AlphaEvolve: A Learning Framework to Discover Novel Alphas in Quantitative Investment

Can Cui
School of Computing
National University of Singapore
cuican@comp.nus.edu.sg

Wei Wang
School of Computing
National University of Singapore
wangwei@comp.nus.edu.sg

Meihui Zhang
School of Computer Science and Technology
Beijing Institute of Technology
meihui_zhang@bit.edu.cn

Gang Chen
College of Computer Science and Technology
Zhejiang University
cg@zju.edu.cn

Zhaojing Luo
School of Computing
National University of Singapore
zhaojing@comp.nus.edu.sg

Beng Chin Ooi
School of Computing
National University of Singapore
ooibc@comp.nus.edu.sg

ABSTRACT

Alphas are stock prediction models capturing trading signals in a stock market. A set of effective alphas can generate weakly correlated high returns to diversify the risk. Existing alphas can be categorized into two classes: Formulaic alphas are simple algebraic expressions of scalar features, and thus can generalize well and be mined into a weakly correlated set. Machine learning alphas are data-driven models over vector and matrix features. They are more predictive than formulaic alphas, but are too complex to mine into a weakly correlated set. In this paper, we introduce a new class of alphas to model scalar, vector, and matrix features which possess the strengths of these two existing classes. The new alphas predict returns with high accuracy and can be mined into a weakly correlated set. In addition, we propose a novel alpha mining framework based on AutoML, called AlphaEvolve, to generate the new alphas. To this end, we first propose operators for generating the new alphas and selectively injecting relational domain knowledge to model the relations between stocks. We then accelerate the alpha mining by proposing a pruning technique for redundant alphas. Experiments show that AlphaEvolve can evolve initial alphas into the new alphas with high returns and weak correlations.

CCS CONCEPTS

• **Applied computing** → **Computers in other domains**; • **Information systems** → *Data mining*; • **Computing methodologies** → Symbolic and algebraic manipulation.

KEYWORDS

stock prediction; search algorithm

1 INTRODUCTION

Alphas are stock prediction models generating trading signals (i.e., triggers to buy or sell stocks). Mining alphas with high returns[1] has been an active research topic in the field of data mining [6, 11, 24, 25]. However, high returns typically come with high risks. To buffer risk, experts from hedge funds[2] aim at identifying a set of weakly correlated[3] formulaic alphas with high returns. The identifying process is illustrated in Figure 1. Alphas are first designed based on experts' different perspectives and then backtested on the markets to ensure weakly correlated returns. Such alphas, as algebraic expressions, have fine properties of simplicity and good generalizability [5].

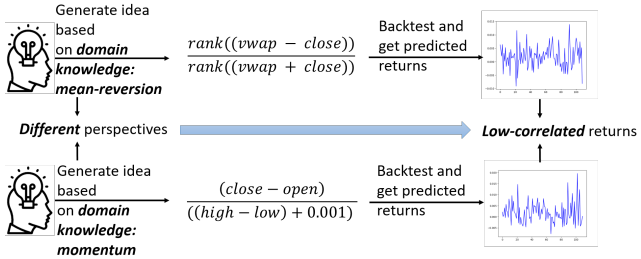


Figure 1: The generating process of formulaic alphas.

Existing AI approaches surpass the experts by designing complex machine learning alphas [3, 8, 10] or using the genetic algorithm to automatically mine a set of formulaic alphas [14, 15]. In the first approach, machine learning alphas are machine learning models that generate trading signals. They can model high-dimensional features and learn from training data to boost generalization performance. However, they are complex in their structures, and thus difficult to mine into a set of alphas with weakly correlated returns and to maintain this set [17]. Further, machine learning alphas designed with domain knowledge are based on strong structural assumptions. To be specific, the injection of relational domain knowledge assumes the returns of the stocks from the same sector change similarly [10]. However, this assumption is less likely to hold for a volatile stock market. In the second approach, the genetic algorithm has two limitations: First, the search space of the algorithm is small because only arithmetic operations are considered. In such a small search space, the algorithm can hardly evolve the initial

[1] In this paper, the return is defined as Today's Stock Return = (Today's Stock Price - Yesterday's Stock Price) / (Yesterday's Stock Price)

[2] Hedge funds are institutional investors and among the most critical players in the stock markets [2].

[3] In this paper, we adopt the standard for weak correlation in hedge funds: the sample Pearson correlation of 15% between portfolio returns of different alphas [13].

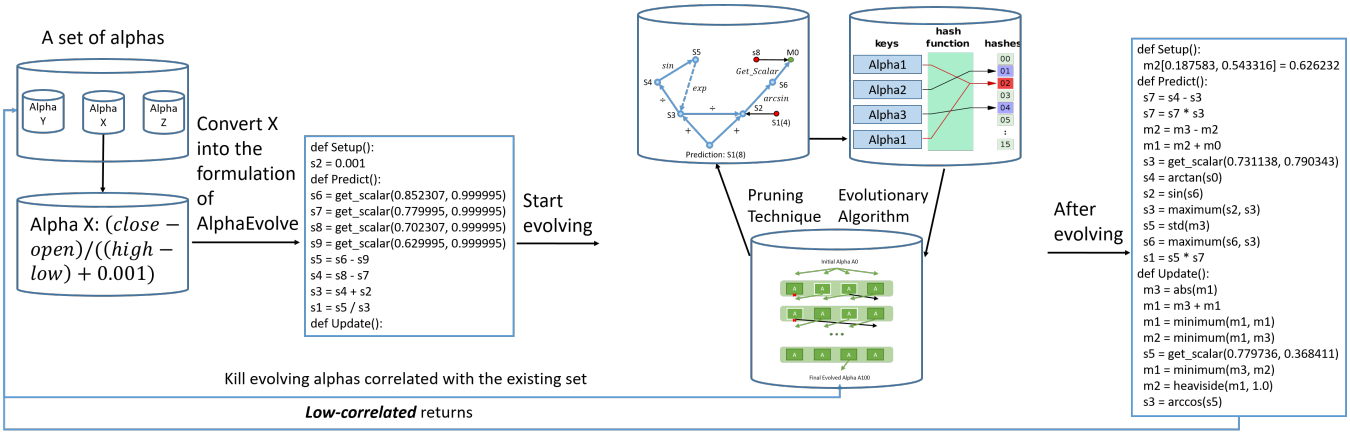


Figure 2: AlphaEvolve architecture overview.

alphas to improve the performance; Second, the algorithm searches formulaic alphas, which only utilize short-term features (e.g., the moving average of last 30 days' close prices).

A framework called AutoML-Zero [21] was recently proposed to discover a neural network from scratch and can thus be considered for alpha mining. Such consideration is advantageous over the genetic algorithm: AutoML-Zero expands the search space by allowing more operation types, i.e., vector or matrix operations, for feature capturing. Consequently, a domain-expert-designed alpha can be further improved. However, merely applying AutoML-zero to alpha mining suffers from the drawbacks of machine learning alphas, i.e., complex alpha structures and ineffective use of relational domain knowledge, leading to ineffective and inefficient alpha mining. Specifically, machine learning alphas typically contain complex components that are difficult to discover from scratch, e.g., graph neural network [10], attention mechanism and LSTM neural networks [3]. Even discovering a two-layer neural network requires intensive computing resources and long searching time (i.e., 100 to 1000 processes over a week) to evaluate 10^{12} candidate alphas [21]. Also, AutoML-Zero considers stocks as independent tasks and thus cannot leverage relational domain knowledge, which provides rich information for modeling the relations between stocks.

