From Flat to Hierarchical: Evolving Tree-structured Thoughts for Fine-grained Alpha Mining

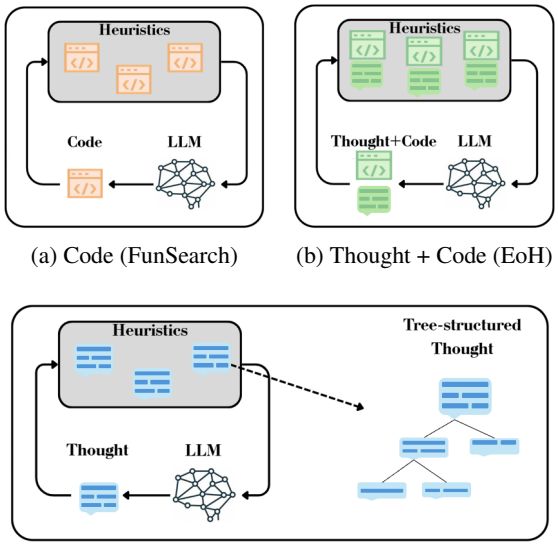
Anonymous ACL submission

Abstract

Alpha mining, aimed at discovering predictive return signals, is typically formulated as symbolic regression. Traditional symbolic methods suffer from search inefficiency and biased prior knowledge. Recently, Large Language Models (LLMs) have emerged as a promising alternative, automatically generating textual thoughts and executable codes to achieve both efficient and interpretable alpha mining. However, existing approaches mostly focus on leveraging LLM's reasoning and reflection capabilities, yet largely neglect the positional bias due to the flat thought representation which restricts efficiency and diversity of the search process. This paper introduces Tree-structured thought Evolution (TreEvo), which evolves hierarchically decomposed thoughts to expand the effective search space. In addition, we propose a set of evolutionary operators tailored to structured thoughts. Experiments on four real-market datasets demonstrate that TreEvo not only obtains competitive alphas with traditional methods in up to 200 times fewer evaluations, but also consistently outperforms LLM-driven EAs across all datasets by 14.31% on average.

1 Introduction

Alpha mining, an important endeavor within financial quantitative trading, is to identify measurable signals that predict asset returns. The core challenge of alpha mining lies in isolating truly predictive factors from noisy, non-stationary data efficiently while ensuring interpretability (Giglio et al., 2022; Belle and Papantonis, 2021). Existing literature primarily tackles this challenge through two distinct computational paradigms. Under the sub-symbolic paradigm, neural networks excel at automatic feature extraction and nonlinear modeling in alpha mining(Abe and Nakayama, 2018; Chen et al., 2024; Jiang, 2021), yet their "black-box" nature severely limits interpretability and restricts widespread financial adoption. Conversely, the symbolic paradigm remains predominant because it generates interpretable factors for rigorous strategy validation and risk management(Černevičienė and Kabašinskas, 2024), though tree-based methods like Genetic Programming (GP)(Zhang et al., 2020) and Reinforcement Learning (RL)(Yu et al., 2023) suffer from an exponentially large search space driven by combinatorial explosion(Vie et al., 2020).



(a) Code (FunSearch) (b) Thought + Code (EoH)

(c) Tree-structured Thought (TreEvo, our work)

Figure 1: Different forms of heuristics in LLM-driven EAs.

Recent advances have explored the integration of Large Language Models (LLMs) with Evolutionary Algorithms (EAs)(Romera-Paredes et al., 2024), which work by iteratively prompting LLMs for evolving the executable codes. Compared to traditional symbolic methods, LLMs provide full expressiveness, under which the search space is no longer limited to manually predefined symbols(Merler et al., 2024). Besides, the massive prior knowledge encoded in LLMs allow for the generation of diverse and context-relevant code

structures(Jiang et al., 2025). Consequently, this leads to faster convergence in possible solutions. More recently, research has expanded to jointly evolve both the underlying thoughts and the generated codes(Liu et al., 2024a). The shift from code-only to thought-and-code evolution enables more cognitively inspired and effective LLM-driven program synthesis.

Such a framework has been explored to foster a more interpretable and efficient search process of symbolic approaches in alpha mining(Li et al., 2024; Shi et al., 2025b). However, existing methods mostly focus on the enhancement of LLM-based evolutionary operators and trajectories, overlooking the underlying conceptual structure of thoughts. Current LLM-driven methods rely on flat-text thoughts, which constrain the capacity to disentangle logical dependencies. The limitation leads to positional bias alongside redundant refinements, inducing search stagnation and premature convergence. In this context, a transition toward hierarchical thought structures holds potential to alleviate the inherent constraints of linear text.

In this paper, we propose Tree-structured thought Evolution (TreEvo), a novel paradigm where the evolution process is performed on thoughts represented as hierarchical tree structures. Each sub-tree encapsulates a sub-component of reasoning, and higher-level nodes form increasingly abstract heuristics. This structural clarity reshapes the effective topology of the search space, transforming an amorphous, high-dimensional flat-text manifold into a more navigable logical landscape that more closely reflects the recursive structure underlying symbolic discovery. We also propose the tailored evolutionary operators for evolving the hierarchical tree-structure. Empirical comparisons with six representative traditional methods and two LLM-driven EAs have been conducted on stock pools across both Chinese and U.S. markets. TreEvo shows significant advantages over traditional approaches in search efficiency, achieving competitive results with much fewer evaluations. The clear superiority of the proposed tree-structure and evolutionary operators are also extensively analyzed by comparing with two representative LLM-driven EAs and the ablation variants.

In summary, our contributions are as follows:

- We propose the tree-structured thought, which explicitly organize the reasoning path of an idea in a hierarchical manner to mitigate positional bias in flat text.
- We propose TreEvo for alpha mining, a novel paradigm where evolution operates at the tree-structured thoughts with the evolutionary operators for fine-grained search.
- We demonstrate that TreEvo not only achieves competitive performance with traditional methods costing up to 200 times fewer evaluations, but also consistently outperforms LLM-driven EAs on four diverse real-market datasets.

2 Background and Related Works

2.1 Alpha Mining

Alpha mining refers to the automated discovery of predictive signals (alphas) for future asset returns. Early approaches relied on handcrafted designs (Fama and French, 1993; Grinold and Kahn, 2000). While recent neural network methods capture complex nonlinear and temporal patterns (Heaton et al., 2017; Kim and Kang, 2019; Qian et al., 2024), they often struggle with interpretability and overfitting. Symbolic approaches, using techniques like GP and RL (Ren et al., 2024b,a; Zhao et al., 2025; Crochepierre et al., 2022), generate transparent, modular alpha formulas. Current research focuses on robustness, regime adaptation, and balancing predictive power with interpretability (Gu et al., 2020; Yeo et al., 2025), positioning symbolic paradigms as principled alternatives to black-box models (Wang et al., 2023).

2.2 LLM-driven EAs

The emergence of LLMs has provided novel opportunities to enhance EAs through their advanced capabilities in natural language understanding, symbolic reasoning, and generative modeling (Zhang et al., 2023; Romera-Paredes et al., 2024). Incorporating implicit knowledge acquired during pretraining, LLMs can support solution generation, guide search trajectories, and enable context-aware evolutionary operators (Hao et al., 2024; Huang et al., 2024; Tian et al., 2025; Ye et al., 2024). Recent studies further explore LLM-driven representation learning, program synthesis, and automated operator design to improve search efficiency and reduce reliance on handcrafted heuristics (Liu et al., 2024b; Lange et al., 2024). Despite challenges related to computational cost, robustness, and interpretability, LLM-augmented EAs represent a

rapidly growing direction with strong potential for advancing metaheuristic optimization.

