# ALPHA$^2$: DISCOVERING LOGICAL FORMULAIC ALPHAS USING DEEP REINFORCEMENT LEARNING

Feng Xu[1,2*], Yan Yin[*], Xinyu Zhang[1,2], Tianyuan Liu[1,2],
Shengyi Jiang[3], and Zongzhang Zhang[1,2†]

[1]National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[2]School of Artificial Intelligence, Nanjing University, Nanjing, China
[3]The University of Hong Kong, Hong Kong, China
xufeng@lamda.nju.edu.cn, yy855@cornell.edu,
{zhangxinyu, liutianyuan}@lamda.nju.edu.cn, syjiang@cs.hku.hk,
zzzhang@nju.edu.cn

## ABSTRACT

Alphas are pivotal in providing signals for quantitative trading. The industry highly values the discovery of formulaic alphas for their interpretability and ease of analysis, compared with the expressive yet overfitting-prone black-box alphas. In this work, we focus on discovering formulaic alphas. Prior studies on automatically generating a collection of formulaic alphas were mostly based on genetic programming (GP), which is known to suffer from the problems of being sensitive to the initial population, converting to local optima, and slow computation speed. Recent efforts employing deep reinforcement learning (DRL) for alpha discovery have not fully addressed key practical considerations such as alpha correlations and validity, which are crucial for their effectiveness. In this work, we propose a novel framework for alpha discovery using DRL by formulating the alpha discovery process as program construction. Our agent, Alpha$^2$, assembles an alpha program optimized for an evaluation metric. A search algorithm guided by DRL navigates through the search space based on value estimates for potential alpha outcomes. The evaluation metric encourages both the performance and the diversity of alphas for a better final trading strategy. Our formulation of searching alphas also brings the advantage of pre-calculation dimensional analysis, ensuring the logical soundness of alphas, and pruning the vast search space to a large extent. Empirical experiments on real-world stock markets demonstrates Alpha$^2$'s capability to identify a diverse set of logical and effective alphas, which significantly improves the performance of the final trading strategy. The code of our method is available at https://github.com/x35f/alpha2.

## 1 INTRODUCTION

In quantitative investment, alphas play a pivotal role in providing trading signals. Serving as the foundation for strategic decision-making, alphas transform raw market data, such as opening and closing prices, into actionable signals such as return predictions. These signals inform traders' decisions and shape their strategies. Uncovering high-performance alphas that can withstand market fluctuations has long been a focal point in financial research.

Alphas are broadly categorized into two groups: formulaic alphas and black-box alphas. Formulaic alphas, in the form of operators and operands, are widely adopted because of their straightforward mathematical expressions and ease of analysis. They encapsulate market dynamics into succinct formulas. A textbook example is $\frac{close-open}{high-low}$, a mean-reversion alpha. Conversely, black-box alphas leverage advanced machine learning algorithms, such as deep learning and tree-based models

[*]Equal Contribution [†]Corresponding Author

(Ke et al., 2017; Chen & Guestrin, 2016), to directly transform a group of inputs to a numerical signal. Despite their high expressivity and handy end-to-end nature, they come with their own set of challenges. The lifetimes can be notably shorter, and training these models demands careful tuning of hyper-parameters. A widely held opinion is that formulaic alphas, given their simplicity and transparency, exhibit resilience to market fluctuations and are often more enduring than the machine-learning counterparts. Our work focuses on the discovery of formulaic alphas.

Traditionally, the discovery of formulaic alphas is often attributed to the intuition and insights of a trader, usually grounded in economic fundamentals. However, as modern computational capabilities advance, algorithmic techniques are also employed to find formulaic alphas (Yu et al., 2023; Cui et al., 2021; Zhang et al., 2020). These methods can identify alphas that satisfy specific criteria without the need for constant human oversight. Genetic Programming (GP) (Koza, 1994) based solutions, such as those detailed in (Zhang et al., 2020), have gained traction as popular tools for alpha discovery. These solutions maintain a population of expressions that undergo stochastic modifications, such as crossover and mutations. AlphaGen (Yu et al., 2023) pioneers the usage of Reinforcement Learning (RL) to discover alphas, which leverages an RL agent to sequentially produce alphas. While their results showcased the potential of RL in this domain, their adoption of the techniques can be improved. These existing works exhibit two weaknesses that hinder their practical use. First, they are not able to find formulaic alphas built from more primitive operators or deeper structures. This problem is even worse for GP-based methods because of their sensitivity to initial population distributions and high computational demands. Second, existing methods tend to use the performance of alpha as the only evaluation metric, producing alphas with high information correlation (IC) and low interpretability.