To address these issues, we first introduce a new class of alphas, which can be effectively mined into a weakly correlated set of alphas. Next, we propose a novel alpha mining framework, AlphaEvolve, for generating the new alphas. An overview of AlphaEvolve is illustrated in Figure 2. A well-designed formulaic alpha is first transformed into the formulation of AlphaEvolve. Then an evolutionary algorithm is initialized with this alpha. During the evolutionary process, a population of candidate alphas is iteratively updated for generating better alphas. Besides, candidate alphas are eliminated if they are correlated with a given set of alphas. Finally, an alpha is generated with weakly correlated high returns. To this end, we first introduce operators extracting scalar inputs from the input matrix. These operators facilitate searching for the new alphas and avoid discovering a complex machine learning alpha from scratch, because the new alphas augmented with extracted inputs are more predictive and thus more likely to be selected in the evolutionary process than complex machine learning alphas, which are less predictive. Next, we further introduce operators to selectively inject sector-industry relations of all stocks into an alpha. Lastly, we propose a pruning technique to boost search efficiency. This technique helps to avoid redundant calculations in alpha search with a large number of tasks.

In this paper, we make the following contributions:

- We first introduce a new class of alphas with intriguing strengths: like formulaic alphas, these alphas can model scalar features and thus are simple to mine into a weakly correlated set, but, like machine learning alphas, they are high-dimensional data-driven models utilizing long-term features. We then propose a novel alpha mining framework, AlphaEvolve, to generate the new alphas. To the best of our knowledge, we are the first to solve the stock prediction problem based on AutoML and the first to tackle the problem of mining weakly correlated alphas.
- We enable AlphaEvolve to selectively inject relational domain knowledge without any strong structural assumption in an alpha.
- We propose an optimization technique to accelerate alpha mining by pruning redundant alphas.
- We conduct extensive experimental study on AlphaEvolve using the stock price data of NASDAQ. The results show that AlphaEvolve generates alphas with weakly correlated high returns.

The remainder of the paper is structured as follows. Section 2 defines the problem formulation. Section 3 introduces the evolutionary algorithm AlphaEvolve is based on. Section 4 elaborates on the optimization of AlphaEvolve. Section 5 analyses the experiment results. Section 6 reviews the background of the alpha mining problem. Section 7 concludes this paper.

2 PROBLEM FORMULATION

We adopt the alpha definition used by traders in hedge funds [13]. An alpha is a combination of mathematical expressions, computer source code, and configuration parameters that can be used, in

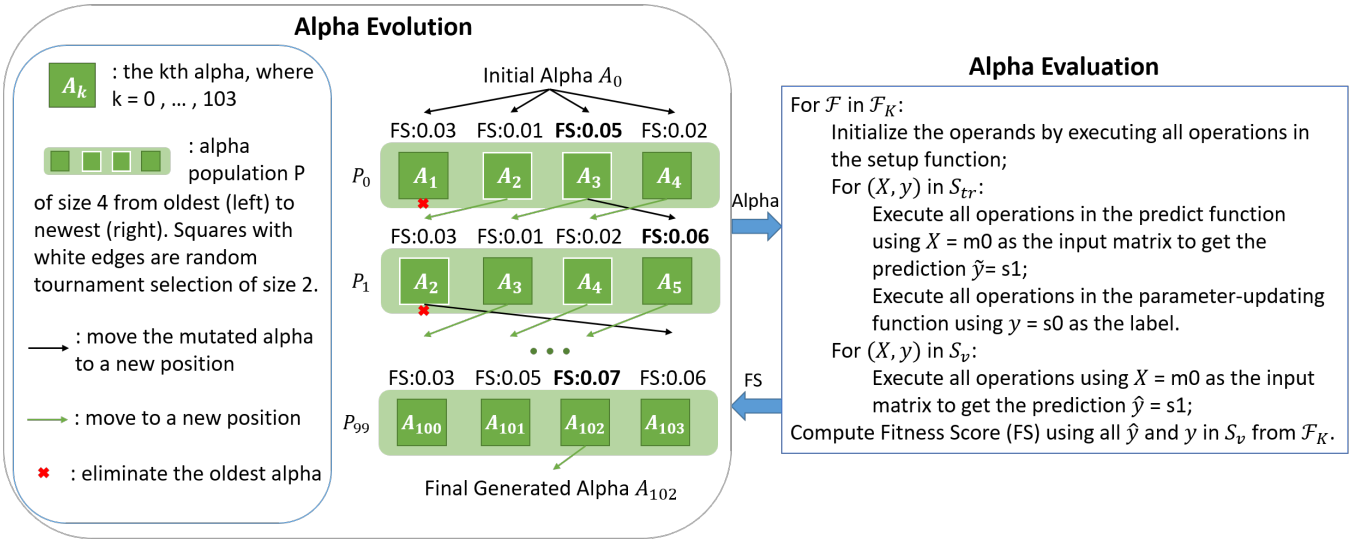


Figure 3: The evolution and evaluation processes in alpha mining.

combination with historical data, to make predictions about future movements of stocks [22]. Under this definition, an alpha is formulated in AlphaEvolve as a sequence of operations, each of which comprises an operator (i.e., OP), input operand(s), and an output operand. For each operand, we use s, v, m to denote a scalar, a vector, and a matrix respectively. Let us refer back to Figure 2. The fifth operation $s5 = s6 - s9$ in the Predict() component of the alpha X before evolving has $s5$ as the output operand, $s6$ and $s9$ as the input operands, and $-$ as the OP. Specifically, the operand $s5$, $s6$, and $s9$ are the sixth, seventh, and tenth scalar operand respectively. Special operands are predefined, e.g., the input feature matrix $m0$, the output label $s0$, and the prediction $s1$. Note that an operand can be overwritten and therefore only the last $s1$ in the predict function is the final prediction. The allowable OPs are composed of our proposed OPs and basic mathematical operators for scalars, vectors, and matrices [21].

Each alpha consists of three components: a setup function to initialize operands, a predict function to generate a prediction, and a parameter-updating function to update parameters if any parameters are generated during the evolution. These three components correspond to def Setup(), def Predict(), and def Update() respectively in the examples in Figure 2. In the prediction function, the new class of alphas improves the formulaic alphas by extracting features from vectors and matrices. Additionally, including scalars into operations makes the new alphas less complex than machine learning alphas, which only allow operations of high-dimensional vectors or matrices. In the parameter-updating function, we define the operands that are updated in the training stage (i.e., features) and passed to the inference stage as parameters. Unlike intermediate calculation results which are only useful for a specific prediction, these operands, as features of long-term training data, improve the alpha's inference ability. The new alphas with these parameters are advantageous over formulaic alphas which cannot utilize long-term historical data.