2.3 Hierarchical Text

Hierarchical text modeling has evolved from simple sequential processing toward architectures that explicitly capture nested structural dependencies in language. Early studies showed that multi-level representations improve semantic modeling and alleviate semantic dilution in flat architectures (Yang et al., 2016). Subsequent work demonstrated that hierarchical structures enhance long-document representation and content aggregation beyond the capacity of standard models (Cohan et al., 2018; Zhang et al., 2019). More recent research highlights their role in recursive reasoning, knowledge navigation, and interpretability in complex information settings (He et al., 2025; Huang et al., 2025). However, existing studies primarily focus on hierarchical text classification, whereas comparatively limited attention has been given to treating hierarchical text itself as a structured search space for LLMs.

3 Method

3.1 Main Idea

TreEvo encodes each candidate alpha represented as a tree $\mathcal{T} = (V, E)$, where V is the set of reasoning units (e.g., logical steps), and $E \subseteq V \times V$ defines the hierarchical dependencies between them. Each node $v_i \in V$, corresponds to a distinct semantic or computational sub-component, and the root node v_0 encodes the expression of the overall alpha factor. The hierarchy reflects the structural organization of human cognition, which is widely understood to proceed from abstract, global concepts toward more concrete, fine-grained elaborations.

The hierarchical structure induces a structured parameterization of the search space $\mathcal{S} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots\}$ by mapping candidate alphas onto a compositional tree topology $\mathcal{T}$, where structural variation is formally expressed as a set of transformation operators $\Phi\colon \mathcal{T} \to \mathcal{T}'$ over sub-trees rather than modifications to a monolithic sequence. This topology partitions the search space into weakly coupled regions $\mathcal{S} = \bigcup_k \mathcal{S}_k$ associated with different functional roles, which constrains how changes propagate across the representation. As a result, semantic dependencies are mediated through explicit structural interfaces instead of implicit global entanglement. The search process is therefore organized around operations on bounded structural units with controllable scopes, forming a landscape $\mathcal{L} = (\mathcal{S}, \Phi)$ in which alternative configurations arise through systematic recombination and rearrangement of these units.

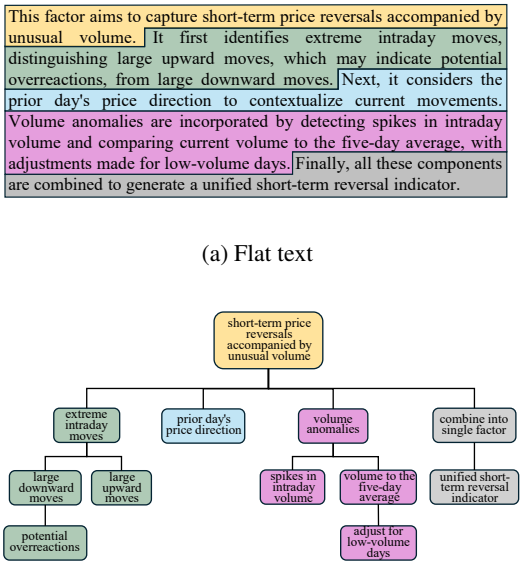


Figure 2: An example of a set of paired flat text and tree-structured text.

3.2 Tree-structured Thought Representation

Figure 2 shows an example of a flat text and its corresponding hierarchical tree representation used to organize complex concepts or computations. The sentences with the same color in 2a and 2b correspond to each other. At the highest level of the tree-structured text, a root node represents the overall entity or idea, which is decomposed into several child nodes capturing its main reasoning steps. Each child node can further branch into its own children, reflecting more finer-grained operations that collectively define the meaning of their parent node. This layer-wise decomposition continues recursively until the leaf nodes that represent the most basic, indivisible units in the structure. The edges between nodes indicate the compositional relationships, where parent nodes depend on or aggregate information from their children. The hierarchical levels correspond to different degrees of abstraction, with higher levels summarizing or coordinating the details specified at lower levels.

In TreEvo, the entire tree-structured thought is generated by LLM as a holistic structured entity, with hierarchical information explicitly encoded

using structured symbolic delimiters. Each node and its children are separated by these symbols, establishing clear boundaries that distinguish reasoning units at different levels of abstraction. This allows the model to produce a complete hierarchical representation in one step, while preserving the modularity and compositionality of its constituent sub-trees. By maintaining these explicit structural cues, the generated thought provides a coherent framework that can later be systematically explored and modified, supporting principled evolution of alternative tree configurations.

On this basis, TreEvo is significantly different from the well-known Chain-of-Thought(CoT) (Wei et al., 2022) and Tree-of-Thought(ToT) (Yao et al., 2023). CoT treats a thought as a sequence of reasoning steps. ToT also organizes a thought as a sequence of reasoning steps, and integrates multiple related thoughts as a tree. That is, sibling reasoning steps under the same sub-tree typically share the same reasoning prefix. In contrast, TreEvo regards a whole tree as a thought. Consequently, the generation of the thoughts are also different. In CoT, the whole thought is generated by constructively appending reasoning steps. In ToT, the generation of a thought is to find the optimal complete branch from the root to one leaf. In TreEvo, we need to search the space of tree-structured representations, where EA can be adopted.

3.3 Evolutionary Operators

We propose five evolutionary operators to generate new individuals in the search process. Collectively, they are designed to balance recombination, variation, and simplification within the tree-structured search space. The crossover operator primarily emphasizes recombination of coherent reasoning modules, facilitating the emergence of new structural compositions. The mutation operators focus on introducing controlled variations at different levels, supporting both global exploration and local refinement of candidate structures. The pruning operator prioritizes structural simplification, encouraging more compact and parsimonious representations while maintaining functional validity.

Crossover: Given two individuals $\mathcal{T}_1$ and $\mathcal{T}_2$, we define crossover as integrating sub-trees:

$$\mathcal{T}' = \mathcal{T}_1 \oplus_{v_i, v_j} \mathcal{T}_2$$

where $v_i \in \mathcal{T}_1$, $v_j \in \mathcal{T}_2$. $\oplus$ denotes the integration of v_i and v_j. This operator preserves semantic validity and allows meaningful recombination of reasoning units.

Mutation-R: It generates an individual by generating a completely different tree $\widetilde{\mathcal{T}}$ from current population $\{\mathcal{T}_1, ..., \mathcal{T}_n\}$:

$$\mathcal{T}' = \widetilde{\mathcal{T}}$$

This ensures the diversity of individuals in the population.

Mutation-I: It operates $\mathcal{T}$ by selecting an internal node v_k and replacing its sub-tree with a newly generated, syntactically valid and semantically plausible tree $\widetilde{\mathcal{T}}$:

$$\mathcal{T}' = \mathcal{T} \ominus_{v_k} \widetilde{\mathcal{T}}$$

where $\ominus$ denotes the sub-tree or node replacement rooted at node v_k. This allows localized alteration of logical steps while preserving the overall structure and intent of the thought.

Mutation-F: It operates $\mathcal{T}$ by selecting a leaf node v_k and replacing it with a newly generated, syntactically valid and semantically plausible node v':

$$\mathcal{T}' = \mathcal{T} \ominus_{v_k} v'$$

This allows refinements on parameter only while preserving the overall structure and intent of the thought.

