ALPHAREF: REFERENCE-GUIDED AUTOREGRESSIVE FORMULA GENERATION FOR FORMULAIC ALPHA MINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Discovering predictive formulaic alpha factors from noisy stock-market data is a challenging problem in quantitative finance. Existing symbolic regression methods often search over a vast expression space with limited reuse of established factor structures. Recently, some LLM-based approaches, such as Alpha-GPT, incorporate existing factor knowledge into prompts to guide formula generation. However, effectively using reference formulas to guide generation while preserving the ability to explore beyond them remains a challenge. We propose AlphaRef, a reference-guided autoregressive framework for formulaic alpha mining. AlphaRef converts formulas from the Alpha101 and Alpha158 libraries into reverse Polish notation and progressively masks their suffixes to obtain variable-length prefixes. These prefixes provide generation conditions at different levels of specificity. A Transformer-based generator combines reference prefixes with market data and factor-pool information to autoregressively generate candidate formulas. During training, a reference-decay schedule gradually reduces the use of reference prefixes, enabling a transition from reference-conditioned completion to generation without explicit references. Experiments on the CSI 300, CSI 500, and CSI 1000 stock universes in the Chinese A-share market show that AlphaRef achieves higher mean information coefficients and rank information coefficients than the baselines in the main comparison. Under the evaluated backtesting settings, AlphaRef also achieves the highest Sharpe ratios and compound annual growth rates across all three stock universes. These results support the effectiveness of reference-guided autoregressive generation for formulaic alpha mining.

1 INTRODUCTION

In a dynamic stock trading market, generating excess returns over a benchmark is often the primary goal of many investors and institutions. Researchers typically resort to a variety of methods to forecast the future direction of the stock market and profit from it Qian et al. (2007). However, the inherently low signal-to-noise ratio of the stock market makes it difficult to predict market movements reliably. The process of predicting stock trends from historical stock data and stock-related information is referred to as alpha factor mining Tulchinsky (2019). Therefore, discovering alpha factors that are both sound and persistently predictive is a critical problem in quantitative finance.

Existing alpha mining approaches fall mainly into two paradigms. The first paradigm is based on machine learning and deep learning, including tree-based models such as LightGBM Ke et al. (2017) and XGBoost Chen & Guestrin (2016), as well as deep learning methods proposed in Xie et al. (2025); Zhang et al. (2025b). The methods in this paradigm directly output the prediction results of the returns. These deliver strong predictive performance but operate as opaque black boxes, as their internal mining logic and outputs lack interpretability. The second paradigm recasts alpha mining as a formula-generation problem Huang et al. (2026); Yu et al. (2023): alpha factors are represented as mathematical expressions, which allows researchers to analyze the economic rationale embodied in a formulaic alpha factor and makes the mining process far more transparent. For example, the expression TsMean(volume, 10d) denotes the average trading volume over the most recent ten days. Formulaic alpha research has historically evolved through two stages: manual construction and symbolic regression (SR)-based methods. Early work relied predominantly on manual

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

approaches, where experts derived factor formulas from economic theory, intuition, or statistical analysis Kakushadze (2016). However, manual methods are constrained by individual knowledge breadth and thinking patterns, leading to incomplete exploration of the alpha formula space and low discovery efficiency in complex financial markets. With advances in artificial intelligence, the field shifted to SR-based methods, including genetic programming (GP) and deep learning approaches. GP methods employ evolutionary algorithms on expression trees to automatically search and evolve factor formulas Cui et al. (2021); Zhang et al. (2020). More recent deep-learning-based SR methods Chen et al. (2026); Zhu & Zhu (2025) use metrics such as IC and cross-factor correlation as rewards to guide networks toward higher-probability high-performing factors. Despite their excellent predictive power, these methods share a common limitation: they train from scratch on each stock dataset and fail to exploit knowledge from prior discoveries.

More recently, two branches of work have further advanced formulaic alpha factor mining. In this direction, AlphaCFG Yang et al. (2026) formulates alpha discovery as a tree-structured linguistic Markov decision process over a grammar-defined search space solved by grammar-aware Monte Carlo tree search, whereas TLRS Zhao et al. (2025) shapes dense intermediate rewards by measuring the subsequence-level similarity between partially generated expressions and expert-designed formulas to guide generative policy training. For LLM-based paradigms, works like Alpha-GPT Wang et al. (2023) use established factors as initial references, leveraging prompt engineering and iterative evaluation feedback to guide LLMs in mining and refining factor formulas; AlphaBench Duan et al. (2026) provides a standardized benchmark for systematically evaluating LLM capabilities across the full alpha mining workflow. Both directions have achieved encouraging results. Inspired by them, we propose AlphaRef, an alpha mining framework that uses an existing set of alpha formulas as reference and, through an autoregressive generation approach, produces new alpha factors with stronger predictive power on the basis of the original formulas.

We use Alpha101 Kakushadze (2016) and Alpha158 Yang et al. (2020) as reference libraries. These two sets are widely adopted and rank among the established benchmark formula sets in the field of formulaic alpha factor mining. A further difficult problem is how to incorporate the prior knowledge embedded in the reference formulas into our model. To this end, we draw on the methods of discrete diffusion models Austin et al. (2021); That work introduces a noising forward process that progressively applies masking to discrete sequences. Inspired by these works, we design a masking scheme: factor formulas are first represented as Reverse Polish Notation (RPN) sequences, then masks are progressively added from the end of each formula toward its front to sample reference prefixes of varying lengths. In the generation stage, the model takes an input formula segment, generates one token, appends it to the end of the segment, and feeds the extended segment back into the model, thereby autoregressively producing a complete formula van den Oord et al. (2016). Meanwhile, we also add alpha factor pool encoding and stock data encoding to the encoder, which strengthens AlphaRef's ability to mine alpha factors. Our main contributions can be summarized as follows:

1. We propose AlphaRef, an alpha mining model that discovers formulaic alpha factors through reference-guided autoregressive generation.