We aim to search for the best alpha from all possible alphas constrained by the maximum allowable number of operations for each component, the allowable OPs for each component, and the maximum allowable number of operands for scalars, vectors, and matrices.

An alpha is evaluated over a set of tasks $\mathcal{F}_K$, where K is the number of tasks. Each task is a regression task for a stock, mapping an input feature matrix $\mathcal{X} \in \mathbb{R}^{f \times w}$ to a scalar label of return y, where f is the number of feature types and w is the input time window in days. The pair of $\mathcal{X}$ and y defines a sample. All samples S are split into a training set S_{tr}, a validation set S_v, and a test set S_{te}.

3 EVOLUTIONARY ALGORITHM FOR ALPHA MINING

AlphaEvolve is based on the evolutionary algorithm, which is an iterative selection process for the best alpha under a time budget. The process is illustrated in Figure 3:

(1) In the first iteration, AlphaEvolve is initialized by a starting parent alpha, e.g., A_0 in Figure 3. A population P_0 is generated by mutating the parent alpha. Two types of mutations are performed on the parent alpha to generate a child alpha: (1) randomizing operands or OP(s) in all operations; (2) inserting a random operation or removing an operation at a random location of the alpha.

(2) Each alpha of P_0 is evaluated on the tasks $\mathcal{F}_K$. The evaluation process outputs a fitness score as shown on the right side of Figure 3. We use the Information Coefficient (IC) as the fitness score for alpha i (Eq. 1), where $\hat{\mathbf{y}}_t^{(i)} = \left(\hat{y}_{t,1}^{(i)}, \ldots, \hat{y}_{t,K}^{(i)}\right)$ is the vector of predictions at date t, $\mathbf{y}_t$ is the vector of corresponding labels, corr is the sample Pearson correlation, N is the number of samples in S_v.

$$IC_i = \frac{1}{N} \sum_{t=1}^{N} corr\left(\hat{\mathbf{y}}_t^{(i)}, \mathbf{y}_t\right) \tag{1}$$

(3) The alpha with the highest fitness score in a randomly selected set of fixed size, called the tournament, is selected as the new parent alpha. For example, the alpha with the highest fitness score in the tournament of P_0, A_3, is selected as the parent alpha.

(4) In the subsequent iterations, a new population is generated by adding the mutated parent alpha into the previous population and eliminating the oldest alpha. For example, P_1 is generated by adding A_5 mutated from A_3 and eliminating A_1 from P_0 in Figure 3.

(5) If the training budget is exhausted, the alpha with the highest fitness score in the population is selected as the evolved alpha.

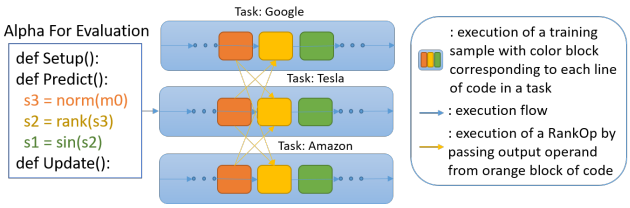


Figure 4: An example of the execution of the RelationOp.

4 OPTIMIZATION

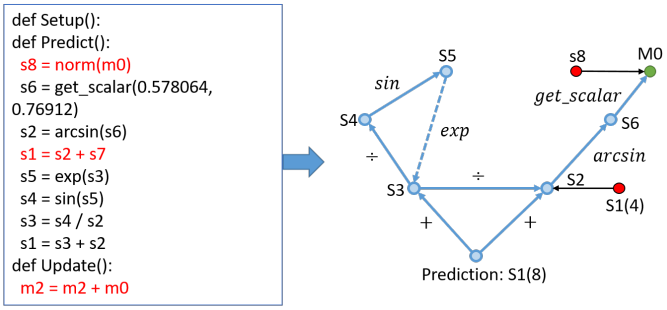
This section introduces the proposed OPs and the pruning technique for redundant alphas to optimize the alpha search.

4.1 RelationOp and ExtractionOp

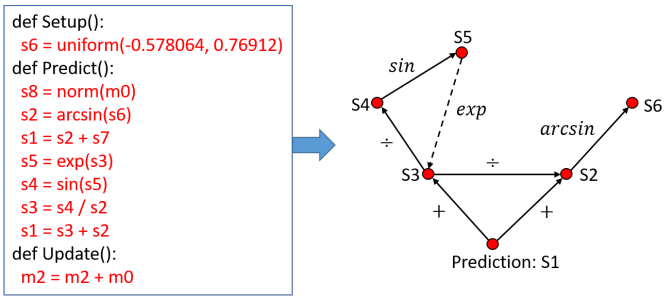
Unlike conventional AutoML frameworks where tasks are mutually independent [18, 21], our tasks $\mathcal{F}_K$ are related because the stocks predicted in $\mathcal{F}_K$ are classified into sectors and industries in a stock market. Modeling such relations in a stock market effectively improves the prediction accuracy [10]. We design a set of OPs, called RelationOps, to model such relations: they calculate an output operand based on input scalar operands calculated in the current task and other related tasks in the same sector (industry).

For the execution of an operation with the RelationOp on a sample $s^{(a)} \in S$ where $a \in \mathcal{F}_K$, the input operands are the output operands calculated on the samples $\mathbf{s}^{\mathcal{F}} \subset S$ where $\mathcal{F}$ is a set of related tasks. For the example shown in Figure 4, the execution of $s2 = rank(s3)$ (in yellow) has its inputs calculated as $s3 = norm(m0)$ (in orange) on the samples of the same time step from all related tasks, where *norm* calculates the Frobenius norm of a matrix. The output operand and $\mathcal{F}$ are determined by the types of RelationOps: (1) RankOp outputs the ranking of the input operand calculated on $s^{(a)}$ among those calculated on $\mathbf{s}^{\mathcal{F}_K}$; (2) RelationRankOp outputs the ranking of the input operand calculated on $s^{(a)}$ among those calculated on $\mathbf{s}^{\mathcal{F}_I}$ where $\mathcal{F}_I \subset \mathcal{F}_K$ are the tasks in the same sector (industry); (3) RelationDemeanOp calculates the difference between the input operand calculated on $s^{(a)}$ and the mean of those calculated on $\mathbf{s}^{\mathcal{F}_I}$.

We define OPs extracting a scalar feature from $\mathcal{X}$ and OPs extracting a vector feature from $\mathcal{X}$ as GetScalarOps and GetVectorOps respectively, or called ExtractionOps in general. Once an ExtractionOp is selected in a mutation step, $\mathcal{X}$ serves as a pool for selecting a scalar or a vector, and thus the actual input of an alpha can be $\mathcal{X}$ or just a scalar, a column, or a row of $\mathcal{X}$.



(a) An alpha with redundant operations.



(b) A redundant alpha.

Figure 5: Examples of the redundancy pruning process.