From the view of practical strategies for real-market data, alphas should satisfy two properties. First, as outlined in (Tulchinsky, 2019), diversity among alphas plays a important role in constructing robust trading strategies. This diversity helps mitigate the risk of overfitting, ensuring that strategies remain resilient when facing market volatility. Second, alphas should be logically sound according to certain rules, such as dimension consistency. For example, performing an addition between the open price and volume should be avoided, since they are not of the same dimension. GP-based methods directly modify the structure of expressions, while AlphaGen constructs an expression in the form of Reverse Polish Notation, token by token. Both methods can only perform dimensional analysis after an alpha is fully constructed. Being unable to prune the search space in advance, a lot of computational efforts are wasted.

One key challenge of alpha discovery lies in its large search space. To illustrate, consider a task involving 40 binary operators and 20 operands. For an alpha constituted of up to 15 operators, the search space swells to an overwhelming size of approximately $10^{63}$. Performing brute-force search on this space is impractical. In the AlphaGo class of algorithms (Mankowitz et al., 2023; Silver et al., 2017; 2016), RL-guided Monte Carlo Tree Search (MCTS) has demonstrated strong ability in finding solutions in large search spaces, such as Go, Chess, Shogi, Starcraft, and assembly programs.

To address the challenges observed in previously discussed frameworks and to discover alphas for practical use, we present a novel alpha discovery approach that combines RL with MCTS to generate alphas that are logical and less correlated. Drawing inspiration from AlphaDev (Mankowitz et al., 2023), we conceptualize an alpha as a program, akin to an assembly program, assembled incrementally. Such programs can be seamlessly translated into expression trees to be calculated. Meanwhile, such construction of an alpha can easily prune the search space in advance according to predefined rules. We then encapsulate this generation process within an environment. Subsequently, by leveraging a refined value estimation and policy guidance from DRL, we efficiently focus the search for diverse, robust, and high-performance alphas. Empirical studies validate the efficacy of our framework, confirming that alphas searched via our method surpass those discovered through traditional methods, in terms of its performance, correlation, and validity.

The primary contributions of our work are:

- We reconceptualize the task of generating formulaic alphas as a program generation process. The assembly of the alpha program enjoys the benefits of pruning the search space to a large extent.
- We present a novel search algorithm for formulaic alpha generation, utilizing the strength of DRL.
- Our experimental results validate the efficacy of our approach. We achieve a substantial reduction in search space and demonstrate the capability to discover logical, diverse, and effective alphas.

## 2 RELATED WORKS

**Symbolic Regression**: Symbolic Regression (SR) is a machine learning technique aiming to discover mathematical expressions to fit a dataset. The search for alphas can be seen as a way for SR to predict the market's return, which is highly correlated with our problem setting. However, the data in the financial market typically has a low signal-to-noise ratio, making accurate predictions from expressions of operators and operands impossible. Techniques like genetic programming, Monte Carlo Tree Search, and neural networks have been applied to symbolic regression. Mundhenk et al. (2021) introduce a hybrid neural-guided GP approach, utilizing the power of RL to seed the GP population. Sahoo et al. (2018) use a shallow neural network structured by symbolic operators to identify underlying equations and extrapolate to unseen domains. Kamienny et al. (2023) propose a MCTS-based method, using a context-aware neural mutation model to find expressions.

**Auto Generation of Formulaic Alphas**: Formulaic alphas provide interpretable trading signals based on mathematical expressions involving market features. Automated discovery of alphas has gained traction in quantitative trading. Genetic Programming has been widely applied to find trading strategies by evolving mathematical expressions. AutoAlpha (Zhang et al., 2020) uses Principal Component Analysis to navigate the search path from existing alphas. AlphaEvolve (Cui et al., 2021) utilizes AutoML techniques to a new class of alphas that are different from previous classes of formulas and machine learning models. Yu et al. (2023) first propose to use RL to generate formulaic alphas in the form of Reverse Polish Notation, and introduce a framework to automatically maintain the best group of alphas. Although the result of AlphaGen shows great improvement over previous methods, their framework of the RL task can be further improved. Due to the sparse nature of alpha search space, the Markov Decision Process defined in AlphaGen leads to highly volatile value estimations, and the policy generates similar alpha expressions.

