
AlphaBuild: Generating Formulaic Alphas on a Wider Range of Stock Data

Xiyuan Yang, Mowen Ruan, Zhijie Chen

Abstract

In the field of quantitative analysis, it has become a standard paradigm to transform historical market data into alpha factors that predict future market trends. Formulaic alphas, in particular, are often more favored due to their high interpretability. Existing methods of automatically mining formulaic alphas generally adopt either Genetic Programming or Reinforcement Learning (RL). The former method often struggles to mine pools of alphas with optimized combined performance due to the computational complexity of navigating an exponentially growing search space; applications of the latter also face challenges, because time bottlenecks hinder the exploitation of market-wide data, in turn limiting the factors' expressive powers.

We propose AlphaBuild, an RL-based methodology which employs Proximal Policy Optimization to mine a pool of factors that optimizes the performance of downstream models on a broader database. Using the downstream model performance directly as the reward, we maintain multiple smaller-sized factor pools, and obtain the final factors using methods including Principal Component Analysis. This methodology leverages smaller pool sizes to reduce the time complexity of a single mining procedure, while also increasing the degree of parallelism of the process and capturing whole market information through random sampling. Experiment results demonstrate that AlphaBuild is capable of mining pools of high-quality formulaic alphas that achieve superior performance in backtests.

1 Introduction

In the field of quantitative analysis, it is nowadays common practice to transform raw historical stock data (*e.g.*, opening price, lowest price, vwap) into quantities that can predict future returns and risks of the securities. Such quantities are called alpha factors, or simply alphas. Due to its profound connection to investment outcomes, discoveries of high-quality alphas with strong predictive power have become one of the central topics in this domain.

While the construction of alphas can take various forms, they share a common core idea: an alpha is any function that takes in historical stock data (both cross-sectional and time-series) as input, and outputs a predictive score for each security, indicating its predicted recent return. Current methods of automatically mining alphas can be divided into two categories. The first category is machine learning-based. Historical data are used to train large models to predict future returns (the label). Examples of this paradigm include LSTM [Hochreiter and Schmidhuber, 1997] and REST [Xu et al., 2021]. These methods often excel in capturing the non-linearities and high-order patterns present in financial data, and therefore exhibit stronger expressive power comparing to their formulaic counterparts. However, those black boxes often have very poor interpretability, making them prone to overfitting. On the other end of the spectrum are formulaic factors. Formulaic factors, as its name suggests, have an explicit expression. At the cost of complexity, these factors are highly interpretable, and are often backed by concrete economic rationale. Those properties make formulaic factors generally more robust, and are more favored in most situations. Kakushadze [2016] exemplifies such factors.

Formulaic alphas are traditionally constructed by human experts by expertise, experience, and instincts; yet methods are also emerging to automate this process [Cui et al., 2021, Zhang et al., 2020, Lin et al., 2019, Yu et al., 2023]. Primarily adopting Genetic Programming (GP), these methods face a common limitation. Due to the exponential growth of search space when simultaneously mining a pool of alphas (the Cartesian product of multiple single-factor search spaces), it is computationally infeasible to directly find a pool of alphas that optimizes the performance of the downstream combination model. As a compromise, individual alphas are mined independently and then combined and filtered by their correlation (*e.g.*, mutual IC). However, experiments [Yu et al., 2023] demonstrate that such methodology is invalid in the sense that it does not optimize the performance of the downstream combination model.

To address this issue, we adopt a reinforcement learning-based method to mine a pool of alphas that optimizes the performance of downstream combination models. We employ a sequence generator (with constraints) to generate valid alpha factor expressions; a policy gradient-based algorithm is used to train the generator. Contrary to traditional methods of mining single alphas, we directly use the performance of the downstream model as the reward function. This methodology can automatically produce a collection of formulaic alphas along with a set of weights that achieves optimal downstream performance. Furthermore, dimension reduction techniques are employed upon factors obtained from RL. By detecting and extracting key patterns learned by the pool of factors and discarding redundant information, we obtain a smaller pool of post-processed factors that are more mutually independent, and yet inherit most expressive power from the original pool. This simplification of factor pools allows for a more robust downstream model, and facilitates the interpretation of factor contributions.

Experiment results demonstrate that our factor mining pipeline is capable of automatically mining a compact pool of formulaic alpha factors that optimizes downstream performance from historical stock data.

2 Related Works

GP-based alpha mining This kind of method employs Genetic Programming [Cui et al., 2021, Zhang et al., 2020, Lin et al., 2019]. Each factor is uniquely associated with an expression tree, *i.e.*, its chromosome. A crossover is performed by swapping a subtree between two parent factors to produce a new offspring; a mutation is performed by altering a single node (an operator, a variable, or a constant) within a factor’s associated expression tree. By continuously applying these genetic operators and selecting for better-performing factors over many generations, a population of “fittest” (with strongest predictive power) factors is finally obtained. However, GP-based methods still search for individually well-performing factors at their cores, while neglecting the efficiency of factors’ combination; for these methods, it is generally considered impossible to directly search for a *set* of alphas that work best together under a certain combination model, even if the combination model is a linear one, due to the exploding complexity of maintaining a large population.

Alpha mining through reinforcement learning To adapt to traditional auto-regressive generators, a factor expression is converted to and represented by its associated reverse Polish notation. Using downstream performance as the reward function for RL, the agent is trained to write a set of legal formulaic alpha expressions that optimize downstream combination models’ performance. Compared to GP-based methods, RL-based methods are more effective at navigating a large search space and finding a whole pool of alphas with high performance. This method of alpha mining is exemplified by AlphaGen [Yu et al., 2023]. However, such methods struggle to handle market-wide data due to time bottlenecks. Therefore, the predictive power of factors discovered via these methods is often limited.

3 Methodology

3.1 Problem Formulation

We consider the historical data of k features of n stocks over T trading days. The information of stock j on day t is depicted as a vector $x_{t,j} \in \mathbb{R}^k$. Then the information of the market on day t is depicted as a matrix

$$X_t = [x_{t,1} \quad \dots \quad x_{t,n}]^T \in \mathbb{R}^{n,k}.$$

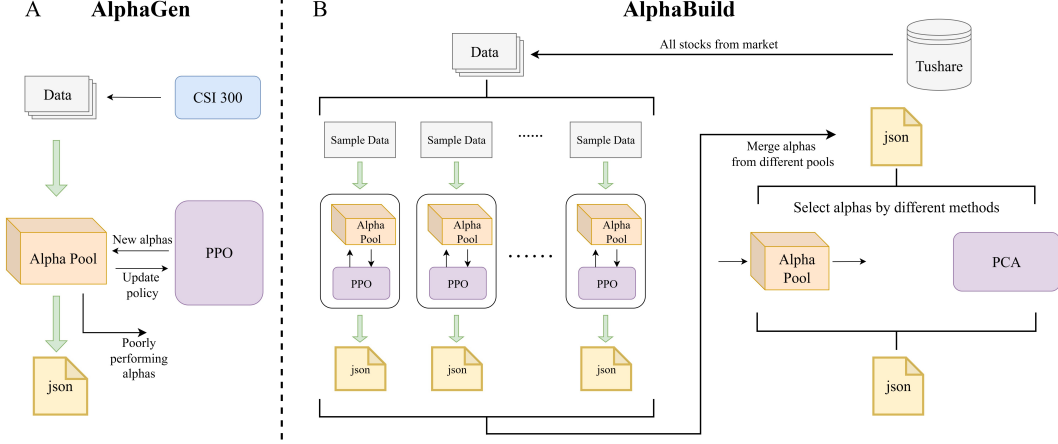


Figure 1: (A) AlphaGen uses data from CSI 300. It maintains an alpha pool with a fixed size. Proximal Policy Optimization (PPO) is employed to consistently mine new alphas and optimize the performance of the entire pool. (B) Our methodology samples data from market-wide historical stock data. This practice allows for stronger generalization capability of the model. AlphaBuild mines multiple smaller-sized alpha pools, from which final alphas are selected. The final selection can be performed either by a similar method to pool maintenance, or by Principal Component Analysis (PCA).