Pruning: It operates by identifying and removing structurally redundant or semantically neutral sub-trees from $\mathcal{T}$. Given the sub-tree of node v_k, if it satisfies the redundancy criteria, it is replaced with a simpler expression $\widehat{\mathcal{T}}$:

$$\mathcal{T}' = \mathcal{T} \ominus_{v_k} \widehat{\mathcal{T}}$$

This facilitates structural simplification and encourages parsimony in the thought representation.

All evolutionary operators are implemented at the prompt level with explicit constraints that preserve the logical and semantic coherence of individuals. During each operation, LLM is guided to maintain structural consistency and respect the hierarchical dependencies within the tree representation. As a result, newly generated individuals remain semantically valid and logically self-consistent, ensuring that evolutionary modifications operate on well-formed reasoning structures rather than arbitrary textual variations.

3.4 Framework

The evolution process maintains a population of N candidate alphas, which are represented as tree-structured thoughts. The population are iteratively

evolved with LLM. At each iteration, given N parent thoughts, N new offspring thoughts are generated by LLM with one of three operator types: crossover operator, mutation operators, or pruning operator. To evaluate those N new thoughts, LLM is first applied to generate one piece of executable code based on each new thought, then each code is run with the given market dataset to calculate the predictive performance. After that, a new population of N candidate thoughts are selected among the N parents and N offspring, based on their individual evaluated performance. More specific steps are described in Appendix G.

4 Experiments

The experimental studies focus on three key research questions (RQs):

- **RQ1:** How tree-structure contributes to LLM-driven EAs for fine-grained search process?
- **RQ2:** How does our proposed method compared to existing methods on alpha mining?
- **RQ3:** Can the idea of tree-structure also be applied to existing LLM-driven EAs? How does the evolutionary operators contribute?

4.1 Experimental Settings

4.1.1 Dataset

Four widely used stock pools that serve as benchmarks are considered, encompassing both Chinese A-shares market (CSI300, CSI500) and U.S. stock market (SPX, NDX). Considering about the reproducibility, we chose six widely used raw data features, opening, closing, highest, lowest prices (OHLC), trading volume (volume), and volume-weighted average price (vwap), for the whole experiments. All price and volume data are forward-adjusted for corporate actions to ensure consistency over time. The dataset is split chronologically into training set (2016/01/01-2020/01/01), validation set (2020/01/01-2021/01/01) and test set (2021/01/01-2024/01/01).

4.1.2 Baselines

The framework is compared with eight baseline methods. Six of them are traditional approaches: **XGBoost** (Chen and Guestrin, 2016), **LightGBM** (Ke et al., 2017), **GP** (Zhang et al., 2020), **AlphaGen** (Yu et al., 2023), **AlphaForge** (Shi et al., 2025a) and **QFR** (Zhao et al., 2025). The remaining two are LLM-driven EAs for code generation: **EoH** (Liu et al., 2024a) and **ReEvo** (Ye et al., 2024). 22 arithmetic operators are used for combing the above six data features in traditional approaches (see Appendix C), while no restriction for the operators of LLM-driven EAs given the embedded prior knowledge.

For traditional approaches, we adopt the evaluation budgets specified in their original implementations. The hyperparameters are consistent with their original papers. Specifically, the hyperparameter settings of XGBoost and LightGBM follow those provided in Qlib(Yang et al., 2020a), which consume 1000 evaluations. GP adopts the default hyperparameter configuration in gplearn, which consumes 40000 evaluations. AlphaGen, AlphaForge and QFR follow the configurations reported in their original papers, which consume at least 50000 evaluations. For LLM-driven EAs, we set the evaluation budget for obtaining the best alpha to a small number of 200 evaluations. Models utilize the Qwen3-Max model (Yang et al., 2025) to generate both thoughts and codes of candidate solutions, and the population size N is fixed to 10. For TreEvo, p_R and p_I is set to 0.4, p_F is set to 0.2. The performance of an algorithm on each testing dataset is calculated by averaging over 5 independent runs of the above process, to ensure fairness and consistency.

4.1.3 Evaluation Metrics

To measure the predictive performance of each algorithm, we employ two metrics commonly used across the quantitative finance industries. IC measures the cross-sectional Pearson correlation between factor values $F \in \mathbf{R}^{n\times T}$ and the corresponding future 5-day returns $R \in \mathbf{R}^{n\times T}$ over T trading days and n stocks as

$$IC(F,R) = \frac{1}{T}\sum_{t=1}^{T} corr(f_t, r_t) \tag{1}$$

$$= \frac{1}{T}\sum_{t=1}^{T} \frac{\sum_{i=1}^{n}(f_{i,t}-\overline{f}_t)(r_{i,t}-\overline{r}_t)}{\sqrt{\sum_{i=1}^{n}(f_{i,t}-\overline{f}_t)^2 \sum_{i=1}^{n}(r_{i,t}-\overline{r}_t)^2}}. \tag{2}$$

It evaluates the predictive power of a mined alpha with values ranging from -1 to 1, and is also adopted as the training loss or search objective for the compared methods. RankIC is defined as the Spearman rank correlation between $\mathbf{F}$ and $\mathbf{R}$ as

$$RankIC(F, R) = IC(rank(F), rank(R)), \tag{3}$$

where $rank()$ assigns ranks to the elements of F or R each day across all stocks, capturing monotonic relationships and exhibiting greater robustness to outliers and scale variations.

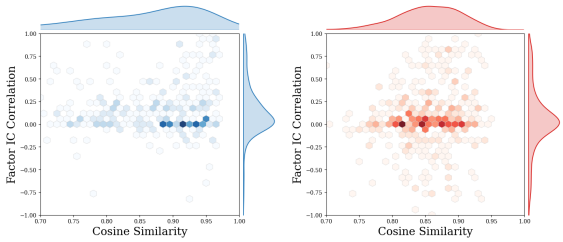
4.2 Results and Analysis

4.2.1 Comparisons of Text Representations

To illustrate the advantage of tree-structured text in enabling fine-grained search within LLM-driven evolutionary algorithms, we conducted experiments to examine how the LLM applies changes to tree-structured versus flat text. As a first experiment, we generated 10 sets of paired samples with both formats, each consisting of 10 sub-ideas, and use a simple prompt to select a single sub-idea as the variation point for generating a new individual. To eliminate order bias, half of the sets were generated starting with tree-structured text and the other half with flat text (details in Appendix D). This setup allows a direct comparison of the modification flexibility and locality afforded by each representation.

Figure 7 presents the distribution of variation points of both forms of text. The distribution of variation points indicates a notable distinction between tree-structured and flat text representations. In tree-structured text, ideas are selected relatively uniformly, suggesting that the model can exploit a wide range of positions for modification, thereby facilitating fine-grained and diverse search. In contrast, flat text exhibits an uneven distribution, with certain positions being preferentially selected while others are seldom modified, implying a bias toward localized changes and restricted exploration. Consistent with the observations, the tree-structured representation achieves a mean MSE of 0.00869 relative to the uniform distribution, whereas the flat-text representation records a higher value of 0.01829, corresponding to an approximate 52.49% reduction.

We further examine how the tree-structured representation contributes to the optimization characteristics of the search space. Specifically, we focus on the relationship between textual cosine similarity and factor IC correlation of the paired texts. For each pair, we initially utilized a prompt to randomly generate a base factor text under a given representation form. Then, we employed the identical prompt to instruct the LLM to apply a modification to this base text, thereby producing its corresponding variant (details in Appendix E).



(a) Tree-structured Text (b) Flat Text

Figure 3: Comparison between distribution of textual cosine similarity and factor IC correlation in tree-structured and flat text formats.