2. By sampling formula fragments from the existing set of formulaic alpha factors, we incorporate the corresponding prior knowledge into the model's mining process, enabling the model to leverage such prior knowledge when discovering novel alpha factors.Meanwhile, we introduce a reference decay scheduling strategy during training to gradually reduce the reliance on reference prefixes, which endows the model with the capability to explore beyond the scope of prior knowledge.

3. We conduct extensive experiments to demonstrate the effectiveness of the proposed model AlphaRef. Specifically, experiments on the A-share market show that our model outperforms the baseline models we compare against. Meanwhile, we also compare with the Alpha101 and Alpha158 factor libraries we use, demonstrating that our model can mine more effective factors on the basis of the original factors. Finally, we simulate a simulated trading under the stated assumptions and show that the alpha factors mined by our model are capable of generating excess returns.

2 PRELIMINARIES

2.1 ALPHA FACTOR DEFINITION

We take the stock data as $S \in \mathbb{R}^{N\times K\times T}$, a three-dimensional tensor, where N denotes the number of different stocks, K denotes the number of features (open, close, high, low, volume, and vwap), and T denotes the number of trading days considered. For each trading day $t \in \{1, 2, \ldots, T\}$ and each stock i, there is the historical feature matrix for stock i $s_{it} \in \mathbb{R}^{K\times t}$, which represents the feature sequence of that stock over the past t days. Letting s_{it} be the i-th row of $S_t \in \mathbb{R}^{N\times K\times t}$, an alpha factor can be defined as a function f such that $z^t = f(S_t),\ z_t \in \mathbb{R}^N$, where each entry of the vector z_t is the factor's predicted value for the corresponding stock.

2.2 FORMULAIC ALPHA

A formulaic alpha factor is predictive functional formula that maps daily stock market features to cross-sectional factor values for return forecasting. Raw inputs cover basic quotes such as open, close, and trading volume, which capture historical market information within a rolling window. A formulaic alpha consists of operands (market features and constants) and two classes of operators: fundamental mathematical operators for cross-sectional calculations, and time-series statistical operators that aggregate rolling-window statistics to capture temporal market patterns. Traditional alpha formulas are represented as expression trees, with operands as leaf nodes and operators as internal nodes, which early genetic programming-based factor mining methods rely on. To adapt to deep learning paradigms, recent studies adopt RPN to standardize formula representation Shi et al. (2025). Converting expression trees into RPN sequences via post-order traversal transforms factor mining into a sequence generation task, providing structured and standardized inputs for alpha discovery models.

2.3 GROUP RELATIVE POLICY OPTIMIZATION

Group Relative Policy Optimization (GRPO) Shao et al. (2024) is a critic-free on-policy reinforcement learning algorithm specifically designed for large language model alignment. Unlike standard PPO Schulman et al. (2017) that requires a separate critic network for value estimation, GRPO eliminates the critic module and leverages *group relative reward normalization* to compute advantage estimates, substantially reducing computational and memory overhead for LLM fine-tuning.

Given a policy $\pi_\theta(a \mid s)$ with parameters θ, for each input state s, GRPO samples a group of G responses $\{a_1, \ldots, a_G\}$ with corresponding scalar rewards $\{r_1, \ldots, r_G\}$. The group-based advantage is defined as:

$$\hat{A}_i = \frac{r_i - \mu_r}{\sigma_r + \epsilon} \tag{1}$$

where $\mu_r = \frac{1}{G}\sum_{j=1}^{G} r_j$ and $\sigma_r = \sqrt{\frac{1}{G}\sum_{j=1}^{G}(r_j - \mu_r)^2}$ denote the group reward mean and standard deviation, respectively. ϵ is a small constant for numerical stability.

GRPO adopts the clipped surrogate objective of PPO for stable policy update. Let θ_{old} denote the frozen policy parameters of the last iteration, and the probability ratio $ratio(\theta) = \pi_\theta(a_t \mid s_t)/\pi_{\theta_{\text{old}}}(a_t \mid s_t)$. The final training objective is:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{(s,a)\sim\pi_{\theta_{\text{old}}}}\left[\min\left(ratio(\theta)\hat{A},\ \text{clip}\big(ratio(\theta), 1-\gamma, 1+\gamma\big)\hat{A}\right)\right] \tag{2}$$

where $\hat{A}$: group relative advantage; $\gamma \in (0, 1)$: clipping threshold (typically 0.2) that constrains policy update magnitude; G: group sample size. By normalizing rewards within each group, GRPO focuses on the *relative ranking* of model outputs, which is well-suited for preference-based alignment tasks.

3 METHODS

Our factor mining process mainly consists of three parts: reference preprocessing, factor mining, and factor combination. In the pre-processing part, we use a sequential masking scheme to sample

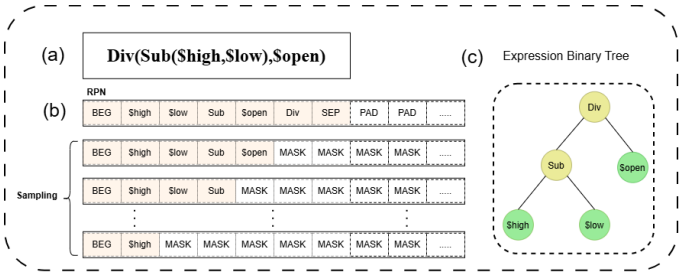


Figure 1: (a) The alpha factor expression. (b) The Reverse Polish Notation of the factor and the sampling procedure for factor formulas. (c) The binary-tree structure of the factor formula.

different reference prefixes from complete alpha formulas, which serve as the reference for the model during factor mining. In the mining part, we adopt a Transformer Vaswani et al. (2017)as the network backbone, and encode the stock data, the factors in the alpha factor pool, and the reference prefixes as network inputs to predict the next token. In the combination part, following the method in Shi et al. (2025), we combine factors dynamically.

