AlphaTree: Structure-Aware Reinforcement Learning for Formulaic Alpha Mining

1st Xiaolong Yang
School of Future Technology
South China University of Technology
Guangzhou, China
bruceyang@bayescrest.com

2nd Yang Wang*
School of Future Technology
South China University of Technology
Guangzhou, China
ftwangyang@mail.scut.edu.cn

3rd Ya-Hui Jia*
School of Future Technology
South China University of Technology
Guangzhou, China
jia.yahui@foxmail.com

4th Qiang Yang
School of Artificial Intelligence
Nanjing University of Information Science and Technology
Nanjing, China
mmmyq@126.com

5th An Song
GF Asset Management
Guangzhou, China
safe.song@qq.com

***Abstract*—Formulaic alpha discovery is crucial in quantitative finance but remains challenging due to the vast search space for expressions. Existing reinforcement learning methods treat alpha formulas as flat token sequences, failing to capture their inherent tree structure. In this paper, we propose AlphaTree, which uses a Tree-structured Long Short-Term Memory (Tree-LSTM) network to encode the hierarchical structure of mathematical expressions during generation. Unlike sequential models that process tokens left-to-right, Tree-LSTM captures operator-operand relationships through bottom-up information propagation over the expression tree. Furthermore, we integrate Tree-LSTM with distributional reinforcement learning to address non-stationarity and reward sparsity. Experiments on the CSI300 and CSI500 datasets demonstrate that AlphaTree significantly outperforms existing methods, achieving higher information coefficients and producing substantial improvements in risk-adjusted returns. Our code is available at https://github.com/ftwangyang/AlphaTree.**

***Index Terms*—Formulaic alpha discovery, Tree-LSTM, Deep reinforcement learning, Quantitative finance.**

I. Introduction

The discovery of effective trading signals, known as alphas, represents a fundamental challenge in quantitative finance [1], [2]. Formulaic alphas, expressed as mathematical expressions combining market features through operators, offer distinct advantages over black-box machine learning models due to their interpretability, compactness, and generalizability [3]–[5]. However, the vast search space of potential expressions, which scales exponentially with the available features and operators, makes automatic alpha discovery exceptionally difficult [6], [7].

Recent advances have reformulated alpha discovery as a sequential decision-making problem within the Markov Decision Process (MDP) framework [8]–[10]. AlphaGen [11] pioneered token-by-token generation using Proximal Policy Optimization (PPO). AlphaQCM [12] further addressed the non-stationarity and reward sparsity through distributional Reinforcement Learning (RL). These methods employ Reverse Polish Notation (RPN) to represent formulaic alphas as token sequences, using Long Short-Term Memory (LSTM) networks as feature extractors to encode the current state of expression generation [13].

However, treating alpha formulas as flat sequences fundamentally misrepresents their mathematical structure. Consider the expression $(-1) \times \mathrm{TsRank}(\mathrm{Rank}(\mathrm{Low}), 9)$: while its RPN representation is a linear sequence, the computation follows a tree structure where $\mathrm{Rank}(\mathrm{Low})$ forms a complete subexpression feeding into TsRank. Sequential LSTM processes tokens left-to-right but cannot capture that distant tokens may form tight computational dependencies. This structural mismatch leads to inefficient exploration and suboptimal alpha discovery. Mathematical expressions are naturally tree-structured, and the state encoder should respect this structure to enable better compositional understanding [14].

To address this limitation, we propose AlphaTree, a structure-aware method for formulaic alpha discovery that explicitly leverages the tree structure of mathematical expressions. Unlike prior methods that process RPN tokens left-to-right, AlphaTree performs bottom-up computation on expression trees, naturally capturing operator-operand dependencies regardless of sequential distance. This structural alignment enables more effective exploration in the vast search space of formulaic alphas. Specifically, we make the following contributions:

- We introduce a Tree-structured Long Short-Term Memory (Tree-LSTM) network as the feature extractor for alpha discovery, enabling the agent to learn structure-aware representations that respect the compositional semantics of mathematical expressions.
- We develop a seamless integration of Tree-LSTM with the distributional RL framework, where Tree-LSTM provides hierarchical state encoding while the Q-network and

*Corresponding author.

This work is supported in part by the National Natural Science Foundation of China under Grant 62576172, in part by the introduced innovative R&D team of Guangdong under Grant 2023ZT10L145.

quantile network guide action selection through learned value estimates and uncertainty quantification.

We conduct comprehensive experiments on the CSI300 and CSI500 datasets, demonstrating that AlphaTree consistently outperforms existing baselines, yielding notable gains in information coefficient and risk-adjusted returns.

The remainder of this paper is organized as follows. Section II reviews the background and related work. Section III details the Tree-LSTM architecture and its integration with distributional RL. Section IV provides extensive experimental evaluation. Section V concludes the paper.

II. Background and Related Work

A. Problem Formulation

We consider the task of discovering a pool $\mathcal{F} = \{f_1, f_2, \ldots, f_P\}$ of synergistic formulaic alphas for predicting stock returns. Each alpha f_p maps historical market data H_{l-1} to cross-sectional alpha values $\alpha_{p,l} = f_p(H_{l-1}) \in \mathbb{R}^E$ for E stocks at trading day l. The meta-alpha is constructed as a linear combination:

$$\hat{\alpha}_l = \sum_{p=1}^{P} \alpha_{p,l} \hat{\beta}_p, \tag{1}$$

where $\hat{\beta}_p$ are learned coefficients. The objective is to maximize the Information Coefficient (IC), the Pearson correlation between the meta-alpha and future returns, measuring the predictive power of the discovered alphas.