As shown in Figure 3, the joint distribution of textual cosine similarity and factor IC correlation illustrates distinct patterns between the two representation forms. In the flat text, the similarities are highly clustered within a narrow textual similarity range of 0.8 to 0.9, while the corresponding factor IC correlations scatter widely across the entire range from -1 to 1. In contrast, the tree-structured text exhibits a significantly broader and more uniform distribution of textual similarity, spanning from 0.7 to 0.95 with a flatter marginal density curve. Moreover, the vast majority of paired texts maintain positive factor IC correlations, showing a more stable distribution with fewer extreme negative anomalies. This indicates that the tree-structured representation effectively smooths the optimization landscape, ensuring that structural variations yield more stable and directional search.

4.2.2 Comparisons with Traditional Methods

Table 1 shows the performance comparisons of traditional approaches and TreEvo on CSI300 and CSI500. Overall, TreEvo shows comparable performance to the baselines in terms of IC, illustrating its competitiveness in capturing the magnitude of future returns. On CSI300, TreEvo's delivers an IC improvement of over 1.99% over the strongest baseline. On CSI500, TreEvo remains in the top tier, surpassing all baseline models in IC except for QFR. These results highlight the benefit of incorporating LLM guidance into the alpha mining.

Figure 4 indicates the high efficiency of TreEvo in alpha mining compared to traditional approaches. TreEvo achieves higher IC within 5 times fewer evaluations than XGBoost and LightGBM and up

Table 1: Comparisons of IC and RankIC with Traditional Approaches

Method	CSI300		CSI500	
	IC	RankIC	IC	RankIC
XGBoost	0.0197 (±0.0025)	0.0221 (±0.0013)	0.0169 (±0.0016)	0.0209 (±0.0021)
LightGBM	0.0165 (±0.0012)	0.0244 (±0.0025)	0.0172 (±0.0022)	0.0241 (±0.0017)
GP	0.0224 (±0.0017)	0.0278 (±0.0038)	0.0254 (±0.0028)	0.0297 (±0.0014)
AlphaGen	0.0259 (±0.0010)	0.0301 (±0.0046)	0.0337 (±0.0023)	**0.0405 (±0.0047)**
AlphaForge	0.0287 (±0.0014)	**0.0351 (±0.0029)**	0.0319 (±0.0022)	0.0346 (±0.0026)
QFR	**0.0302 (±0.0032)**	0.0346 (±0.0044)	**0.0374 (±0.0039)**	0.0403 (±0.0031)
TreEvo	**0.0308 (±0.0048)**	**0.0349 (±0.0057)**	**0.0362 (±0.0051)**	**0.0393 (±0.0049)**

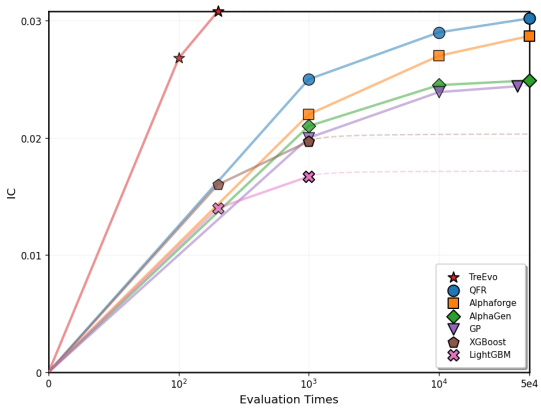


Figure 4: Comparison of search efficiency with Traditional Approaches on CSI300.

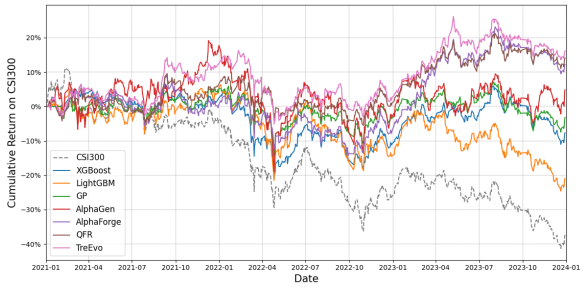


Figure 5: Cumulative rewards of traditional approaches on CSI300.

to 200 times fewer than the remaining traditional approaches. This efficiency underscores the effectiveness of LLM-driven search methods in narrowing the search space with the embedded prior knowledge and facilitating faster convergence toward high-quality solutions. The reduced resource demands, combined with improved predictive accuracy, position TreEvo as a highly practical and worth exploring framework for alpha mining. Wall-clock time and token costs are shown in Appendix F.2.

In Figure 5, we further depict the cumulative return of each method on the CSI300 index with the commonly used Top-50/Drop-5 trading strategy. The trading strategy selects the top 50 stocks based on their alpha scores with a daily rebalancing frequency. At each rebalance date, the bottom five stocks in the current portfolio are removed and replaced with five new candidates from the updated top-ranked stocks. Figure 5 demonstrates the superiority of TreEvo over the traditional approaches in terms of cumulative return under this strategy. Except for the very short initial period, TreEvo maintains the leading position throughout the backtest. On the whole, TreEvo achieves 21.17% in the cumulative return and more than 58.42% excess return by compared to the CSI300 market benchmark (i.e., the gray dash curve in the figure). Moreover, as the CSI300 index continuously goes down, the increasing curve of TreEvo demonstrates that real alphas that are less correlated to beta have been mined.

To conclude, the performance of TreEvo not only shows its superiority over representative approaches in the quantitative finance industries, but also supports that LLM-driven EAs can be revolutionary tools for alpha mining. LLM-driven EAs can automatically generate executable codes of powerful alphas in minutes rather than traditionally daily, and without much human efforts in the manufacturing loop.

4.2.3 Comparisons with LLM-driven EAs

Table 2 reports the comparative results of TreEvo against two existing LLM-driven EAs across all datasets. The results demonstrate that TreEvo generally outperforms the compared approaches, indicating that the alpha signals discovered by our method possess stronger linear associations with future returns across different markets. Overall, TreEvo outperforms the strongest baseline by an average of 14.31%. More specifically, on CSI300, SPX and NDX, the improvement made by TreEvo on IC is over 10% , which is significant. Notably, it is observed that EoH consistently beats ReEvo on all cases. This shows that evolving thoughts and codes separately is better than solely evolving the codes. Moreover, given TreEvo outperforms EoH, this table further suggests that evolving tree-

Table 2: Performance Comparisons of LLM-driven EAs

Dataset	Metric	EoH	ReEvo	TreEvo
CSI300	IC	0.0238 (±0.0028)	0.0211 (±0.0017)	**0.0308 (±0.0033)**
	RankIC	0.0262 (±0.0037)	0.0269 (±0.0028)	**0.0349 (±0.0057)**
CSI500	IC	0.0348 (±0.0036)	0.0247 (±0.0031)	**0.0362 (±0.0041)**
	RankIC	0.0363 (±0.0046)	0.0283 (±0.0042)	**0.0393 (±0.0049)**
SPX	IC	0.0284 (±0.0026)	0.0224 (±0.0033)	**0.0317 (±0.0035)**
	RankIC	0.0326 (±0.0030)	0.0268 (±0.0036)	**0.0355 (±0.0037)**
NDX	IC	0.0254 (±0.0029)	0.0252 (±0.0024)	**0.0285 (±0.0031)**
	RankIC	0.0297 (±0.0034)	0.0302 (±0.0027)	**0.0316 (±0.0043)**

structured thoughts is more effective than plain-text thoughts, which verifies the major hypothesis of this work.