3.1 Alpha Formula Sampling

In this paper we adopt a masking scheme that progressively adds masks to a reference formula in order, from the end of the formula toward its front. We call this masking procedure reference sampling and use the term throughout this paper. The method is illustrated in Figure 1. Specifically, we first represent each formula in Reverse Polish Notation and, during encoding, prepend a `BEG` token and append a `SEP` token to the formula to delimit its length. Starting from the position immediately before `SEP`, we add one mask at a time toward the `BEG` position, thereby sampling reference prefixes of different lengths; the final sampled segment contains only `BEG` and its subsequent token. Assuming that a formula together with `BEG` and `SEP` contains N tokens in total, in principle $N-3$ distinct reference prefixes can be sampled.We apply this procedure to partial factor formulas in Alpha101 and all factor formulas in Alpha158. The extracted reference prefixes are first deduplicated to form a candidate pool, from which we randomly sample a subset per generation batch as references for new formula synthesis. This design allows the model to efficiently acquire both the syntax rules and the prior knowledge encoded in the original formulas. Moreover, we design a dynamic decay mechanism for sampled formula references for AlphaRef training. Starting from a specified batch, the proportion of referenced sampled formulas declines every fixed n batches. The reduced portion is replaced by an all mask, an empty reference containing only the BEG token. Eventually, the model generates from scratch without relying on sampled formula references. This mechanism facilitates the discovery of novel alpha factors. Additional details on the utilization of sampled formulas are given in the Appendix D.2.1.

3.2 Alpha Mining

3.2.1 Multi-source Encoding

The overall network architecture and training pipeline of our model are illustrated in Figure 2. The initial inputs to our model consist of three components.First, we sample stock data from the training set, e.g., randomly selecting sequences of 500 distinct stocks within a consecutive 150-day time window. Let $\boldsymbol{S} \in \mathbb{R}^{t' \times n \times k}$ denote the sampled stock data, where n is the number of sampled stocks, t' is the length of the sampled time window, and k denotes the number of stock-specific features.

$$\begin{aligned} \boldsymbol{S}_{\text{flat}} &= \text{Reshape}(\boldsymbol{S}) \in \mathbb{R}^{t' \times (n \cdot k)}, \\ \boldsymbol{O}_s &= \text{LayerNorm}\big(\boldsymbol{S}_{\text{flat}} \boldsymbol{W}_s + \boldsymbol{b}_s\big) \in \mathbb{R}^{t' \times h}. \end{aligned} \tag{3}$$

where $\text{Reshape}(\cdot)$ denotes the tensor-reshaping operation, and $\text{LayerNorm}(\cdot)$ denotes the layer-normalization function. $\boldsymbol{W}_s \in \mathbb{R}^{(n \cdot k) \times h}$, $\boldsymbol{b}_s \in \mathbb{R}^h$, and h is the hidden dimension.

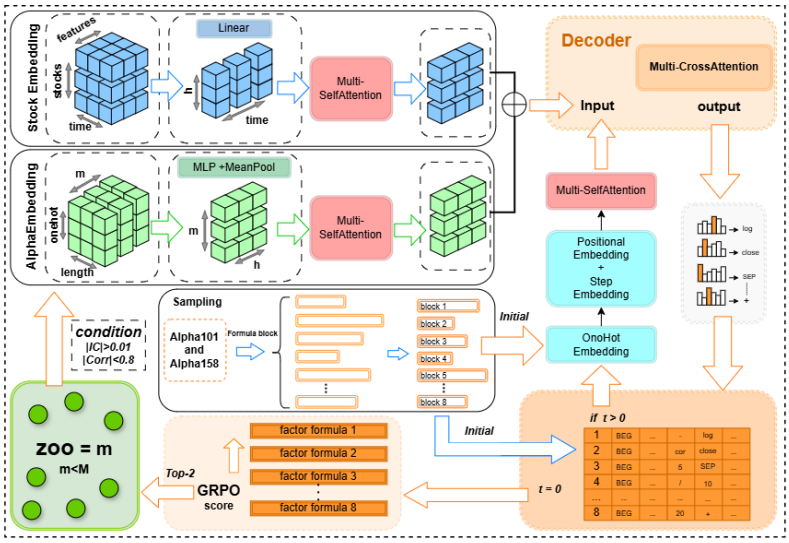


Figure 2: AlphaRef adopts the Transformer as its backbone architecture. Using an autoregressive generation paradigm, the model outputs one token at each step, where each token represents an operator, stock feature, or constant. The generated token is appended to the end of the reference formula segment until a complete alpha formula is formed. After scoring, the top-2 candidates are selected, followed by a judgment on whether to add them to the factor zoo.

Second, factors in the factor zoo are represented via one-hot encoding, and all formulas inside the factor zoo are encoded as model embeddings. Let f_i denote each factor in the zoo, M denote the maximum capacity of the zoo, and m denote the actual number of factors contained in the zoo. Let $\boldsymbol{F} \in \mathbb{R}^{m \times L \times D}$ denote the encoded representations of all factors in the zoo, where L is the maximum formula length and D is the dimension of the one-hot encoding. Note that the zoo contains no factors at initialization, so this input branch is inactive; it takes effect only after factors are added into the zoo.

$$\begin{aligned} \boldsymbol{F}_{\text{pos}} &= \boldsymbol{F} + \boldsymbol{PE}, \quad \boldsymbol{PE} \in \mathbb{R}^{L \times D}. \\ \boldsymbol{H} &= \text{MLP}(\boldsymbol{F}_{\text{pos}}), \quad \in \mathbb{R}^{m \times L \times h}, \\ \boldsymbol{H}_F &= \text{Mean}(\boldsymbol{H}, \text{dim} = 1), \quad \in \mathbb{R}^{m \times h}. \end{aligned} \tag{4}$$

where $\boldsymbol{PE}$ denotes positional encoding, $\text{MLP}(\cdot)$ denotes the multi-layer perceptron, and $\text{Mean}(\cdot)$ denotes the mean-pooling operation. The argument $\text{dim} \in \{0, 1, 2\}$ specifies the dimension along which reduction is performed.