As illustrated in Fig. 1, the expression $(-1) \times \mathrm{TsRank}(\mathrm{Rank}(\mathrm{Low}), 9)$ can be represented both as a token sequence and as an expression tree. While prior methods [15] encode the sequential RPN representation using LSTM, our method reconstructs the expression tree from RPN and applies Tree-LSTM to capture hierarchical dependencies directly.

B. Related Work

1) Evolutionary Algorithms: Genetic Programming (GP) dominated early formulaic alpha mining. Cui *et al.* [6] and Zhang *et al.* [16] evolve expression trees via mutation and crossover, yielding interpretable formulas. However, GP methods suffer from premature convergence and sensitivity to initialization. The stochastic nature of genetic operators lacks gradient-guided directionality, resulting in sample inefficiency on large feature spaces [17].

2) Generative Models: A common challenge in alpha discovery is that optimizing a single objective often leads to many similar signals, whereas a diverse set of uncorrelated alphas is more desirable for portfolio construction. Shi *et al.* [18] proposed a two-stage framework that first generates candidate factors using a variational autoencoder and then dynamically combines them to form a portfolio. Chen *et al.* [19] introduced generative flow networks to sample formulas with probability proportional to their reward, and used a graph neural network (GNN) to encode each complete expression tree for evaluation. While these methods encourage diversity and incorporate structure via GNN encoding, the structural information is extracted only after a full formula is generated (post-hoc) rather than during the generation process.

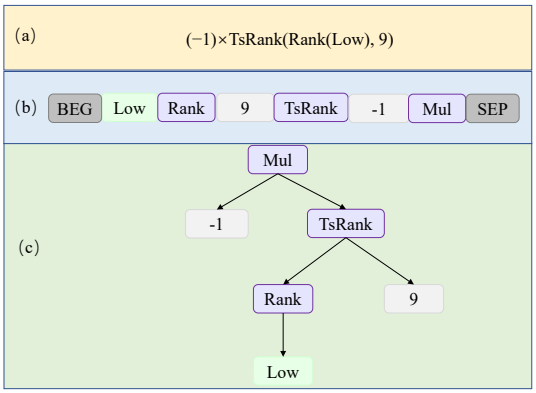


Fig. 1: Motivation for structure-aware encoding. (a) Original mathematical expression. (b) RPN sequence loses hierarchical structure, sequential LSTM struggles to connect distant but semantically related tokens like 'Low' and its operator 'Rank'. (c) Tree structure explicitly captures the computation flow: 'Rank' directly operates on 'Low', and this subexpression feeds into 'TsRank'. Tree-LSTM leverages this structure for better state representation.

3) Transformer and Large Language Model methods: Viewing symbolic mathematics as a language, Xu *et al.* [20] introduced an end-to-end Transformer that directly generates complete formulaic risk factors, including constants. Using a hybrid symbolic-numeric vocabulary and Broyden–Fletcher–Goldfarb–Shanno refinement for predicted parameters, their method achieves orders-of-magnitude faster inference than symbolic regression benchmarks on high-frequency data. Large Language Models (LLMs) have also been explored [21]. Cao [22] leverages chain-of-thought reasoning for alpha generation, while Tang *et al.* [23] combine LLM-driven exploration with regularization against alpha decay. Shi *et al.* [24] guide LLMs via Monte Carlo Tree Search for more efficient exploration.

4) Reinforcement Learning methods: More recently, researchers have framed alpha discovery as a sequential decision process that can be tackled with reinforcement learning. Yu *et al.* [11] were the first to generate alphas token-by-token using a PPO-based agent. Zhu [12] addressed the non-stationary and sparse-reward nature of the environment by using distributional RL with Quantile Conditional Moment learning. Zhao *et al.* [25] argued that PPO is suboptimal for this problem due to the deterministic state transitions in formula construction. They proposed a variance-bounded REINFORCE algorithm with a specialized baseline, incorporating an information ratio term into the reward to encourage more stable factors. Xu *et al.* [26] extended the RL framework to discover logical combinations of alphas, using enhanced exploration strategies. While these RL-based methods improved the efficiency of alpha discovery, they all rely on sequential encoders over RPN tokens, which do not explicitly capture the hierarchical structure of mathematical

expressions.

Despite the variety of approaches, a fundamental limitation remains: existing methods typically encode an alpha formula as a flat sequence of tokens, which neglects the formula's true computation graph. Complex expressions have hierarchical dependencies that sequential models struggle to learn. In this work, we address this gap by introducing a Tree-LSTM to encode the partial expression at each step of generation, preserving the structural context throughout the reinforcement learning process.

III. Methodology

A. Alpha Discovery as a Markov Decision Process

We formulate formulaic alpha discovery as a MDP $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. The environment dynamics are non-stationary (because adding new alphas changes the portfolio's performance distribution over time), and rewards are sparse. Unlike prior methods that represent the state as a simple token sequence [12], we define each state in terms of an expression tree to capture structural information.

State Space. A state $x_t \in \mathcal{X}$ corresponds to a partially constructed expression, represented by a partial expression tree T_t. The initial state x_0 is an empty tree (no tokens). To maintain interpretability and avoid overfitting, we impose a maximum tree depth, which limits the complexity of any generated expression.