ExtractionOps facilitate searching for the new class of alphas and avoid discovering a complex machine learning alpha from scratch. Specifically, in the evolutionary process, the fitness score of an alpha augmented with extracted scalar inputs is usually high among a population, and thus this alpha is likely to survive to the next population as a parent alpha. Iteratively this process guides the evolution towards the new alpha instead of a machine learning alpha that does not allow scalar inputs.

4.2 Pruning Technique

Searching for the best alpha in a large space efficiently is challenging. Earlier work [21] increases efficiency by avoiding repeated evaluations of an alpha. It fingerprints an alpha by its predictions on a small set of samples and uses the fingerprint to terminate any repeated alphas with the same predictions on the set of samples in subsequent searches. However, this method is inefficient for two reasons. First, this method evaluates redundant operations and alphas. Second, the cost of testing on the set of samples is higher in our problem because the number of tasks K is larger than the number of the original problem. Further, we cannot approximate all the tasks with a small subset because the stocks predicted in the tasks vary greatly given the noisy nature of the stock price data.

We thus propose an optimization technique by pruning redundant operations and alphas as well as fingerprinting without evaluation. Specifically, the fingerprint of an alpha is built by pruning redundant operations and alphas before evaluation, and transforming the strings of the alpha's remaining operations into numbers. If the fingerprint is matched in the cache, the fitness score stored in the cache is reused. Otherwise, the alpha is evaluated to get its fitness score and then hashed into the cache.

The redundancy pruning process prunes the operations that do not contribute to the calculation between the input feature matrix $m0$ and the prediction $s1$. The process works as follows. First, we represent an alpha as a graph with operators as edges and operands as nodes. The prediction $s1$ is the root node. Next, starting from the root node, we iteratively check a node's redundancy by finding the operation where the node is the output operand and checking if the input operands of this operation are redundant nodes. The check returns true if a leaf is $m0$. Finally, we prune the operation with a redundant output operand. This process is illustrated in Figure 5, where the red, blue, and green nodes are the redundant operands, the necessary operands, and $m0$ respectively, and the dashed edge is the operator with the input operand calculated at the last time step. In Figure 5a, part of the alpha are redundant operations: The fourth operation of the Predict() component with output $s1$, denoted by $s1(4)$ in the graph, is redundant since it is overwritten by $s1(8)$, which is used as the prediction; The operation with output $s8$ is redundant since $s8$ does not contribute to the calculation of $s1(8)$. In Figure 5b, the alpha is redundant since $m0$ is not used to calculate the prediction.

We discuss two typical scenarios in the evolutionary process on how the pruning technique increases search efficiency. In the early stage of the evolutionary process, an alpha usually has more redundant operations than useful ones. These redundant operations can be pruned by the pruning technique. In the later stage, an alpha with no redundancy tends to be vulnerable to random mutations, e.g., deleting a random operation would invalidate the prediction. Consequently, this alpha would possibly become a redundant alpha after random mutations and would be pruned.

5 EXPERIMENTAL STUDY

In this section, we compare the performance of AlphaEvolve among different initializations, and with baselines including the genetic algorithm and complex machine learning alphas. Then we study the effectiveness of the parameter-updating function, the selective injection of relational domain knowledge, and the pruning technique.

5.1 Dataset

We use the 5-year (2013-2017) stock price data from a major stock market NASDAQ. The 5-year data consists of 1220 days in total and is split into sets of 988, 116, and 116 days for training, validation, and test, respectively. Two types of stocks are filtered out in the data preprocessing stage: (1) the stocks without sufficient samples and (2) the stocks reaching too low prices during the selected period. The first is because they are less traded and thus only bring the noise to the model, while the second is because they are too risky for investors. After filtering, there are 1026 stocks left. Each type of the features is normalized by its maximum value across all time steps for each stock.

5.2 Baselines and Settings

We use the following baselines for comparison:

(1) *alpha_G* is the searched alpha by the genetic algorithm, which is a popular alpha mining approach discussed in Section 1.
(2) Rank_LSTM is a variant of the LSTM model with its output mapped to a fully connected layer.
(3) RSR is a variant of Rank_LSTM by adding a graph component, which is designed with the injection of relational domain knowledge by connecting stocks in the same sector (industry). The RSR model is reported with the best performance in the dataset [10].
(4) *alpha_AE_D* is the evolved alpha by AlphaEvolve, initialized with a domain-expert-designed alpha. See details of the domain-expert-designed alpha in the alpha before evolving in Figure 2.
(5) *alpha_AE_NOOP* is the evolved alpha by AlphaEvolve with no initialization.
(6) *alpha_AE_R* is the evolved alpha by AlphaEvolve, initialized with an alpha designed randomly.
(7) *alpha_AE_NN* is the evolved alpha by AlphaEvolve, initialized with a two-layer neural network alpha.

We shall now describe the setting for each method. For AlphaEvolve, we use the population size of 100 and the tournament size of 10. The mutation probability of each operation is set to 0.9. The dimensions f and w for the input feature matrix X are 13. The first four features are the moving averages of the close prices over 5, 10, 20, and 30 days, respectively. The next four are the close prices' volatilities over 5, 10, 20, and 30 days, respectively. The last five are the open price, the high price, the low price, the close price, and the volume, respectively. The minimum number of the operations in each function is set to 1 and the maximum number to 21, 21, and 45, respectively. We choose the size of the maximum allowed scalar, vector, and matrix operands to be 10, 16, and 4, respectively. During the evolutionary process, we train our alpha by one epoch for fast evaluation.

For the genetic algorithm, the input and the output are the same as those of AlphaEvolve. The sizes of the generation and the tournament are the same as AlphaEvolve. Apart from that, we follow the implementation details in [15]: the probability of crossover, subtree mutation, hoist mutation, point mutation, and point replace are set to 0.4, 0.01, 0, 0.01, and 0.4, respectively.

For Rank_LSTM and RSR, each model's input is a vector of the close prices' moving averages over 5, 10, 20, and 30 days for each of the input stocks, while the output is the predicted return. Following the experiment settings of [10], we fine-tune the hyper-parameters for Rank_LSTM. To be specific, the grid for the sequence length, the number of units, and the hyperparameter balancing the loss terms are [4, 8, 16, 32], [32, 64, 128, 256], and [0.01, 0.1, 1, 10], respectively. The learning rate is set to 0.001. The best set of the hyperparameters is selected based on the performance on S_v. This set is used for reporting the performance of Rank_LSTM and then getting the pre-trained embeddings for RSR following the original implementation. The average of the testing results is reported by performing 5 runs with different random seeds.

5.3 Evaluation Metrics

Apart from the IC, we use the Sharpe ratio to measure risk-adjusted returns of a portfolio built based on an alpha. We first introduce how this portfolio is built. After that, we define the portfolio return and the Sharpe ratio.

Table 1: Mining weakly correlated alpha with an existing domain-expert-designed alpha.