## 3 PROBLEM FORMULATION

### 3.1 DEFINITION OF ALPHA

In this study, we focus on finding a day-frequency trading strategy in a stock market consisting of $n$ distinct stocks spanning $T$ trading days. For a stock dataset consists of $D$ trading days, on every trading day $d \in \{1, 2, ..., D\}$, each stock $i$ is represented by a feature vector $x_{d,i} \in \mathbb{R}^{m\tau}$. This vector encapsulates $m$ raw features, including open, close, high prices, etc, in the past $\tau$ days. $\tau$ is decided according to the expression of the alpha and availability of data. An alpha is a function $\zeta$ that transforms the features of a stock into a value $z_{d,i} = \zeta(x_{d,i}) \in \mathbb{R}$. These values of alphas are subsequently utilized in a combination model to form the trading signals. In the rest part of the paper, we omit the date index $i$ for brevity and operate on the $D$-day stock dataset.

### 3.2 EVALUATION METRICS FOR AN ALPHA

To assess an alpha's efficacy, the primary metric used is the Information Correlation (IC), which is computed as the average Pearson Correlation Coefficient between the alpha value $z$ and market return $\mu$ in $D$ days. It is mathematically expressed as:

$$\mathrm{IC}(z, \mu) = \frac{\sum_{d=1}^{D} \frac{\mathrm{Cov}(z_d, \mu_d)}{\sigma_{z_d} \sigma_{\mu_d}}}{D}, \tag{1}$$

where $\mathrm{Cov}$ computes the covariance and $\sigma$ computes the standard deviation.

Additional metrics used to further evaluate alphas include Rank IC, Max Draw Down (MDD), turnover (TVR), and Sharpe Ratio. While these metrics are not the primary targets of optimization within our framework, they hold substantial importance in practical trading scenarios and provide a comprehensive understanding of alpha performance, and our framework can be easily customized to these evaluation metrics.

| Category | Examples |
|---|---|
| Unary | Abs, Ln, Sign, ... |
| Binary | Add, Sub, Mul, TS-Mean, ... |
| Ternary | Correlation, Covariance, ... |
| Indicator | Start, End |

(a) Operators

| Category | Examples |
|---|---|
| Scalar | 0, 0.1, 0.5, 1, 3, 5, 15, ... |
| Matrix | open, close, high, low, vwap, ... |
| Register | Reg0, Reg1, ... |
| Placeholder | Null |

(b) Operands

Table 1: Operators and operands

| Operator | Operand1 | Operand2 | Operand3 | Register |
|---|---|---|---|---|
| Start | Null | Null | Null | |
| Sub | close | open | Null | Reg0 |
| Sub | high | low | Null | Reg1 |
| Div | Reg0 | Reg1 | Null | Reg0 |
| End | Null | Null | Null | |

Table 2: An example alpha program of $\frac{close-open}{high-low}$

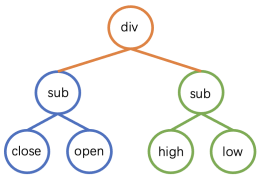


Figure 1: Expression tree

# 4 METHODOLOGY

## 4.1 ALPHA DISCOVERY AS PROGRAM GENERATION

Formulaic alphas are structured compositions of operators and operands. Drawing inspiration from AlphaDev's approach to algorithm generation, we reconceptualize the task of alpha discovery as constructing an "alpha program".

### 4.1.1 OPERATORS, OPERANDS AND INSTRUCTIONS

An alpha program is built from a series of instructions, where each instruction is characterized as a 4-element tuple $(Operator,\ Operand1,\ Operand2,\ Operand3)$. Operators are grouped into unary, binary, ternary, and indicator types based on the type and number of operands they engage. The indicator operators mark the start and end of an alpha program. Operand types include scalar values, matrix data, register storage, and a placeholder. Scalar operands are parameters for operators. Matrix operands are input features of the market, such as $open$ and $close$. Registers are used for storing intermediate results. The placeholder operand, $Null$, is used to align the instructions to a 4-element tuple. Examples of operators and operands are provided in Tab. 1.

### 4.1.2 TRANSLATING AN ALPHA PROGRAM INTO A COMPUTATION TREE

To actually compute an alpha program, we need to convert the program into formats that the computer can understand. A computational tree is built from alpha instructions in a bottom-up way. Tab. 2 and Fig. 1 provide an example of this transformation. Programs begin with the instruction tuple $(Start,\ Null,\ Null,\ Null)$. Then, instructions are translated one by one to build an expression tree. The color coding in the table corresponds to the colored nodes of the expression tree. Each colored node in the tree is the result of the execution of its matching colored instruction in the alpha program. The instructions marked by blue and green fill two registers. Note that the register assignment is implicit, if an instruction doesn't utilize registers, its output is stored in the first available register. Then, the instruction marked by orange performs a division between the values in the two registers. For an instruction employing a single register, the output replaces the current value. When an instruction involves two registers, the computed result replaces the Reg0 value while Reg1 is emptied. The $(End,\ Null,\ Null,\ Null)$ instruction marks the termination of an alpha program. Evaluating an alpha program involves reading values from the Reg0 register either during program construction or after its completion. This implicit approach to register management has been proven effective in our experiments.