The third component is the encoding procedure for reference prefixes. Let $\text{ref} \in \mathbb{R}^{l \times D}$ denote the sampled reference formula segment, where l is the length of the formula segment.Let x_t denote the token generated when there are t decoding steps remaining, with $0 \leq t < L - l$.

$$\begin{aligned} \mathbf{REF}_t &= \text{concat}\left(\mathbf{ref}, x_{L-l-1}, \ldots, x_{t+1}\right) \in \mathbb{R}^{(L-t+1) \times D}, \\ \mathbf{REF}_{\text{pos,times}} &= \mathbf{REF}_t + \mathbf{PE} + \mathbf{TE}, \quad \mathbf{PE}, \mathbf{TE} \in \mathbb{R}^{(L-t+1) \times D}, \\ \mathbf{REF}_{\text{final}} &= \text{MLP}\left(\mathbf{REF}_{\text{pos,times}}\right), \quad \mathbf{REF}_{\text{final}} \in \mathbb{R}^{(L-t+1) \times h}. \end{aligned} \tag{5}$$

where x_{t+1} denotes the token to be predicted at the current step, a `MASK` placeholder before being sampled. $\boldsymbol{TE}$ denotes the remaining generation step encoding for autoregressive generation. Each of the three encoded branches is fed into a multi-head self-attention module. The outputs from the stock branch and the factor-zoo branch are concatenated and treated as the $\boldsymbol{K}, \boldsymbol{V}$ matrices, while the output from the reference-formula branch serves as the $\boldsymbol{Q}$ matrix for multi-head cross-attention. More details of the model architecture are provided in the Appendix D.1.

3.2.2 GRPO TRAINING PIPELINE

In this work, GRPO is adopted for reinforcement-learning optimization of the factor-generation model. Within-group relative reward signals are constructed based on factor IC and inter-factor correlation metrics to iteratively update the generation policy. Concretely, formulas are categorized according to syntactic and numerical validity, as shown in Table 1.

Table 1: Reward table for candidate formulas. Rewards and penalties are assigned to candidate formulas based on their syntactic validity and degeneracy.

Category	Reward Function
Valid formula	$\big(e^{10\cdot \mathrm{abs_ic}} - 1\big) \cdot \big(0.8 - \mathrm{max_corr}\big)$
Invalid or degenerate formula	$\mathrm{random}(-0.2, -0.1)$

Here, $\mathrm{abs_ic}$ denotes the absolute value of the information coefficient (IC) of the formulaic alpha factor, and $\mathrm{max_corr}$ represents the absolute value of the maximum correlation between the newly generated alpha factor and the existing factors in the factor zoo. The syntactic validity of generated formulas is judged by their computational grammar. Degenerate formulas are defined as formulas that only exhibit stock selection predictive power for a fraction of stocks. The criteria for identifying degenerate formulas are detailed in Appendix D.2.2.

We use score to denote the reward of a formula, computed from its absolute IC and its maximum correlation with the existing zoo factors (see Table 1). The advantage A_i of the i-th sampled formula is the reward standardized within the batch:

$$A_i = \frac{\mathrm{score}_i - \mu_r}{\sigma_r} \tag{6}$$

where score_i is the reward of the i-th formula, and μ_r and σ_r are the mean and standard deviation of the rewards of all B formulas in the batch. To prevent invalid and degenerate formulas from receiving positive advantages, we clip μ_r with a lower bound of -0.1; we also clip σ_r with a lower bound of 0.1 to avoid numerical instability.

To mitigate the accumulated log-probability bias induced by variable formula lengths, we normalize the log-probability of each sampled formula by its effective sequence length. Let $\mathcal{M}_i$ denote the set of masked positions of the i-th formula, i.e., the positions generated by the policy, which exclude the reference-formula prefix, the BEG token, and the PAD padding. The log-probability is summed only over these masked positions:

$$\log \pi_i = \mathrm{lp}_i / l_i, \tag{7}$$

where lp_i is the sum of the log-probabilities over $\mathcal{M}_i$ under a given policy, and l_i is the effective sequence length, i.e., the number of tokens up to the SEP token. This normalization is applied to the new, old, and reference policies, yielding $\log \pi_i^{\mathrm{new}}$, $\log \pi_i^{\mathrm{old}}$, and $\log \pi_i^{\mathrm{ref}}$, respectively.

Let B be the number of formulas generated per batch. We update the model using only the top-n formulas with the highest rewards, and add qualified candidate formulas to the factor zoo. The probability ratio between the updated policy and the old sampling policy is

$$r_i = \exp\Big(\mathrm{clamp}\big(\log \pi_i^{\mathrm{new}} - \log \pi_i^{\mathrm{old}},\, -20,\, 20\big)\Big), \tag{8}$$

where π^{new} denotes the updated policy with trainable parameters θ, and π^{old} denotes the frozen policy snapshot used for sampling at the beginning of the current training step. The clamp operator with thresholds -20 and 20 prevents numerical overflow. The policy loss is then

$$\mathcal{L}_{\mathrm{policy}} = -\frac{1}{|\mathcal{S}_{\mathrm{top}}|} \sum_{i \in \mathcal{S}_{\mathrm{top}}} \min\Big(r_i A_i,\ \mathrm{clip}\big(r_i, 1-\epsilon_{\mathrm{clip}}, 1+\epsilon_{\mathrm{clip}}\big) A_i\Big), \tag{9}$$

where $\mathcal{S}_{\mathrm{top}}$ is the index set of the top-n highest-reward samples within the batch, and $\epsilon_{\mathrm{clip}} = 0.2$ is the clipping coefficient that bounds the magnitude of each policy update.