We define a factor to be a function $f : \mathbb{R}^{n,k} \rightarrow \mathbb{R}^n$ which maps the information of the market on one day to the predicted future scores of the n stocks. Let \mathcal{F} denote the factor pool, which is a list of factors.

To avoid unnecessary complication in the subsequent processes (e.g., the validity of the method in 3.2.2), we introduce a normalization operator $\mathcal{N}(\cdot)$, which is defined as

$$\mathcal{N}(v) = \frac{v - \bar{v}}{\sigma(v)},$$

where $\sigma(\cdot)$ is the standard deviation. This operator applies a linear transformation to a vector such that its elements have a mean of 0 and a standard deviation of 1. Consequently, the normalized vector is unit length. We will henceforth omit writing \mathcal{N} before a factor or a return vector. In other words, we tacitly treat f as $\mathcal{N} \circ f$ and y as $\mathcal{N}(y)$, where $y_t \in \mathbb{R}^n$ is the future return of the n stocks on day t .

To quantify the predictive power of a factor and the correlation between two factors, we introduce the Pearson correlation coefficient $\rho(\cdot, \cdot)$.

$$\rho(v, w) = \frac{\text{Cov}(v, w)}{\sigma(v) \cdot \sigma(w)} = \frac{\langle v, w \rangle}{\|v\| \cdot \|w\|},$$

Note that $\rho(\cdot, \cdot)$ is invariant under the $\mathcal{N}(\cdot)$ operator. The predictive power of a factor is measured by its (mean) information coefficient (IC).

$$\text{IC}_t = \rho(f(X_t), y_t), \quad \bar{\text{IC}} = \bar{\rho}_t(f(X_t), y_t) = \frac{1}{T} \sum_{t=1}^T \text{IC}_t.$$

Let the downstream combination model be $c(X; \mathcal{F}, \theta)$, where θ is the parameter vector of the combination model. We use a linear model here, and $w = \theta$ denotes the weights. The problem can then be formulated as the optimization problem below, where $c(\cdot; \mathcal{F}, \theta)$ can be treated as a “mega-factor”.

$$\mathcal{F}^* = \operatorname{argmax}_{\mathcal{F}} \left\{ \max_{\theta} \bar{\rho}_t(c(X_t; \mathcal{F}, \theta), y_t) \right\}.$$

3.2 Modeling the Alpha Generation Process

3.2.1 Sequential Representation of a Factor

Because traditional auto-regressive generators only handle sequences, a formulaic factor needs to be represented by a linear sequence. We use the reverse Polish notation (RPN) to uniquely represent a factor expression. A factor is then represented by a linear sequence of *tokens*. A token is a fundamental building block of factors. It can be an operator, a variable, a constant, or a sequence indicator (BEG for the beginning, and SEP for the end of an expression). An example of a factor and its sequential representation is shown below.

Expression: $-10.0 + \$high \cdot \$change$

BEG	-10.0	\$high	\$change	Mul	Add	SEP
-----	-------	--------	----------	-----	-----	-----

Figure 2: An Example of a Factor and its Sequential Representation

3.2.2 Calculation of Reward

The methodology maintains a factor pool of a fixed size m along with an optimal weight vector $w \in \mathbb{R}^m$. Intuitively speaking, the reward for writing a valid factor is measured by the performance of the combination model after merging the new factor. The loss function $\mathcal{L}(w)$ is defined as the sum of MSE and a Lasso regularization term.

$$\mathcal{L}(w) = \frac{1}{nT} \sum_{t=1}^T \|c(X_t; \mathcal{F}, w) - y_t\|^2 + \lambda \cdot \|w\|_1,$$

where λ is the regularization strength.

The reward is calculated as follows. Add the new factor to the existing pool, with a randomly initialized weight for it. The optimal weight is calculated by gradient descent with respect to the loss function defined above. Remove the factor with the least absolute weight. The reward is given by (with updated \mathcal{F} and w)

$$r = \bar{\rho}_t(c(X_t; \mathcal{F}, w), y_t).$$

In the particular case where $\lambda = 0$, MSE can be minimized directly by least squares. $[y_1 \dots y_T]$ is projected onto the subspace of $\mathbb{R}^{n,T}$ spanned by $\{[f_j(X_1) \dots f_j(X_T)]\}_{j=1}^m$. We shall formulate the problem in a more intuitive and more computationally simple language. A factor f_j is here represented by a vector $v_j \in \mathbb{R}^{nT}$, obtained by squashing the matrix $[f_j(X_1) \dots f_j(X_T)]$. Let $Y \in \mathbb{R}^{nT}$ be the vector obtained by squashing the matrix $[y_1 \dots y_T]$. Let $M = [v_1 \dots v_m] \in \mathbb{R}^{nT,m}$. The goal is to find $w^* \in \mathbb{R}^m$ such that

among the w 's that minimize $\|Mw - Y\|$, w^* has minimal 2-norm.

Theorem 3.1. *A solution to the problem defined above is given by*

$$w^* = V\Sigma^+U^TY,$$

where $M = U\Sigma V^T$ is the singular value decomposition of M .

See Appendix F for the proof of this theorem.

3.2.3 Generation Procedure as a Markov Decision Process

We model the alpha generation procedure as a (non-stationary) Markov decision process (MDP). The key components of this MDP are defined as follows.

States A state is an initial segment of a valid RPN, which is a sequence of tokens. The length of an RPN is capped at 20 to prevent overfitting. Hence there are only finite states.

Actions An action attempts to append a token to the end of the current state. Yet not all sequences of tokens are valid RPNs. Any actions that violate one of the following syntactic rules are masked.

- A stack is used to attempt to evaluate the expression. No errors should occur before the end of the sequence.
- A time-series operator must take a time-delta (*e.g.*, 5d means a time difference of 5 days) as its last parameter.
- A multi-token expression should not equal a constant.
- The SEP token is allowed only if the current state is an RPN.

Syntactic validity does not suffice for the validity of an RPN. A syntactically valid RPN can have semantic invalidities, such as division by 0, a Log operator having a negative operand, etc. Such errors are not directly detected during the generation procedure. Instead, expressions with semantic invalidities receive a reward of -1, the least possible value for a reward (because $\rho(\cdot, \cdot) \in [-1, 1]$). That is to say, we discourage the agent from writing a semantically invalid expression.

It can be shown that syntactic and semantic validity together ensure the validity of a (partial) RPN.