We analyze the correlation among the best factors discovered by each method (see Figure 8). The heatmaps reveal that ReEvo suffers from significant redundancy, characterized by consistently high positive correlations, while EoH exhibits extreme absolute correlations in both directions, suggesting that its factors are largely interdependent or simple linear transformations of one another. In sharp contrast, TreEvo yields a much sparser correlation profile with significantly lower off-diagonal coefficients. This comparison demonstrates that TreEvo effectively explores a broader functional space and captures more independent alpha signals, thereby mitigating the risk of information overlap and providing a more robust foundation for factor ensemble and portfolio construction. Additionally, we compared TreEvo with existing LLM-based alpha mining frameworks and evaluated its performance across different LLMs (see Appendix F.4 and F.5). We also conducted walk-forward validation to evaluate the the out-of-sample performance of the proposed method (see Appendix F.6).

4.2.4 Ablation Studies

To clarify whether the major performance gain stems from the tree-structured thoughts or the proposed evolutionary operators, a variant of ReEvo is designed by solely evolving its codes with the proposed tree-structured thoughts, denoted as TReEvo. By comparing TReEvo with ReEvo, it suffices to show how tree-structure contributes to the performance, as they share the same evolutionary operators. By comparing TReEvo with the proposed TreEvo, the novel evolutionary operators are ablated.

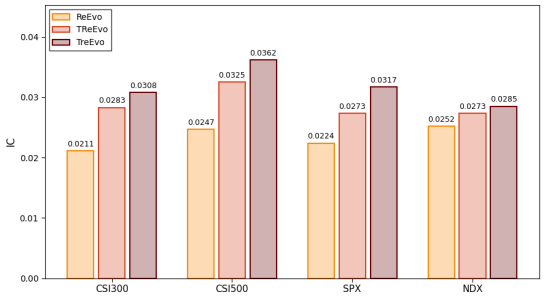


Figure 6: Comparison results of ablation studies of ReEvo, TReEvo and TreEvo on 4 databases

Figure 6 depicts the performance of ReEvo, TReEvo, and TreEvo on four datasets. TReEvo consistently outperforms ReEvo in terms of IC, verifying the advantages of the tree-structured thoughts. The superiority of TreEvo over TReEvo reveals that the proposed evolutionary operators are more compatible with tree-structured thoughts. More specifically, the ICs of TReEvo lead 23.98% over that of ReEvo on average. Comparatively, the ICs of the proposed TreEvo gain advantage over TReEvo by 10.19%. These present that the tree-structured thought is more dominant than the novel evolutionary operators to improve the performance of LLM-driven EAs on alpha mining. Meanwhile, the evolutionary operators should also be tailored to be compatible with the hierarchical thoughts, otherwise the improvements would be limited.

5 Conclusion

This paper proposes a novel LLM-driven alpha mining paradigm based on tree-structured thoughts, which enables more expressive and hierarchical representations of heuristics. By leveraging the compositional nature of trees, our approach reduces semantic coupling between sub-components, allowing for finer-grained reasoning and more flexible manipulation of intermediate thought steps, making it particularly suitable for complex search spaces such as formulaic alpha. Experiments show that, under a relatively small evaluation budget, our method achieves performance competitive with traditional approaches. The proposed method also demonstrates stronger performance compared with existing LLM-driven EAs under then same evaluation budget.

Limitations

An important aspect of TreEvo concerns the explicit hierarchical text representation employed in this work. While the framework adopts a structured encoding with explicit delimiters to express tree topology, we do not systematically examine which forms of hierarchical representation are most effective. Different encoding schemes may introduce distinct inductive biases into the evolutionary process, particularly in how operators enable fine-grained structural control. The choice of representation therefore has meaningful implications for the precision and interpretability of evolutionary search. Investigating alternative hierarchical encodings and their influence on fine-grained search behavior represents a promising direction for further study.

In practical deployments, certain obstacles remain. LLM queries limit the number of individuals that can be generated within a fixed time budget, leading LLM-driven EAs to typically explore fewer candidates than traditional methods. According to experimental observations, mutation operations under the hierarchical representation tend to produce incremental structural updates. Pruning partially alleviates this effect, though introducing more diverse transformations during mutation should further improve exploration. In addition, for constrained tasks such as alpha mining, invalid outputs may still occur when domain constraints are explicitly specified, necessitating additional validation and correction.

References

Masaya Abe and Hideki Nakayama. 2018. Deep learning for forecasting stock returns in the cross-section. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 273–284. Springer.

Vaishak Belle and Ioannis Papantonis. 2021. Principles and practice of explainable machine learning. *Frontiers in big Data*, 4:688969.

Jurgita Černevičienė and Audrius Kabašinskas. 2024. Explainable artificial intelligence (xai) in finance: a systematic literature review. *Artificial Intelligence Review*, 57(8):216.

Luyang Chen, Markus Pelger, and Jason Zhu. 2024. Deep learning in asset pricing. *Management Science*, 70(2):714–750.

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 615–621, New Orleans, Louisiana. Association for Computational Linguistics.

Laure Crochepierre, Lydia Boudjeloud-Assala, and Vincent Barbesant. 2022. Interactive reinforcement learning for symbolic regression from multi-format human-preference feedbacks. In *IJCAI*, pages 5900–5903.

Eugene F Fama and Kenneth R French. 1993. Common risk factors in the returns on stocks and bonds. *Journal of financial economics*, 33(1):3–56.

Stefano Giglio, Bryan Kelly, and Dacheng Xiu. 2022. Factor models, machine learning, and asset pricing. *Annual Review of Financial Economics*, 14(1):337–368.

Richard C Grinold and Ronald N Kahn. 2000. The efficiency gains of long–short investing. *Financial Analysts Journal*, 56(6):40–53.

Shihao Gu, Bryan Kelly, and Dacheng Xiu. 2020. Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33(5):2223–2273.

Hao Hao, Xiaoqun Zhang, and Aimin Zhou. 2024. Large language models as surrogate models in evolutionary algorithms: A preliminary study. *Swarm and Evolutionary Computation*, 91:101741.

Zifan He, Yingqi Cao, Zongyue Qin, Neha Prakriya, Yizhou Sun, and Jason Cong. 2025. HMT: Hierarchical memory transformer for efficient long context language processing. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8068–8089, Albuquerque, New Mexico. Association for Computational Linguistics.

James B Heaton, Nick G Polson, and Jan Hendrik Witte. 2017. Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry*, 33(1):3–12.

Beichen Huang, Xingyu Wu, Yu Zhou, Jibin Wu, Liang Feng, Ran Cheng, and Kay Chen Tan. 2024. Exploring the true potential: Evaluating the black-box optimization capability of large language models. *arXiv preprint arXiv:2404.06290*.

Haoyu Huang, Yongfeng Huang, Yang Junjie, Zhenyu Pan, Yongqiang Chen, Kaili Ma, Hongzhi Chen, and James Cheng. 2025. Retrieval-augmented generation

with hierarchical knowledge. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 6044–6060, Suzhou, China. Association for Computational Linguistics.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2025. A survey on large language models for code generation. *ACM Transactions on Software Engineering and Methodology*.

Weiwei Jiang. 2021. Applications of deep learning in stock market prediction: recent progress. *Expert Systems with Applications*, 184:115537.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30.

Sangyeon Kim and Myungjoo Kang. 2019. Financial series prediction using attention lstm. *arXiv preprint arXiv:1902.10877*.

Robert Lange, Yingtao Tian, and Yujin Tang. 2024. Large language models as evolution strategies. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 579–582.

