https://doi.org/10.1007/s11704-025-41061-5

RESEARCH ARTICLE

A hybrid approach to formulaic alpha discovery with large language model assistance

Shuo YU[1,2], Hong-Yan XUE[1,2], Xiang AO[1,2,3]✉, Qing HE[1,2]✉

1. Key Lab of AI Safety, Chinese Academy of Sciences, Key Lab of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
2. University of Chinese Academy of Sciences, Chinese Academy of Sciences, Beijing 100049, China
3. Institute of Intelligent Computing Technology, Suzhou 215000, China

Received September 30, 2024; accepted March 17, 2025

E-mail: aoxiang@ict.ac.cn; heqing@ict.ac.cn

Abstract

In the domain of quantitative trading, the imperative is to translate historical financial data into predictive signals, commonly referred to as alpha factors, which serves to anticipate future market trends. Notably, formulaic alphas that are expressible via explicit mathematical formulas are highly sought after by certain investors for better interpretability. The evolving landscape of technology has witnessed the increasing deployment of large language models (LLMs) across various domains, which raises the question of whether LLMs can be effective in the context of formulaic alpha-mining tasks. This paper presents several paradigms aimed at integrating LLMs into the optimization loop of alpha mining, including scenarios where an LLM serves as the sole alpha generator, as well as instances where LLMs enhance existing frameworks. Empirical evaluations on real-world stock data demonstrate significant performance improvements, with our hybrid method achieving an average information coefficient (IC) of 0.0515, a 75% improvement over the baseline — a state-of-the-art reinforcement learning-based framework; backtesting further reveals a cumulative excess return more than double the baseline framework. These results underscore the potential of LLM-enhanced approaches in advancing formulaic alpha discovery and driving innovation in quantitative trading.

Keywords

computational finance; stock trend forecasting; large language model

1 Introduction

Alpha factors, commonly referred to as alphas, are quantifiable signals derived from historical stock data, designed to be indicative of future market trends [1]. These alphas range from simple rules, such as identifying stocks with above-average trading volume today as likely outperformers tomorrow, to sophisticated models like specialized deep neural networks tailored for stock trend forecasting [2–6]. Alphas play a central role in quantitative trading, since their predictive accuracy directly impacts investment returns. Consequently, identifying robust alphas that maximize returns while mitigating risks remains a priority for practitioners in quantitative finance [7].

Among the diverse types of alphas, formulaic alphas hold particular appeal for risk-conscious investors. Defined through explicit mathematical expressions, these alphas map input features (e.g., prices and volumes) to expected returns in a transparent and interpretable manner. For instance, the simple formula $\mathrm{Var}(\mathrm{close}, 20d)$ computes the 20-day variance of closing prices, offering insights into market volatility trends. Such interpretability distinguishes formulaic alphas from complex, non-formulaic models like deep neural networks, which often function as opaque black boxes.

Traditionally, formulaic alphas were manually crafted based on established financial principles [8,9]. However, the growing demand for new alphas has driven the adoption of automated discovery methods. Genetic programming (GP) emerged as a prominent approach, utilizing evolutionary algorithms to generate new alpha expressions [10–12]. While effective at optimizing individual alphas, GP methods often overlook synergies within alpha sets, leading to redundancies and missed opportunities for performance enhancements. More recently, reinforcement learning (RL) frameworks have addressed this limitation by generating synergistic collections of alphas [13]. Nevertheless, RL-based methods face challenges such as the overly random exploration of the search space, resulting in excessively complex expressions prone to overfitting and entrapment in local optima during training.

Amid these advancements, the emergence of large language models (LLMs) offers a promising avenue for further innovation.

Pretrained on extensive human language corpora, LLMs encode a vast repository of implicit knowledge in its multitude of parameters, including logical reasoning and domain-specific insights. These capabilities have catalyzed transformative applications across fields beyond natural language processing, including data analysis, medical information retrieval, and quantitative trading [14–20]. In the context of formulaic alpha generation, LLMs exhibit potential not only due to their embedded financial knowledge but also because of the close resemblance between the task of expression generation and of language modeling. This alignment thus raises intriguing questions: *How can we harness the capabilities of LLMs to enhance alpha diversity and performance?* and *What strategies can effectively integrate LLMs into existing frameworks to address their limitations while maximizing their synergy?*

To address these questions and overcome the limitations of existing frameworks, we propose a hybrid approach to alpha set generation that seamlessly integrates LLMs into RL-based optimization loops. Our framework leverages the symbolic reasoning capabilities of LLMs to enhance alpha diversity and predictive accuracy while maintaining computational efficiency.

First, we evaluate the standalone performance of an LLM as an alpha generator by providing it with syntactic and domain-specific guidance for generating formulaic alpha expressions. The generated alphas are then evaluated using historical stock data, and the model's outputs are optimized iteratively based on an optimization loop resembling the one used in traditional (RL-based or GP-based) frameworks. In this iterative process, information on the alpha combination's state is reported to the LLM with a custom feedback mechanism. This process demonstrates the potential of LLMs to generate interpretable and effective alpha sets autonomously.

In our efforts to enhance the existing RL-based alpha generator, we suggest employing the LLM alpha generator to produce an initial pool of alphas for warm-starting the process. Building upon the aforementioned capabilities, we alternate between the RL generator and the LLM-based generator while optimizing the alpha combination. Careful scheduling schemes are applied to keep the total LLM invocation cost in a reasonable range. This strategy aims to generate alphas that lie beyond the distribution of the RL generator, mitigating overfitting and steering the framework away from local optima. This hybrid approach ensures a dynamic exchange between the two paradigms, leading to improved performance.

We validate our framework through experiments on real-world datasets. These experiments include both numerical correlation analyses, such as the information coefficient (IC) between predicted and actual returns, and investment simulations (backtests) to evaluate the practical effectiveness of our framework under realistic trading conditions.

Our work makes the following key contributions:

- We propose an approach to utilizing LLMs for formulaic alpha generation, leveraging their inherent financial knowledge to produce interpretable and diverse alpha expressions.
- We introduce a hybrid optimization framework that integrates LLM-generated alphas into an RL-based workflow. This includes scheduling and feedback mechanisms to maximize synergy between the two methods while managing computational overhead.
- We demonstrate the effectiveness of our approach through comprehensive experiments, achieving significant improvements in predictive metrics and interpretability over baseline methods.

2 Preliminaries

2.1 Stock dataset

We consider a stock dataset comprising n stocks and spanning T trading days. The state of the i-th stock on the t-th trading day is denoted by a feature vector $x_{t,i} \in \mathbb{R}^m$. For brevity in notation, we introduce $\tilde{x}_{t,i}$ as a "rolling" feature vector, which concatenates τ days' worth of features for a stock. In other words, $\tilde{x}_{t,i} = x_{t-\tau+1,i} \| \cdots \| x_{t,i} \in \mathbb{R}^{m\tau}$. Finally, let $\tilde{X}_t = (\tilde{x}_{t,1}, \ldots, \tilde{x}_{t,n})^\top \in \mathbb{R}^{n \times m\tau}$ be the matrix containing all the stocks' rolling feature vectors on a specific trading day.

2.2 Formulaic alpha

An alpha factor is characterized as a function that maps a feature matrix to alpha values for all stocks. Formally, this can be expressed as $f : \mathbb{R}^{n \times m\tau} \to \mathbb{R}^n$. When unambiguous, we will use the term "alpha" to refer to both the function f and its outcome $f(\tilde{X})$ in the subsequent sections.

A formulaic alpha is an alpha factor that is defined through an explicit mathematical formula. These formulas are composed using a predefined set of features and operators. The features include raw price and volume information of the stocks, such as the highest price on a specific trading day. The operators consist of various common mathematical operations like "$x + y$" and "$|x|$", as well as time series statistical functions like $\mathrm{Mad}(x, \Delta t)$, representing the mean absolute deviation of a value x in the recent Δt days. For instance, a specific formulaic alpha denoted as "$\mathrm{Var}(\mathrm{open} - \mathrm{close}, 20\mathrm{d})$" computes the variance of the difference between a stock's open and close prices over the last 20 days.

We follow the work of AlphaGen [13], utilizing the same set of features and operators for alpha generation. The list of operators used in our experiments are shown in Appendix A.1. In the case of the RL-based alpha generator, it is necessary to discretize the space of constants. For time difference constants, we select from predefined intervals: 1d (1 day), 5d, 10d, 20d, and 40d. Numerical constants are chosen from the set $\{0.01, 0.5, 1, 2, 5, 10, 30\}$ and their corresponding negative values. On the other hand, for the LLM-based alpha generator, which we introduce subsequently, discretization of the constant space is not necessary. The LLM generates continuous constants freely, and all values produced by the model are accepted without restriction.

