genetic programming works by evolving expression trees to create factors. Over time, the evolutionary process may favor certain types of tree structures or mathematical operators that consistently yield better results. As a result, many of the top-performing factors might share similar structural elements, leading to high correlation between them.

2. **Overfitting to Historical Data**:
   GP models optimize factors based on historical data. If the optimization process strongly overfits to specific patterns in the data, it can result in factors that capture the same market behavior or trend. Even though they appear different, these factors may react similarly to certain market conditions, increasing their correlation.

3. **Limited Diversity in Features**:
   The factors generated by GP are typically based on a limited set of underlying features (e.g., price, volume, moving averages). If the model is using the same features repeatedly in different combinations, the resulting factors may end up being highly correlated because they are ultimately based on the same input data.
4. **Selection Pressure**:
   In genetic programming, high-performing factors are more likely to be selected for reproduction and mutation. If certain mathematical forms or combinations of features consistently perform well, they will dominate the population, leading to many similar factors being generated and retained in the top 20.
5. **Narrow Search Space**:
   GP may explore only a narrow subset of the potential factor space due to limitations in mutation and crossover operations. If the search space is not broad enough, the algorithm may converge on a set of similar solutions that perform well, but are not sufficiently diverse.

Q4-a: Rank, Compare and Filter Generated Alpha(alpha_ranking.py)

Breakdown of what each function does:

1. load_alphas(): Loads CSV files containing alpha data from a specified folder into a DataFrame.
2. rank_alphas(): Ranks alphas based on their correlation with a target variable, filtering out weak correlations.
3. cal_corr(): Calculates the correlation matrix of the remaining alphas, identifying and removing highly correlated factors (above a defined threshold).
4. save_unique_alphas(): Saves the filtered, unique alphas back to a new folder.
5. main(): Ties all the steps together—loading, ranking, filtering, and saving unique alphas.

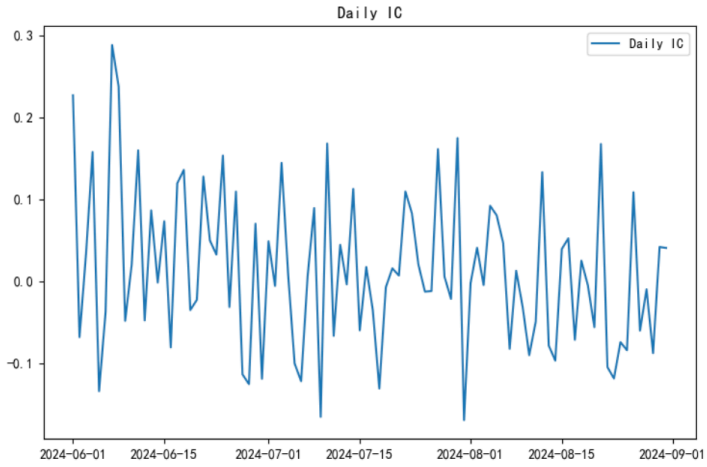
The detailed running result of alpha ranking function is presented in Q2-b step3.