Transition The next state is generated by appending to the end of the current state the token corresponding to the action taken, with a probability of 1. Here, a state is a sequence of tokens, and an action is a token.

Reward No reward (reward = 0) is given for partial RPNs. When a SEP is written down or the maximal allowed length is reached, the RPN is parsed and the reward is calculated (*cf.* 3.2.2). Because the reward depends on the current factor pool, the MDP is non-stationary.

Discount factor The discount factor γ quantifies the agent’s preference for immediate rewards over future rewards. Because it is generally considered harder to discover longer yet equally effective alphas, we set $\gamma = 1$ and do not penalize longer expressions.

3.3 Training with Policy Gradient-based Methods

Based on the MDP defined above, we use Proximal Policy Optimization (PPO) [Schulman et al., 2017] to optimize a stochastic policy network, which maps the current state to a probability distribution over all possible actions. Because of the validity requirements for RPN alpha expressions, we adopt Maskable PPO [Huang and Ontanon, 2022], a variant of PPO that incorporates an invalid action mask mechanism to only sample from the set of legal actions.

During the mining process, we focus on the performance of the entire alpha pool. Instead of assessing individual alphas, we select alphas that optimize the overall performance. When the current size of the pool is less than m , the newly mined alpha is simply added to it. When the size limit is reached, the process in 3.2.2 is executed. In this way, we maintain a dynamic alpha pool and continuously optimize its overall performance as the algorithm progresses.

3.4 Post Processing for Dimension Reduction

Distinct factors, despite being individually mined, can exhibit significant correlation. This redundancy can compromise the robustness and predictive power of downstream models (*e.g.*, portfolio allocation models) and complicate the interpretation of factor contributions. To mitigate this issue, we employ dimension reduction techniques. The goal is to transform the initial set of factors into a more compact set of less correlated components that preserve the essential information.

3.4.1 Principal Component Analysis

Among various dimension reduction techniques, Principal Component Analysis (PCA) aims to eliminate linear relationships within a set of factors. Fix a positive integer $1 \leq p \leq nT$. Each of the m factors is represented by a vector in \mathbb{R}^{nT} .¹ Let $A \in \mathbb{R}^{m, nT}$ be the matrix whose rows are the

¹It can be treated as a matrix squashed into a vector.

factor vectors. The goal is to find an orthonormal list of p vectors that “best describe” the distribution of the original vector set. More formally, PCA finds a list $\{w_j\}_{j=1}^p \subseteq \mathbb{R}^{nT}$ of orthonormal vectors, called the principal components of A , which is the solution to the following optimization problem.

$$w_j = \underset{\|w\|=1}{\operatorname{argmax}} \sigma(Aw) \quad \text{subject to} \quad w_i^\top w_j = 0, \forall i < j.$$

Theorem 3.2. *The j -th principal component of $A \in \mathbb{R}^{m,nT}$ is an eigenvector of $A^\top A$ corresponding to its j -th greatest eigenvalue, with each eigenvalue included as many times as its multiplicity.*

See Appendix F for the proof of this theorem.

By selecting an appropriate value for p , or by truncating the eigenvalues of $A^\top A$ at a certain lower bound threshold, we obtain a smaller, orthogonal (independent) set of factors that captures most patterns learned by the original factors. The extra clarity and simplicity bring benefits to downstream models without sacrificing much performance.

3.4.2 Factor Filtering

We employ a similar method to 3.2.2 to select a final pool of alphas from existing ones. To be specific, we maintain a factor pool (a list of factors along with their weights) that maximizes the target function

$$\text{Ref}(\text{open}, -11)/\text{Ref}(\text{open}, -1) - 1.$$

The selection procedure is iterative. A candidate factor is added to the pool and the weights are optimized, after which the factor with the least absolute weight is removed. This cycle of addition, optimization and removal is repeated for all factors mined previously.

4 Experiments

4.1 Data and Evaluation Settings

4.1.1 Data

While AlphaGen ([Yu et al., 2023]) uses CSI 300 and CSI 500 indices of China A-shares as the stock set, our approach enhances the model’s generalization capability by leveraging data from all 3,000 stocks across the China A-shares market². To achieve this purpose, we perform data calibration (change the dataset from CSI 300 to all stock data in China A-shares) before training. We select 10 raw features, shown in Table 1. The target is set to be the 10-day return of the stocks, selling/buying at the opening price, calculated by

$$\text{Ref}(\text{open}, -11)/\text{Ref}(\text{open}, -1) - 1.$$

The dataset is split by date into a training set (2016/01/01 to 2020/12/31), a validation set (2020/01/01 to 2021/12/31), and a test set (2021/01/01 to 2025/01/01).

Table 1: Raw Features for Factor Mining

Metric	Definition
OPEN	First trade price of the day.
HIGH	Highest price reached during the day.
LOW	Lowest price reached during the day.
CLOSE	Last trade price of the day.
PRE_CLOSE	Previous day’s closing price.

Continued on the next page ...

²Our data is obtained from <https://tushare.pro/>, a free, open source Python financial data interface package which collects and stores stocks and other financial data.

Table 1: Raw Features for Factor Mining

Metric	Definition
CHANGE	$\text{CLOSE} - \text{PRE_CLOSE}$.
PCT_CHG	$\frac{\text{CHANGE}}{\text{PRE_CLOSE}} \times 100\%$.
VOLUME	Total shares traded daily.
AMOUNT	Total value traded daily; approximately $\text{VOLUME} \times \text{CLOSE}$.
ADJ_FACTOR	Multiplier for price adjustments (<i>e.g.</i> , due to stock splits).

4.1.2 Evaluation Methods and Metrics

To compare our framework with existing formulaic alpha generation approaches, we first apply (plain) RL algorithms such as Maskable PPO, and LLM-based selection methods with prompt engineering to emulate manual factor selection. We then optimize both upstream (data adaptation) and downstream (factor filtering and dimension reduction) processes building upon plain RL algorithms (*cf.* 4.2.2). Our optimized methods exhibit superior performance compared to baseline methods in backtests (*cf.* 4.3).

We primarily adopt two metrics to measure the performance of our models, namely IC and Rank IC (RIC), where the latter is defined as

$$\text{RIC}_t = \text{Corr}(\text{rank}(f_{i,t}), \text{rank}(r_{i,t+1}))$$

to mitigate the impact of outliers.

To emulate manual factor mining process, we implement a prompt engineering framework in which an LLM generates candidate factors from given stock data, and automated multi-round evaluation is then performed. Prompt templates and outputs are detailed in Appendix E.

4.2 Training Results

4.2.1 Replication Experiment

We begin by replicating experiments conducted by AlphaGen [Yu et al., 2023]. Building upon their work, we adjust the parameters to address the identified time bottleneck.

Figure 3, 4, 5 and 6 in Appendix B respectively demonstrate the best IC value, mean length of generated expressions, and mean value of IC and RIC, with different pool sizes (20 and 100) and random seeds. The results indicate that larger pool sizes lead to both higher factor quality and better IC and RIC, and that with sufficient training steps, the model’s performance converges for distinct random seeds. However, excessive iteration steps can lead to premature convergence of factor expressions towards overly simplified forms, while significantly prolonging training duration.