2.3 Optimization objective

Following standard practices, we evaluate the effectiveness of an alpha using the IC, measuring the correlation between the alpha values and the actual trend it predicts. Formally, let the centered and

normalized value of the vector $v \in \mathbb{R}^n$ be the vector $\mathcal{N}(v)$ with components

$$\mathcal{N}(v)_i = \frac{v_i - \bar{v}}{\sqrt{\sum_{j=1}^{n} \left(v_j - \bar{v}\right)^2}}.$$

For each trading day t, given the prediction target $y_t \in \mathbb{R}^n$ and an alpha function f, the IC σ between the alpha values $z_t = f(\tilde{X}_t)$ and the target is simply given by $\sigma(z_t, y_t) = \mathcal{N}(z_t) \cdot \mathcal{N}(y_t)$. The mean IC $\sigma(f) = \mathbb{E}_t\left[\sigma(f(\tilde{X}_t), y_t)\right]$ over all trading days serves as a metric to assess the performance of an alpha across the entire dataset.

As previously discussed, the expressivity of a single formulaic alpha falls short in capturing the complexity of the stock market [13]. Consequently, in practical applications, it is common to rely on a collection of formulaic alphas. These alphas are then aggregated with a model, forming a "mega-alpha" that serves as the basis for subsequent trading decisions. We represent the combination model that aggregates the set of alphas as $c(\cdot; \mathcal{F}, \theta)$, where θ corresponds to the parameters of the model, and $\mathcal{F} = \{f_1, \ldots, f_k\}$ is a set of k alpha factors. In our experiments, the combination model $c(\cdot; \theta, \mathcal{F})$ is chosen to be a linear combination model $c(X; \mathcal{F}, w) = \sum_{i=1}^{k} w_i f_i(X)$, which preserves the interpretability of the resulting alpha combination.

Ultimately, the objective of alpha mining is defined as the optimization problem $\operatorname{argmax}_{\mathcal{F}} \max_{\theta} \sigma(c(\cdot; \mathcal{F}, \theta))$. In other words, the parameters θ are trained to maximize the mean IC of the combination model's output for each set of alphas $\mathcal{F}$. The goal is then to identify the set of alphas that maximizes the mean IC of the optimally trained combination model.

3 Methodology

3.1 LLM for few-shot alpha generation

We leverage the LLM's few-shot learning ability to generate expressions, asking it to create new alpha expressions based on a few given valid examples. Since a limited set of valid examples cannot capture all the syntactic specifications required for alpha expressions, it is essential to first formalize the structure of valid expressions into a detailed syntactic template. Additionally, we must provide the LLM with a set of operators and features that can be used in the construction of these expressions, along with their corresponding semantic or mathematical interpretations.

Provided with the syntactic specification and several examples, the LLM is able to generate valid alpha expressions for around 90% of the cases. However, in rare situations, invalid expressions may be produced due to misunderstandings of certain operators. For example, the operator "Max" should be used in the form of $\mathrm{Max}(x, t)$ for finding the maximum value of a time series x over the past t days, but sometimes the LLM confuses it with the operator $\mathrm{Greater}(x, y)$, which picks the larger value from the provided x and y. Such issues can be addressed through rule-based corrections in the expression parser. For detailed explanation, see Appendix A.2. In extremely rare cases, the model may generate completely invalid results that cannot be corrected by these rules, such as unbalanced parentheses or made up nonexistent operators and features. For these unfixable instances, further interactions aiming at correcting these errors are avoided. This is because that the LLM tends to adhere too closely to context, and attempts at fixing the errors often result in more compounding errors. As a result, these invalid outputs are simply discarded.

By integrating syntactic guidance and using the LLM's inherent generative capabilities, a broad range of alpha expressions can be produced, which can then be evaluated or refined for their effectiveness in the later steps.

Using this approach, we can even construct a framework that relies entirely on the LLM for alpha set mining. The simplest method is to directly instruct the LLM to generate a set of alphas (see *P1*) and then optimize a combination model based on this set. This approach is referred to as single-step LLM-based alpha set generation. However, this framework has an obvious drawback: the LLM lacks access to the actual training and validation data, meaning it receives no feedback on how the alphas it generates perform on real stock data. As a result, the alphas are generated randomly, with no guarantee of their effectiveness. To improve upon their effectiveness, further interactive iteration over the generated alpha set is needed.

3.2 Multi-round interactions

We propose an iterative feedback approach to enhance the performance of the purely LLM-based alpha generator, the diagram of which is illustrated in Fig. 1 as the LLM-based optimization loop. In this method, evaluation results are fed back to the LLM, prompting it to generate new alphas based on the provided feedback. This iterative process enables the LLM to refine its outputs and progressively improve its performance. For the LLM to improve upon past experience, it is essential to supply the model with basic statistics and evaluation results of the current alpha pool, similar to how reinforcement learning frameworks inform an agent about the performance of new alphas within a given environment via reward assignment.

Several considerations arise regarding the type of information to encode in the messages sent to the LLM. The most basic information includes the components of the alpha combination, their individual IC values, and their respective weights in the combination model. More advanced information could encompass the conversation history from previous iterations or the complete update history of the alpha pool, detailing which alphas are added or removed at each step and the performance metrics of the alpha combination before and after these edits.

Regardless of the information sent to the LLM, the overall procedure for alpha set generation remains the same. Initially, we request the LLM to generate an initial set of alphas. In each subsequent iteration, we supply the LLM with details about the current alpha combination, and optionally, information regarding the conversation or update history, before requesting the generation of a new set of alphas. Next, we attempt to insert each newly generated alpha into the existing set one at a time. If the pool exceeds its capacity after an insertion, we remove the alpha with the lowest absolute weight from the set. This process is repeated for a predetermined number of iterations. The whole procedure is given by Algorithm 1 in pseudo-code format.

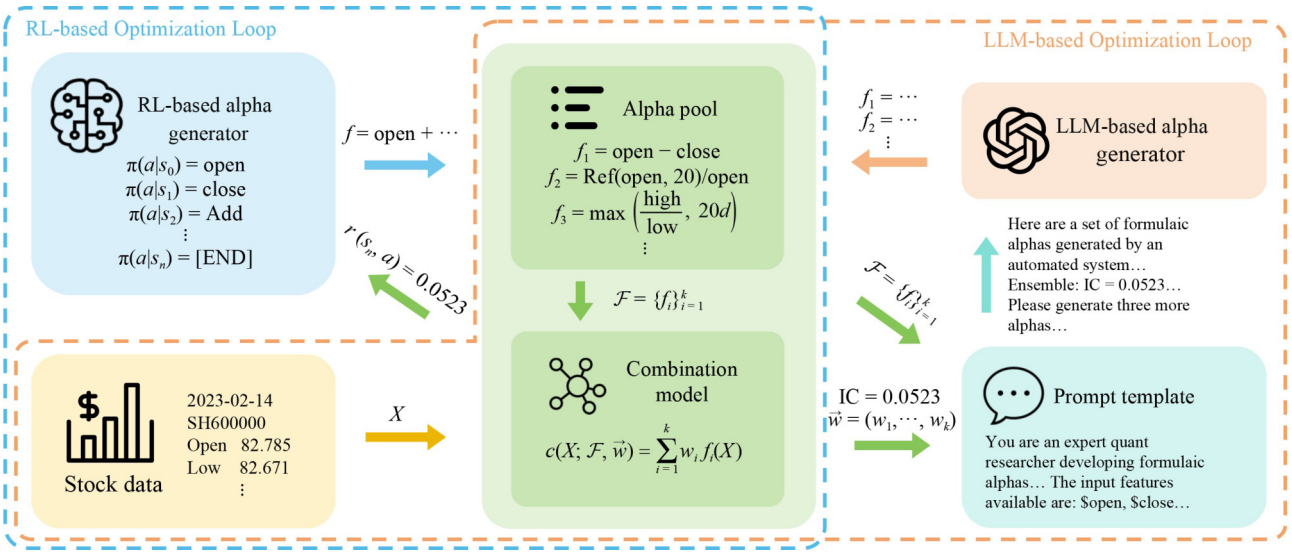

Fig. 1 A diagram showing the working process of the RL-LLM hybrid alpha generator HARLA. The RL-based optimization loop and the LLM-based one share the same alpha pool to continuously and alternately improve upon it. Notice that the procedure of the two feedback loops closely resembles each other