To regularize the distribution shift between the new policy and the reference policy, we adopt a KL divergence term approximated by the mean log-probability difference:

$$D_{\mathrm{KL}} = \frac{1}{B} \sum_{i=1}^{B} \big(\log \pi_i^{\mathrm{new}} - \log \pi_i^{\mathrm{ref}}\big), \tag{10}$$

where π^{ref} denotes the reference policy. The reference policy is not updated by gradients but is gradually updated via an exponential moving average (EMA) of the main policy π^{new}, which stabilizes training and mitigates mode collapse:

$$\pi^{\text{ref}} \leftarrow \lambda\, \pi^{\text{ref}} + (1-\lambda)\, \pi^{\text{new}}, \quad \lambda = 0.995, \tag{11}$$

where λ is the smoothing coefficient controlling the tracking speed of the reference policy relative to the main policy.

Entropy regularization encourages policy diversity by penalizing overly peaked distributions. The entropy of the new policy is the length-normalized sum of per-token output entropies:

$$\mathcal{H} = \frac{1}{B} \sum_{i=1}^{B} \frac{1}{l_i} \sum_{t \in \mathcal{M}_i} H\big(\pi^{\text{new}}(\cdot \mid s_{i,t})\big), \tag{12}$$

where $s_{i,t}$ denotes the decoding state at the t-th position of the i-th formula, consisting of the stock features, the factor-zoo context, the prefix tokens generated so far, and the decoding-time encoding; $\pi^{\text{new}}(\cdot \mid s_{i,t})$ is the token-vocabulary distribution output by the new policy conditioned on $s_{i,t}$; and $H(\cdot)$ denotes the entropy of a distribution. The normalization by l_i and the average over B remove the influence of variable formula lengths and batch composition.

The overall objective is formulated as

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{policy}} + \alpha\, D_{\text{KL}} - \beta\, \mathcal{H}, \tag{13}$$

where α and β are weighting coefficients ($\alpha = 0.01$, $\beta = 0.01$ in our experiments).

3.3 Factor Combination

For the combination stage, we follow the method proposed in Shi et al. (2025). Instead of fixing a static set of alphas, that framework dynamically re-selects the mined alphas and combines them linearly. Specifically, each day we select the most effective alpha factors based on their historical performance for linear combination. The benefit of this scheme is that both the composition of the combined factors and the weights among factors can be adjusted according to different market conditions, which effectively mitigates factor decay while retaining interpretability. Compared with complex non-linear combinations, this approach strikes a balance between effectiveness and efficiency.

4 EXPERIMENTS

We conduct extensive experiments to evaluate the performance of our proposed model, aiming to answer four questions: **Q1:** Can AlphaRef discover predictive alpha factors? **Q2:** Does our model mine factors that are higher-quality on the basis of the reference formulas? **Q3:** Are the individual modules we propose in the model effective? **Q4:** Can the alpha factors mined by our model generate returns in the evaluated backtesting setting?

4.1 Experimental Setup

Datasets. We use the daily real-market data of all A-share stocks, with six features: daily open, close, high, low, volume, and vwap. The target is the 20-day holding-period return, computed as $\text{Ref}(\text{close}, -20)/\text{close} - 1$, where $\text{Ref}(\text{close}, -20)$ denotes the closing price 20 days in the future. The dataset is chronologically divided into a training set (2010-01-01 to 2020-12-31), a validation set (2021-01-01 to 2022-06-30), and a test set (2022-07-01 to 2025-06-30). Our experiments focus on the constituent stocks of the CSI 300, CSI 500, and CSI 1000 stock universes, on which we conduct comprehensive experiments.

Baselines.

1. Machine Learning-based Methods: These methods directly forecast stock returns without explicit alpha formulas, using 250-day raw features to predict 20-day forward returns. **LSTM:** Hochreiter & Schmidhuber (1997). **MLP:**Rumelhart et al. (1986). **XGBoost:**Chen & Guestrin (2016). **LightGBM:**Ke et al. (2017).

2. Symbolic-regression-based Methods: These methods produce alpha factors as interpretable mathematical formulas. **GP:**Koza (1992). **AlphaForge:** Shi et al. (2025). **AlphaGen:** Yu et al. (2023). **AlphaSAGE:** Chen et al. (2026). **AlphaQCM:** Zhu & Zhu (2025).

Evaluation metrics. We evaluate the model performance using seven widely adopted quantitative metrics, including IC, ICIR, Rank IC, Rank ICIR, Sharpe Ratio (SR), Compound Annual Growth Rate (CAGR), and Maximum Drawdown (MDD). Detailed descriptions of the baseline models and evaluation metrics are provided in Appendix E.

4.2 Main Results

We conduct a series of experiments on the CSI 300, CSI 500, and CSI 1000 stock universes and evaluate the model's performance with various metrics.

To answer **Q1**, we compare our proposed model with a set of baseline models. As shown in Table 2, AlphaRef generally outperforms the baseline models across the experimental settings.

Table 2: Performance comparison on three market universes. Bold and underlined numbers denote the best and second-best results, respectively.