Consequently, our optimization focuses on three goals:

- maintaining the RL model’s performance during the adaptation of market-wide data;
- achieving moderate factor expression complexity;
- cutting excessive training time, especially for large pool sizes (100) and wide data scope (all 3,000 stocks across the China A-shares market), without sacrificing much performance.

4.2.2 Optimization of Sampling and Factor Filtering

In the experiments, we observe that as the factor pool expands (`pool_size` = 1, 10, 30, 50, 100), it takes an increasing amount of time to mine the same number of factors. A major reason is that each update requires computing the mutual IC between every pair of factors in the pool. Hence the time needed for those steps is linearly related to the pool size³. To improve time efficiency without sacrificing factors’ quality, we optimize the factor mining process as follows.

³This assertion can be verified by analysis of Theorem 3.1.

- **Data sampling:** Due to the substantial computation bottleneck caused by the scale of the adapted dataset (5-year history, 3,000 stocks), we divide the factor mining process into two steps. Step 1 generates a broad set of candidate factors using only 10% randomly sampled stock time-series data to reduce runtime. Step 2 filters those factors by applying full-sample evaluations (a refined selection), obtaining the final factor pool.
- **Trade-offs in pool size selection:** The original experiment’s large pool size and iteration steps result in slow mining speeds. However, excessively reducing either the pool size or iteration steps leads to shorter expression length and higher repetition rates of simplistic factors (*e.g.*, basic factors such as \$amount). To reach an equilibrium, we adopt the following three parameter sets:

- (a) pool_size = 1, steps = 50000, random_seed_num = 300.
- (b) pool_size = 10, steps = 100000, random_seed_num = 30.
- (c) pool_size = 100, steps = 100000, random_seed_num = 3.

In the parameters settings above, total numbers of mined factors $\text{factor_num} = \text{pool_size} \times \text{random_seed_num}$.

Hence, our optimization of plain RL algorithms primarily focuses on two components:

- implementing data sampling and factor filtering mechanisms with calibrated pool size parameters for mining acceleration;
- integrating PCA to streamline factor representation (*cf.* 3.4.1).

4.3 Investment Simulation

To ensure that the generated factors perform effectively in the real, noisy stock market, we backtest them using the following strategy. We compound the generated factors based on their expressions and weights, using it as a scoring function to assess and rank all available stocks on a given day (excluding special cases such as trading suspensions). We then buy the stocks ranking top 15. The sell signal is triggered when any of the held stocks falls out of the top 500 ranking. The proceeds from these sales are then used to purchase the highest-ranked available stocks. Table 3 outlines all the hyperparameter settings.

We perform 5 backtests, using factors generated by 5 different methods:

1. **(Baseline)** 20 factors generated by LLM, serving as an emulation of manual factor mining;
2. **(Baseline)** 20 factors generated by plain RL algorithms used by AlphaGen.
3. 500 factors generated by optimized RL methods after data calibration and data sampling for speedup.
4. 100 features derived from the previous 500 factors by dimension reduction using PCA.
5. 101 factors obtained by applying additional factor filtering algorithm to the previous 500 factors.

Figure 7, 8, 9, 10 and 11 in Appendix B respectively demonstrate the backtesting result of the 5 methods listed above. The blue benchmark line represents the CSI 500 Index during the period⁴. The red strategy line represents our model’s cumulative return. The yellow excess line indicates the excess return rate after subtracting market price fluctuations ($\text{Excess Rate} = \text{Strategy Return Rate} - \text{Benchmark Return Rate}$). Table 2 presents more detailed backtesting results, including metrics such as total return, win rate, excess return, and annualized excess return.

Compared to baseline methods, the optimization for our methods, including data calibration, PCA dimension reduction and factor filtering, leads to an improvement of all metrics during backtests. Leveraging market-wide stock data for data calibration enhances generalizability. PCA and factor filtering enable the factor pool to be distilled without sacrificing much performance.

⁴Due to variations in strategy selection, the end times of different backtests exhibit slight discrepancies.

Table 2: Backtest Results by Different Methods

Method	Total Return	Maximum Drawdown	Sharpe Ratio	Win Rate	Excess Return	Ann. Excess Return
LLM Response	71.05%	24.86%	0.896	55.21%	81.12%	16.04%
Plain RL	74.82%	21.82%	0.941	55.31%	85.82%	16.79%
Optimized RL (500 Factors)	121.98%	19.24%	1.230	57.79%	136.50%	24.07%
PCA	103.55%	19.45%	1.120	56.55%	116.54%	21.26%
Factor Filtering	111.58%	21.81%	1.134	57.28%	126.03%	19.52%

5 Conclusion

We present AlphaBuild, an RL-based methodology for automated formulaic factor generation. By mining multiple smaller-sized pools and later applying dimension reduction techniques including PCA and factor filtering, our methodology addresses the time bottleneck present in current RL-based methods. This improvement enables the model to leverage market-wide data and in turn discover more robust alphas with stronger predictive power and combined downstream performance. Experiment results demonstrate that AlphaBuild is capable of mining a pool of high-quality formulaic alphas which exhibits superior performance in backtests compared to baseline methods.

References

- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.
- Wentao Xu, Weiqing Liu, Chang Xu, Jiang Bian, Jian Yin, and Tie-Yan Liu. Rest: Relational event-driven stock trend forecasting. In *Proceedings of the Web Conference 2021, WWW '21*, pages 1–10, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383127. doi: 10.1145/3442381.3450032. URL <https://doi.org/10.1145/3442381.3450032>.
- Zura Kakushadze. 101 formulaic alphas. *Wilmott Magazine*, 84:72–80, 12 2016. doi: 10.2139/ssrn.2701346. URL <https://ssrn.com/abstract=2701346>. Available at SSRN.
- Can Cui, Wei Wang, Meihui Zhang, Gang Chen, Zhaojing Luo, and Beng Chin Ooi. Alphaevolve: A learning framework to discover novel alphas in quantitative investment. In *Proceedings of the 2021 International conference on management of data*, pages 2208–2216, 2021.
- Tianping Zhang, Yuanqi Li, Yifei Jin, and Jian Li. Autoalpha: an efficient hierarchical evolutionary algorithm for mining alpha factors in quantitative investment, 2020. URL <https://arxiv.org/abs/2002.08245>.
- Xiaoming Lin, Ye Chen, Ziyu Li, and Kang He. Stock alpha mining based on genetic algorithm. *Technical Report, Huatai Securities Research Center*, 2019.
- Shuo Yu, Hongyan Xue, Xiang Ao, Feiyang Pan, Jia He, Dandan Tu, and Qing He. Generating synergistic formulaic alpha collections via reinforcement learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5476–5486, 2023.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- Shengyi Huang and Santiago Ontanon. A closer look at invalid action masking in policy gradient algorithms. *The International FLAIRS Conference Proceedings*, 35, 05 2022. doi: 10.32473/flairs.v35i.130584.

A Full Hyperparameters

All experiments are conducted on a server equipped with 8 NVIDIA RTX 3090 GPUs.