Algorithm 1 Iterative alpha combination optimization with LLM

Input: Alpha set size k, update size l, iteration steps s, initial alpha set $\mathcal{F}_0$ (optional)
Output: Optimized alpha combination $c(\cdot; \mathcal{F}, w)$, where $\mathcal{F} = \{f_1, f_2, \ldots, f_k\}$, $w = (w_1, w_2, \ldots, w_k)^\top$.

if $\mathcal{F}_0$ *is provided* **then**
  $\mathcal{F} \leftarrow \mathcal{F}_0$;
else
  $\mathcal{F} \leftarrow \text{InvokeLLM}(P_1(k))$;
$w \leftarrow \text{rand}(\mathbb{R}^k)$;
Optimize $c(\cdot; \mathcal{F}, w)$ w.r.t w with gradient descent;
for $i \leftarrow 1$ **to** s **do**
  Evaluate c and format the prompt p with template P_2;
  $\mathcal{F}' = \{f'_1, f'_2, \ldots, f'_l\} \leftarrow \text{InvokeLLM}(p)$;
  foreach $f \in \mathcal{F}'$ **do**
    $w \leftarrow w \| \text{rand}(\mathbb{R})$;
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{f\}$;
    Optimize c incrementally;
    if $|\mathcal{F}| > k$ **then**
      $j \leftarrow \text{argmin}_j |w_j|$;
      $w \leftarrow \{w_1, \ldots, w_{j-1}, w_{j+1}, \ldots, w_{|\mathcal{F}|}\}$;
      $\mathcal{F} \leftarrow \mathcal{F} \setminus \{f_j\}$;
return c;

3.3 Integration of LLM into existing framework

We now address the challenge of integrating the LLM-based alpha combination generator and optimizer into an existing framework. A key issue with the RL-based alpha set generator is that, due to its freedom to explore the entire search space of valid alpha expressions, it often produces overly complex expressions that may capture spurious correlations in the training data. This can result in the model becoming trapped in overfitting local optima. Introducing LLM-generated alphas into the alpha set disrupts these overfitting tendencies, providing new, more generalizable expressions and thereby enhancing the efficiency and effectiveness of the search process.

Given that the iterative update scheme of the LLM-based generator closely mirrors the RL optimization loop, we propose merging the two into a unified optimization process. However, invoking LLMs incurs significantly higher costs compared to running the RL loop. To manage the higher computational cost of invoking LLMs, we limit their use to specific trigger conditions. One effective approach is to invoke the LLM after the RL-based alpha set generator has failed to improve the alpha combination's performance for a significant number of steps. Another practical strategy is to invoke the LLM periodically, after a predefined number of RL environment simulation steps. In our experiments, the latter approach proves effective, achieving substantial performance gains while maintaining computational and financial costs within a manageable range.

The hybrid optimization process proceeds as follows. First, the LLM-based generator generates an initial alpha set following Algorithm 1, which is then continuously refined by the RL-based alpha set generator. The update history of the alpha pool is recorded to support the LLM's iterative prompting strategy if necessary. When the trigger condition for LLM invocation is met, the least significant alphas in the current combination are removed to create space for new alphas. The LLM-based multi-round optimization algorithm then refines the alpha set, using the current alpha set state as the initial input. Upon completion of the LLM-based loop, the updated alpha set is returned to the RL-based optimizer for further refinement. This alternating optimization continues until an early-stop condition is triggered or the maximum number of total optimization steps is reached.

A diagram of the whole hybrid framework is shown in Fig. 1, and the psuedo-code is given by Algorithm 2.

3.4 Prompt design

The goal of the prompt design process is to enable the LLM to generate valid and effective formulaic alpha expressions, adhering to strict syntactic requirements while maintaining interpretability and

Algorithm 2 Hybrid alpha set optimization framework

Input: Alpha set size k, LLM update size l, force removed alpha amount m, LLM iteration steps s_L, max RL environment simulation steps s_E

Output: Optimized alpha combination $c(\cdot;\mathcal{F},w)$

$c(\cdot;\mathcal{F},w) \leftarrow \mathrm{Alg}_1(k,l,s_L,\{\})$;
for $i \leftarrow 1 \ldots s_E$ **do**
  Run RL step and update $\mathcal{F}, w$ accordingly (as described in [13]);
  if *LLM invocation condition met* **then**
    while $|\mathcal{F}| > k - m$ **do**
      $j \leftarrow \mathrm{argmin}_j |w_j|$;
      $w \leftarrow \{w_1, \ldots, w_{j-1}, w_{j+1}, \ldots, w_{|\mathcal{F}|}\}$;
      $\mathcal{F} \leftarrow \mathcal{F} \setminus \{\{j\}\}$;
    $c(\cdot;\mathcal{F},w) \leftarrow \mathrm{Alg}_1(k,l,s_L,\mathcal{F})$;
  if *early-stop condition met* **then**
    break;
return c;

diversity. Achieving this required addressing challenges such as ensuring syntactic validity, minimizing redundant outputs, and leveraging the model's domain knowledge to produce meaningful expressions.

When designing our prompts, we generally follow well-established practices in prompt engineering. For instance, it is widely recognized that clear and precise instructions yield more effective responses, and that specifying what to do—rather than what to avoid—is often more beneficial. Additionally, encouraging the model subtly within the instruction can potentially influence its output positively [21,22]. These principles form the baseline of our prompt design. However, crafting prompts for formulaic alpha-mining required domain-specific adaptations and refinements to optimize the outcomes.

A critical aspect of our method is ensuring the LLM follows instructions to generate alphas in the correct form. Crafting precise and clear instructions is vital, as generating syntactically valid expressions consistently is a prerequisite for producing high-quality alpha factors. A model incapable of reliably generating valid expressions would not be useful for generating effective alphas. We design multiple prompts to describe the alpha expression syntax and find that the results consistently align with our expectations, demonstrating the model's ability to follow domain-specific instructions.

Beyond achieving syntactic correctness, another critical consideration is how to guide the LLM toward generating alphas that enhance the performance of the current alpha combination. During the iterative optimization process, various forms of contextual information can be collected, such as the current alpha pool state, weights assigned to individual alphas, and the impact of previous alpha insertions or removals on combination performance. With a model that reliably generates syntactically valid expressions, the challenge shifts to determining what subset of this information should be included in the prompt to maximize its effectiveness. To address this, we test several different prompting strategies, including providing the model with performance metrics from previous iterations, as well as details on which alphas had the most significant impact on performance. These strategies help us identify the optimal balance of information to give the model in order to generate expressions that contribute meaningfully to the alpha combination's optimization.

For a detailed comparison of these strategies, please refer to Subsection 4.3.

4 Experiments

In this section, we design a series of experiments to assess various aspects of the proposed framework. The experiments cover topics including the overall performance of the framework under correlation analysis and simulated investment, impact of different prompting strategies, computational efficiency assessment, and studies on curious cases.

4.1 Experiment settings

4.1.1 Stock data

Our experiments were conducted using raw stock price and volume data sourced from the China A-shares stock market[1]. Each stock on every trading day is characterized by six raw features ($m = 6$), including high, low, open, and close prices, volume, and volume-weighted average price (VWAP). The dataset spans from 01/01/2012 to 06/30/2023. The initial ten years of data (01/01/2012–12/31/2021) are allocated for model training, while the remaining period (01/01/2022–06/30/2023) is reserved for testing. The prediction target is defined as the 20-day return rate of the close price, expressed as "$\mathrm{Ref}(\mathrm{close}, -20)/\mathrm{close} - 1$" in formulaic representation. Our experiments are conducted on the constituents of the CSI300 index, an important subset of the China A-shares stock market.

4.1.2 Selection of LLM

Formulaic alpha expressions impose strict requirements on the precise format of the outputs. Simply outlining the rules would consume a significant portion of the limited context available to an LLM. This stringent requirement emphasizes the critical importance of the model adhering precisely to the provided instructions. This necessity arises from the lack of a viable method for processing a response that includes non-existent operators, contrived feature names, or grammatical errors. The only viable options are either discarding such responses or instructing the model to self-correct the expression. The latter, however, consumes even more space within the already limited context. If the model frequently fails to adhere to the prescribed rules, its potential contribution to existing methods becomes questionable. Consequently, older and comparatively "smaller" LLMs that do not carefully follow human instructions, such as GPT-3 [23], are unsuitable for this task.

Since the advent of the InstructGPT technique [24], a variety of language models capable of following instructions have emerged. While ChatGPT [14] stands as a representative example, open-source

[1] The price and volume data are backward-dividend-adjusted based on the adjustment factors on 01/01/2009.

alternatives like Llama [25] and ChatGLM [26] have also gained attention. Unfortunately, to maintain the integrity of our evaluation, it is crucial to allocate a sufficiently large time span for stock data. A key concern is that the training corpora of newer models overlap with our testing period, potentially providing access to "future data" which could compromise the validity of our test results.