Alpha	Sharpe ratio	IC	Correlation with the existing alpha
alpha_D_0	4.111784	0.013159	NA
alpha_AE_D_0	**21.323797**	**0.067358**	0.030301
alpha_G_0	13.034052	0.048853	-0.103120

We use the long-short trading strategy, a popular hedge fund strategy [13], to build a portfolio to evaluate an alpha's investment performance. At a time step t, the strategy selects stocks based on the ranking of predicted returns of all stocks to long (i.e., buy) and to short (i.e., borrow stocks to sell). The long position V_l^t is built by buying the stocks with the top 50 predicted returns. The long position gains when the prices go up and the stocks are sold to win the price differences. The short position V_s^t is built by borrowing the stocks with the bottom 50 predicted returns and selling them for cash. The short position gains when the prices go down and the stocks are bought and returned to win the price differences. These long-short positions are balanced by the cash position C^t. The reason is that with V_l^t and V_s^t fluctuating with gains or losses, we want to stick to a fixed investment plan (i.e., a fixed ratio between the two positions) to avoid large risk exposure on either side. The net asset value (NAV) is defined as $NAV^t = V_l^t + V_s^t - C^t$ and the portfolio return is defined as $R_p^t = (NAV^t - NAV^{t-1})/NAV^{t-1}$. The portfolio returns on S_v or S_{te} is a vector $\mathbf{R}_p = \left(R_p^1, \ldots, R_p^N\right)$, where N is the size of S_v or S_{te}.

The Sharpe ratio is defined as $SR = (\bar{R}_p - R_r)/\sigma_p$, where $\bar{R}_p$ is the average of $\mathbf{R}_p$, R_r is the risk-free rate[4], and σ_p is the volatility of the portfolio calculated as the standard deviation of $\mathbf{R}_p$. Both $\bar{R}_p$ and σ_p are annualized over 252 trading days.

5.4 Performance Evaluation

5.4.1 Alpha Mining Performance Comparisons. We run AlphaEvolve with each of the initializations and the genetic algorithm for five rounds. For AlphaEvolve, the best alpha with the highest Sharpe ratio among all initializations is selected into a set $\mathcal{A}$ after each round. To achieve low correlation, we discard alphas correlated with any alpha in $\mathcal{A}$ above a cutoff during the evolutionary process, where the correlation is calculated using portfolio returns on S_v and the cutoff is set at 15% by the standard in hedge funds [13]. Note that as the number of rounds and the size of $\mathcal{A}$ increase, the difficulty of generating an alpha with low correlation increases. The same process is applied to the genetic algorithm. The time budget is set to 60 hours in each round. Notation-wise, we use the last digit number in the alpha name to represent the round number (starting from 0). In the last round, the alphas in $\mathcal{A}$ are used as initializations, each denoted with B followed by a number referring to the round generating the alpha. Note that we do not set a cutoff with the domain-expert-designed alpha *alpha_D_0* because its Sharpe ratio and IC are too low compared to the evolved alphas and its correlations with the evolved alphas are weak. These results are observed in Table 1, in which *alpha_AE_D_0* and *alpha_G_0*, with the cutoffs set for the correlations with *alpha_D_0*, are compared with *alpha_D_0*.

[4]Following [13], we set R_r equal to 0 for simplicity.

Table 2: Performance of weakly correlated alpha mining.

Alpha	Sharpe ratio	IC	Correlation with the best alphas
alpha_AE_D_0	**21.323797**	**0.067358**	NA
alpha_G_0	13.034052	0.048853	NA
alpha_AE_D_1	**13.580572**	**0.056703**	-0.303845
alpha_G_1	4.407823	0.037521	-0.267428
alpha_AE_D_2	**15.067808**	**0.052464**	-0.040815
alpha_G_2	-1.936161	0.000779	-0.065011
alpha_AE_D_3	**4.901069**	**0.028437**	-0.202224
alpha_G_3	-1.971355	0.000000	-0.014054
alpha_AE_B0_4	**9.502871**	**0.032155**	0.137851
alpha_G_4	NA	NA	NA

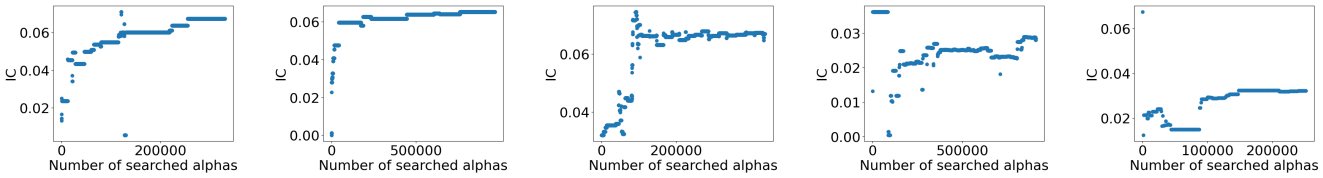
Table 3: Performance of weakly correlated alpha mining for different initializations.

Alpha	Sharpe ratio	IC	Correlation with the best alphas
alpha_AE_D_0	**21.323797**	**0.067358**	NA
alpha_AE_NOOP_0	12.126316	0.046382	NA
alpha_AE_R_0	10.718915	0.047780	NA
alpha_AE_NN_0	14.576585	0.057008	NA
alpha_AE_D_1	13.580572	0.056703	-0.303845
alpha_AE_NOOP_1	11.858020	0.044230	-0.393348
alpha_AE_R_1	12.186153	0.050688	0.008778
alpha_AE_NN_1	**14.175835**	**0.065209**	-0.240786
alpha_AE_D_2	15.067808	0.052464	-0.040815
alpha_AE_NOOP_2	12.309789	0.051791	-0.337799
alpha_AE_R_2	**18.629571**	**0.066962**	-0.177144
alpha_AE_NN_2	13.606091	0.052847	-0.242090
alpha_AE_D_3	**4.901069**	0.028437	-0.202224
alpha_AE_NOOP_3	3.873076	0.015012	0.007269
alpha_AE_R_3	3.660071	0.023426	-0.002207
alpha_AE_NN_3	3.070874	**0.031879**	-0.033980
alpha_AE_B0_4	**9.502871**	**0.032155**	0.137851
alpha_AE_B1_4	3.649546	0.014045	-0.003357
alpha_AE_B2_4	12.275912	0.059586	0.217138
alpha_AE_B3_4	3.803120	0.021081	-0.064233

The comparison results with the genetic algorithm are shown in Table 2, where the best alpha in a round is marked in bold. We observe that the Sharpe ratio and the IC of the genetic algorithm deteriorate with more cutoffs, showing that it does not do well in mining weakly correlated alphas. This result is due to the smaller search space of the genetic algorithm. Due to the consecutive low performances of *alpha_G_2* and *alpha_G_3*, we stop the search for *alpha_G_4*.

The comparison results among different initializations are shown in Table 3. *alpha_AE_D_x* shows the best results for $x = 0, 3$ as well as *alpha_AE_B0_4* in the last round. These show that AlphaEvolve can generate good alphas by leveraging a well-designed alpha. The decrease in performance for *alpha_AE_D_1* and *alpha_AE_D_2* is because the cutoff is set for the correlations with *alpha_AE_D_0*, which has the same initialized alpha. For the same reason, after the cutoff is set for the correlations with *alpha_AE_NN_1*, *alpha_AE_NN_x* has dropped in performance significantly for $x = 2, 3$. A similar drop in performance is also observed for *alpha_AE_R_3*. Such drops are observed for most initializations, but *alpha_AE_D_x* is nonetheless the second-best for $x = 1, 2$. *alpha_AE_NOOP_x* shows the worst performances by not being the best alpha in any round due to no initialization.