#### 4.1.3 THE MDP FOR REINFORCEMENT LEARNING TASK

Given the established operators, operands, and instructions, we can construct a task suitable for RL. This task is defined as a Markov decision process (MDP), denoted as $(\mathcal{S}, \mathcal{A}, p, r, \gamma, \rho_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p(\cdot|s, a)$ is the transition probability, $r(s, a) \in [0, R_{\max}]$ is the reward function, $\gamma \in (0, 1)$ is the discount factor, and $\rho_0(s)$ is the initial state distribution.

In our alpha program environment, the state space, $\mathcal{S}$, contains all potential alpha programs. Each state $s$ corresponds to a unique alpha function $\zeta$. A state $s$ is a vectorized representation of the alpha function $\zeta$. The action space, $\mathcal{A}$, is the set of all possible instructions. The transition probability, $p(\cdot|s, a)$, is deterministic, which takes the values 1 for the next alpha program build after applying an action. The reward, denoted as $r(s_t, a_t, s_{t+1})$, is determined by the increase in the evaluation metric when arriving at $s_{t+1}$ after applying action $a_t$. The evaluation metric function is denoted as $\mathrm{Perf}(\zeta)$, which takes the alpha expression of the state as input. Since our transition is deterministic, the reward is computed as $r(s_t, a_t, s_{t+1}) = \mathrm{Perf}(\zeta_{t+1}) - \mathrm{Perf}(\zeta_t)$. The definition of the evaluation metric is primarily IC, but we have refined it, which we will detail later. The discount factor, $\gamma$, is a hyper-parameter to control the length of the alpha program. The initial state distribution, $\rho_0(s)$, invariably starts from an empty program, i.e., its value is 1 for an empty program, and 0 otherwise.

### 4.2 DISCOVERING ALPHAS USING RL

Alpha$^2$ uses a DRL agent to explore the alpha program generation task. The RL algorithm of Alpha$^2$ is similar to that of AlphaDev (Mankowitz et al., 2023), which is a modification of the AlphaZero agent (Silver et al., 2016). DRL guides a MCTS procedure using a deep neural network. The deep neural network takes the current state $s_t$, which is a vectorized representation of the alpha program $\zeta_t$, as input, and outputs action distributions and value predictions. The action distribution predicts the prior probability that an agent should take for each action, and the value predictions predict the cumulative reward that the agent should expect from the current state $s_t$. Since the Alpha series of works has detailed the MCTS process, we do not elaborate on it in this paper. The next paragraphs focus on key improvements that make the search algorithm better for discovering formulaic alphas.

### 4.3 DISCOVERING ROBUST, DIVERSE AND LOGICAL ALPHAS

#### 4.3.1 DISCOVERING ROBUST ALPHAS

Our approach to estimating the value of child nodes introduces a nuanced deviation from conventional methodologies. In traditional MCTS, the mean operator is often used to calculate the values of child nodes. Our empirical findings indicate that this operator falls short during the initial phases of the algorithm, a phenomenon we attribute to the inherent sparsity of formulaic alphas. In the early tree search stage, most alphas yield non-informative signals, leading to arbitrary policy directions. This scenario calls for the adoption of a max operator, which is more adept at navigating the sparse landscape of formulaic alphas. However, simply using the max operator can lead to the discovery of parameter-sensitive alphas, which is not desired. Supporting our observation, Dam et al. (2019) state that using the mean operator leads to an underestimation of the optimal value, slowing down the learning, while the maximum operator leads to overestimation. They propose a power mean operator that computes the value between the average value and the maximum one. In our work, we take a simpler form, and leave the balance between the maximum operator and the mean operator controlled by a hyperparameter. The value estimation for a child node is formulated as

$$Q(s, a) = r(s, a) + \beta \cdot \mathrm{mean}(V_s) + (1 - \beta) \max(V_s),$$

where $\beta \in [0, 1]$ is a hyperparameter controlling the balance between mean and max, $V_s$ is the value backup of a node on state $s$. Also, to further increase the validity of value estimation, especially for the mean operator, the value backup is calculated from the top-k values added to the node. That is, $V_s = \{v_1, ..., v_k\}$, where the $k$ values are stored in a min heap, so that the computation complexity is $O(\log k)$ for each new value. The RL agent, in the simulation phase, operates based on maximizing this $Q$-value. This refined definition of $Q$-value is expected to help discover alphas that are both effective and robust to parameters.