Moreover, preliminary experiments have revealed that common open source models (even ones with a comparatively large amount of parameters like 70B) struggle to adhere to the syntactic requirements of alpha-mining tasks, i.e., they tend to generate valid formulas at a significantly lower success rate. Consequently, we opt to exclusively use ChatGPT-3.5[2)] in our tests, as it is the most suitable choice with robust reasoning capabilities and a training data cutoff that ensures no overlap with the testing period.

4.1.3 Compareison methods

We select two non-LLM methods as baselines for comparison:

- *GP* refers to a genetic programming-based alpha expression generation method, implemented using the gplearn [27] library. Since the GP method can only optimize the performance of individual alphas rather than entire alpha sets [13], we select the alphas that achieve the highest mean IC's while ensuring that the correlation between any pair of alphas in the set does not exceed 0.7. The performance of this optimized alpha combination is then assessed using the same procedure as the other methods.
- *AlphaGen* [13] is a framework that utilizes reinforcement learning for synergistic alpha set discovery. We adhere to the same hyperparameter settings as the original work, using the open-source implementation referenced in the paper.

In our experiments, we evaluate two approaches for integrating LLM assistance into existing alpha set generation frameworks. Given that the GP method is not suitable for alpha set optimization and has poor generalization capabilities, we select AlphaGen as the backbone framework for this evaluation.

- *AlphaGPT** is a modified version of AlphaGPT [28]. The original AlphaGPT was designed with human interaction in mind and did not prioritize the absolute performance of the results. To enhance its effectiveness, several modifications have been made to the workflow. The initial "seed" alphas are generated using the most capable iterative LLM-only alpha set generator via Algorithm 1, after which the AlphaGen framework is employed to continuously refine this set. This approach yields a more synergistic alpha set than would typically result from a GP-based workflow.
- *HARLA* (Hybrid Alpha Generator with Reinforcement Learning and Large Language Model Assistance) represents our full framework, which integrates the LLM generator into the optimization loop of the RL-based AlphaGen framework. In this method, the initial pool is also generated by the LLM-based generator. Additionally, for every 25k steps in the RL optimization loop, the least significant quarter of the alpha pool is discarded, and the most capable LLM generator is invoked to iteratively enhance the remaining set. This process is repeated until a predefined threshold of total steps is met. For the specifics of the procedure, please refer to Algorithm 2.

The size of the alpha sets for all the methods are set to be 20. Several LLM-only alpha generation strategies are also compared, with details provided in a subsequent section.

4.1.4 Evaluation metrics

In our main experiments, we evaluate the performance of the frameworks using four metrics: (mean) IC, ICIR, (mean) Rank IC, and Rank ICIR. For all these metrics, a higher value indicates better performance.

The primary metric, mean IC, measures the linear correlation between predicted alpha values and actual stock trends, as defined in Subsection 2.3. Complementing this, the mean Rank IC evaluates the correlation in rankings between predictions and actual trends using Spearman's rank correlation coefficient. Since many stock selection strategies prioritize relative rankings over absolute predictions for a given trading day, Rank IC is often considered more relevant in practice. However, our framework optimizes for the linear IC due to its computational advantages: the continuous nature of IC facilitates certain optimization techniques, whereas the relatively discrete nature of Rank IC makes such optimizations less straightforward. Despite this distinction, IC and Rank IC tend to follow similar performance trends, collectively assessing the accuracy of stock trend prediction.

The information ratio (IR) measures the excess return of a portfolio over a benchmark, adjusted for return volatility. However, this classic IR depends on both the stock trend prediction model and the specifics of the trading strategy applied, as it requires direct evaluation of a trading agent's market performance in either simulated or real-world settings. To focus solely on the predictive model and avoid reliance on backtesting or strategy-specific contexts, it is common to use simplified metrics derived from IC. The ICIR extends the IC metric by incorporating consistency, calculated as the daily mean IC over an evaluation period divided by the standard deviation of the individual daily IC values. Similarly, the Rank ICIR applies the same formula to the Rank IC values.

Together, IC and Rank IC assess the predictive accuracy of a model, while ICIR and Rank ICIR also account for the stability of these predictions over the test period. This combination of metrics ensures a comprehensive evaluation of both the predictive power and the robustness of the framework's performance.

4.2 Main results

We first compare the performance of our proposed method against several other alpha set-mining methods. The results are shown in Table 1.

The last six months of the testing dataset exhibit significant changes in market characteristics, which negatively impacts

[2)] The model code is `gpt-3.5-turbo-0125`, with training data up to September 2021.

Table 1 Main results on the CSI300 stock pool. Values without parentheses are the means, while values in parentheses are the corresponding standard deviation across 10 runs with different random seeds

Method	IC	ICIR	Rank IC	Rank ICIR
GP	−0.0048	−0.0300	−0.0154	−0.0762
	(0.0169)	(0.1007)	(0.0238)	(0.1250)
AlphaGen	0.0288	0.1957	0.0321	0.2238
	(0.0090)	(0.0650)	(0.0101)	(0.0809)
LLM Single Step	0.0182	0.1224	0.0283	0.1933
	(0.0185)	(0.1358)	(0.0211)	(0.1594)
LLM Force Replace	0.0248	0.1895	0.0424	0.2904
	(0.0195)	(0.1435)	(0.0287)	(0.1993)
LLM w/ Context	0.0301	0.2165	0.0423	0.2963
	(0.0181)	(0.1247)	(0.0230)	(0.1572)
LLM w/ Updates	0.0335	0.2414	0.0456	0.3122
	(0.0118)	(0.0818)	(0.0182)	(0.1109)
LLM w/o Context	0.0396	0.2889	0.0538	0.3821
	(0.0125)	(0.0990)	(0.0181)	(0.1444)
AlphaGPT*	0.0307	0.2224	0.0321	0.2387
	(0.0153)	(0.1250)	(0.0222)	(0.1832)
HARLA	**0.0515**	**0.3612**	**0.0572**	**0.4104**
	(0.0152)	(0.1040)	(0.0161)	(0.1201)

frameworks susceptible to overfitting on the training set. As shown in the results, the GP-based method performs the worst, confirming that filtering-based alpha selection techniques fail to produce a synergistic alpha combination [13]. The original AlphaGen framework achieves moderate performance, but overfitting on the training set is also observed in our experiments, as evidenced by the best-performing combination getting an mean IC over 0.1 on the training set.

AlphaGPT* is given a headstart using the initial alpha pool generated by the LLM-based system. However, since the LLM-based system is only used for the initial "seeding", those alphas are gradually replaced by those generated by the RL-based generator. This leads to only marginally better performance compared to vanilla AlphaGen.

Last but not least, the complete HARLA framework achieves the best performance among the compared methods, across all four evaluation metrics. The results thus demonstrate that the proposed framework is able to improve upon the baseline both on the accuracy and the stability front. This framework enhances the collaboration between the LLM-based generator and the RL-based generator, effectively altering the RL agent's environment by integrating its own generated alphas into the pool. These small perturbations, inbued with the inherent financial knowledge within the LLM, can help the RL-based generator escape local optima, guiding it toward more effective solutions.

4.3 Comparison of different prompting techniques

We will discuss the performance impact of the different prompting techniques in two parts. The first part will demonstrate how different ways of syntactic specification affect how large of a portion the expressions generated by the LLM is valid. The second part compares how different types of information given to the LLM affect the final performance of a purely LLM-based alpha generating framework.

4.3.1 Experiments on expression validity rate

Designing an effective system prompt is critical for ensuring that the LLM generates valid alpha expressions in the correct format. To evaluate the impact of different prompts on expression validity, we tested four distinct system prompts, each providing varying levels of syntactic guidance. In these experiments, the LLM is tasked with generating 10 alpha expressions per interaction, repeated 150 times for each prompt type. While the output for each test case should be 1,500 expressions, in practice, the LLM occasionally returned fewer or greater than 10 expressions in a few interactions.

The generated expressions are expected to conform to a nested operator format in a "function call" style, i.e., $\mathrm{Op}(\mathrm{args}\ldots)$. For consistency, even basic arithmetic operations are required to adhere to this style (e.g., $\mathrm{Add}(\mathrm{open},\mathrm{close})$) to avoid issues regarding operator precedence. The expressions generated by the LLM are deduplicated and validated with a recursive-descent parser to check

whether they follow the prescribed grammar. Common errors discovered during parsing include unbalanced parentheses, undefined operators or input feature names, and incorrect operator arities or argument types.