The first four rounds show a decreasing trend in both the Sharpe ratio and the IC because the accumulative cutoffs increase the search difficulty. This increasing difficulty is also shown in the evolutionary trajectories of the best alphas from all rounds in $\mathcal{A}$



(a) *alpha_AE_D_0* **(b)** *alpha_AE_NN_1* **(c)** *alpha_AE_R_2* **(d)** *alpha_AE_D1_3* **(e)** *alpha_AE_B0_4*

Figure 6: Evolutionary trajectories for the best alphas in all rounds.

in Figure 6, where the ICs decrease and fluctuate as the round increases. This trend reverses when we set the previous best alphas as the initialized alphas in the last round. It shows the capability of AlphaEvolve in generating weakly correlated alphas with all cutoffs. Note that we choose *alpha_AE_B0_4* as the best alpha instead of *alpha_AE_B2_4* since the latter correlates with the previous alphas above 15%.

5.4.2 Study of The Evolved Alphas. We first study the best alphas from all rounds in $\mathcal{A}$ and then perform the ablation study on the parameter-updating functions. To ease readability, we first change the raw output of AlphaEvolve, e.g., the form of the evolved alpha in Figure 2, into a compact set of equations. Then we divide the equations into three parts: M, P, and U. In the training stage, the predict function is M and the parameter-updating function is U, while in the inference stage, the prediction function consists of M and P. M is used in both stages to pass parameters between the stages.

For *alpha_AE_D_0* (Eq. 2 to Eq. 9), $S4_{t-2}$ and $S2_{t-2}$ are updated by Eq. 6 and Eq. 7 respectively in the training stage, and then passed to M as the parameters at the beginning of the inference stage. These parameters affect output operands $S1_t$ in Eq. 2 and $S3_{t-1}$ in Eq. 3 by initializing the input operands. Then the parameter $S4_{t-2}$ is overwritten in Eq. 4. The remaining parameter $S2_{t-2}$ is used in an upper bound $\arcsin(S2_{t-2})$ for an expression of the temporal difference (i.e., trend) of the high prices in Eq. 3. This bound will be overwritten once it is less than the trend, upon which the model becomes a formulaic alpha. This is because a formulaic alpha is a special case of the new alpha with no parameters. The prediction is the fraction with an expression of bounded trend feature on high prices as the numerator and another trend feature as the denominator in Eq. 2. Therefore, this alpha makes trading decisions based on the trends of high prices, constrained by the historically updated bound $\arcsin(S2_{t-2})$.

$$M: \quad S1_t = \tan(S3_{t-1})/\cos(S4_{t-2} - \arcsin(high\_price_{t-1})) \tag{2}$$
$$S3_{t-1} = \min(S4_{t-2} - \arcsin(high\_price_{t-1}), \arcsin(S2_{t-2})) \tag{3}$$
$$P: \quad S4_{t-2} = \arctan(\arcsin(high\_price_{t-3})) \tag{4}$$
$$S2_{t-2} = \arctan(S3_{t-2}) \tag{5}$$
$$U: \quad S4_{t-2} = \tan(heaviside(S1_{t-2})) \tag{6}$$
$$S2_{t-2} = \arccos(norm(norm(M2_{t-4}, axis = 0))) \tag{7}$$
$$M2_{t-4} = \min(abs(abs(M1_{t-4})), broadcast(broadcast(S0_{t-4}), axis = 1)) \tag{8}$$
$$M1_{t-4} = matmul(M2_{t-5}, M1_{t-5}) \tag{9}$$

For *alpha_AE_NN_1* (Eq. 10), it is a complex formula using *relation_rank* operator and *high_price*.

$$M: S1_t = \log(\cos(\arcsin(\min(tsrank(abs(relation\_rank(\arctan(\sin(\sin(\exp(high\_price_{t-2}))))))))), \log(\sin(\arctan(\sin(\sin(\exp(high\_price_{t-1}))))))) \tag{10}$$

For *alpha_AE_R_2* (Eq. 11 to Eq. 16), the parameter $M2_{t-2}$ is updated recursively with an expression of the input feature matrix $M0_{t-2}$ (Eq. 14 and Eq. 15). $S2_{t-2}$ is a trend feature based on the comparison between $high\_price_{t-4}$ and a recursively compared feature of $high\_price_{t-5}$ (Eq. 12 and Eq. 13). Thus in the inference stage, we can observe from Eq. 11 that the alpha makes trading decision based on the volatility of the historically updated features $M2_{t-2}$, the trend feature based on high prices $S2_{t-2}$ and the recent return $S0_{t-3}$.

$$M: \quad S1_t = std(M2_{t-2}) \times (\arctan(S0_{t-3}) - S2_{t-2}) \times S2_{t-2} \tag{11}$$
$$P: \quad S2_{t-2} = \max(\sin(S3_{t-3}), high\_price_{t-4}) \tag{12}$$
$$S3_{t-3} = \max(S3_{t-4}, \max(\sin(S3_{t-4}), high\_price_{t-5})) \tag{13}$$
$$U: \quad M2_{t-2} = abs(M1_{t-2}) \tag{14}$$
$$M1_{t-2} = M2_{t-3} + heaviside(\min(M2_{t-3}, \min(M2_{t-3} + M1_{t-3}, M2_{t-3})), 1) + M0_{t-2} \tag{15}$$
$$S2_{t-2} = low\_price_{t-10} \tag{16}$$

For *alpha_AE_D_3* (Eq. 17 to Eq. 19), a lower bound of the transpose of the input feature matrix $M0_{t-2}$ is set for an expression of $M0_{t-2}$ to recursively update the parameter $M1_{t-2}$ (Eq. 19). At the beginning of the inference stage, $M1_{t-2}$ is passed to M and P as initial matrices $M1_{t-2}$ and $M1_{t-3}$ respectively (Eq. 17 and Eq. 18). Then $M1_{t-3}$ recursively compares with an expression of $M0_{t-2}$ (Eq. 18). Finally, the prediction is the standard deviation of another comparison result between $M1_{t-2}$ and an expression of another input feature matrix $M0_{t-1}$ (Eq. 17), showing that this alpha trades based on the volatility of an expression of the previous day's features $M0_{t-1}$ bounded by the historically updated features $M1_{t-2}$. Note that once $M1_{t-3}$ is larger than $heaviside(M0_{t-2}, 1)$ (Eq. 18), this alpha becomes a formula without parameters.

$$M: \quad S1_t = std(\min(heaviside(M0_{t-1}, 1), M1_{t-2})) \tag{17}$$
$$P: \quad M1_{t-2} = \max(M0_{t-2}, \min(heaviside(M0_{t-2}, 1), M1_{t-3})) \tag{18}$$
$$U: \quad M1_{t-2} = \max(transpose(M0_{t-2}), \max(M0_{t-2}, \min(heaviside(M0_{t-2}, 1), M1_{t-3}))) \tag{19}$$

For *alpha_AE_B0_4* (Eq. 20 to Eq. 22), the parameter $M1_{t-2}$ is updated recursively with an expression of $M0_{t-2}$ and an expression of $M0_{t-4}$ (Eq. 21 and Eq. 22). The prediction is based on the comparison between the inverse of $close\_price_{t-3}$ and the expression of $MV30_{t-4}$ (i.e., the moving average of the close prices over the last 30 days calculated at $t-4$), and the standard deviation of $M1_{t-2}$ (Eq. 20). Thus, this alpha makes trading decisions based on the recent

Table 4: Ablation study of the parameter-updating function.