Predictive Metrics												
Method	CSI300				CSI500				CSI1000			
	IC$^{\uparrow}$ (%)	ICIR$^{\uparrow}$	RankIC$^{\uparrow}$ (%)	RankICIR$^{\uparrow}$	IC$^{\uparrow}$ (%)	ICIR$^{\uparrow}$	RankIC$^{\uparrow}$ (%)	RankICIR$^{\uparrow}$	IC$^{\uparrow}$ (%)	ICIR$^{\uparrow}$	RankIC$^{\uparrow}$ (%)	RankICIR$^{\uparrow}$
MLP	4.18	0.304	5.35	0.381	5.05	0.402	6.08	0.530	4.93	0.546	5.10	0.593
LSTM	2.26	0.213	2.61	0.221	4.74	0.420	4.73	0.431	4.69	0.567	5.44	0.603
XGBoost	2.23	0.173	1.77	0.128	4.21	0.378	4.25	0.393	6.06	0.532	6.42	0.627
LightGBM	1.99	0.183	2.68	0.255	3.69	0.343	3.56	0.346	4.91	0.504	5.25	0.538
GP	4.62	0.327	6.30	0.366	3.33	0.248	6.69	0.409	4.64	0.375	10.53	0.673
AlphaGen	4.41	0.395	5.93	0.418	4.61	0.423	8.04	0.570	5.37	0.507	10.14	0.753
AlphaForge	5.09	0.405	6.44	0.454	4.21	0.320	6.99	0.477	6.78	0.521	11.12	0.748
AlphaQCM	5.72	0.464	6.82	0.457	6.35	0.407	9.16	0.628	7.02	0.571	10.64	0.742
AlphaSAGE	5.06	0.359	6.63	0.443	4.69	0.357	8.41	0.583	7.21	0.593	11.50	0.817
Ours	**8.13**	**0.486**	**10.02**	**0.562**	**6.98**	**0.446**	**11.01**	**0.685**	**9.02**	**0.617**	**13.77**	**0.872**

Method	CSI300			CSI500			CSI1000		
	SR$^{\uparrow}$	CAGR$^{\uparrow}$ (%)	MDD$^{\downarrow}$ (%)	SR$^{\uparrow}$	CAGR$^{\uparrow}$ (%)	MDD$^{\downarrow}$ (%)	SR$^{\uparrow}$	CAGR$^{\uparrow}$ (%)	MDD$^{\downarrow}$ (%)
MLP	0.359	6.08	34.96	0.430	7.92	32.76	0.266	3.56	36.98
LSTM	0.368	6.51	31.26	0.545	10.48	31.96	0.545	11.04	33.76
XGBoost	0.345	5.83	35.53	0.496	9.84	34.68	0.529	11.03	35.77
LightGBM	0.296	4.45	34.44	0.389	6.76	33.93	0.438	8.24	34.94
GP	0.468	8.90	33.78	0.407	7.20	34.30	0.571	11.79	33.68
AlphaGen	0.505	9.48	**30.30**	0.497	9.52	32.10	0.617	12.86	33.14
AlphaForge	0.452	8.28	32.55	0.447	8.22	33.84	0.607	12.68	**32.98**
AlphaQCM	0.472	9.16	32.40	0.595	12.41	**31.71**	0.635	13.06	34.34
AlphaSAGE	0.444	8.30	34.37	0.470	8.90	34.16	0.613	12.90	34.01
Ours	**0.614**	**13.12**	34.38	**0.616**	**12.60**	32.07	**0.672**	**14.67**	33.41

To answer **Q2**, we evaluate the performance of the original factors from Alpha101 and Alpha158 on our test set using the same combination methodology. As shown in Table 3, under the same dynamic OLS combination, AlphaRef consistently outperforms the reference formula set, demonstrating its ability to mine more effective alpha factors. For a more detailed comparison between AlphaRef and the reference formula set, please refer to Appendix F.

Table 3: Performance comparison on the CSI 300, CSI 500 and CSI 1000. Bold and underlined numbers denote the best and second-best results, respectively.

Method	CSI300				CSI500				CSI1000			
	IC$^{\uparrow}$(%)	ICIR$^{\uparrow}$	RankIC$^{\uparrow}$(%)	RankICIR$^{\uparrow}$	IC$^{\uparrow}$(%)	ICIR$^{\uparrow}$	RankIC$^{\uparrow}$(%)	RankICIR$^{\uparrow}$	IC$^{\uparrow}$(%)	ICIR$^{\uparrow}$	RankIC$^{\uparrow}$(%)	RankICIR$^{\uparrow}$
Alpha101	4.75	0.357	6.43	0.467	4.19	0.375	7.33	0.622	5.13	0.494	8.22	0.700
Alpha158	4.52	0.286	6.51	0.366	2.32	0.178	4.26	0.291	4.83	0.399	8.08	0.562
Alpha101+158	5.32	0.333	7.30	0.411	2.76	0.210	5.04	0.345	5.48	0.450	8.87	0.631
Ours	**8.13**	**0.486**	**10.02**	**0.562**	**6.98**	**0.446**	**11.01**	**0.685**	**9.02**	**0.617**	**13.77**	**0.872**

4.3 Effects of the Different Modules of AlphaRef

To answer **Q3**, we conduct ablation studies on each component to verify the contributions of the stock market data embedding module, the alpha factor zoo formula embedding module, and the reference formula encoding module. For the reference formula module, we remove formula references entirely so that the model explores and generates alpha formulas from scratch. As shown in Figure 3, every component brings performance gains to AlphaRef. Among them, the reference formula

module yields the most prominent performance improvement. This result verifies the effectiveness of our proposed AlphaRef.

4.4 Performance under Simulated Backtesting

To answer **Q4**, we conduct backtests on CSI 300 over the same time period as our test set, and the results are shown in Figure 4. Aligned with our prediction target, we construct 20 sub-portfolios for backtesting, where each sub-portfolio initiates trades on a distinct day within the first 20 trading days. This holding scheme reduces sensitivity to the choice of portfolio initiation date, accounts for transaction costs, and remains consistent with our prediction objective. Specifically, one sub-portfolio is launched on each of the first 20 trading days. For each sub-portfolio, we select the top 20% ranked stocks by the combined alpha signal for holding positions, and liquidate all positions after a 20-trading-day holding period. This procedure repeats in subsequent cycles until fewer than 20 trading days remain. We set the transaction cost to 0.05%.