The system prompts tested are summarized below. For the complete prompts, please refer to Appendix A.3.1.

- Text: Describes all syntatic requirements using natural language, without providing examples.
- BNF: Utilizes a relaxed form of Backus-Naur Form (BNF) to describe the syntactic requirements.
- Func: Specifies correct operator usage through examples, while all other components are described in natural language like in Text.
- Func + CE: Similar to Func, but includes examples of invalid expressions drawn from common errors made by the LLM alpha generator.

The comparison results are presented in Table 2. The column "Valid% w/o Fix" reports the percentage of alpha expressions that are valid without any additional corrections. Some invalid cases can be corrected automatically by applying operator aliases in the parsing procedure, and such adjusted validity rates are listed in the "Valid% w/ Fix" column. Details on the operator aliasing process can be found in Appendix A.2.

Notably, the Func prompt yields the highest rate of valid expressions, both before and after the corrections. This emphasizes the importance of precise syntactic specifications, as the method relying solely on vague language (Text) obtains the lowest success rate. Additionally, using straightforward examples rather than abstract descriptions is crucial for clarity, evidenced by the slightly poorer performance of BNF compared to Func. Finally, the inferior results of Func + CE compared to Func suggest that providing counterexamples may confuse the LLM, leading to the generation of expressions resembling those invalid examples.

The results demonstrate that well-designed prompts are essential for maximizing the LLM's ability to generate syntactically valid alpha expressions. Providing clear and straightforward examples of correct usage, as in the Func prompt, proved to be the most effective strategy, while counterexamples introduced unintended complexities. These findings emphasize the importance of balancing explicit guidance with simplicity when designing prompts.

4.3.2 Comparison of LLM-based alpha generators

To investigate the impact of different alpha set update strategies and prompting techniques on the performance of a pure LLM-based alpha set generator, we compare results from five distinct strategies. In all experiments, the LLM is instructed to generate three new alphas at each step, or to fill the alpha set to its capacity of 20 if it is not yet full. Unless stated otherwise, the LLM does not receive the conversation history from previous iterations. Basic statistics of the alpha combination, including individual mean IC's and combination model weights for the alphas, are provided to the LLM at each iteration. The results are displayed in the middle section of Table 1. Please refer to Appendix A.3.2 for the entirety of the prompts and templates.

- LLM Single Step: The LLM is asked to generate a set of alphas just once, after which the resulting alpha combination is evaluated for performance.
- LLM Force Replace: The three worst-performing alphas are always dropped at each step and replaced with new alphas generated by the LLM, regardless of the performance of the newly generated alphas.
- LLM w/ Context: The conversation history is always provided to the LLM for reference.
- LLM w/ Updates: The full update history of the alpha pool is made available to the LLM for reference.
- LLM w/o Context: Only the basic statistics of the alpha pool are given to the LLM.

The single-step generator lacks the opportunity to incorporate feedback and iteratively refine the alpha pool, which results in it having the poorest performance among the compared strategies. Despite this limitation, it can still achieve a slight correlation with the prediction target, likely due to the LLM's inherent knowledge of quantitative trading and alpha factors. The force replace strategy performs only marginally better, indicating that an optimization algorithm is essential for effective alpha selection, as the LLM alone cannot reliably discern which alphas are more effective based solely on the alpha set statistics.

Among the other three strategies, the simplest strategy, LLM w/o Context, yields the best results. This outcome suggests that retaining the conversation history can introduce compounding errors, as it may include previously generated invalid expressions, trapping the LLM-based generator in a detrimental feedback loop and causing the iteration process to halt prematurely. While the strategy that includes formatted alpha set updates mitigates this issue, the sheer volume of complex information may overwhelm the LLM, as the effectiveness

Table 2 Comparison of expressions validity rate under different methods of prompting. The values in parentheses stand for the amount of duplicated expressions. Duplicates do not count towards the validity rate calculation

Method	#Expressions	Valid% w/o Fix/%	Valid% w/ Fix/%
Func	1498 (108)	**80.4**	**89.6**
BNF	1499 (122)	77.3	87.1
Func + CE	1530 (113)	75.9	87.6
Text	1500 (100)	69.6	85.4

of previous alpha additions or removals may not translate to the current optimization context.

Given its superior performance, LLM w/o Context serves as the underlying strategy for the full HARLA framework.

4.4 Assessment of computational costs

One critical aspect of evaluating the practicality of the proposed hybrid RL/LLM framework is its computational cost, particularly when compared with the baseline RL-only approach. Table 3 provides a summary of key metrics collected during the training process.

The results indicate that the hybrid framework incurs a modest increase in training time. This is primarily due to periodic LLM invocations, which introduce changes to the alpha pool and, consequently, to the RL agent's environment. These changes dislodge the RL-based alpha generator from its previously achieved stable state, requiring the generation and evaluation of additional alphas. As a result, the total training time and number of alpha evaluations increase.

Despite this, the computational burden introduced by the LLM invocations themselves is relatively low. With each interaction typically completing in a matter of seconds and only a few hundred invocations per experiment, the total time spent on LLM interactions constitutes only a small fraction of the overall training time. The cost-efficiency of the LLM interactions is further underscored by their minimal API cost, averaging less than $0.25 per full experiment. This demonstrates that the integration of LLMs does not impose a significant financial or temporal burden on the framework.

In summary, while the hybrid method entails slightly higher computational costs than the baseline, this added overhead is acceptable when weighed against the substantial gains in alpha quality and diversity. These improvements translate into enhanced predictive performance, underscoring the cost-effectiveness of the proposed approach.

4.5 Case study

We select several representative alpha expressions generated by the RL-based generator and the LLM-based generator respectively, to examine the distinct characteristics of each.

- RL1: Add(Greater(Mul(Greater(2, Add(close,0.01)),−0.5), Greater(Mul(Sub(Min(low, 20d), vwap), high),−2)),0.01)
- RL2: Div(close, Greater(vwap, Log(open)))
- RL3: Sub(Corr(Sub(−2, vwap), volume, 40d), −2)
- RL4: Greater(−1, Mad(Mean(high, 40d), 5d))
- RL5: Min(Add(−5, Log(Max(EMA(Log(open), 40d), 20d))), 10d)
- LLM1: Max(Div(high, low), 20d)
- LLM2: Div(EMA(high, 20d), EMA(low, 10d))
- LLM3: Mean(Mul(0.2, high), 20d)
- LLM4: Greater(WMA(close, 10d), Ref(close, 20d))
- LLM5: Cov(Ref(volume, 10d), close, 30d)

Table 3 Comparison of computational costs between the baseline RL framework (AlphaGen) and the proposed hybrid RL/LLM framework (HARLA)

Comoutational cost	AlphaGen	HARLA
Training time (hours)	7.68 ± 0.82	8.68 ± 1.12
Alpha evaluations	19856 ± 4324	22496 ± 4320
LLM interactions	–	274.8 ± 44.9
User/system tokens	–	380295 ± 76685
Output tokens	–	20258 ± 5290

Upon reviewing these expressions, we observe that the alphas generated by the RL-based generator are often more complex and intricate, sometimes containing redundancies and convoluted structures. This complexity likely stems from the RL-based generator's lack of inherent financial knowledge. It searches freely across the entire space of valid expressions, often uncovering obscure patterns that are difficult to interpret using conventional financial reasoning. However, a set of alphas generated solely in this manner may be more prone to overfitting, as it could capture spurious patterns specific to the training data that do not generalize well to new data.

Conversely, the alphas generated by the LLM-based generator exhibit more structured and interpretable characteristics, drawing from well-established financial principles. For instance, LLM1 calculates the maximum ratio of the high to the low price over the past 20 days, serving as a measure of volatility. Higher values indicate periods of increased price fluctuations. Similarly, LLM5 evaluates the 30-day covariance between the 10-day lagged volume and the closing price, potentially capturing accumulation or distribution trends. These LLM-generated alphas are generally more straightforward for human practitioners to understand but may lack the ability to capture complex relationships hidden in the data.

In conclusion, the RL-based and LLM-based generators complement each other well. While the LLM-based generator leverages its inherent financial knowledge to produce interpretable alphas, the RL-based generator explores the search space more broadly, potentially identifying intricate patterns and relationships that might elude conventional logic. Combining the strengths of both systems yields the most effective results, blending comprehensible financial logic with innovative and data-driven insights.

4.6 Investment simulation

The backtest results, as shown in Fig. 2, highlight the relative performance of various alpha generation methods under a simple trading strategy in a more realistic investing environment. This strategy aims to minimize excessive trading by focusing on the top 50 stocks while capping trades to five stocks at a time.