Action Space. An action $a_t \in \mathcal{A}$ adds a token (operand, operator, or constant) to the expression. The action vocabulary $\mathcal{A}$ is constructed from a feature set $\mathcal{V}$ (e.g., Open, Close, Volume), an operator set $\mathcal{O}$ (e.g., $+$, $-$, Rank, Delay), and numerical constants. At any given step, only a subset of actions $\mathcal{A}_{\text{valid}} \subseteq \mathcal{A}$ is valid based on the partial expression T_t. For example, choosing a binary operator is only valid if there are at least two completed subexpressions on the stack that can serve as operands. These validity constraints ensure that every sequence of actions corresponds to a syntactically valid formula.

Transition Dynamics. The state transition function $\mathcal{P}$ is deterministic. Given the current state x_t (expression tree T_t) and a selected action a_t, the next state is obtained by placing the token a_t into the appropriate position in the tree:

$$x_{t+1} = \mathcal{P}(x_t, a_t) = \text{Extend}(T_t, a_t), \tag{2}$$

where Extend operation attaches a new node to the tree (or fills in a pending operand slot) according to the formula construction rules. An episode (i.e., generating one formula) terminates when the expression tree is complete, all required children for each operator have been provided, or when a special end-of-expression token is generated.

Reward Function. To address the sparsity of rewards, we assign $r_t = 0$ for all intermediate steps before the formula is complete. A non-zero reward is only given at the terminal step M of an episode. If the generated expression is invalid (e.g., an incomplete formula or division-by-zero error), we assign a penalty $r_M = -1$. Otherwise, for a valid formula α_{new}, we evaluate its marginal contribution to the current alpha pool:

$$r_M = IC(\mathcal{F} \cup \{\alpha_{\text{new}}\}) \; - \; IC(\mathcal{F}), \tag{3}$$

where $IC(\cdot)$ measures the information coefficient of the meta-alpha constructed from the given pool of alphas. In essence, the reward is the improvement in the meta-alpha's predictive correlation when the newly discovered alpha is added to the portfolio.

Discount Factor. We set $\gamma = 1$ because each episode has a finite horizon (bounded by the maximum formula length) and our goal is to maximize the undiscounted terminal reward of each formula.

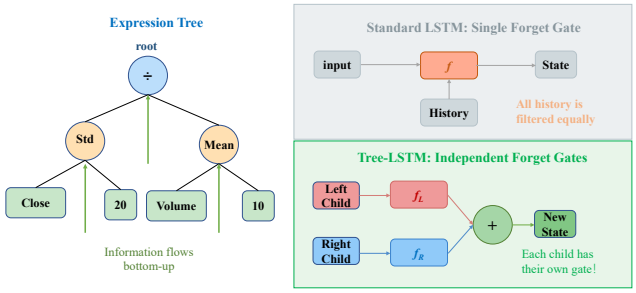


Fig. 2: Tree-LSTM state encoding for a formula. Information flows upward from feature nodes (leaves) through operator nodes to the root. Each operator node combines information from its children using child-specific forget gates (f_L, f_R), allowing it to filter each child's contribution independently.

B. Tree-LSTM for Structure-Aware State Encoding

A key innovation of AlphaTree is the use of a Tree-LSTM network to encode the state (partial formula) in a structure-aware manner. Mathematical expressions have an inherent tree structure, and we design the state encoder to exploit this. Consider a simple formula $\frac{\text{Std}(\text{Close},20)}{\text{Mean}(\text{Volume},10)}$. This formula naturally forms a binary tree with the division as the root node and two subtrees corresponding to the numerator and denominator. A sequential LSTM processing the RPN token sequence ('Close, 20, Std, Volume, 10, Mean, Div') would have to implicitly learn this structure, whereas a Tree-LSTM can encode it explicitly. Fig. 2 illustrates how Tree-LSTM processes an expression tree, propagating information from leaf nodes (features) through intermediate operators to the root, capturing the hierarchical computation structure.

In a Tree-LSTM, each node in the expression tree computes its hidden state from its children's states. For a node j with children $C(j)$, let e_j be the embedding of the token at node j (e.g., feature embedding or operator embedding). Let (h_k, c_k) be the hidden and cell state of child $k \in C(j)$. We first aggregate information from all children:

$$\tilde{h}_j = \sum_{k \in C(j)} h_k. \tag{4}$$

Then compute gate activations and the new states as:

$$i_j = \sigma\Big(W^{(i)} e_j + U^{(i)} \tilde{h}_j + b^{(i)}\Big), \tag{5}$$

$$o_j = \sigma\Big(W^{(o)} e_j + U^{(o)} \tilde{h}_j + b^{(o)}\Big), \tag{6}$$

$$u_j = \tanh\Big(W^{(u)} e_j + U^{(u)} \tilde{h}_j + b^{(u)}\Big), \tag{7}$$

$$f_{jk} = \sigma\Big(W^{(f)} e_j + U^{(f)} h_k + b^{(f)}\Big), \quad \forall k \in C(j), \tag{8}$$

$$c_j = i_j \odot u_j \; + \sum_{k \in C(j)} f_{jk} \odot c_k, \tag{9}$$

$$h_j = o_j \odot \tanh(c_j), \tag{10}$$

where σ is the sigmoid function and $\odot$ denotes element-wise multiplication. Here, i_j, o_j, and f_{jk} are the input, output, and forget gates (with one forget gate f_{jk} for each child k), and u_j is the candidate cell state.

Notably, unlike a standard LSTM, which uses a single forget gate for all past information, the Tree-LSTM uses a separate forget gate f_{jk} for each child. This allows the model to selectively retain or forget information from each subexpression independently. After computing the states in a bottom-up fashion, the root node's hidden state h_{root} provides an encoding of the entire expression. We denote the Tree-LSTM encoding of the partial expression T_t as :

$$s_t = \psi(T_t) = h_{\text{root}} \in \mathbb{R}^d, \tag{11}$$

where d is the hidden dimension of the Tree-LSTM.