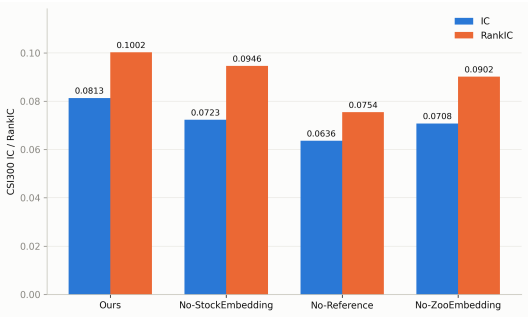


Figure 3: **No-StockEmbedding** denotes the ablation of stock data embedding.**No-ZooEmbedding** denotes the ablation of embeddings for factors from the factor zoo.**No-Reference** means that no reference formulas are utilized during model training.

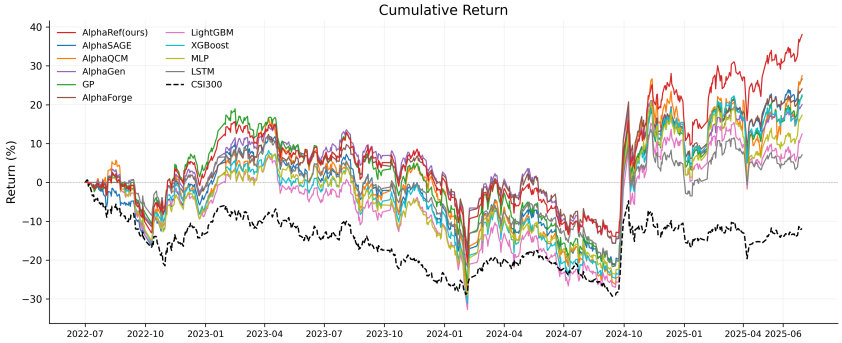


Figure 4: Backtest results on CSI 300.

We validate the masking mechanism employed
in our framework through comparative experiments against a random subtree masking baseline on syntax trees. To further assess the cross-market applicability of our method, we retrain AlphaRef on U.S. equity data and evaluate it on the S&P 500 constituent dataset. Complete results of both experiments are provided in Appendix G.

5 Conclusion

In this paper, we propose AlphaRef, a framework that generates alpha factors in an autoregressive, reference-guided manner. Instead of generating factors from scratch, AlphaRef leverages known formulaic alphas as reference-prefix conditioning to encourage the discovery of candidate factor expressions. Our proposed reference sampling method provides a novel way to incorporate prior knowledge from the reference library into the model and offers a new perspective on alpha factor mining. Extensive experiments show that our proposed AlphaRef outperforms existing methods on real-world stock market datasets in terms of key evaluation metrics. The simulated trading backtests also demonstrate its effectiveness and practical value in real quantitative investment applications.

Reproducibility Statement

To ensure the reproducibility of the results presented in this work, we have made we have efforts to document our methodology and provide well-organized experimental resources. The complete training pipeline of AlphaRef, including the reward function, GRPO loss, and factor zoo admission rules, is elaborated in detail in both the main text and the appendix. Detailed specifications of all baseline models and evaluation metrics are available in Appendix E. We have submitted the full anonymous source code as supplementary material, which includes the implementation of AlphaRef, all baseline methods, the factor evaluation toolkit, and the complete scripts for model training and result evaluation. The data preprocessing pipeline and full experimental hyperparameter configurations are also documented in the supplementary materials, allowing independent replication of all results reported in this paper.

References

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, volume 34, pp. 17981–17993, 2021.

Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 936–945, 2021.

Kathleen Champion, Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116 (45):22445–22451, 2019. doi: 10.1073/pnas.1906995116.

Binqi Chen, Hongjun Ding, Ning Shen, Jinsheng Huang, Taian Guo, Luchen Liu, and Ming Zhang. Alphasage: Structure-aware alpha mining via gflownets for robust exploration. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=zRKF4ln2VE.

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

Can Cui, Wei Wang, Meihui Zhang, Gang Chen, Zhaojing Luo, and Beng Chin Ooi. Alphaevolve: A learning framework to discover novel alphas in quantitative investment. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*, pp. 2208–2216, 2021. doi: 10.1145/3448016.3457324. arXiv:2103.16196.

Yitong Duan, Yu Shi, Tianping Zhang, and Jian Li. Alphabench: Benchmarking large language models in formulaic alpha factor mining. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=YvXQ3sQ04n.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

Haotong Huang, Jie Peng, Zezhen Ding, Pingzhi Li, and Tianlong Chen. Alphaformer: End-to-end symbolic regression of alpha factors with transformers. In *Conference on Parsimony and Learning*, volume 328, pp. 61–82. PMLR, 2026. URL https://proceedings.mlr.press/v328/huang26a.html.

Zura Kakushadze. 101 formulaic alphas. *Wilmott*, (84):72–81, 2016. doi: 10.1002/wilm.10525.

Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA, 1994.

William La Cava, Lee Spector, and Kourosh Danai. ϵ-lexicase selection for regression. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '16, pp. 741–748, 2016. doi: 10.1145/2908812.2908898.

Mikel Landajuela, Chak Shing Lee, Jiachen Yang, Ruben Glatt, Claudio P. Santiago, Ignacio Aravena, Terrell Mundhenk, Garrett Mulcahy, and Brenden K. Petersen. A unified framework for deep symbolic regression. In *Advances in Neural Information Processing Systems*, volume 35, pp. 33985–33998, 2022.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: 10.1038/nature14539.