Among the compared methods, HARLA is shown to be the most effective, generating the largest cumulative excess return over the test period. The figure clearly illustrates that while all methods experience a decline in performance towards the end of the testing period, likely due to a significant change in market conditions, the fully hybrid framework emerges to be the most resilient. It retains the lead it had built during 2022, and suffers the least from the adverse

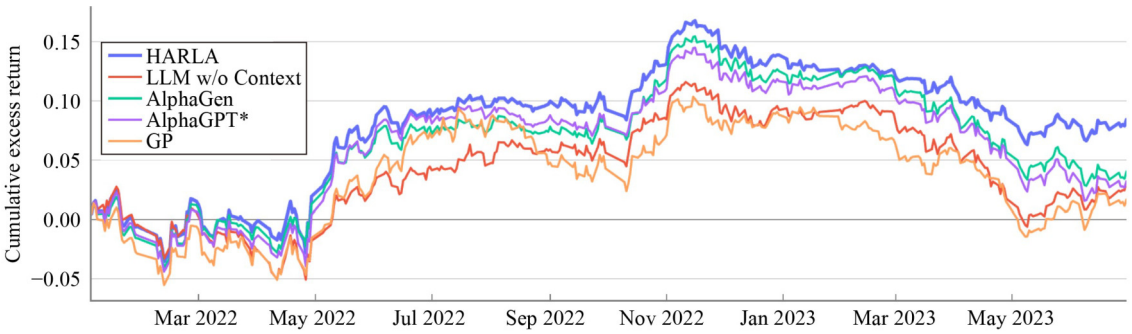

Fig. 2 Backtest results on CSI 300. The lines track the cumulative excess returns of simulated trading agents averaged across multiple runs utilizing the various alpha set generating techniques

market conditions in 2023 compared with other methods. The results prove that combining RL-based exploration with LLM-based financial knowledge is beneficial for robust alpha generation.

5 Related work

5.1 Formulaic alpha

Automatically mining formulaic alphas poses a considerable challenge, mainly because evaluating their performances relies on assessing formulas against stock data —a process that lacks differentiability. Genetic programming methods, skilled at manipulating tree structures like those of mathematical expressions, became early contenders for autonomous alpha generators. Initiatives by Lin et al. [10,11] extend the gplearn library [27] to handle alpha generation, yielding promising results. AutoAlpha [29] utilizes a set of tricks to aid the process of alpha generation, incorporating mutual-IC filtering to enhance the diversity of the generated alphas. AlphaEvolve [12] generates new alphas by evolving upon known alphas, utilizing computational graphs as an intermediate representation instead of expression trees. Recognizing the limitations of frameworks generating individual alphas and subsequently applying filtering for a more diversified alpha pool, AlphaGen [13] is based on a reinforcement learning-based framework. This innovative approach focuses on generating and optimizing a pool of alphas as a whole, resulting in alphas that work more synergistically and significantly improved performance.

5.2 Non-formulaic approaches

Unconstrained by the stringent requirements of explicit formulaic structures, research on non-formulaic alphas has experienced significant progress. Given that stock data inherently exhibit time series characteristics, the application of time-series deep learning models, such as LSTMs [2] and Transformers [30], is suitable for the task of stock trend forecasting. The approach introduced by Zhao et al. [4], known as DoubleAdapt, attempts to leverage online learning techniques to address the challenges posed by distribution shifts. Additionally, methods incorporating more heterogeneous data forms, as proposed by Xu et al. [5,6], aim to enhance existing practices in stock trend forecasting.

Methods that are of simpler forms, for example basic methods like ARIMA [31] or decision tree algorithms like XGBoost [32] and LightGBM [33], have also long been employed in stock trend forecasting. Unlike their "modern" deep learning counterparts, these models are comparatively less complex, and, consequently, they are sometimes considered interpretable. However, the interpretability of an ensemble of decision trees with high tree depths remains a debatable point.

5.3 Applications of LLMs

As the number of parameters in language models scales up, these models absorb an increasingly vast amount of knowledge from extensive human corpora datasets. Surprisingly, capabilities seem to emerge organically as the parameter count grows. Consequently, these newfound capabilities are harnessed for applications extending beyond natural language processing-related fields. These applications encompass tasks such as graph-based reasoning [15,16], data analysis [17], medical information retrieval [18], quantitative trading [19,20], and more.

A notable contribution in the realm of applying LLMs closely related to our work is AlphaGPT [28]. Unlike our focus on utilizing LLMs to enhance the alpha generator's performance, AlphaGPT leverages the natural language comprehension abilities of LLMs to improve the interpretability of alphas. In AlphaGPT, a human user provides instructions to the LLM, which then analyzes the user's intentions and generates initial seeds for further mutations using a GP-based optimizer. The optimized alphas and explanations automatically provided by the LLM are then presented back to the user.

6 Conclusion

In this paper, we explored various methods to leverage the intrinsic financial knowledge within the extensive parameters of LLMs for formulaic alpha generation. First, we proposed a method to instruct the LLM to produce alpha expressions. We employed this approach alongside a combination model in an optimization loop to evaluate the potential of LLMs as standalone alpha generators. Additionally, we presented a strategy to integrate the LLM into the existing state-of-the-art RL-based alpha generation framework for enhancements. Our experiments were conducted on a real-world stock dataset, and utilized numerical correlation analysis and investment simulation results as evaluation metrics. These experiments revealed promising results for LLMs in the alpha generation task. Also, we observed notable performance improvements over existing methods, emphasizing the beneficial impact of the usage of LLMs.

In the future, we plan to explore additional ways to leverage LLMs for alpha mining. Given the extensive financial knowledge embedded in LLMs, we envision employing them beyond direct alpha generation. One potential direction is to more deeply integrate the LLM into the RL update loop: using the LLM's output to guide the reward assignment of the RL agent directly. This would enable the RL agent to learn from the structure of the expressions emitted by the LLM, potentially mitigating the problem of overfitting and enhancing the interpretability of the alphas.

Moreover, the application of LLMs to assist in generating mathematical expressions extends beyond investment-related tasks. We posit that employing similar techniques in other domains requiring interpretable models with explicit formulaic forms represents a promising direction for future research.

Acknowledgements

The research work was supported by the National Key R&D Program of China (No. 2022YFC3303302), the National Natural Science Foundation of China (Grant Nos. 62476263 and U2436209), the Project of Youth Innovation Promotion Association CAS, Beijing Nova Program 20230484430, and the Innovation Funding of ICT, CAS (No. E461060).

Competing interests

The authors declare that they have no competing interests or financial conflicts to disclose.

Appendix A

A.1 List of operators

There are four types of operators used in our framework. The four types break down into two groups: cross-section operators, and time-series operators. Cross-section operators (indicated by "CS" in the table) only deal with data on the current trading day, while time-series operators (indicated by "TS") take into consideration data from a consecutive period of time. Each of the two groups further divides into unary (indicated by "U") and binary (indicated by "B") operators that apply to one or two series respectively. The full list of operators is shown in Table A1.

To ensure more consistent parsing of alpha expressions, the LLM is explicitly instructed to use the arithmetic operators in their functional forms as given above. For example, it is instructed to use $\mathrm{Add}(x,y)$ instead of the more common form of $x+y$.

A.2 Expression parser

For validating and analyzing the formulaic expressions emitted by the LLM, we employ a simple recursive-descent parser that converts the expressions into tree form. The parsing rules are designed to be as lenient as possible, allowing the parser to overlook common error

Table A1 All the operators used in our framework. (CS: cross-section, TS: time-series, U: unary, B: binary)

Operator	Category	Descriptions
$\mathrm{Abs}(x)$	CS–U	The absolute value $\lvert x\rvert$.
$\mathrm{Log}(x)$	CS–U	Natural logarithmic function $\log(x)$.
$\mathrm{Add}(x,y)$, $\mathrm{Sub}(x,y)$, $\mathrm{Mul}(x,y)$, $\mathrm{Div}(x,y)$	CS–B	Arithmetic operators.
$\mathrm{Greater}(x,y)$, $\mathrm{Less}(x,y)$	CS–B	The larger/smaller one of the two values.
$\mathrm{Ref}(x,t)$	TS–U	The expression x evaluated at t days before the current day.
$\mathrm{Mean}(x,t)$, $\mathrm{Med}(x,t)$, $\mathrm{Sum}(x,t)$	TS–U	The mean/median/sum value of the expression x evaluated on the recent t days.
$\mathrm{Std}(x,t)$, $\mathrm{Var}(x,t)$	TS–U	The standard deviation/variance of the expression x evaluated on recent t days.
$\mathrm{Max}(x,t)$, $\mathrm{Min}(x,t)$	TS–U	The maximum/minimum value of the expression x evaluated on the recent t days.
$\mathrm{Mad}(x,t)$	TS–U	The mean absolute deviation $\mathbb{E}[\lvert x-\mathbb{E}[x]\rvert]$ of the expression x evaluated on the recent t days.
$\mathrm{Delta}(x,t)$	TS–U	The relative difference of x compared to t days ago, $x-\mathrm{Ref}(x,t)$.
$\mathrm{WMA}(x,t)$, $\mathrm{EMA}(x,t)$	TS–U	Weighted moving average and exponential moving average of the expression x evaluated on the recent t days.
$\mathrm{Cov}(x,y,t)$	TS–B	The covariance between two time series x and y in the recent t days.
$\mathrm{Corr}(x,y,t)$	TS–B	The Pearson's correlation coefficient between two time series x and y in recent t days.

types typically found in LLM-generated expressions. The parser is case-insensitive and does not enforce the presence of the symbol "d" (days) for time difference constants. For example, "Ema(open, 30d)", "EMA(Open, 30)", "ema(open, 30.)" are all considered valid by the parser, meaning to take the opening price's 30-day exponential moving average (EMA).