Table 3: Hyperparameter Settings

Hyperparameter	Value
Hyperparameters for Reinforcement Learning	
λ (Lasso regularization strength)	5×10^{-3}
device	torch.device("cuda:0")
γ (discount factor of MDP)	1.0
entropy coefficient	0.01
batch size	128
policy: features extractor class	LSTMSharedNet
LSTM: number of layers	2
LSTM: dropout rate	0.1
LSTM: dimension of hidden states	128
Hyperparameters for Backtest Results	
Backtest period	2021/01/01 to 2025/01/01
Initial capital	RMB 10,000,000
Rebalance frequency	1
Maximum holding	200
Minimum sell rank	500
Loss threshold	0.05
Profit threshold	0.05
Benchmark	CSI 500 Index ⁵
Commission	0.0003
Transfer fee	0.00001
Stamp duty	0.0005

B Replication Experiment Results

Figure 3, 4, 5 and 6 respectively demonstrate the best IC value, mean length of generated expressions, and mean value of IC and RIC, with different pool sizes (20 and 100) and random seeds.

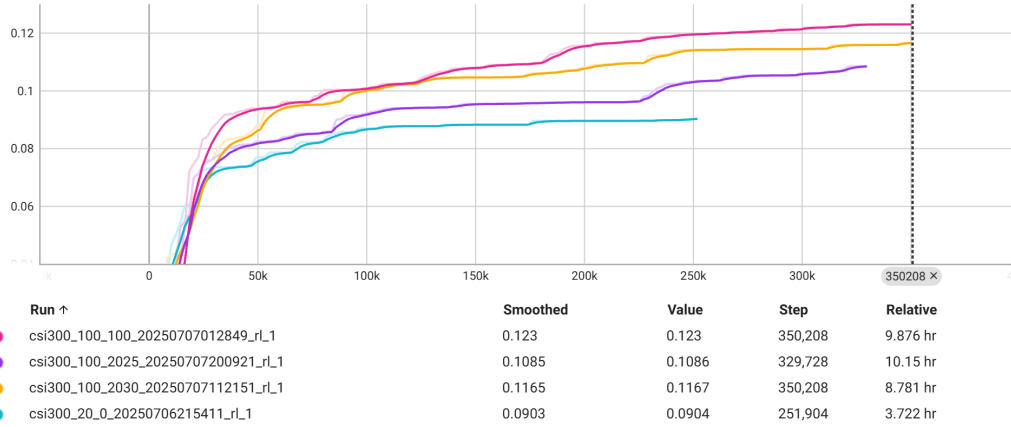


Figure 3: Best IC with Different Hyperparameters

⁵The CSI 500 Index reflects the performance of 500 small-cap and mid-cap A-share stocks traded on the Shanghai and Shenzhen stock exchanges. It excludes the CSI 300 constituents (the 300 largest stocks by market capitalization) to focus on smaller companies.