C. Integration with Distributional Reinforcement Learning

To effectively search the immense formula space, we integrate the Tree-LSTM encoder with a distributional reinforcement learning strategy [27] that can handle the non-stationary, high-variance rewards. An overview of our action selection method is shown in Fig. 3. It consists of three main components: the Tree-LSTM feature extractor, a Q-network for value estimation, and a quantile network for uncertainty estimation.

1) Network Architecture: The Tree-LSTM encoder $\psi(\cdot)$ produces a state vector $s_t = \psi(T_t)$ for the current partial expression. This state representation is then passed to a Q-network and a quantile network [28]. The Q-network outputs a scalar Q-value for each possible action:

$$\hat{Q}(x_t, a) = [\text{MLP}_Q(s_t)]_a, \qquad \forall a \in \mathcal{A}. \tag{12}$$

The quantile network outputs K quantile values $\{\hat{\theta}_1, \ldots, \hat{\theta}_K\}$ characterizing the return distribution for each action.

2) Uncertainty Estimation and Action Selection: We use the Quantile Conditional Moment (QCM) method [12] to estimate return variance from quantile outputs:

$$\hat{h}(x_t, a) = \sum_{k=1}^{K-1} w_k \Big(\hat{\theta}_{k+1}(x_t, a) - \hat{\theta}_k(x_t, a)\Big)^2, \tag{13}$$

Algorithm 1 Training of AlphaTree

Require: Market data $\mathcal{D}$, feature set $\mathcal{V}$, operator set $\mathcal{O}$, total steps N
Ensure: Alpha factor pool $\mathcal{F}$

```
Initialize empty alpha pool F ← ∅
Initialize Tree-LSTM encoder ψ, Q-network, quantile network
Initialize target networks and replay buffer B
for step = 1 to N do
    Initialize empty expression tree T_0
    while expression not complete do
        Encode tree state s_t = ψ(T_t)                ▷ Eq. (11)
        Compute Q-values Q̂(x_t, a)                    ▷ Eq. (12)
        Estimate uncertainty ĥ(x_t, a)                ▷ Eq. (13)
        Select action a_t                             ▷ Eq. (14)
        Extend tree T_{t+1} = Extend(T_t, a_t)        ▷ Eq. (2)
    end while
    Compute terminal reward r_M                       ▷ Eq. (3)
    Store transitions in B, sample minibatch
    Update networks by minimizing L                   ▷ Eq. (15)
    Soft update target networks
    if r_M > 0 then
        F ← F ∪ {α_new}
    end if
end for
return Alpha pool F
```

where w_k is a weight corresponding to the interval between adjacent quantiles. We adopt an upper confidence bound strategy for action selection:

$$a_t = \arg\max_{a \in \mathcal{A}_{\text{valid}}} \Big\{ \hat{Q}(x_t, a) \; + \; \lambda \sqrt{\hat{h}(x_t, a)} \Big\}, \tag{14}$$

where $\mathcal{A}_{\text{valid}}$ is the set of syntactically valid actions and λ is the exploration coefficient.

3) Training Procedure: We train the network using experiences from a replay buffer. The quantile network is updated via quantile regression loss $\mathcal{L}_\theta$, and the Q-network via temporal difference loss $\mathcal{L}_Q$ [28]. The total loss is:

$$\mathcal{L} = \mathcal{L}_Q + \mathcal{L}_\theta. \tag{15}$$

The complete training procedure of AlphaTree is summarized in Algorithm 1.

IV. Experiments

A. Experimental Setup

1) Datasets: We evaluate our method on two widely used benchmark datasets from the Chinese A-share market. The CSI300 dataset comprises the largest 300 stocks by market capitalization, representing large-cap companies with high liquidity. The CSI500 dataset includes the 500 largest stocks, excluding those in the CSI300, and covers mid-cap companies with potentially higher volatility. For both datasets, we use daily Open-High-Low-Close-Volume (OHLCV) data and adopt a chronological split to prevent look-ahead bias: the training

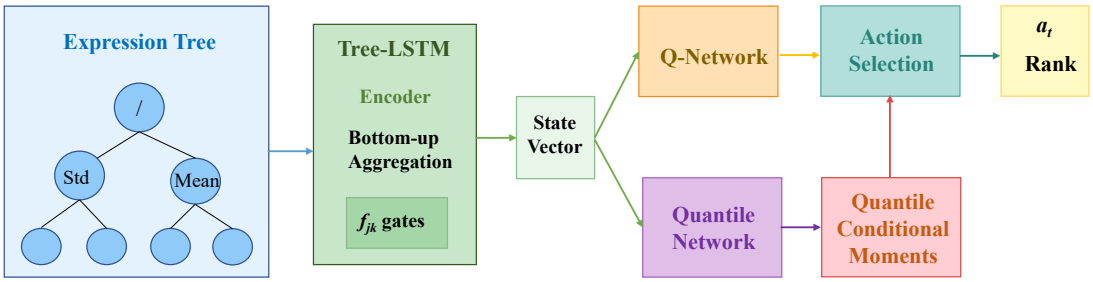


Fig. 3: Illustration of AlphaTree's action selection mechanism. The Tree-LSTM encodes the current expression tree into a state vector, which is fed into a Q-network and a quantile network. The Q-network outputs expected returns for actions, while the quantile network estimates return distributions. The agent selects the action with the highest upper-confidence-bound value, balancing exploitation and exploration through an uncertainty-based bonus.

period spans from January 1, 2010, to December 31, 2020; the validation period covers January 1, 2021, to December 31, 2021; and the test period extends from January 1, 2022, to December 31, 2023.