Zhiwei Li, Ran Song, Caihong Sun, Wei Xu, Zhengtao Yu, and Ji-Rong Wen. 2024. Can large language models mine interpretable financial factors more effectively? a neural-symbolic factor mining agent model. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 3891–3902.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. 2024a. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, pages 32201 – 32223.

Shengcai Liu, Caishun Chen, Xinghua Qu, Ke Tang, and Yew-Soon Ong. 2024b. Large language models as evolutionary optimizers. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE.

Matteo Merler, Katsiaryna Haitsiukevich, Nicola Dainese, and Pekka Marttinen. 2024. In-context symbolic regression: Leveraging large language models for function discovery. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pages 589–606, Bangkok, Thailand.

Hao Qian, Hongting Zhou, Qian Zhao, Hao Chen, Hongxiang Yao, Jingwei Wang, Ziqi Liu, Fei Yu, Zhiqiang Zhang, and Jun Zhou. 2024. Mdgnn: Multi-relational dynamic graph neural network for comprehensive and dynamic stock investment prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 14642–14650.

Tao Ren, Ruihan Zhou, Jinyang Jiang, Jiafeng Liang, Qinghao Wang, and Yijie Peng. 2024a. Riskminer: Discovering formulaic alphas via risk seeking monte carlo tree search. In *Proceedings of the 5th ACM International Conference on AI in Finance*, pages 752–760.

Weizhe Ren, Yichen Qin, and Yang Li. 2024b. Alpha mining and enhancing via warm start genetic programming for quantitative investment. *arXiv preprint arXiv:2412.00896*.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, and 1 others. 2024. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475.

Hao Shi, Weili Song, Xinting Zhang, Jiahe Shi, Cuicui Luo, Xiang Ao, Hamid Arian, and Luis Angel Seco. 2025a. Alphaforge: A framework to mine and dynamically combine formulaic alpha factors. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 12524–12532.

Yu Shi, Yitong Duan, and Jian Li. 2025b. Navigating the alpha jungle: An llm-powered mcts framework for formulaic factor mining. *arXiv preprint arXiv:2505.11122*.

Haoxiang Tian, Xingshuo Han, Guoquan Wu, An Guo, Yuan Zhou, Jie Zhang, Shuo Li, Jun Wei, and Tianwei Zhang. 2025. An llm-empowered adaptive evolutionary algorithm for multi-component deep learning systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 20894–20902.

Aymeric Vie, Alissa M Kleinnijenhuis, and Doyne J Farmer. 2020. Qualities, challenges and future of genetic algorithms: a literature review. *arXiv preprint arXiv:2011.05277*.

Hanchen Wang, Tianfan Fu, Yuanqi Du, Wenhao Gao, Kexin Huang, Ziming Liu, Payal Chandak, Shengchao Liu, Peter Van Katwyk, Andreea Deac, and 1 others. 2023. Scientific discovery in the age of artificial intelligence. *Nature*, 620(7972):47–60.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Xiao Yang, Weiqing Liu, Dong Zhou, Jiang Bian, and Tie-Yan Liu. 2020a. Qlib: An ai-oriented

quantitative investment platform. *arXiv preprint arXiv:2009.11189*.

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

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. 2024. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608.

Wei Jie Yeo, Wihan Van Der Heever, Rui Mao, Erik Cambria, Ranjan Satapathy, and Gianmarco Mengaldo. 2025. A comprehensive review on financial explainable ai. *Artificial Intelligence Review*, 58(6):1–49.

Shuo Yu, Hongyan Xue, Xiang Ao, Feiyang Pan, Jia He, Dandan Tu, and Qing He. 2023. Generating synergistic formulaic alpha collections via reinforcement learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5476–5486.

Fangfang Zhang, Yi Mei, Su Nguyen, and Mengjie Zhang. 2023. Survey on genetic programming and machine learning techniques for heuristic design in job shop scheduling. *IEEE Transactions on Evolutionary Computation*, 28(1):147–167.

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

Xingxing Zhang, Furu Wei, and Ming Zhou. 2019. HIBERT: Document level pre-training of hierarchical bidirectional transformers for document summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5059–5069, Florence, Italy. Association for Computational Linguistics.

Junjie Zhao, Chengxi Zhang, Min Qin, and Peng Yang. 2025. Quantfactor reinforce: mining steady formulaic alpha factors with variance-bounded reinforce. *IEEE Transactions on Signal Processing*.

A Datasets

CSI300 tracks the 300 largest and most liquid A-share stocks, which represents the large-capitalization (large-cap) segment of China's market.

CSI500 consists of 500 mid-cap and small-cap A-share stocks, which reflects the behaviors of a broader market besides the large-cap stocks, capturing the growth potential of more dynamic and innovative stocks.

SPX (S&P500) includes 500 leading publicly traded companies in U.S., covering all major industries and reflecting the overall American economic.

NDX (NASDAQ100) consists of 100 largest non-financial companies listed on the NASDAQ stock exchange, representing a broad spectrum of technology, consumer, and healthcare sectors in the U.S. economy.

B Baselines

Traditional Approaches: XGBoost (Chen and Guestrin, 2016) is an implementation of gradient boosted decision trees with regularization and second-order optimization for accurate and scalable prediction on tabular data, provided by Qlib (Yang et al., 2020b). **LightGBM** (Ke et al., 2017) is a gradient boosting framework using histogram-based binning and leaf-wise tree growth for fast and efficient training on large datasets, also provided by Qlib. **GP** (Zhang et al., 2020) is an EA, iteratively searching interpretable symbolic formulas through evolutionary operators like crossover and mutation, integrated by gplearn[1]. **AlphaGen** (Yu et al., 2023) is an RL framework where PPO-based RL and tokenized Reverse Polish Notation formula generation optimize portfolio-level performance to produce synergistic formulaic alpha sets. **AlphaForge** (Shi et al., 2025a) is an RL framework combines a generative-predictive neural network for formulaic alpha factor mining with a dynamic factor combination model. **QFR** (Zhao et al., 2025) is an RL approach uses a REINFORCE policy without critic, introducing greedy-baseline variance reduction and information-ratio based reward shaping to mine stable, interpretable alpha factors.

LLM-driven EAs: EoH (Liu et al., 2024a) is a framework that combines LLMs and EA to evolve both thoughts and executable code, enabling efficient automatic heuristic design. **ReEvo** (Ye et al.,

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

2024) integrates LLMs with EA to solely generate executable codes, strengthened with reflective reasoning.

C Arithmetic Operators

The predefined operators used in traditional symbolic approaches are listed in Table 3.

Table 3: Operators Used in Symbolic Approaches

Category	Operator
Cross-Section	abs(x), log(x) $+, -, \times, /, >, <$
Time-Series	ts_mean(x, l), ts_med(x, l), ts_sum(x, l) ts_std(x, l), ts_var(x, l) ts_max(x, l), ts_min(x, l) ref(x, l), mad(x, l), delta(x, l) wma(x, l), ema(x, l) ts_cov(x, y, l), ts_corr(x, y, l)

D Variation Point Modification

We generated paired samples consisting of tree-structured and flat texts containing 10 ideas. These ideas encompass a hierarchy ranging from high-level abstractions to concrete details. For each paired sample, a simple prompt was used to instruct the LLM to select a single idea as a variation point, thereby generating a new individual. The experiment was conducted across 10 independent sets of samples: half of them were generated as tree-structured text first then flat text, the other half were flat text first then tree-structured text.