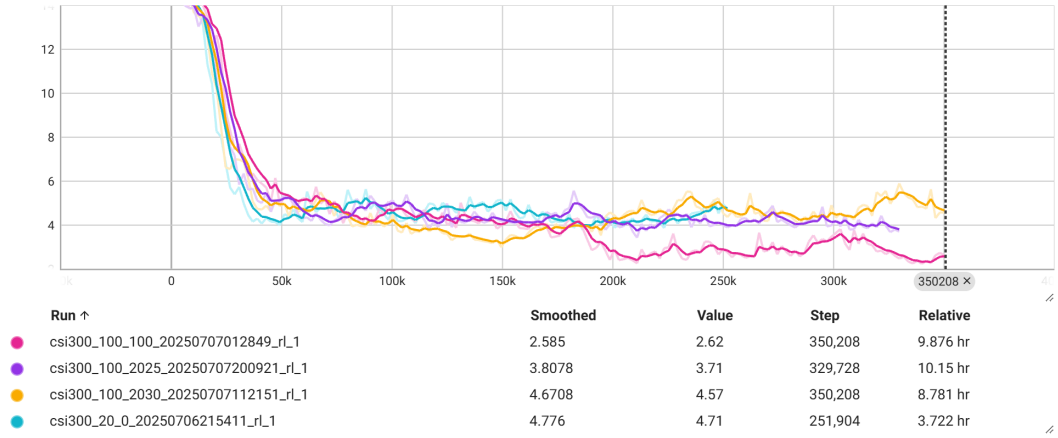


Figure 4: Mean Length of Generated Expressions with Different Hyperparameters

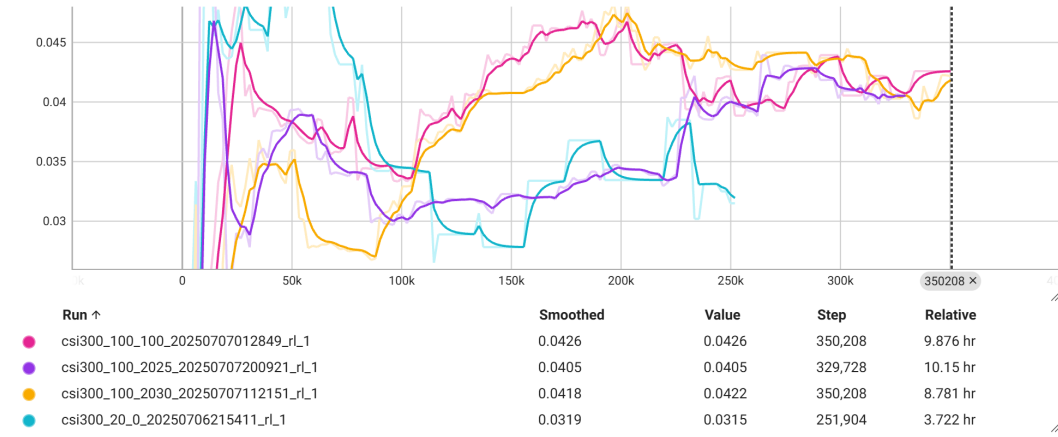


Figure 5: Mean Value of IC with Different Hyperparameters

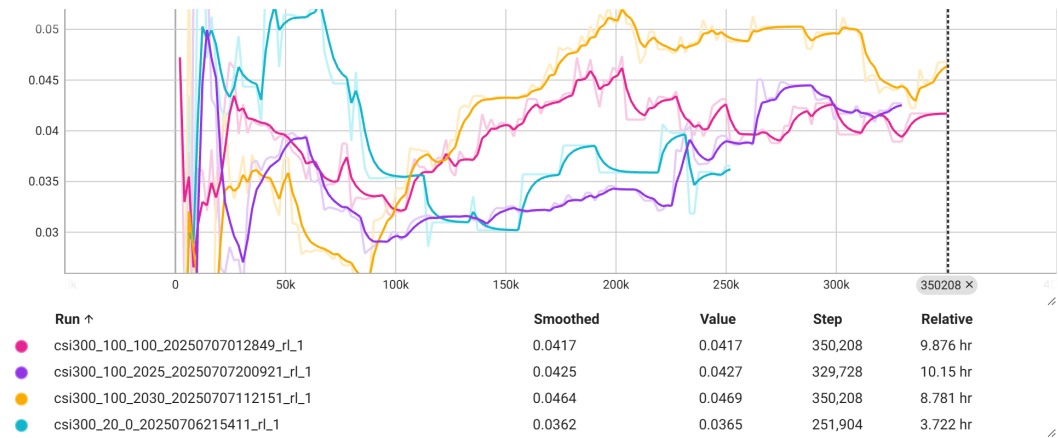


Figure 6: Mean Value of RIC with Different Hyperparameters

C Backtest Results

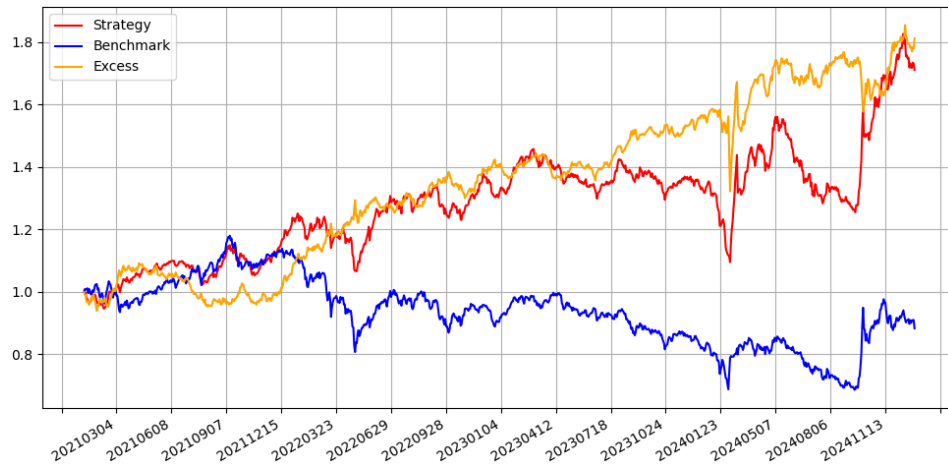


Figure 7: Backtest Result by LLM

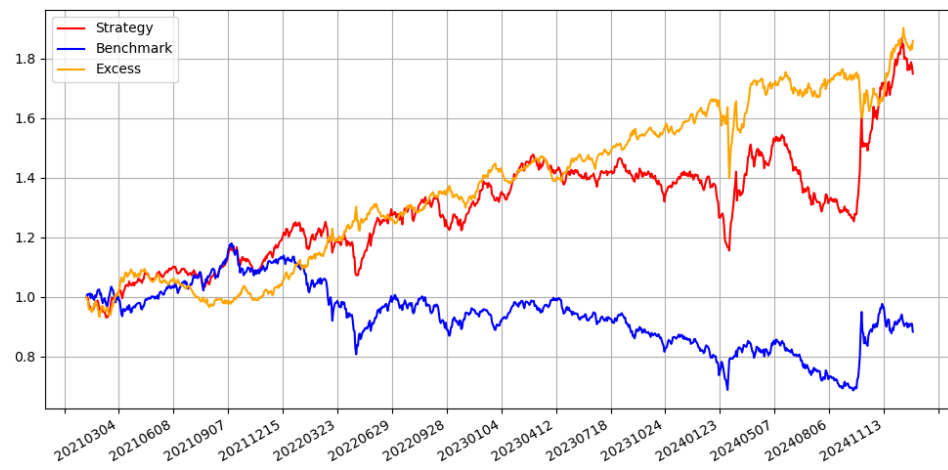


Figure 8: Backtest Result by Plain RL Methods

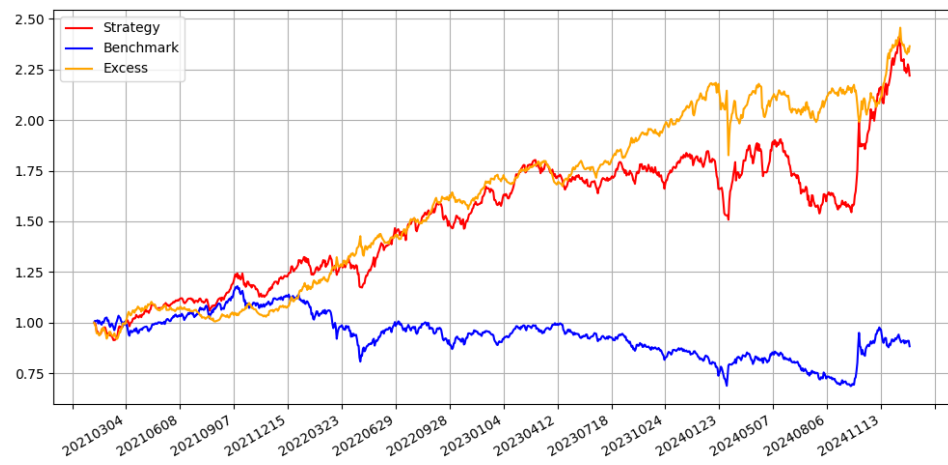


Figure 9: Backtest Result by Optimized RL methods (500 Factors)

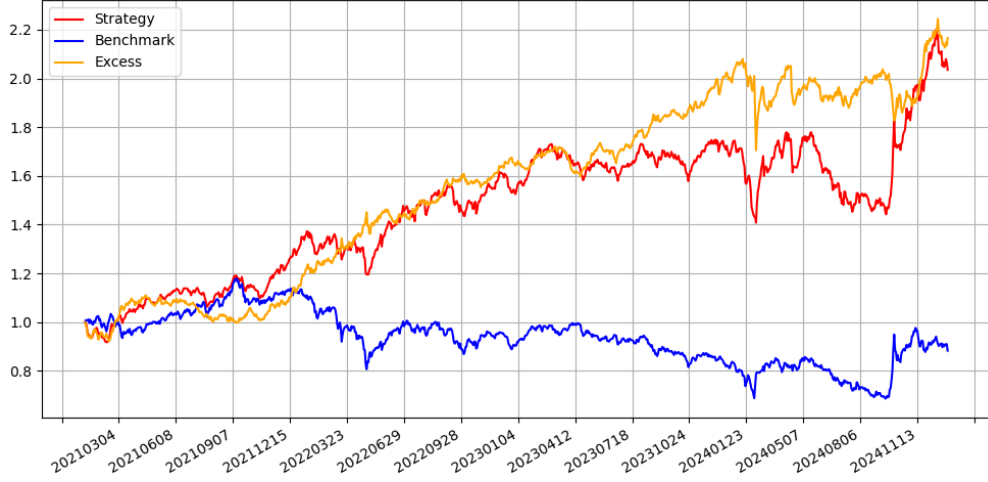


Figure 10: Backtest Result with PCA

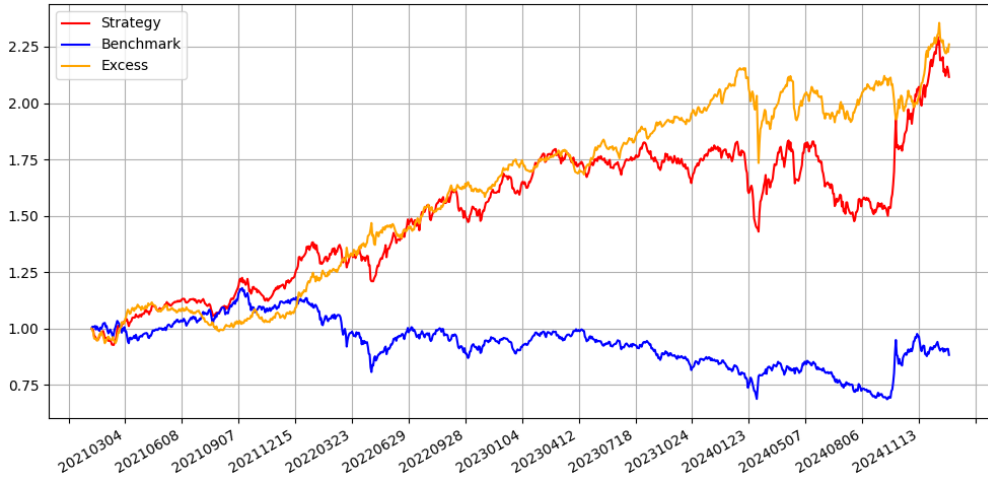


Figure 11: Backtest Result with Factor Filtering

D Generated Factor Library

Only part of the generated factor pool is exhibited here due to space limitations.