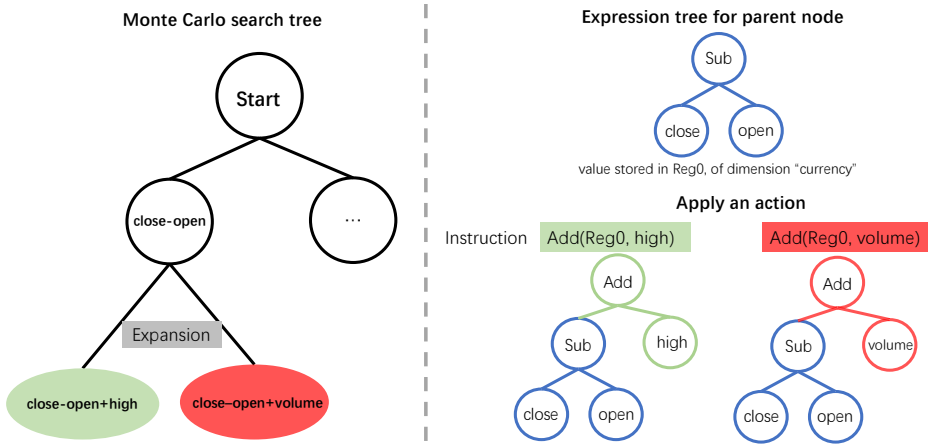


Figure 2: Example of pruning the search space according to dimensional consistency. The left part shows an ongoing search process of MCTS. The right part shows the expression trees of the corresponding alpha expressions. For an alpha expression $close - open$, which is of dimension $currency$, consider adding $high$ or $volume$ to the expression. Adding $high$ is allowed since it is of the same dimension $currency$, while adding $volume$ is forbidden because it is of another dimension. The nodes marked in green illustrates the addition of $high$, and the nodes marked in red illustrates the addition of $volume$.

#### 4.3.2 Discovering diverse alphas

As outlined in (Tulchinsky, 2019), diversity among alphas helps buliding robust trading strategies. In our framework, we incorporate such a target within the evaluation function. The diversity can be quantified by computing the correlation between alphas. For an alpha function $\zeta_t$ to be evaluated, we first compute its alpha value $z_t$ on all stocks and trading days. Then, for an already discovered set of alpha values $G = \{z^1, z^2, ..., z^n\}$, where $n$ is the number of alphas. We compute the max correlation with the current alpha value, i.e., $\text{MaxCorr}(z_t, G) = \max_i \text{IC}(z_t, z^i)$. The evaluation metric is discounted according to the max Pearson Correlation Coefficient between the alpha value and the mined alpha set:

$$\text{Perf}(\zeta_t) = (1 - \text{MaxCorr}(z_t, G)) \cdot \text{IC}(z_t, \mu),$$

This evaluation metric function encourages the discovery of low-correlation alphas by assigning higher value to alphas with low correlation with the mined alpha set, and discourages the discovery of highly correlated alphas by reducing the value of alphas with high correlation. In this way, Alpha$^2$ can continuously discover diverse alphas.

#### 4.3.3 Ensuring the Dimensional Consistency of Alphas

In real-world applications, especially in financial contexts, meaningful interactions between features are vital. Combining disparate features can lead to spurious relationships, making trading strategies derived from them unreliable. In the SR field, the dimension of individual input features is generally overlooked. For SR approaches in deep learning, normalization of features typically occurs during the pre-processing phase. Yet, for alpha discovery, where data is extended over both time and the count of assets, integrating normalization within the search process is preferable. This is from the fact that normalization inherently alters the data's semantics. While AlphaGen and GP-based methods have produced alphas with impressive statistical metrics, these often lack an underlying logical rationale. A defining quality of a coherent alpha expression is the dimensional consistency among its operators and operands. For instance, summing up variables like price and trade volume is fundamentally wrong due to their divergent distributions and different dimensions. Tab. 3 lists the dimensions of the basic input features. Note that the di-

| Feature | Dimension |
|---|---|
| open | currency |
| close | currency |
| high | currency |
| low | currency |
| vwap | currency |
| volume | unit |