2) Baseline Methods: We compare AlphaTree against representative methods from four categories:

- *Traditional Machine Learning:* We include MLP, LightGBM [29], and XGBoost [30] as non-formulaic baselines that directly predict returns from raw features.
- *Genetic Programming:* We compare with GP [6], which evolve expression trees via mutation and crossover operations.
- *Generative Models:* We compare with AlphaForge [18], which couples generative factor proposals with dynamic recombination for diverse alpha portfolios.
- *Reinforcement Learning:* We include AlphaGen [11], which pioneered PPO-based token-by-token alpha generation, and AlphaQCM [12], which addresses non-stationarity via distributional RL with QCM.

For all baseline methods, we adopt the hyperparameter settings as specified in their original papers and thus omit detailed descriptions here.

3) Evaluation Metrics: We employ a rolling window backtesting framework. On each trading day, alpha factors are calculated based on the preceding 100 trading days, and their predictive performance is evaluated against the subsequent 20-day forward returns. Following [19], we employ seven metrics covering both predictive power and portfolio performance:

- **Information Coefficient (IC):** The Pearson correlation between alpha values and future returns across stocks. IC measures the linear predictive power of the alpha factor.
- **IC Information Ratio (ICIR):** The ratio of mean IC to its standard deviation. ICIR quantifies the stability and reliability of the alpha's predictive signal over time.
- **Rank Information Coefficient (RankIC):** The Spearman rank correlation between alpha values and future returns. RankIC is robust to outliers and captures monotonic relationships.
- **Rank IC Information Ratio (RankICIR):** The ratio of mean RankIC to its standard deviation, measuring the consistency of rank-based predictions.
- **Annualized Return (AR):** The annualized portfolio return achieved by trading based on the alpha signal, reflecting the practical profitability of the discovered factors.
- **Maximum Drawdown (MDD):** The largest peak-to-trough decline in portfolio value during the evaluation period. We report the negative of MDD so that higher values indicate smaller drawdowns.
- **Sharpe Ratio (SR):** The risk-adjusted return computed as the ratio of excess return to portfolio volatility, measuring return per unit of risk.

4) Implementation Details: All experiments were conducted on a machine equipped with an Intel® Core™ i9-14900K CPU and an NVIDIA RTX 4090D GPU. Each experiment is repeated with 10 different random seeds, and we report the mean of all metrics. Our network architecture adheres to the design principles established in Yu *et al.* [11] to ensure a fair comparison. Both the Q-network and quantile network use fully connected heads, and we adopt the hyperparameter configuration from [12] for the distributional RL components. The detailed hyperparameter settings are summarized in Table I.

TABLE I: Hyperparameter settings.

Hyperparameter	Value
Tree-LSTM layers	2
Tree-LSTM hidden dimension	128
Tree-LSTM dropout rate	0.1
Q-network hidden layers	2×64
Quantile embedding dimension	64
Number of quantiles K	64
Huber loss threshold κ	1.0
Optimizer	Adam
Learning rate	5×10^{-5}
Batch size	128
Replay buffer capacity	100,000
Exploration coefficient λ	1.0
Alpha pool size P	20
Total training steps N	3×10^5

TABLE II: Experimental results on CSI300 dataset (test period: 2022-01-01 to 2023-12-31). Best results are in **bold**, second best are underlined. All metrics are better when higher.

Method	Correlation Metrics				Portfolio Metrics		
	IC	ICIR	RankIC	RankICIR	AR	MDD	SR
LightGBM	0.0239	0.2453	0.0022	0.0203	2.58%	-22.37%	0.5849
XGBoost	0.0342	0.3377	0.0200	0.1806	3.11%	-19.69%	1.9291
MLP	0.0330	0.3220	0.0161	0.1490	5.26%	-15.85%	1.7601
GP	0.0611	0.2724	0.0362	0.2066	6.53%	-11.26%	2.7810
AlphaForge	0.0490	0.3461	0.0568	0.3188	4.22%	-14.34%	1.8348
PPO	0.0614	0.3820	0.0546	0.3671	4.18%	-15.18%	0.7298
AlphaQCM	0.0766	0.4423	0.0452	0.3241	6.12%	-11.00%	2.9520
AlphaTree (Ours)	**0.0818**	**0.4599**	**0.0573**	**0.4001**	**6.77%**	**-10.13%**	**3.0912**

TABLE III: Experimental results on CSI500 dataset (test period: 2022-01-01 to 2023-12-31). Best results are in **bold**, second best are underlined. All metrics are better when higher.

Method	Correlation Metrics				Portfolio Metrics		
	IC	ICIR	RankIC	RankICIR	AR	MDD	SR
LightGBM	0.0432	0.2827	0.0522	0.3586	2.31%	-17.69%	0.4951
XGBoost	0.0317	0.3849	0.0437	0.5309	2.08%	-19.22%	0.5140
MLP	0.0412	0.2971	0.0369	0.3902	1.99%	-18.67%	0.4187
GP	0.0659	0.4795	0.0598	0.5284	2.72%	-15.89%	0.8949
AlphaForge	0.0668	0.3751	0.0682	0.6059	2.20%	-14.65%	0.8340
PPO	0.0612	0.3673	0.0547	0.4485	1.64%	-15.91%	0.5508
AlphaQCM	0.0796	0.4804	0.0838	0.6700	4.24%	-13.03%	1.4529
AlphaTree (Ours)	**0.0841**	**0.6169**	**0.0917**	**0.7171**	**5.12%**	**-11.83%**	**1.4882**