Table 4: Factors Generated by LLM

Factors
Corr(\$high,\$volume,20d)
Log(Div(EMA(\$close,10d),EMA(\$close,50d)))
Abs(Sub(EMA(\$open,30d),30.0))
Mul(Ref(\$close,10d),Ref(\$volume,15d))
Log(Div(Mean(\$high,10d),Mean(\$low,10d)))
Less(Mean(Ref(\$low,10d),40d),10.0)
Abs(Div(Mean(\$vwap,20d),Ref(\$vwap,5d)))
Div(EMA(\$close,20d),Mean(\$volume,10d))
Cov(Ref(\$high,10d),\$low,20d)
Div(Mad(\$volume,20d),Abs(EMA(\$close,10d)))

Table 4: Factors Generated by LLM

Factors
EMA(Min(\$vwap, 10d), 30d)
Less(Max(\$open, 10d), 30.0)
Abs(Div(Ref(\$high, 10d), Mean(\$low, 20d)))
Ensemble: IC = 0.0620, RIC = 0.0743

Table 5: Factors Generated by Optimized RL Methods

Factors
Div(-0.01, \$high)
Mul(Greater(5.0, \$amount), -0.01)
Sub(10.0, \$amount)
Greater(-0.5, \$amount)
Abs(Mul(5.0, Sub(\$volume, -10.0)))
Sub(0.5, \$amount)
Greater(-30.0, Max(Abs(\$volume), 1d))
Log(Mul(1.0, Mul(\$amount, \$pre_close)))
Add(5.0, Mul(\$amount, 2.0))
Greater(\$low, Mul(\$amount, 10.0))
Greater(\$amount, Rank(\$pre_close, 1d))
Div(\$amount, -1.0)
Div(Add(Log(Div(5.0, \$amount))), -1.0), 30.0)
Add(\$amount, -30.0)

E LLM Prompt Templates and Output

LLM model settings: gpt-4o-mini-2024-07-18.

Basic LLM prompt templates are shown below⁶:

Prompt and Output of LLM-based Response

```

1 [system] You are an expert quant researcher developing formulaic alphas.
2
3 # Specification
4 The formulaic alphas are expressed as mathematical expressions.
5 An expression can be a real constant between -30 and 30, an input feature, or an
  operator applied with its operands.
6 The input features available are: $open, $close, $high, $low, $volume, $vwap.
7 The operators, their descriptions, and their required operand types are listed in
  the table below. The operands x and y denote expressions, and t denotes a time
  span in days between "1d" and "50d".
8
9 Abs(x): absolute value
10 Log(x): logarithm
11 Add(x,y): add
12 Sub(x,y): subtract
13 Mul(x,y): multiply
14 Div(x,y): divide
15 Greater(x,y): larger one of two expressions
16 Less(x,y): smaller one of two expressions
17 Ref(x,t): the input expression at t days before
18 Mean(x,t): mean in the past t days
19 Sum(x,t): total sum in the past t days
20 Std(x,t): standard deviation in the past t days

```

⁶Several settings are from Yu et al. [2023].

```

21 Var(x,t): variance in the past t days
22 Max(x,t): maximum in the past t days
23 Min(x,t): minimum in the past t days
24 Med(x,t): median in the past t days
25 Mad(x,t): mean Absolute Deviation in the past t days
26 Delta(x,t): difference of the expression between today and t days before
27 WMA(x,t): weighted moving average in the past t days
28 EMA(x,t): exponential moving average in the past t days
29 Cov(x,y,t): covariance between two time-series in the past t days
30 Corr(x,y,t): correlation of two time-series in the past t days
31
32 Some examples of formulaic alphas:
33 - Abs(Sub(EMA(open,30d),30.))
34 - Max(WMA(open,10d),20d)
35 - Cov(Ref(volume,10d),open,50d)
36 - Greater(0.1,volume)
37
38 ## Limits
39 - You may not need to access any real-world stock data, since I will provide you
    with enough information to make a decision.
40 - You should give me alphas that are of medium length, not too long, nor too short.
41 - Do not use features or operators that are not listed above.
42
43 [user] Please generate 20 alphas that you think would be indicative of future stock
    price trend. Each alpha should be on its own line without numbering. Please do
    not output anything else.
44
45 [assistant] Abs(Sub(EMA(close,10d),EMA(open,10d)))
46 Corr(Delta(vwap,1d),volume,20d)
47 Greater(Mul(Std(high,15d),0.5),Div(vwap,Mean(volume,10d)))
48 ... More generated factors are omitted.
49 Less(Add(volume,Delta(close,5d)),Abs(Sub(close,low)))
50 Min(Abs(Sub(vwap,Ref(vwap,10d))),15d)
51
52 [script] Invalid expressions: ['Max(Mul(Med(high,10d),0.8),Sum(low,5d)) ']
53 [script] Ensemble on train: IC = 0.0442, Rank IC = 0.0587
54 [script] Ensemble on test #1: IC = 0.0699, Rank IC = 0.0586
55 [script] Ensemble on test #2: IC = 0.1135, Rank IC = 0.1235
56 [script] Ensemble on test #3: IC = -0.0179, Rank IC = 0.0166
57
58 [user] Here is a set of formulaic alphas generated by an automated system. These
    alphas are combined with a linear model into the final predictive signal. The
    alphas and the combined signal are tested on real-world dataset, and the IC/
    Rank IC metrics of them, together with the alphas' weights in the linear model
    is reported as follows:
59 Corr(Sub(close,Ref(close,5d)),volume,20d): IC = -0.0424, weight = -0.0384 This is a
    good alpha!
60 Greater(EMA(close,15d),Mean(vwap,10d)): IC = 0.0105, weight = 0.0060 This is a good
    alpha!
61 ...
62 Min(Abs(Sub(vwap,Ref(vwap,10d))),15d): IC = 0.0017, weight = -0.0001 This alpha
    doesn't contribute much.
63 WMA(Sub(close,open),20d): IC = -0.0108, weight = -0.0001 This alpha doesn't
    contribute much.
64 Med(Sub(close,open),15d): IC = -0.0032, weight = 0.0000 This alpha doesn't
    contribute much.
65 Ensemble: IC = 0.0442, Rank IC = 0.0587
66 According to the result, please generate ten more alphas, not similar to the
    insignificant ones. The most insignificant alphas will be replaced with the new
    ones to potentially boost the performance. Again, one on each line without
    numbering, and do not output anything else.
67
68 [assistant] Sub(Mean(high,20d),Med(low,15d))
69 Greater(Ref(vwap,10d),Abs(Sub(vwap,open)))
70 ... More generated factors are omitted.