Table 3: Dimension of features

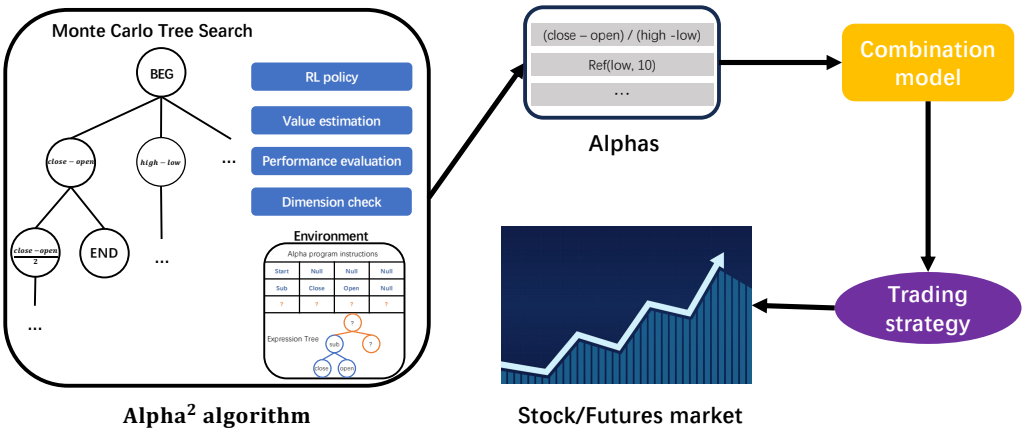


Figure 3: The pipeline for the generation of strategy

mension changes as the expression gets complicated. Our approach innovates by imposing rules that constrict the search space right from the node expansion stage, before actually evaluating the alpha, which is a feature not achievable in preceding methods. We maintain a record of an expression's dimension within each register, allowing for a preemptive filtration of nodes based on the dimensional requisites specified by the operators when expanding an MCTS tree. Our method to construct an alpha ensures that once a segment of the alpha's expression tree passes the dimensional check, it does not need to be reassessed. An example of the dimension system is illustrated in Fig. 2.

Traditional methods, including AlphaGen and GP-based paradigms, due to their structural limitations, are unable to prune the search space in advance. AlphaGen incrementally constructs alphas, token by token. Coupled with its use of Reverse Polish Notation for expressions, pinpointing a token's exact location within the final expression tree is ambiguous, which does not support the search space pruning. GP-based methods, with their mutation operations, remain unaware of the overall structure of expression trees, and can even perform cyclic modifications. Thus, performing dimension check is not achievable before the alpha is generated. Employing these dimension restrictions results in a large reduction in nodes at every level compared to the unrestrained counterpart. By reducing the complexity and potential combinations in the search space, we can focus on discovering alphas that are logical. This can also reduce the chances of overfitting, which is a significant concern in quantitative finance.

### 4.4 Pipeline to generate a trading strategy

Our method focuses on generating alphas and does not provide an end-to-end solution. Alpha$^2$ first produces alphas using RL-guided MCTS with the refined value estimation, performance evaluation, and dimension check. Then, a combination model takes the alphas as input and generates a trading strategy. The combination model can be customized to meet user demand, e.g., linear regression, deep neural networks, and gradient boosting trees. Fig. 3 shows the pipeline for a practical adoption of our method.

## 5 Experiments

In the experiment section, we aim to demonstrate the efficacy of our method compared to existing approaches. Our primary focus is to answer the three questions:

- Can Alpha$^2$ generate diverse and good alphas?
- Can alphas mined by Alpha$^2$ perform better than previous methods?
- How do alphas mined by Alpha$^2$ perform in the real-world market?

### 5.1 Experiment Setup

**Data:**. The data is acquired from the Chinese A-shares market through baostock[1]. Six raw features are selected to generate the alphas: {open, close, high, low, volume, vwap}. The target of our

[1]https://www.baostock.com

method is to find alphas that have high IC with the 20-day return of the stocks. The dataset is split into a training set (2009/01/01 to 2018/12/31), a validation set (2019/01/01 to 2020/12/31), and a test set (2021/01/01 to 2023/12/31). We use the constituents of the CSI300 and CSI500 indices of China A-shares as the stock set.

**Baselines:** Our method is compared to several machine learning models. **MLP** uses a fully connected neural network to process the input data to strategy signals. **XGBoost** and **LightGBM** are gradient boosting frameworks. **AlphaGen** and **gplearn** [2] are representative methods for generating a collection of alphas. We follow the open source implementations of AlphaGen [3] and Qlib [4] (Yang et al., 2020) to produce the results.

**Alpha Combination:** Our method, Alpha$^2$, only considers the problem of generating alphas. We use the XGBoost as the combination model. The XGBoost model is trained to fit the alpha signals to the return signals on the training dataset, using the top 20 generated alphas ranked by IC. Then, the trained model is fixed and used to predict the test dataset.