Alpha	Sharpe ratio	IC	Correlation with the best alphas
alpha_AE_D_0	21.323797	**0.067358**	NA
alpha_AE_D_0_P	**21.516798**	0.057707	NA
alpha_AE_R_2	**18.629571**	**0.066962**	-0.177144
alpha_AE_R_2_P	-0.344734	0.003149	-0.094286
alpha_AE_D_3	4.901069	**0.028437**	-0.202224
alpha_AE_D_3_P	**5.697408**	0.026347	-0.241651
alpha_AE_B0_4	**9.502871**	**0.032155**	0.137851
alpha_AE_B0_4_P	-0.004294	-0.001908	-0.097541

Table 5: Performance comparisons with the complex machine learning alphas.

Alpha	Sharpe ratio	IC
alpha_AE_D_0	**21.323797**	**0.067358**
alpha_AE_NN_1	14.175835	0.065209
Rank_LSTM	5.385036+/-1.608296	0.027490+/-0.009336
RSR	5.647131+/-0.522782	0.018623+/-0.000794

close price $close\_price_{t-3}$, the trend of close prices $MV30_{t-4}$, and the volatility of the historically updated features $M1_{t-2}$.

$$M: \quad S1_t = \tan(\tan(\min(1/close\_price_{t-3}, \arcsin(\arcsin(MV30_{t-4})))))/std(M1_{t-2}) \tag{20}$$

$$U: M1_{t-2} = matmul(M2_{t-2}, M0_{t-2}M0_{t-2}) \tag{21}$$

$$M2_{t-2} = transpose(\max(heaviside(abs(M2_{t-3})), broadcast(vector\_uniform(0.314561, -0.187581) + norm(M0_{t-4}, axis = 0), axis = 0))) \tag{22}$$

In Table 4, we perform the ablation study of the parameter-updating function, where each alpha without the parameter-updating function is denoted with an additional P. This ablation study is crucial given the previous observation that the new alpha can be converted into a formulaic alpha. We observe that all the parameter-updating functions are effective by increasing the fitness scores, i.e., ICs, in the inference stage. However, the Sharpe ratios do not change with the increasing ICs for *alpha_AE_D_0_P* and *alpha_AE_D_3_P*. This is expected because the Sharpe ratio of a portfolio depends on the top and bottom stock rankings while the IC measures rankings of all stocks.

5.4.3 Comparisons With The Complex Machine Learning Alphas. In Table 5, we compare the complex machine learning alphas with the generated alphas by AlphaEvolve. We observe that both the complex models fail to compete against *alpha_AE_D_0*. The poor performance of RSR is due to the imposition of the relational domain knowledge to the data. The NASDAQ dataset cannot be best explained by the relational domain knowledge because the NASDAQ is a more noisy stock market compared to other stock markets (e.g., the NYSE)[5]: it incorporates many volatile stocks with less capitalization. Therefore, the noisy stock market affected by rapid-changing information cannot be modeled with the static relational knowledge. Besides, Rank_LSTM and RSR are unstable with high standard deviations because their predictions are influenced by random effects (i.e., random seeds). In contrast, the flexibility of the domain knowledge injection by AlphaEvolve leads to the best performance of *alpha_AE_D_0* without the domain knowledge, while

[5]https://finance.yahoo.com/news/nasdaq-vs-nyse-key-differences-200641822.html

Table 6: Efficiency of the pruning technique.

Alpha	Sharpe ratio	IC	Correlation	Number of searched alphas
alpha_AE_D_0	**21.323797**	**0.067358**	NA	309700
alpha_AE_D_0_N	8.898872	0.057817	NA	19500
alpha_AE_NN_1	**13.580572**	**0.056703**	-0.303845	1032700
alpha_AE_NN_1_N	5.148189	0.025506	-0.052586	5700
alpha_AE_R_2	**18.629571**	**0.066962**	-0.177144	429800
alpha_AE_R_2_N	4.575985	0.032180	0.116183	13200
alpha_AE_D_3	**4.901069**	**0.028437**	-0.202224	910100
alpha_AE_D_3_N	4.604322	0.028945	-0.051935	37900
alpha_AE_B0_4	**9.502871**	**0.032155**	0.137851	220100
alpha_AE_B0_4_N	2.775825	0.027594	-0.137923	17300

alpha_AE_NN_1 generated with the relational domain knowledge is weakly correlated with *alpha_AE_D_0* but has a relatively lower Sharpe ratio and IC.

5.4.4 Efficiency of The Pruning Technique. In the ablation study of the pruning technique, we remove this technique from AlphaEvolve but use the prediction of an alpha as the fingerprint. In Table 6, we denote each of the alphas without the technique with an additional letter N. The number of searched alphas is the sum of pruned alphas and evaluated alphas. We observe that, for the baseline methods, the numbers of searched alphas are significantly less than those of AlphaEvolve, leading to ineffective and inefficient alpha mining, which proves the effectiveness of our proposed technique.

6 RELATED WORK

Previous works focus on two classes of alphas: (1) machine learning alphas with vector and matrix operations; (2) formulaic alphas with scalar operations. For the first class, various models are proposed given more efficient deep learning systems to capture high-dimensional features [23]. [25] proposes a novel TITV architecture to model the time-invariant/variant features for the stock index prediction. [20] proposes a two-level attention mechanism to assign importance weights on time steps and stocks. [24] puts up a novel State Frequency Memory (SFM) that decomposes the hidden states of memory cells into multiple frequency components to model different trading activities. [7] uses a neural tensor network to extract event information from news data. [4] encodes time-series images as candlestick (Box and Whisker) charts and models the image feature. [19] creates a simple language of Japanese candlesticks using OHLC data. Domain knowledge injection has proven useful in designing alphas [1, 12]. [10] injects relational knowledge through a novel graph network. In general, this class of alphas is designed by data scientists and typically too complex to mine into a weakly correlated set.

The second class of alphas is widely used in hedge funds. This class of alphas is designed by financial domain experts or mined by the genetic algorithm. [13] studies alphas invested in a hedge fund and their correlation standard. [15] reports alphas mined by the genetic algorithm and their performance. [14] further improves the genetic algorithm by using mutual information as the fitness score to mine nonlinear formulaic alphas. [9] proposes an indexing method to locate similar stock movement patterns. [16] proposes N-dimensional inter-transaction rules to predict stock movements. To our best knowledge, our work is the first to mine alphas based on AutoML and first to generate the new class of alphas combining the strengths of the existing classes.