B. Main Results

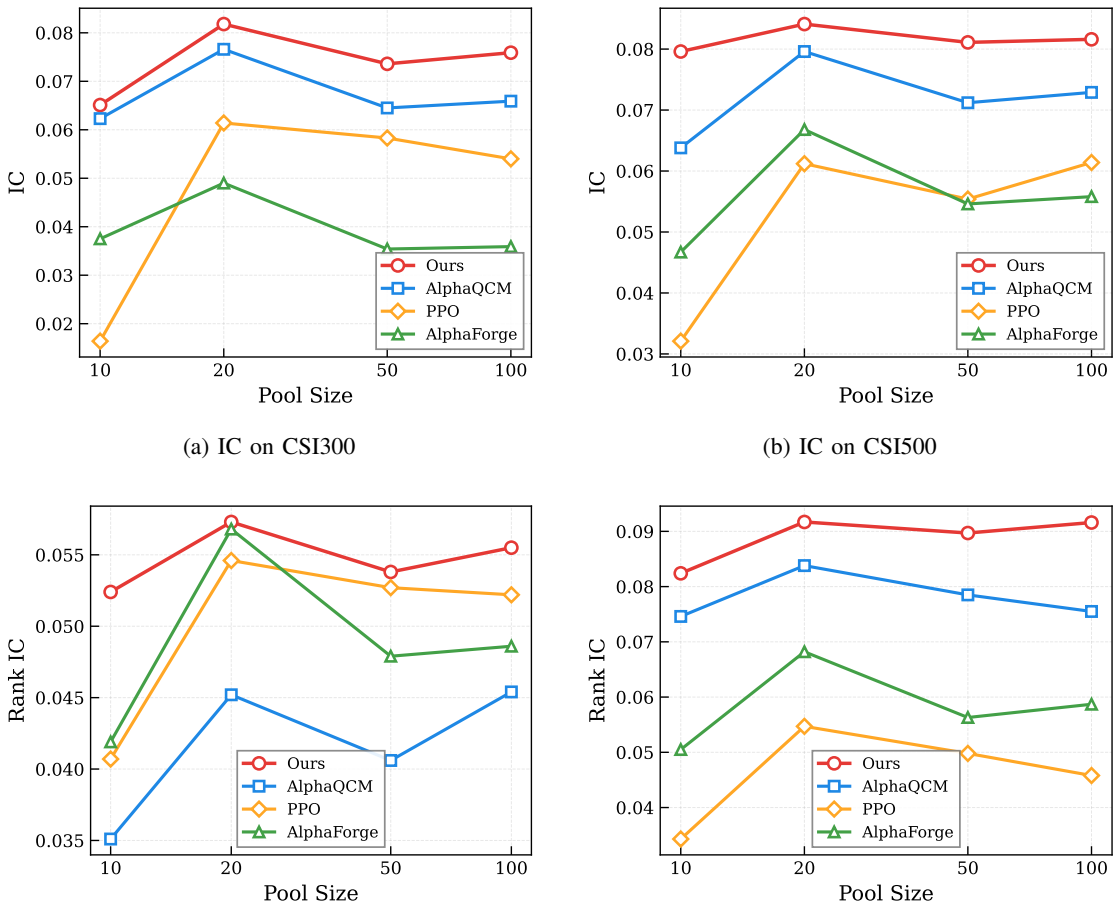
Tables II and III summarize the performance of AlphaTree and baseline methods on the CSI300 and CSI500 test sets, respectively. Overall, AlphaTree outperforms all baselines across every metric on both datasets.

Overall Performance. On CSI300, AlphaTree achieves an IC of 0.0818, which is about 6.8% higher than the next best method (AlphaQCM, IC = 0.0766). On CSI500, AlphaTree's IC is 0.0841 versus 0.0796 for AlphaQCM, a 5.7% improvement. More importantly, AlphaTree delivers substantially more consistent performance: for example, on CSI500 our method's ICIR is 0.6169, a 28.4% increase over AlphaQCM's 0.4804. In terms of portfolio outcomes, AlphaTree attains the highest Sharpe ratio on both CSI300 (3.0912) and CSI500 (1.4882), while also achieving the smallest drawdowns (peak-to-trough declines of about 10.13% and 11.83%, respectively). These results demonstrate superior risk-adjusted returns.

Comparison with Traditional Machine Learning. MLP, LightGBM, and XGBoost exhibit the weakest predictive power. On CSI300, their IC values (0.0239–0.0342) are substantially lower than AlphaTree's 0.0818. Their portfolio performance is also poor, with Sharpe Ratios below 2.0 and maximum drawdowns exceeding 15%. While these methods capture nonlinear patterns, they produce black-box predictions lacking interpretability. AlphaTree generates transparent mathematical expressions suitable for practical deployment.

Comparison with Genetic Programming. GP represents the classical method for formulaic alpha discovery through evolutionary search. GP achieves reasonable IC values of 0.0611 on CSI300 and 0.0659 on CSI500, demonstrating its ability to discover meaningful mathematical expressions. However, the ICIR values reveal a critical weakness: GP's 0.2724 on CSI300 is only 59% of AlphaTree's 0.4599, indicating highly inconsistent predictions over time. This temporal instability stems from the stochastic nature of genetic operators (mutation and crossover), which lack the gradient-guided directionality that RL provides. While GP achieves competitive annualized returns (6.53% on CSI300), its lower Sharpe Ratio (2.7810 vs. 3.0912) and higher maximum drawdown (11.26% vs. 10.13%) suggest that the discovered alphas are less robust during adverse market conditions. These results highlight the advantage of our learning-based method, which systematically explores the expression space while maintaining prediction stability.