E Alignment Between Textual Similarity and Factor Correlation

For each representation, we generated paired samples consisting of two texts containing an initial text and a corresponding variant generated from it. The initial text is produced by an LLM under a specified representation, and the variant is obtained by prompting the LLM to modify the original text under the same prompt, resulting in a structurally related counterpart. Textual similarity is defined as the cosine similarity between embeddings of the two texts. Factor correlation is measured as the IC between the corresponding factors, defined as the correlation between their factor value time series.

F Additional Experimental Results

F.1 Variation Point Distribution

Figure 7 shows the variation point distribution of 10 samples.

F.2 Wall-Clock Time and Token Cost

We compare the Wall-Clock time and token cost under the experimental settings in our work. The result is provided in Figure 4. While LLM-based approaches costs less time than RL-based approaches, GP exhibits the lowest time cost. We further execute GP with a time budget of 20 min for 5 times, achieving an average IC of 0.0241 ± 0.0021. Under the same time budget, TreEvo still outperforms GP.

Table 4: Wall-clock time and token cost comparison across algorithms.

Algorithms	Wall-Clock Time	Token Cost
GP	10 min	–
AlphaGen	4 hour	–
AlphaForge	6 hour	–
QFR	4.5 hour	–
TreEvo	20 min	654K
ReEvo	18 min	414K
EoH	40 min	632K

F.3 Factor Correlations

Figure 8 shows the correlation among the best factors discovered by each method.

F.4 Comparison with LLM Alpha Mining Frameworks

Table 5 exhibits the performance of TreEvo compared to recent LLM alpha mining frameworks.

Table 5: Performance comparison across different markets with LLM alpha mining frameworks.

Dataset	Metric	RD-Agent	AlphaAgent	TreEvo
csi300	IC	0.0267	0.0278	**0.0308**
	RankIC	0.0315	0.0329	**0.0349**
csi500	IC	0.0283	0.0341	**0.0362**
	RankIC	0.0344	0.0387	**0.0393**
SPX	IC	0.0249	**0.0323**	0.0317
	RankIC	0.0286	**0.0364**	0.0355
NDX	IC	0.0214	0.0258	**0.0285**
	RankIC	0.0263	0.0302	**0.0316**

F.5 Comparison on Different LLMs

Table 6 exhibits the performance of TreEvo on different LLMs.

F.6 Walk-forward Validation

We apply walk-forward validation on CSI 300 by splitting the 2023 test period into quarterly segments (Q1–Q4), training each time on the preceding one year of data. The results, reported in terms

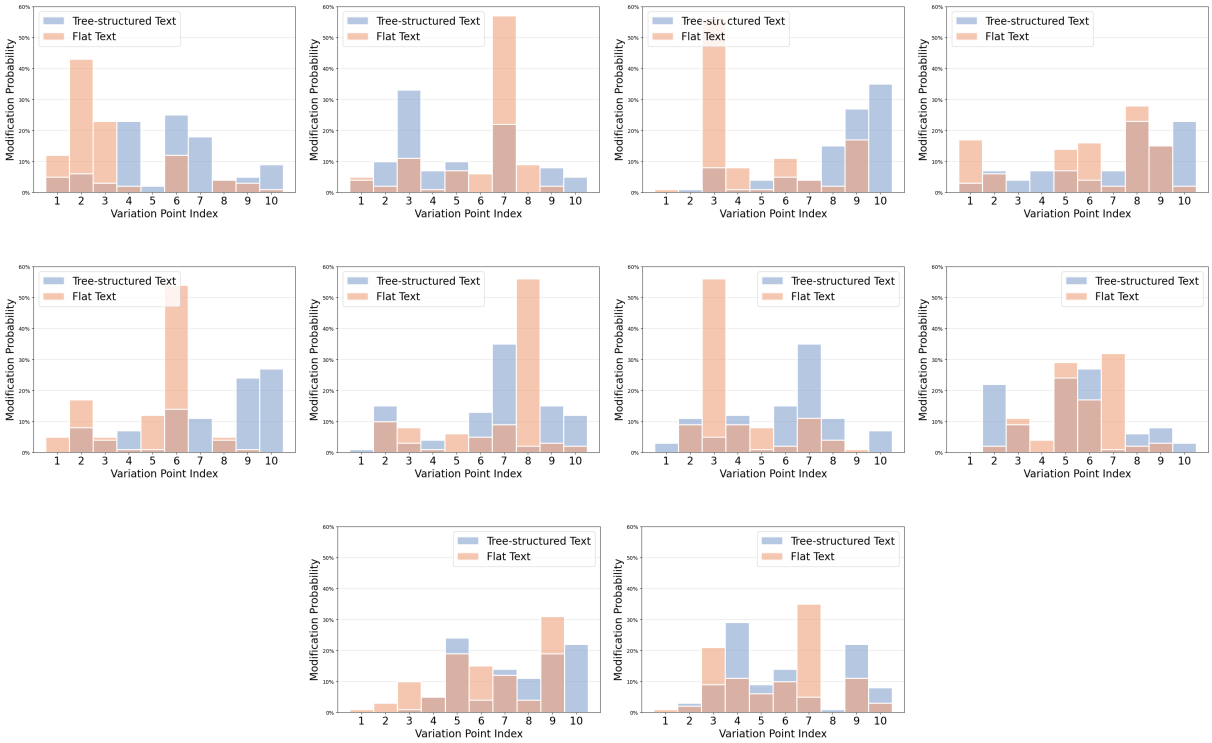


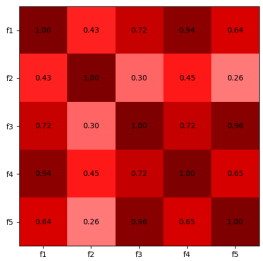
Figure 7: Visualization of the remaining variation point distribution. For the first five images, The tree-structured text is generated and then transformed to flat text. For the last five images, The flat text is generated and then transformed to tree-structured text.

Table 6: Performance comparison of TreEvo on different LLMs.

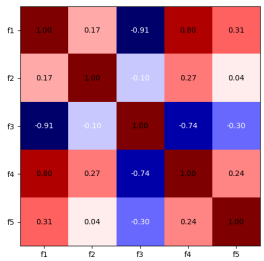
LLM	csi300		csi500		SPX		NDX	
	IC	RankIC	IC	RankIC	IC	RankIC	IC	RankIC
Qwen3-Max	0.0308	0.0349	0.0362	0.0393	0.0317	0.0355	0.0285	0.0316
DeepSeek V3	0.0287	0.0337	0.0346	0.0375	0.0302	0.0347	0.0289	0.0331
Gemini3 pro	0.0315	**0.0368**	**0.0378**	**0.0416**	**0.0334**	**0.0371**	**0.0327**	**0.0363**
GPT5.1	**0.0318**	0.0342	0.0358	0.0405	0.0323	0.0355	0.0309	0.0340

Table 7: Quarterly performance comparison across methods.

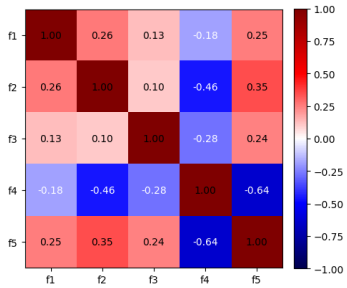
Quarter	EoH			ReEvo			TreEvo		
	IC	RankIC	ER	IC	RankIC	ER	IC	RankIC	ER
Q1	0.0253	0.0319	0.0306	0.0228	0.0256	0.0144	**0.0354**	**0.0392**	**0.0427**
Q2	0.0232	0.0259	0.0137	0.0227	0.0269	**0.0189**	**0.0278**	**0.0325**	0.0158
Q3	0.0273	0.0324	0.0301	0.0218	0.0261	0.0105	**0.0343**	**0.0384**	**0.0431**
Q4	0.0249	0.0284	0.0223	0.0205	0.0245	0.0067	**0.0326**	**0.0359**	**0.0356**



(a) ReEvo



(b) EoH



(c) TreEvo

Figure 8: IC correlation analysis of the best factors from 5 independent runs for LLM-driven EAs on CSI300.

of IC, RankIC, and excess return (ER), are shown in Table 7.