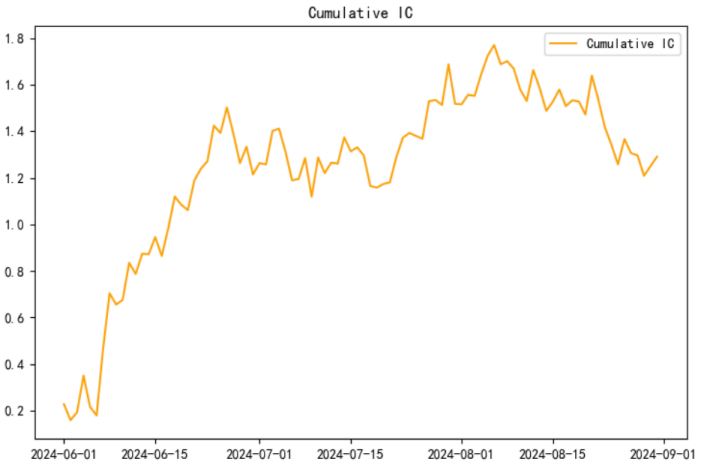
Q4-b: Alpha Backtest

Selected alphas backtest result

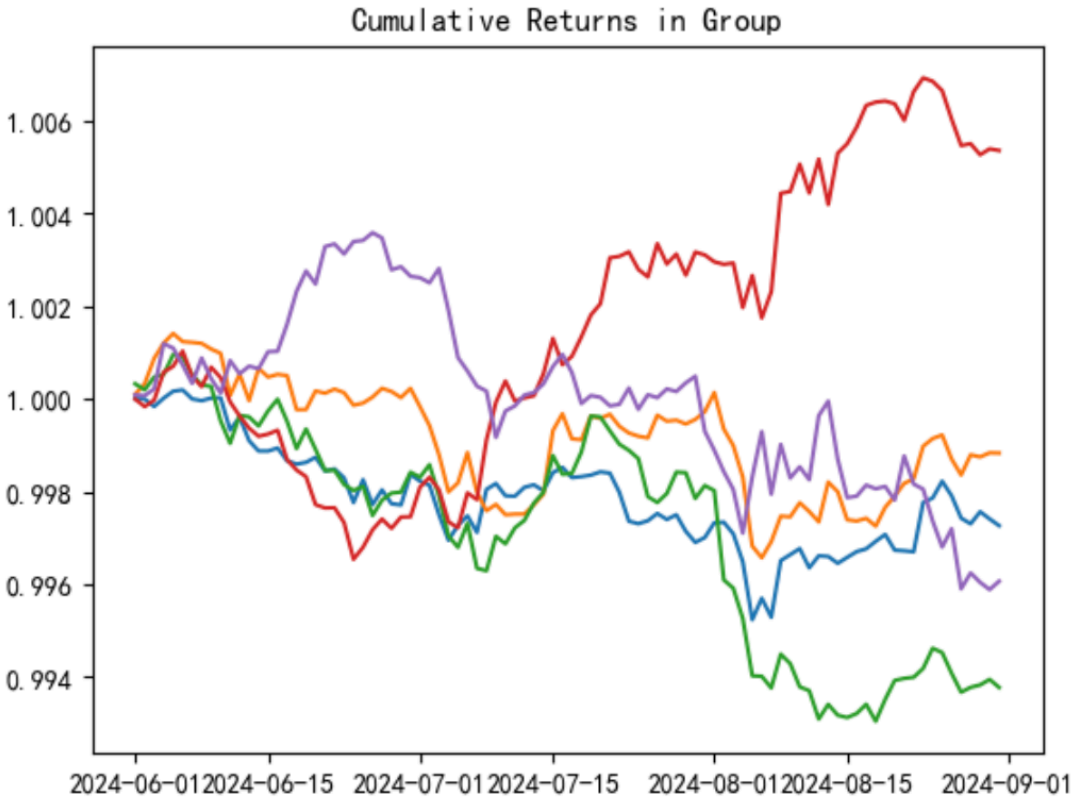
- **Mul(Greater(Constant(10.0),Mad(Min(Max($close,10),20),10)),$volume)**: Generated by RL model

Daily IC & Cumulative IC Plot:



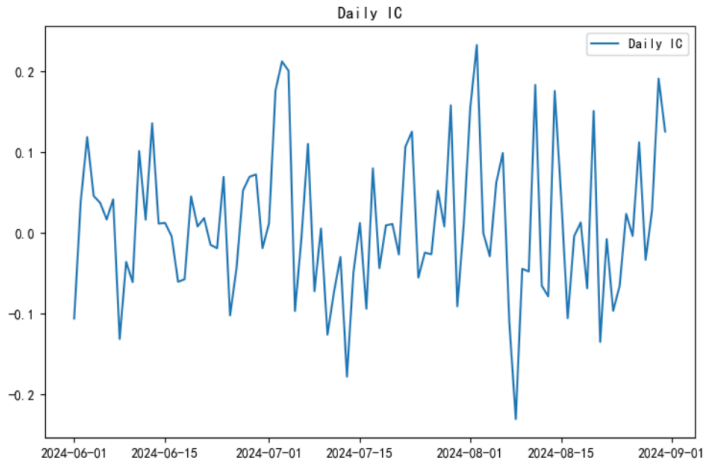


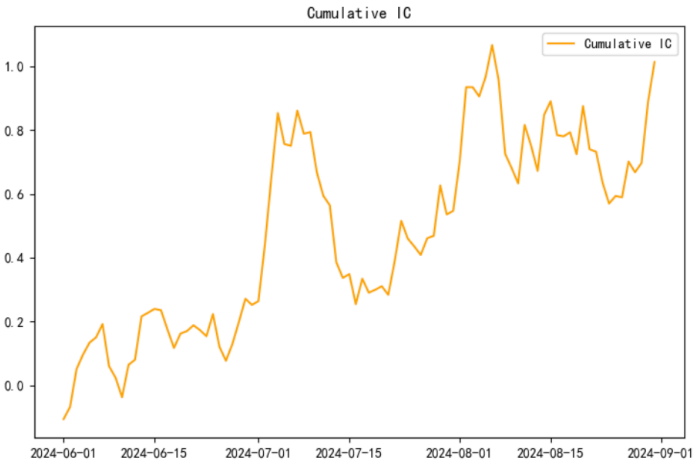
Cumulative Return in Groups:



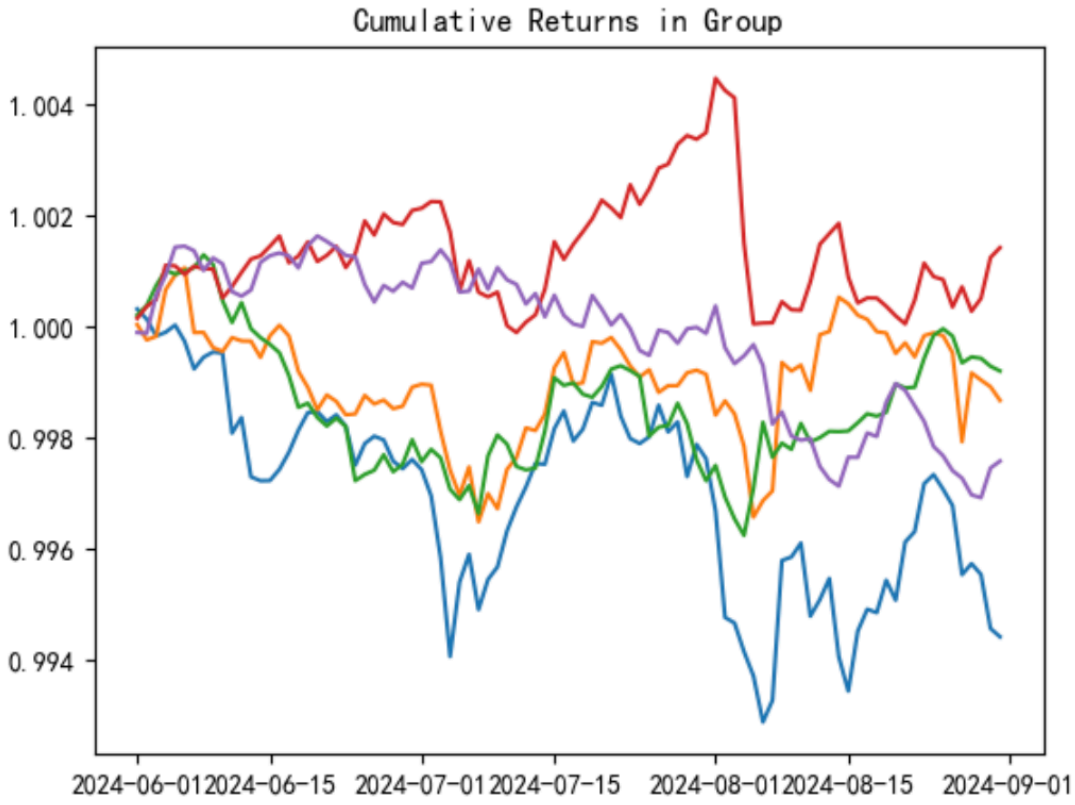
- **Corr($open,$low,30)**: Generated by GP model

Daily IC & Cumulative IC Plot:





Cumulative Return in Groups:



From the above result, I find that although Reinforcement learning generated factors may exhibit lower IC values compared to those produced by genetic programming, but they **tend to be more stable** for several reasons:

1. **Exploration and Generalization**: RL focuses on exploration across the entire solution space, which can result in discovering more robust, generalizable factors that perform steadily, even if their IC is lower.
2. **Reward-Based Optimization**: RL directly optimizes factor performance using a reward function, aligning better with the ultimate objective, leading to more consistent results across different market conditions.

3. **Noise Management**: RL's iterative approach helps it adapt to noisy environments, which enhances the stability of the factors compared to GP, which may overfit to specific market conditions during evolution. This adaptability results in more reliable, though slightly less aggressive, factors.

Q5: Conclusion

In this project, I familiarized and further developed the alphagen framework to generate alpha factors for crypto market. I began by researching methods such as genetic programming and reinforcement learning to generate alpha factors, understanding the strengths and weaknesses of each. I then learned to calculate Information Coefficient and tested alpha performance using techniques like simple long-short strategies. I created Python scripts to load, rank, and backtest alpha factors, while visualizing results with cumulative returns and correlation heatmaps. This involved integrating `pandas`, `matplotlib`, and other libraries for effective analysis.

Through this process, I deepened my understanding of financial factor research and backtesting, learning how to combine theory with practical coding to analyze complex data in an efficient and structured way.