```



```

71 Mul(Std(open,20d),Abs(Sub(close,Mean(vwap,10d))))
72
73 ...More iterations
74
75 [user] Here are a set of formulaic alphas generated by an automated system. These
      alphas are combined with a linear model into the final predictive signal. The
      alphas and the combined signal are tested on real-world dataset, and the IC/
      Rank IC metrics of them, together with the alphas' weights in the linear model
      is reported as follows:
76 Corr(Sub(close,Ref(close,5d)),volume,20d): IC = -0.0424, weight = -0.0352
77 Mean(Sub(EMA(open,30d),Med(close,20d)),25d): IC = -0.0217, weight = -0.0182
78 Div(Mad(Sub(high,low),25d),Std(vwap,12d)): IC = -0.0241, weight = -0.0164
79 ...
80 Corr(Sub(vwap,Mean(close,15d)),Delta(volume,5d),30d): IC = -0.0393, weight = -0.0058
81 Corr(Delta(open,5d),Mean(volume,20d),30d): IC = -0.0065, weight = -0.0033
82 Div(Mean(Mul(volume,Delta(close,5d)),15d),Mad(Sub(high,low),20d)): IC = -0.0071,
      weight = 0.0015
83 Ensemble: IC = 0.0718, Rank IC = 0.0776
84 According to the result, please generate three more alphas, not similar to the
      insignificant ones. The most insignificant alphas will be replaced with the new
      ones to potentially boost the performance. Again, one on each line without
      numbering, and do not output anything else.
85 [assistant] Mean(Sub(Ref(volume,10d),EMA(close,20d)),30d)
86 Corr(Add(Delta(vwap,15d),Sub(high,low)),Mean(open,25d),20d)
87 Div(Max(EMA(Sub(close,low),15d),20d),Mad(volume,30d))

```

F Proof of the Theorems

Lemma F.1. Suppose V, W are finite-dimensional inner product spaces and $T \in \mathcal{L}(V, W)$. Then there exists orthonormal lists e_1, \dots, e_m in V and f_1, \dots, f_m in W such that

$$Tv = s_1 \langle v, e_1 \rangle f_1 + \dots + s_m \langle v, e_m \rangle f_m$$

for every $v \in V$, where s_1, \dots, s_m are the positive singular values of T .

Proof. Let s_1, \dots, s_n denote the singular values of T , where $n = \dim V$. By the positivity of T^*T and the spectral theorem, there exists an orthonormal basis e_1, \dots, e_n of V such that

$$T^*T e_k = s_k^2 e_k$$

for each $1 \leq k \leq n$. For each $1 \leq k \leq m$, let

$$f_k = \frac{T e_k}{s_k}.$$

If $1 \leq j, k \leq m$, then

$$\langle f_j, f_k \rangle = \frac{1}{s_j s_k} \langle T e_j, T e_k \rangle = \frac{1}{s_j s_k} \langle e_j, T^* T e_k \rangle = \frac{s_k}{s_j} \langle e_j, e_k \rangle = \delta_{j,k},$$

where δ is the Kronecker delta. Thus f_1, \dots, f_m is an orthonormal list in W .

For $v \in V$,

$$Tv = T \left(\sum_{j=1}^n \langle v, e_j \rangle e_j \right) = \sum_{j=1}^n \langle v, e_j \rangle T e_j = \sum_{j=1}^m s_j \langle v, e_j \rangle f_j.$$

□

Corollary F.2. Suppose $A \in \mathbb{R}^{p,n}$ is nonzero. Then there exists a matrix $U \in \mathbb{R}^{p,m}$ with orthonormal columns, a diagonal matrix $\Sigma \in \mathbb{R}^{m,m}$ with positive diagonal entries, and a matrix $V \in \mathbb{R}^{n,m}$ with orthonormal columns such that

$$A = U \Sigma V^T.$$

Proof. Lemma F.1 holds *mutatis mutandis* for a matrix. □

Proof of Theorem 3.2. Suppose $A = [a_1 \ \dots \ a_m]^T$. Without loss of generality, suppose the sum of rows of A is 0. Let s_j 's, e_j 's and f_j 's be the same as in Lemma F.1 for matrix A . Note that $Aw = [a_1^T w \ \dots \ a_m^T w]$ has an average entry value of 0.

$$\sigma(Aw) \propto \|Aw\|^2 = \left\| \sum_{j=1}^{nT} s_j \langle w, e_j \rangle f_j \right\|^2 = \sum_{j=1}^{nT} s_j^2 \langle w, e_j \rangle^2,$$

where e_j 's and f_j 's are expanded properly for large j 's with $s_j = 0$.

Recall that singular values are in decreasing order. We prove the theorem by induction. The base case $k = 1$ is trivial: $w_1 = e_1$. Suppose the desired result holds for all integers less than or equal to k . For $k + 1$, the desired result follows from that

$$\langle w, e_j \rangle = 0, \forall j \leq k \quad \text{by induction hypothesis.}$$

Now the induction is closed and the proof is completed. \square

Lemma F.3. Suppose V, W are finite-dimensional inner product spaces, $T \in \mathcal{L}(V, W)$, and the positive singular values of T are s_1, \dots, s_m . Suppose e_1, \dots, e_m and f_1, \dots, f_m are orthonormal lists in V and W such that

$$Tv = s_1 \langle v, e_1 \rangle f_1 + \dots + s_m \langle v, e_m \rangle f_m$$

for every $v \in V$. Then

$$T^\dagger w = \frac{\langle w, f_1 \rangle}{s_1} e_1 + \dots + \frac{\langle w, f_m \rangle}{s_m} e_m$$

for every $w \in W$. Here T^\dagger is the Moore-Penrose inverse of linear map T , defined as

$$T^\dagger = (T|_{\ker T})^{-1} P_{\text{im } T},$$

where P is the orthogonal projection operator onto \cdot .

Proof. Let

$$v = \frac{\langle w, f_1 \rangle}{s_1} e_1 + \dots + \frac{\langle w, f_m \rangle}{s_m} e_m.$$

Then

$$\begin{aligned} Tv &= \frac{\langle w, f_1 \rangle}{s_1} T e_1 + \dots + \frac{\langle w, f_m \rangle}{s_m} T e_m \\ &= \langle w, f_1 \rangle f_1 + \dots + \langle w, f_m \rangle f_m \\ &= P_{\text{im } T} w. \end{aligned}$$

Observe that $v \in (\ker T)^\perp$. Then $v = T^\dagger w$ holds by definition. \square

Proof of Theorem 3.1. By Lemma F.3 (*mutatis mutandis*),

$$M^+ = V \Sigma^+ U^T,$$

where M^+ is the Moore-Penrose inverse of matrix M , and U, Σ, V are the same as in Corollary F.2 for matrix M . By definition of orthogonal projection and Moore-Penrose inverse, one solution to the problem in question is

$$w^* = M^+ Y = V \Sigma^+ U^T Y.$$

\square