**Evaluation Metric:** Two metrics, IC and Rank IC, are used to measure the performance of the models. The definition of IC is given in Eq. 1. The rank information coefficient (Rank IC) measures the correlation between the ranks of alpha values and the ranks of future returns. It is defined as the Spearman's Correlation Coefficient between the ranks of the alpha values and the future return, $\rho(z_d, r_d) = \mathrm{IC}(\mathrm{rk}(z_d), \mathrm{rk}(r_d))$, where $\mathrm{rk}(\cdot)$ is the ranking operator.

**Code:** For an efficient and accelerated experimental process, our implementation is based on the pseudo-code that AlphaDev provides [5], with computational aspects handled by Jax. Experiments are run on a single machine with an Intel Core 13900K CPU and 2 Nvidia A5000 GPUs.

### 5.2 IC and Correlation of Generated Alphas

| Method | IC | Correlation |
|---|---|---|
| gplearn | 0.0164±0.0167 | 0.7029±0.1824 |
| AlphaGen | 0.0257±0.0153 | 0.3762±0.6755 |
| **Ours** | **0.0407±0.0219** | **0.1376±0.3660** |

Table 4: Statistics of IC and correlations of mined alphas on CSI300.

For a robust strategy, the alphas are expected to be diverse, having low Pearson Correlation Coefficients with each other. To answer the first question, we compute the correlations between alphas generated by gplearn, AlphaGen, and Alpha$^2$. The result is shown in Tab. 4.

From the table, we can see that Alpha$^2$ generates the best set of alphas in terms of IC. Meanwhile, it generates the most diverse set of alphas, as measured by the mean correlation. It is worth noting that gplearn generates a set of alphas with high correlations. The high correlation results from the minor mutation of constants in the alpha expressions after it gets trapped in a local optima. With the more diverse and better alpha set, Alpha$^2$ has greater potential to generate a more robust trading strategy.

### 5.3 Performance of Generated Alphas

To answer the second question, we run the baselines and our method on the CSI300 and CSI500 stock datasets and evaluate them on the two metrics. The results are shown in Tab. 5.

The first three methods, MLP, XGBoost, and LightGBM, combine an existing set of alphas from Qlib. They have a worse performance due to the usage of the open-source set of alphas. On the other hand, gplearn and AlphaGen are based on formulaic alphas generated by themselves. Alphas generated by gplearn and AlphaGen perform better on the test dataset. Although AlphaGen has designed a framework to filter alphas, it neither ensures the validity of alphas upon generation nor

[2] https://github.com/trevorstephens/gplearn

[3] https://github.com/RL-MLDM/AlphaGen

[4] https://github.com/microsoft/qlib

[5] https://github.com/google-deepmind/alphadev

| Method | CSI300 | | CSI500 | |
|---|---|---|---|---|
| | IC | Rank IC | IC | Rank IC |
| MLP | 0.0123±0.0006 | 0.0178±0.0017 | 0.0158±0.0014 | 0.0211±0.0007 |
| XGBoost | 0.0192±0.0021 | 0.0241±0.0027 | 0.0173±0.0017 | 0.0217±0.0022 |
| LightGBM | 0.0158±0.0012 | 0.0235±0.0030 | 0.0112±0.0012 | 0.0212±0.0020 |
| gplearn | 0.0445±0.0044 | 0.0673± 0.0058 | 0.0557±0.0117 | 0.0665±0.0154 |
| AlphaGen | 0.0500±0.0021 | 0.0540±0.0035 | 0.0544±0.0011 | 0.0722±0.0017 |
| **Ours** | **0.0576±0.0022** | **0.0681±0.0041** | **0.0612±0.0051** | **0.0731±0.0093** |

Table 5: Performance on CSI300 and CSI500 in the test dataset.