Additionally, several operator aliases are also applied to address frequent misunderstandings of operator meanings by the LLM: The operator "Max" is also mapped to "Greater," "Min" to "Less," and "Delta" to "Sub." This aliasing of the operators provides more leniency for the LLM-generated output. For example, the operator $\mathrm{Delta}(x,t)$ accepts a time series and a time delta, computing the difference between the time series and its t-day lagged counterpart; on the other hand, the operator $\mathrm{Sub}(x,y)$ simply accepts two time series and calculates their difference. However, since the two words "delta" and "sub(traction)" are similar semantically, the LLM sometimes mistakes "Delta" for "Sub" and passes it two time series. In this case, the parser processes the sub-expression $\mathrm{Delta}(x,y)$ as if it were $\mathrm{Sub}(x,y)$.

A.3 Prompt templates

A.3.1 System prompts

The system prompt describing the syntactic requirements with a more "natural language" style and showing function signature examples for the operators (Func) is listed below in full.

```
You are an expert quant researcher developing formulaic alphas.

# Specification

The formulaic alphas are expressed as mathematical expressions.
An expression can be a real constant between -30 and 30, an input feature, or
      an operator applied with its operands.
The input features available are: $open, $close, $high, $low, $volume, $vwap.
The operators, their descriptions, and their required operand types are listed
       in the table below. The operands x and y denote expressions, and t
      denotes a time span in days between "1d" and "50d".

Abs(x): absolute value
Log(x): logarithm
Add(x,y): add
Sub(x,y): subtract
Mul(x,y): multiply
Div(x,y): divide
Greater(x,y): larger one of two expressions
Less(x,y): smaller one of two expressions
Ref(x,t): the input expression at t days before
Mean(x,t): mean in the past t days
Sum(x,t): total sum in the past t days
Std(x,t): standard deviation in the past t days
Var(x,t): variance in the past t days
Max(x,t): maximum in the past t days
Min(x,t): minimum in the past t days
Med(x,t): median in the past t days
Mad(x,t): mean Absolute Deviation in the past t days
Delta(x,t): difference of the expression between today and t days before
WMA(x,t): weighted moving average in the past t days
EMA(x,t): exponential moving average in the past t days
Cov(x,y,t): covariance between two time-series in the past t days
Corr(x,y,t): correlation of two time-series in the past t days

Some examples of formulaic alphas:
- Abs(Sub(EMA(open,30d),30.))
- Max(WMA(open,10d),20d)
- Cov(Ref(volume,10d),open,50d)
- Greater(0.1,volume)

## Limits

- You may not need to access any real-world stock data, since I will provide
      you with enough information to make a decision.
- You should give me alphas that are of medium length, not too long, nor too
      short.
- Do not use features or operators that are not listed above.
```

Another system prompt candidate uses a "relaxed form" (the constant ranges are given in natural language) of BNF. Since additional details need to be incorporated into the operator list, we represent it using a table in Markdown format. Below is the specification part, as the remaining instructions are identical to the one listed earlier.

```
The grammar of such formulaic alphas is given in BNF as follows:

alpha ::= expr
expr ::= feature | constant | (unarycsop "(" expr ")") | (binarycsop "(" expr
      "," expr ")") | (unarytsop "(" expr "," timedelta ")") | (binarytsop "("
       expr "," expr "," timedelta ")")
feature ::= "open" | "close" | "high" | "low" | "volume" | "vwap"
timedelta ::= time in days between "1d" and "50d"
constant ::= real number between -30 and 30
unarycsop ::= "Abs" | "Log"
binarycsop ::= "Add" | "Sub" | "Mul" | "Div" | "Greater" | "Less"
unarytsop ::= "Ref" | "Mean" | "Sum" | "Std" | "Var" | "Max" | "Min" | "Med" |
       "Mad" | "Delta" | "WMA" | "EMA"
binarytsop ::= "Cov" | "Corr"

The tokens you have available are:

| Operator | Type | Description |
| --- | --- | --- |
| Abs | unarycsop | Absolute value |
| Log | unarycsop | Logarithm |
| Add | binarycsop | Add |
| Sub | binarycsop | Subtract |
| Mul | binarycsop | Multiply |
| Div | binarycsop | Divide |
| Greater | binarycsop | Pick the greater one of the two expressions |
| Less | binarycsop | Pick the smaller one of the two expressions |
| Ref | unarytsop | The input expression at t days before |
| Mean | unarytsop | Mean in the past t days |
| Sum | unarytsop | Total sum in the past t days |
| Std | unarytsop | Standard deviation in the past t days |
| Var | unarytsop | Variance in the past t days |
| Max | unarytsop | Maximum in the past t days |
| Min | unarytsop | Minimum in the past t days |
| Med | unarytsop | Median in the past t days |
| Mad | unarytsop | Mean Absolute Deviation in the past t days |
| Delta | unarytsop | Difference of the expression between today and t days
      before |
| WMA | unarytsop | Weighted Moving Average in the past t days |
| EMA | unarytsop | Exponential Moving Average in the past t days |
| Cov | binarytsop | Covariance in the past t days |
| Corr | binarytsop | Correlation in the past t days |
```

The version that includes counterexamples of invalid expressions (Func + CE) modifies the demonstration of examples as follows. These counterexamples are taken from typical errors encountered when using the previous prompting methods.

```
Some examples of VALID formulaic alphas:
- Abs(Sub(EMA(open,30d),30.))
- Max(WMA(open,10d),20d)
- Cov(Ref(volume,10d),open,50d)
- Greater(0.1,volume)
Some examples of INVALID formulaic alphas and why they are invalid, please
      AVOID generating alphas like these:
- Abs(Sub(EMA(close,10d),EMA(close,50d)) (Unbalanced parentheses)
- Abs(EMA(close,10d)-EMA(close,50d)) (You must use the operator name "Sub"
      instead of the symbol "-")
- Max(Relog(Div(high,low)),20d) (Relog is not a valid operator name)
```

The latest version, which employs natural language to explain the operand requirements for each operator (Text), introduces the following snippet prior to the full operator list. The operator list itself remains the same as in the version using BNF.

```
Cross-section unary operators (unarycsop) take only one expression operand,
      while cross-section binary operators (binarycsop) take two expression
      operands. Similarly, time-series unary operators (unarytsop) take one
      expression operand and then a time span, and time-series binary
      operators (binarytsop) take two expression operands and a time span.
```

A.3.2 Interaction prompt templates

Using the system prompt mentioned above, the user requests the generation of an initial set of alphas with the instruction prompt P1.

Listing 1 Prompt for generating an initial alpha set.

```
Please generate <n> alphas that you think would be indicative of future stock
      price trend. Each alpha should be on its own line without numbering.
      Please do not output anything else.
```

After the generated alpha set is combined and evaluated, the evaluation results are reported back to the LLM using a structured feedback prompt. An example is given as P2. The feedback prompt follows a consistent template, though some sections may be omitted depending on the selected strategy. The complete user feedback instruction is quite lengthy, so only a representative example of its structure is provided here, with repetitive parts excluded for clarity.