Comparison with Reinforcement Learning Methods. Among RL methods, the progression from PPO to AlphaQCM to AlphaTree reveals the cumulative benefits of methodological improvements. PPO, which pioneered token-by-token alpha generation, achieves IC values of 0.0614 and 0.0612 on CSI300 and CSI500, respectively. However, it suffers from poor portfolio performance with Sharpe Ratios of only 0.7298 and 0.5508, likely due to its inability to handle the non-stationary reward dynamics inherent in alpha discovery. AlphaQCM addresses this limitation through distributional RL with QCM-based uncertainty estimation, substantially improving both



(a) IC on CSI300

(b) IC on CSI500

(c) RankIC on CSI300

(d) RankIC on CSI500

Fig. 4: Performance comparison under different pool sizes.

predictive metrics (IC of 0.0766 and 0.0796) and portfolio returns (Sharpe Ratios of 2.9520 and 1.4529). AlphaTree builds upon this foundation by incorporating structure-aware encoding via Tree-LSTM. The ICIR improvement from PPO (0.3820) to AlphaQCM (0.4423) to AlphaTree (0.4599) on CSI300 demonstrates that each architectural enhancement contributes to more stable alpha predictions. On CSI500, AlphaTree's ICIR of 0.6169 represents a 67.9% improvement over PPO's 0.3673, underscoring the importance of structural representation in complex markets.

Comparison with Generative Models. AlphaForge, which couples generative factor proposals with dynamic recombination, represents a recent method emphasizing alpha diversity. It achieves competitive RankIC values (0.0568 on CSI300 and 0.0682 on CSI500), indicating reasonable robustness to outliers. However, AlphaForge lags in other critical metrics. On CSI300, its IC of 0.0490 is only 59.9% of AlphaTree's 0.0818, and its Sharpe Ratio of 1.8348 is less than 60% of ours (3.0912). The annualized returns also differ substantially: 4.22% for AlphaForge versus 6.77% for AlphaTree. On CSI500, while AlphaForge maintains a lower maximum drawdown (14.65%) than most baselines, its Sharpe Ratio of 0.8340 remains far below AlphaTree's 1.4882. These results suggest that while generative diversity is valuable for avoiding alpha decay, it must be combined with effective state representation to exploit the compositional structure of mathematical expressions fully. AlphaTree's tree-aware encoding provides precisely this capability, enabling both diverse exploration and high-quality alpha generation within a unified framework.

C. Effect of the Pool Size

We also investigate how the size of the alpha pool P (number of alphas in the portfolio) affects performance. In this analysis, we vary $P \in \{10, 20, 50, 100\}$ and evaluate AlphaTree against PPO, AlphaQCM, and AlphaForge for each setting. The IC and RankIC results on the test set are plotted in Fig. 4.

Across all pool sizes, AlphaTree maintains a clear performance advantage over the baselines on both CSI300 and CSI500. Notably, the relationship between pool size and performance is not monotonic. For AlphaTree (and other methods to a lesser extent), performance peaks around $P = 20$ and then slightly declines as P increases further. For example, on CSI300, AlphaTree's IC is highest at $P = 20$ (0.0818) and drops off at $P = 50$ and $P = 100$. A similar trend is

observed for RankIC. We attribute this behavior to a trade-off between diversity and signal quality: a larger pool can capture more diverse patterns, but it may also introduce weaker or redundant alphas that dilute the overall signal. In our experiments, a pool size of around 20 achieves a good balance, providing enough diversity without too much noise. Importantly, AlphaTree's structured approach appears robust for different P; even with $P = 100$, it still outperforms the baselines by a significant margin. This robustness suggests that the structure-aware encoding helps the agent find a set of alphas that collectively perform well, regardless of portfolio size.

V. Conclusion

In this paper, we proposed AlphaTree, a structure-aware method for formulaic alpha discovery that leverages the inherent tree structure of mathematical expressions. By replacing sequential LSTM with Tree-LSTM as the feature extractor, our method achieves structure-aware state encoding that respects the compositional semantics of alpha formulas. The integration with distributional RL enables effective handling of non-stationarity and reward-sparsity challenges in the alpha discovery MDP. Comprehensive experiments on the CSI300 and CSI500 datasets demonstrate that AlphaTree significantly outperforms state-of-the-art methods across all evaluation metrics. The improvements are particularly pronounced on the larger CSI500 dataset, suggesting that structure-aware representation learning becomes increasingly beneficial as market complexity grows.

Future work could explore more sophisticated tree architectures, such as Graph Neural Networks, for handling shared subexpressions, and extend the method to other domains that require mathematical expression generation.

References

[1] M. Altman, *Handbook of contemporary behavioral economics: foundations and developments*. Routledge, 2015.

[2] I. Tulchinsky, *Finding Alphas: A quantitative approach to building trading strategies*. John Wiley & Sons, 2019.

[3] F. Feng, X. He, X. Wang, C. Luo, Y. Liu, and T.-S. Chua, "Temporal relational ranking for stock prediction," *ACM Transactions on Information Systems (TOIS)*, vol. 37, no. 2, pp. 1–30, 2019.

[4] Q. Ding, S. Wu, H. Sun, J. Guo, and J. Guo, "Hierarchical multi-scale gaussian transformer for stock movement prediction." in *International Joint Conference on Artificial Intelligence*, 2020, pp. 4640–4646.