7 CONCLUSIONS

In this paper, we introduce a new class of alphas and then propose a novel alpha mining framework AlphaEvolve based on AutoML. AlphaEvolve generates the new class of alphas which are different from previous classes of formulas and machine learning models. This class has the advantages of simplicity and generalization ability similar to formulaic alphas, and the ability to be trained by data similar to machine learning alphas. These advantages result in better performances in generating weakly correlated high returns. Consequently, AlphaEvolve provides investors with an automatic solution for low-risk investments with high returns.

8 ACKNOWLEDGEMENT

This research is funded by Singapore Ministry of Education Academic Research Fund Tier 3 with the award number MOE2017-T3-1-007. Meihui Zhang's research is supported by National Natural Science Foundation of China (62050099).

REFERENCES

[1] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (Vancouver, Canada) *(SIGMOD '08)*. Association for Computing Machinery, New York, NY, USA, 1247–1250. https://doi.org/10.1145/1376616.1376746

[2] Charles Cao, Bing Liang, Andrew W Lo, and Lubomir Petrasek. 2018. Hedge fund holdings and stock market efficiency. *The Review of Asset Pricing Studies* 8, 1 (2018), 77–116.

[3] L. Cheng, Y. Huang, and M. Wu. 2018. Applied attention-based LSTM neural networks in stock prediction. In *2018 IEEE International Conference on Big Data (Big Data)*. 4716–4718.

[4] Naftali Cohen, Tucker Balch, and Manuela Veloso. 2019. Trading via Image Classification. *CoRR* abs/1907.10046 (2019). arXiv:1907.10046 http://arxiv.org/abs/1907.10046

[5] Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. 2020. Discovering Symbolic Models from Deep Learning with Inductive Biases. arXiv:2006.11287 [cs.LG]

[6] Shumin Deng, Ningyu Zhang, Wen Zhang, Jiaoyan Chen, Jeff Z. Pan, and Huajun Chen. 2019. Knowledge-Driven Stock Trend Prediction and Explanation via Temporal Convolutional Network. In *Companion Proceedings of The 2019 World Wide Web Conference* (San Francisco, USA) *(WWW '19)*. Association for Computing Machinery, New York, NY, USA, 678–685. https://doi.org/10.1145/3308560.3317701

[7] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. 2015. Deep Learning for Event-driven Stock Prediction. In *Proceedings of the 24th International Conference on Artificial Intelligence* (Buenos Aires, Argentina) *(IJCAI'15)*. AAAI Press, 2327–2333. http://dl.acm.org/citation.cfm?id=2832415.2832572

[8] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. 2016. Knowledge-Driven Event Embedding for Stock Prediction. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. The COLING 2016 Organizing Committee, Osaka, Japan, 2133–2142. https://www.aclweb.org/anthology/C16-1201

[9] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. 1994. Fast Subsequence Matching in Time-Series Databases. *SIGMOD Rec.* 23, 2 (May 1994), 419–429. https://doi.org/10.1145/191843.191925

[10] Fuli Feng, Xiangnan He, Xiang Wang, Cheng Luo, Yiqun Liu, and Tat-Seng Chua. 2019. Temporal relational ranking for stock prediction. *ACM Transactions on Information Systems (TOIS)* 37, 2 (2019), 27.

[11] Gabriel Pui Cheong Fung, Jeffrey Xu Yu, and Wai Lam. 2002. News Sensitive Stock Trend Prediction. In *Advances in Knowledge Discovery and Data Mining*, Ming-Syan Chen, Philip S. Yu, and Bing Liu (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 481–493.

[12] Weiwei Jiang. 2020. Applications of deep learning in stock market prediction: recent progress. arXiv:2003.01859 [q-fin.ST]

[13] Zura Kakushadze. 2016. *101 Formulaic Alphas*. Papers 1601.00991. arXiv.org. https://ideas.repec.org/p/arx/papers/1601.00991.html

[14] Xiaoming Lin, Ye Chen, Ziyu Li, and Kang He. 2019. *Enhancing Stock Alpha Mining Based On Genetic Algorithm*. Papers. Huatai Security Research Center. https://crm.htsc.com.cn/doc/2019/10750101/f75b4b6a-2bdd-4694-b696-4c62528791ea.pdf

[15] Xiaoming Lin, Ye Chen, Ziyu Li, and Kang He. 2019. *Stock Alpha Mining Based On Genetic Algorithm*. Papers. Huatai Security Research Center. https://crm.htsc.com.cn/doc/2019/10750101/f75b4b6a-2bdd-4694-b696-4c62528791ea.pdf

[16] H. Lu, J. Han, and L. Feng. 1998. Stock movement prediction and N-dimensional inter-transaction association rules. In *SIGMOD 1998*.

[17] Zhaojing Luo, Sai Ho Yeung, Meihui Zhang, Kaiping Zheng, Lei Zhu, Gang Chen, Feiyi Fan, Qian Lin, Kee Yuan Ngiam, and Beng Chin Ooi. 2021. MLCask: Efficient Management of Component Evolution in Collaborative Data Analytics Pipelines. arXiv:2010.10246 [cs.SE]

[18] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. 2017. Evolving Deep Neural Networks. (03 2017).

[19] Marko Poženel and Dejan Lavbič. 2019. *Discovering Language of Stocks, Frontiers in Artificial Intelligence and Applications-Databases and Information Systems X, 315*. https://doi.org/10.3233/978-1-61499-941-6-243

[20] Yao Qin, Dongjin Song, Haifeng Cheng, Wei Cheng, Guofei Jiang, and Garrison W. Cottrell. 2017. A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence* (Melbourne, Australia) *(IJCAI'17)*. AAAI Press, 2627–2633.

[21] Esteban Real, Chen Liang, David R So, and Quoc V Le. 2020. AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. *arXiv preprint arXiv:2003.03384* (2020).

[22] I. Tulchinsky. 2015. *Finding Alphas: A Quantitative Approach to Building Trading Strategies*. 1–253 pages. https://doi.org/10.1002/9781119057871

[23] Wei Wang, Meihui Zhang, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Kian-Lee Tan. 2016. Database Meets Deep Learning: Challenges and Opportunities. *SIGMOD Rec.* 45, 2 (Sept. 2016), 17–22. https://doi.org/10.1145/3003665.3003669

[24] Liheng Zhang, Charu Aggarwal, and Guo-Jun Qi. 2017. Stock Price Prediction via Discovering Multi-Frequency Trading Patterns. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Halifax, NS, Canada) *(KDD '17)*. ACM, New York, NY, USA, 2141–2149. https://doi.org/10.1145/3097983.3098117

[25] Kaiping Zheng, Shaofeng Cai, Horng Ruey Chua, Wei Wang, Kee Yuan Ngiam, and Beng Chin Ooi. 2020. TRACER: A Framework for Facilitating Accurate and Interpretable Analytics for High Stakes Applications *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3318464.3389720