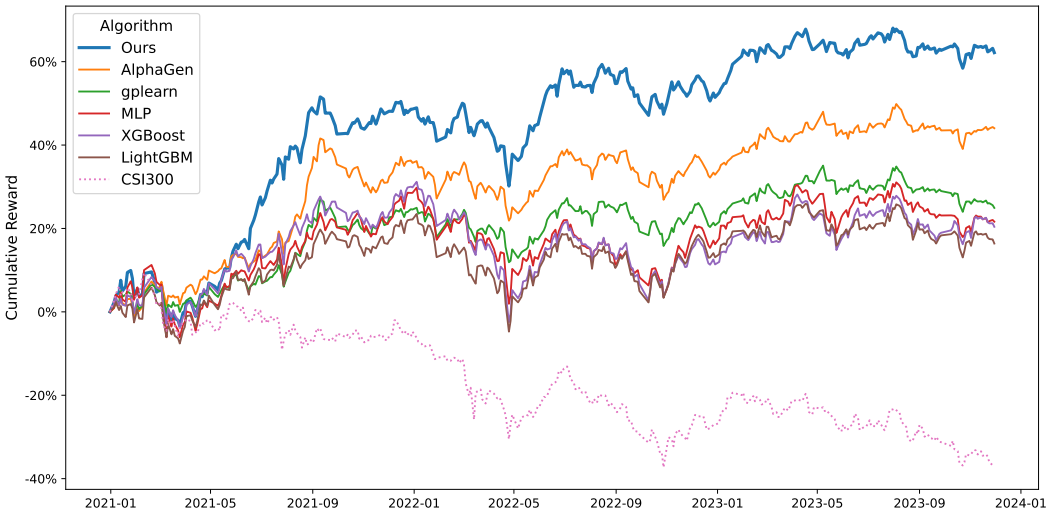


Figure 4: Backtest result on CSI300. The value of the y axis represents the cumulative reward.

emphasizes diversity, which leads to possible performance degradation. We attribute the superior performance of Alpha$^2$ to the logical soundness and diversity of alphas.

### 5.4 Stock Market Backtest

To further validate the effectiveness of our method, we conduct an experiment on a simulated environment. The data is from the Chinese A-shares market in the test period. The trading strategy is a top-k/drop-n strategy. On each trading day, stocks are first sorted according to the alpha values, then the top-$k$ stocks are selected to trade. With at most $n$ stocks traded every day, we try to invest evenly across the $k$ stocks. In out experiment, $k = 50$ and $n = 5$. The result of the backtest is shown in Fig. 4. Our method demonstrates superior performance on the CSI300 stock market.

## 6 Conclusion

In this work, we introduce Alpha$^2$, a novel framework for the discovery of formulaic alphas. Using RL and MCTS, we harness the power of modern machine learning techniques to address the challenges of discovering powerful formulaic alphas in the vast search space. Alpha$^2$ formulates the alpha generation process as a program construction task, using RL-guided MCTS as the search algorithm. Our refined value estimation, performance evaluation and dimension check ensures the discovery of high-quality alphas and a good and robust trading strategy, which are validated in the experiments. From the perspective of search algorithms, vast theoretical research has been done on MCTS and RL. Our method benefits from the existing research. The search algorithm can minimize the regret to the ground truth within theoretical bounds, compared to the mostly empirical results of previous methods. On the engineering side, we propose a novel framework to generate formulaic alphas. This framework allows a general design of search space pruning for formulaic alphas, including but not limited to dimensional consistency rules.

## REFERENCES


Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

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*, pp. 2208–2216, 2021.

Tuan Dam, Pascal Klink, Carlo D'Eramo, Jan Peters, and Joni Pajarinen. Generalized mean estimation in Monte-Carlo tree search. *arXiv preprint arXiv:1911.00384*, 2019.

Pierre-Alexandre Kamienny, Guillaume Lample, Sylvain Lamprier, and Marco Virgolin. Deep generative symbolic regression with Monte-Carlo-tree-search. In *International Conference on Machine Learning*, pp. 15655–15668, 2023.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pp. 3146–3154, 2017.

John R Koza. Genetic Programming as a means for programming computers by natural selection. *Statistics and Computing*, 4:87–112, 1994.

Daniel J. Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, Thomas Köppe, Kevin Millikin, Stephen Gaffney, Sophie Elster, Jackson Broshear, Chris Gamble, Kieran Milan, Robert Tung, Minjae Hwang, A. Taylan Cemgil, Mohammadamin Barekatain, Yujia Li, Amol Mandhane, Thomas Hubert, Julian Schrittwieser, Demis Hassabis, Pushmeet Kohli, Martin A. Riedmiller, Oriol Vinyals, and David Silver. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023.

T Nathan Mundhenk, Mikel Landajuela, Ruben Glatt, Claudio P Santiago, Daniel M Faissol, and Brenden K Petersen. Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053*, 2021.

Subham Sahoo, Christoph Lampert, and Georg Martius. Learning equations for extrapolation and control. In *International Conference on Machine Learning*, pp. 4442–4450, 2018.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Igor Tulchinsky. Finding alphas: A quantitative approach to building trading strategies. John Wiley & Sons, 2019.

Xiao Yang, Weiqing Liu, Dong Zhou, Jiang Bian, and Tie-Yan Liu. Qlib: An AI-oriented quantitative investment platform. *arXiv preprint arXiv:2009.11189*, 2020.

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*, pp. 5476–5486, 2023.

Tianping Zhang, Yuanqi Li, Yifei Jin, and Jian Li. AutoAlpha: An efficient hierarchical evolutionary algorithm for mining alpha factors in quantitative investment. *arXiv preprint arXiv:2002.08245*, 2020.