Listing 2 Template for experimental reports to the LLM.

```
Here are a set of formulaic alphas generated by an automated system. The
      update history of the alpha set, together with how the edits influenced
      the IC performance of the set, is listed below:
...
Added alphas:
  Greater(Delta($high,20d),0.1)
  Div(Ref(Var($close,30d),5d),Ref(Std($high,20d),5d))
  Abs(Div(Mul($close,Ref($open,5d)),Mean($volume,10d)))
  Add(Mul($high,Mean($volume,5d)),Ref(Log($low),5d))
Removed alphas:
  Cov($close,$open,5d)
  WMA($high,20d)
  EMA(Add($high,$low),10d)
  Max($volume,20d)
IC of the combination: 0.0530 -> 0.0552 (increment of 0.0022)
...
These alphas are combined with a linear model into the final predictive signal
      . The alphas and the combined signal are tested on real-world dataset,
      and the IC/Rank IC metrics of them, together with the alphas' weights in
       the linear model is reported as follows:
Corr(close,volume,25d): IC = -0.0444, weight = -0.0333 This is a good alpha!
Div(Mean(volume,20d),Mean(volume,50d)): IC = -0.0240, weight = -0.0173 This is
       a good alpha!
...
Mul(high,Ref(low,5d)): IC = 0.0148, weight = 0.0001 This alpha doesn't
      contribute much.
Ensemble: IC = 0.0558, Rank IC = 0.0652
According to the result, please generate three more alphas, not similar to the
       insignificant ones. The most insignificant alphas will be replaced with
       the new ones to potential boost the performance. Again, one on each
      line without numbering, and do not output anything else.
```

References

[1] Kakushadze Z. 101 Formulaic Alphas. Wilmott, 2016, 2016(84): 72−81

[2] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8): 1735−1780

[3] Zhang L, Aggarwal C, Qi G J. Stock price prediction via discovering multi-frequency trading patterns. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2017, 2141−2149

[4] Zhao L, Kong S, Shen Y. DoubleAdapt: A meta-learning approach to incremental learning for stock trend forecasting. In: Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. 2023, 3492−3503

[5] Xu W, Liu W, Wang L, Xia Y, Bian J, Yin J, Liu T Y. HIST: a graph-based framework for stock trend forecasting via mining concept-oriented shared information. 2021, arXiv preprint arXiv: 2110.13716

[6] Xu W, Liu W, Xu C, Bian J, Yin J, Liu T Y. REST: relational event-driven stock trend forecasting. In: Proceedings of the Web Conference 2021. 2021, 1−10

[7] Qian E E, Hua R H, Sorensen E H. Quantitative Equity Portfolio Management: Modern Techniques and Applications. New York: Chapman and Hall/CRC, 2007

[8] Rosenberg B, Marathe V. The prediction of investment risk: systematic and residual risk. Graduate School of Business, University of Chicago, 1975

[9] Fama E F, French K R. Common risk factors in the returns on stocks and bonds. Journal of Financial Economics, 1993, 33(1): 3−56

[10] Lin X, Chen Y, Li Z, He K. Stock alpha mining based on genetic algorithm. Huatai Securities Research Center, 2019

[11] Lin X, Chen Y, Li Z, He K. Revisiting stock alpha mining based on genetic algorithm. Huatai Securities Research Center, 2019

[12] Cui C, Wang W, Zhang M, Chen G, Luo Z, Ooi B C. AlphaEvolve: a learning framework to discover novel alphas in quantitative investment. In: Proceedings of 2021 International Conference on Management of Data. 2021, 2208−2216

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

[14] OpenAI. Introducing ChatGPT. See Openai. com/index/chatgpt/ website, 2022

[15] Han J, Collier N, Buntine W, Shareghi E. PiVe: prompting with iterative verification improving graph-based generative capability of LLMs. In: Proceedings of Findings of the Association for Computational Linguistics: ACL 2024. 2024, 6702−6718

[16] Wang H, Feng S, He T, Tan Z, Han X, Tsvetkov Y. Can language models solve graph problems in natural language? In: Proceedings of the 37th International Conference on Neural Information Processing Systems. 2023, 30840−30861

[17] Luo Y, Tang N, Li G, Chai C, Li W, Qin X. Synthesizing natural language to visualization (NL2VIS) benchmarks from NL2SQL benchmarks. In: Proceedings of 2021 International Conference on Management of Data. 2021, 1235−1247

[18] Agrawal M, Hegselmann S, Lang H, Kim Y, Sontag D. Large language models are zero-shot clinical information extractors. 2022, arXiv preprint arXiv: 2205.12689

[19] Liu X Y, Wang G, Zha D. FinGPT: democratizing internet-scale data for financial large language models. 2023, arXiv preprint arXiv: 2307.10485

[20] Zhang B, Yang H, Zhou T, Ali Babar M, Liu X Y. Enhancing financial sentiment analysis via retrieval augmented large language models. In: Proceedings of the 4th ACM International Conference on AI in Finance. 2023, 349−356

[21] Saravia E. Prompt engineering guide. See Github. com/dair-ai/Prompt-Engineering-Guide website, 2022

[22] OpenAI. Prompt engineering. See platform.openai.com/docs/guides/text website, 2023

[23] Brown T B, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, Agarwal S, Herbert-Voss A, Krueger G, Henighan T, Child R, Ramesh A, Ziegler D M, Wu J, Winter C, Hesse C, Chen M, Sigler E, Litwin M, Gray S, Chess B, Clark J, Berner C, McCandlish S, Radford A, Sutskever I, Amodei D. Language models are few-shot learners. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. 2020, 159

[24] Ouyang L, Wu J, Jiang X, Almeida D, Wainwright C L, Mishkin P, Zhang C, Agarwal S, Slama K, Ray A, Schulman J, Hilton J, Kelton F, Miller L, Simens M, Askell A, Welinder P, Christiano P, Leike J, Lowe R. Training language models to follow instructions with human feedback.

In: Proceedings of the 36th International Conference on Neural Information Processing Systems. 2022, 2011

[25] Touvron H, Lavril T, Izacard G, Martinet X, Lachaux M A, Lacroix T, Roziere B, Goyal N, Hambro E, Azhar F, Rodriguez A, Joulin A, Grave E, Lample G. LLaMA: open and efficient foundation language models. 2023, arXiv preprint arXiv: 2302.13971

[26] Zeng A, Liu X, Du Z, Wang Z, Lai H, Ding M, Yang Z, Xu Y, Zheng W, Xia X, Tam W L, Ma Z, Xue Y, Zhai J, Chen W, Liu Z, Zhang P, Dong Y, Tang J. GLM-130B: an open bilingual pre-trained model. In: Proceedings of the 11th International Conference on Learning Representations. 2023

[27] Stephens T. Genetic programming in python, with a scikit-learn inspired API: gplearn. See Github. com/trevorstephens/gplearn website, 2016

[28] Wang S, Yuan H, Zhou L, Ni L M, Shum H Y, Guo J. Alpha-GPT: human-AI interactive alpha mining for quantitative investment. 2023, arXiv preprint arXiv: 2308.00016

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

[30] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser Ł, Polosukhin I. Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017, 6000−6010

[31] Mashadihasanli T. Stock market price forecasting using the Arima model: an application to Istanbul, Turkiye. Journal of Economic Policy Researches, 2022, 9(2): 439−454

[32] Chen T, Guestrin C. XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016, 785−794

[33] Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q, Liu T Y. LightGBM: a highly efficient gradient boosting decision tree. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017, 3149−3157

Shuo YU is a PhD candidate of Institute of Computing Technology, Chinese Academy of Sciences, and he is also a student at the University of Chinese Academy of Sciences, China. He received his BS degree in Mathematics from University of Science and Technology of China, China. His research interests include stock trend prediction and alpha factor mining.

Hong-Yan XUE is a PhD candidate of Institute of Computing Technology, Chinese Academy of Sciences, and he is also a student at the University of Chinese Academy of Sciences, China. He received his BE degree in Computer Science from University of Chinese Academy of Sciences, China. His research focuses on reinforcement learning and LLM in decision making.

Xiang AO is an associate professor of Institute of Computing Technology, Chinese Academy of Sciences, China. He received his PhD degree in Computer Science from Institute of Computing Technology, Chinese Academy of Sciences, China in 2015, and BS degree in Computer Science from Zhejiang University, China in 2010. His research interests include text and behavioral data mining for financial and business applications.

Qing HE is a professor as well as a doctoral tutor in the Institute of Computing Technology, Chinese Academy of Sciences, China and he is a professor at the University of Chinese Academy of Sciences, China. He received his B.S. degree from Hebei Normal University, China in 1985, and his M.S. degree from Zhengzhou University, China in 1987, both in mathematics. He received his PhD degree in 2000 from Beijing Normal University, China in fuzzy mathematics and artificial intelligence. His interests include data mining, machine learning, classification, and fuzzy clustering.