[5] K. J. Koa, Y. Ma, R. Ng, and T.-S. Chua, "Diffusion variational autoencoder for tackling stochasticity in multi-step regression stock price prediction," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 2023, pp. 1087–1096.

[6] C. Cui, W. Wang, M. Zhang, G. Chen, Z. Luo, and B. C. Ooi, "Alphaevolve: A learning framework to discover novel alphas in quantitative investment," in *Proceedings of the 2021 International conference on management of data*, 2021, pp. 2208–2216.

[7] J. Zhao, C. Zhang, C. Wang, and P. Yang, "Learning from expert factors: Trajectory-level reward shaping for formulaic alpha mining," *arXiv preprint arXiv:2507.20263*, 2025.

[8] Z. Wang, B. Huang, S. Tu, K. Zhang, and L. Xu, "Deeptrader: a deep reinforcement learning approach for risk-return balanced portfolio management with market conditions embedding," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 1, 2021, pp. 643–650.

[9] T. Ren, R. Zhou, J. Jiang, J. Liang, Q. Wang, and Y. Peng, "Riskminer: Discovering formulaic alphas via risk seeking monte carlo tree search," in *Proceedings of the 5th ACM International Conference on AI in Finance*, 2024, pp. 752–760.

[10] S. Chen, R. Xu, C. Wang, M. Ma, and Z. Miao, "Rl-based explainable factor generation for market risk analysis," in *International Conference on Neural Information Processing*. Springer, 2025, pp. 396–409.

[11] S. Yu, H. Xue, X. Ao, F. Pan, J. He, D. Tu, and Q. He, "Generating synergistic formulaic alpha collections via reinforcement learning," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 5476–5486.

[12] Z. Zhu and K. Zhu, "Alphaqcm: Alpha discovery in finance with distributional reinforcement learning," in *International Conference on Machine Learning*, 2025.

[13] H. Ding, B. Chen, J. Huang, T. Guo, Z. Mao, G. Shao, L. Zou, L. Liu, and M. Zhang, "Alphaeval: A comprehensive and efficient evaluation framework for formula alpha mining," *arXiv preprint arXiv:2508.13174*, 2025.

[14] Q. Zhang, Y. Zhang, X. Yao, S. Li, C. Zhang, and P. Liu, "A dynamic attributes-driven graph attention network modeling on behavioral finance for stock prediction," *ACM Transactions on Knowledge Discovery from Data*, vol. 18, no. 1, pp. 1–29, 2023.

[15] Y. Zhang, P. Zhao, Q. Wu, B. Li, J. Huang, and M. Tan, "Cost-sensitive portfolio selection via deep reinforcement learning," *IEEE Transactions on knowledge and data engineering*, vol. 34, no. 1, pp. 236–248, 2020.

[16] T. Zhang, Y. Li, Y. Jin, and J. Li, "Autoalpha: an efficient hierarchical evolutionary algorithm for mining alpha factors in quantitative investment," *arXiv preprint arXiv:2002.08245*, 2020.

[17] B. K. Petersen, M. Landajuela, T. N. Mundhenk, C. P. Santiago, S. K. Kim, and J. T. Kim, "Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients," *arXiv preprint arXiv:1912.04871*, 2019.

[18] H. Shi, W. Song, X. Zhang, J. Shi, C. Luo, X. Ao, H. Arian, and L. A. Seco, "Alphaforge: A framework to mine and dynamically combine formulaic alpha factors," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 12, 2025, pp. 12 524–12 532.

[19] B. Chen, H. Ding, N. Shen, J. Huang, T. Guo, L. Liu, and M. Zhang, "Alphasage: Structure-aware alpha mining via gflownets for robust exploration," *arXiv preprint arXiv:2509.25055*, 2025.

[20] W. Xu, R. Wang, C. Li, Y. Hu, and Z. Lu, "Hrft: Mining high-frequency risk factor collections end-to-end via transformer," in *Companion Proceedings of the ACM on Web Conference 2025*, 2025, pp. 538–547.

[21] Q. Chen and H. Kawashima, "Adaptive alpha weighting with ppo: Enhancing prompt-based llm-generated alphas in quant trading," *arXiv preprint arXiv:2509.01393*, 2025.

[22] L. Cao, "Chain-of-alpha: unleashing the power of large language models for alpha mining in quantitative trading," *arXiv preprint arXiv:2508.06312*, 2025.

[23] Z. Tang, Z. Chen, J. Yang, J. Mai, Y. Zheng, K. Wang, J. Chen, and L. Lin, "Alphaagent: Llm-driven alpha mining with regularized exploration to counteract alpha decay," in *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, 2025, pp. 2813–2822.

[24] Y. Shi, Y. Duan, and J. Li, "Navigating the alpha jungle: An llm-powered mcts framework for formulaic factor mining," *arXiv preprint arXiv:2505.11122*, 2025.

[25] J. Zhao, C. Zhang, M. Qin, and P. Yang, "Quantfactor reinforce: Mining steady formulaic alpha factors with variance-bounded reinforce," *IEEE Transactions on Signal Processing*, vol. 73, pp. 2448–2463, 2025.

[26] F. Xu, Y. Yin, X. Zhang, T. Liu, S. Jiang, and Z. Zhang, "TextAlpha^2: Discovering logical formulaic alphas using deep reinforcement learning," *arXiv preprint arXiv:2406.16505*, 2024.

[27] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," in *International conference on machine learning*. PMLR, 2018, pp. 1096–1105.

[28] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[29] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[30] T. Chen, "Xgboost: A scalable tree boosting system," *Cornell University*, 2016.