G Framework and Prompts

G.1 Framework

Step 1 Initialization: Initialize a population P of N thoughts by prompting LLM with the initialization prompt. Then ask LLM to generate N codes based the N thoughts and carry out the evaluation of each code respectively.

Step 2 Evolutionary Process: Repeat until the predefined stop criteria are met.

Step 2.1 Offspring Generation: Generate a new offspring population O of N thoughts based on the current population P. For each iteration, the LLM is prompted to apply one of the following three operator types in rotation to generate all N offspring:

- **Crossover:** Select two parent thoughts from P and apply crossover operator to generate a new thought.

- **Mutation:** Select one parent thought from P and apply Mutation-R, Mutation-I, or Mutation-F operator based on specified probabilities p_R, p_I, and p_F to generate a new thought.

- **Pruning:** Select one parent thought from P and apply pruning operator to generate a new thought.

Step 2.2 Evaluation: For each of the N new offspring thoughts in O, use the LLM to generate the corresponding executable code. Evaluate each piece of code in the market dataset.

Step 2.3 Selection: Combine the N parents from P and the N offspring from O into a candidate pool of $2N$ individuals. Select the N best-performing individuals from this pool to form the population P for the next iteration.

Step 3 Output: Output the code of the best performing thought through the whole process.

G.2 Prompts

System Prompt

You are a "Quantitative Factor Tree Semantic Generator".
Your task: Randomly generate a "quantitative factor tree text" that has financial meaning, statistical logic, and computational validity.
Goal:
The generated result must resemble:
- a real quantitative factor
- an executable computation process
- a hierarchical semantic tree structure
- an AST (Abstract Syntax Tree) that progressively decomposes financial semantics into underlying data

---

[Output Format Requirements]

---

1. You must use ASCII tree format
2. The entire output must be a single tree
3. Output only the tree
Do not explain
Do not analyze
Do not add extra text

4. The root node must be:
an abstract financial objective, for example:
- Momentum Strength
- Volatility Compression
- Liquidity Imbalance
- Trend Persistence
- Earnings Quality
- Price Deviation
- Relative Strength

- Return Stability
- Volume Anomaly
- Market Sentiment Shift

5. The tree must satisfy:
abstract financial semantics
- intermediate statistical transformations
- underlying raw data

[Node Type Constraints]

Nodes may only belong to the following categories:
1. Financial semantic nodes
For example:
- Momentum
- Trend
- Volatility
- Liquidity
- Earnings Quality
- Valuation Deviation
- Sentiment Strength
- Relative Price Strength
- Capital Flow Pressure
- Microstructure Imbalance

2. Statistical / mathematical operation nodes
For example:
- addition
- subtraction
- ratio
- difference
- rolling_mean
- rolling_std
- rolling_rank
- zscore
- normalize
- decay
- correlation
- covariance
- regression_beta
- EMA
- rank
- abs
- log
- delta

3. Data leaf nodes
These must ultimately map to real financial data, including only:
- close
- open
- high
- low
- amount
- volume

[Structural Constraints]

---

1. Tree depth constraints:
Minimum depth: 3
Maximum depth: 6
2. Number of child nodes per node:
- unary operations: 1 child
- binary operations: 2 children
- a small number of nodes may have 3 children
3. Degenerate chain structures are forbidden
4. Overly wide structures are forbidden
5. Every layer must have a clear semantic hierarchy:
higher levels are more abstract
lower levels are more concrete

[Financial Semantic Consistency]

---

1. Volatility-related nodes:
may only be composed from:
- rolling_std
- variance
- abs(return)
- squared_return

2. Momentum-related nodes:
may only be composed from:
- return
- ratio
- delta
- EMA trend
- cumulative return
3. Liquidity-related nodes:
may only use the following data:
- volume
- turnover
- amount
- spread

4. Node names must be logically consistent with their subtrees

---

[Time Constraints]

---

1. Future information is forbidden
Forbidden:
- return_t+1
- close_t+1
2. Allowed:
- t
- t-k
- rolling window

[Generation Requirements]

---

1. Each generation must be random
Randomization includes:
- financial theme
- operation combinations
- time windows
- data sources
- normalization methods
- tree structure shape
2. Do not generate repeated templates
3. Do not generate meaningless stacks of financial buzzwords
Incorrect:
market volatility trend liquidity strength factor
4. The tree must be mappable to a real computational expression

Initialization Prompt

$\{seed_tree\}$

Refering to the format of a trivial design above, be very creative to generate a totally different factor from the given factor and give the thought tree of it. Your response only outputs txt code of the thought tree. Format your code as a txt code block: “‘txt ... “‘.

Crossover Prompt

[Crossover] Crossover generates a new factor tree by refering the high-level ideas from parent trees. Choose a sub-idea from one parent tree and combine it with the other parent tree. You are allowed to revise the high level ideas of changing sub-idea to preserve logical consistency and remain interpretable.

[Parent tree 1]
$\{parent_tree_1\}$

[Parent tree 2]
$\{parent_tree_2\}$

[New tree]
The root should represent the shared main idea. Child nodes should be meaningful refinements of their parent. The remaining branches should be limited unchanged. Your response only outputs txt code of the thought tree. Format your code as a txt code block: “‘txt ... “‘.

Mutation-R Prompt

[Mutation] Mutation generate the tree of the new factor which is totally different from all parent trees to ensure diversity. The new tree should be logical and interpretable.

[Parent trees]
$\{parent_trees\}$

[New tree]
Generate a single coherent tree structure. Your response only outputs txt code of the thought tree. Format your code as a txt code block: “‘txt ... “‘.

Mutation-I Prompt

[Sub-idea Mutation] Sub-idea Mutation generate a new tree by changing one sub-idea of the parent tree. First, choose one internal node of the tree. Then introduce a new sub-idea to replace the old sub-idea while keeping logical consistency of the tree. The new tree should be logical and interpretable.

[Parent tree]
$\{parent_tree\}$

[New tree]
Generate a single coherent tree structure. Your response only outputs txt code of the thought tree. Format your code as a txt code block: “‘txt ... “‘.

Mutation-F Prompt

[Parameter Mutation] Parameter Mutation generate a new tree by changing the parameters in the parent tree while maintaining the idea of the tree. The Parameters exists all on leaf nodes in the tree. The new tree should be logical and interpretable.

[Parent tree]
$\{parent_tree\}$

[New tree]
Generate a single coherent tree structure. Your response only outputs txt code of the thought tree. Format your code as a txt code block: “‘txt ... “‘.

Pruning Prompt

[Pruning] Pruning simplifies a thought tree by keeping core idea and removing redundant or low-importance components. The main idea and overall structure should be preserved. Do not introduce any new concepts or rewrite existing ones.

[Parent tree]
$\{parent_tree\}$

[New tree]
Generate a single coherent tree structure. Each remaining node should directly support the core reasoning idea. Your response only outputs txt code of the thought tree. Format your code as a txt code block: “‘txt ... “‘.