Xiaoming Lin, Ye Chen, Ziyu Li, and Kang He. Stock alpha mining based on genetic algorithm. Technical report, Huatai Securities Research Center, 2019a. Huatai Artificial Intelligence Series, Jun. 10, 2019.

Xiaoming Lin, Ye Chen, Ziyu Li, and Kang He. Revisiting stock alpha mining based on genetic algorithm. Technical report, Huatai Securities Research Center, 2019b. Huatai Artificial Intelligence Series, Aug. 7, 2019.

Alberto Moraglio, Krzysztof Krawiec, and Colin G. Johnson. Geometric semantic genetic programming. In *Parallel Problem Solving from Nature – PPSN XII*, volume 7491 of *Lecture Notes in Computer Science*, pp. 21–31, 2012. doi: 10.1007/978-3-642-32937-1_3.

E. E. Qian, R. H. Hua, and E. H. Sorensen. *Quantitative Equity Portfolio Management: Modern Techniques and Applications*. CRC Press, 2007.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0.

Michael D. Schmidt and Hod Lipson. Age-fitness pareto optimization. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pp. 543–544, 2010. doi: 10.1145/1830483.1830584.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Hao Shi, Weili Song, Xinting Zhang, Jiahe Shi, Cuicui Luo, Xiang Ao, Hamid Arian, and Luis Angel Seco. Alphaforge: A framework to mine and dynamically combine formulaic alpha factors. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 12524–12532, 2025. doi: 10.1609/aaai.v39i12.33365. arXiv:2406.18394.

Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan K. Reddy. Transformer-based planning for symbolic regression. In *Advances in Neural Information Processing Systems*, volume 36, pp. 45907–45919, 2023.

Igor Tulchinsky. *Finding Alphas: A Quantitative Approach to Building Trading Strategies*. John Wiley & Sons, 2019.

Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, pp. 2603–2611. PMLR, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Saizhuo Wang, Hang Yuan, Leon Zhou, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. Alpha-GPT: Human-AI interactive alpha mining for quantitative investment. *arXiv preprint arXiv:2308.00016*, 2023.

Yanzhen Xie, Li Huang, Xueqin Chen, Qiang Gao, Guisong Liu, and Kunpeng Zhang. Progressive dependency representation learning for stock ranking in uncertain risk contrasting. In *Proceedings of the 31st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2025. doi: 10.1145/3690624.3709189.

Wentao Xu, Weiqing Liu, Lewen Wang, Yingce Xia, Jiang Bian, Jian Yin, and Tie-Yan Liu. HIST: A graph-based framework for stock trend forecasting via mining concept-oriented shared information, 2021.

Han Yang, Dong Hao, Zhuohan Wang, Qi Shi, and Xingtong Li. Alpha discovery via grammar-guided learning and search, 2026.

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

Shuo Yu, Hongyan Xue, Xiang Ao, Feiyang Pan, Jia He, Dandan Tu, and Qing He. Generating synergistic formulaic alpha collections via reinforcement learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23)*, 2023. doi: 10.1145/3580305.3599831. arXiv:2306.12964.

Hengzhe Zhang, Qi Chen, Bing Xue, Wolfgang Banzhaf, and Mengjie Zhang. RAG-SR: Retrieval-augmented generation for neural symbolic regression. In *International Conference on Learning Representations*, 2025a.

Liheng Zhang, Charu Aggarwal, and Guo-Jun Qi. Stock price prediction via discovering multi-frequency trading patterns. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pp. 671–679, 2017. doi: 10.1145/3097983.3098117.

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

Xu Zhang, Zhengang Huang, Yunzhi Wu, Xun Lu, Erpeng Qi, Yunkai Chen, Zhongya Xue, Qitong Wang, Peng Wang, and Wei Wang. Multi-period learning for financial time series forecasting. In *Proceedings of the 31st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2025b. doi: 10.1145/3690624.3709422.

Junjie Zhao, Chengxi Zhang, Chenkai Wang, and Peng Yang. Learning from expert factors: Trajectory-level reward shaping for formulaic alpha mining, 2025.

Canmiao Zhou and Han Huang. Syntax-aware retrieval augmentation for neural symbolic regression. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13137–13147, 2025.

Zhoufan Zhu and Ke Zhu. Alphaqcm: Alpha discovery in finance with distributional reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 80463–80479. PMLR, 2025.

Hossein Zolfagharinia, Mehdi Najafi, Shamir Rizvi, and Aida Haghighi. Unleashing the power of tweets and news in stock-price prediction using machine-learning techniques. *Algorithms*, 17(6): 234, 2024. doi: 10.3390/a17060234.

A THE USE OF LARGE LANGUAGE MODELS (LLMS)

Large language models (LLMs) were used for linguistic polishing and revision of the manuscript, and for verifying the consistency between the manuscript text (including formulas, notations, and reported results) and the implementation. LLMs were not involved in the research design, the derivation of algorithms, the implementation or execution of experiments, the scientific analysis of empirical results, or the generation of any research ideas. All intellectual contributions and experimental findings are entirely the authors' own.

B RELATED WORK

Machine Learning-based Alpha Factors. A large body of work treats each stock as an independent time series and regresses its return from its own history. Early work applies generic approximators — multilayer perceptrons LeCun et al. (2015), gated recurrent networks such as the LSTM Hochreiter & Schmidhuber (1997) and Transformer encoders Vaswani et al. (2017), while gradient-boosted trees such as XGBoost Chen & Guestrin (2016) and LightGBM Ke et al. (2017) remain strong on fixed hand-designed factor sets. A second wave injects finance-specific structure into the model: the State Frequency Memory network Zhang et al. (2017) decomposes the memory cell into discrete frequency channels via a Discrete Fourier Transform, capturing trading patterns at several periodicities. Others widen the information set instead. Zolfagharinia et al. (2024) augment technical indicators with Twitter and news coverage counts, with gains peaking during market-panic periods, and Xu et al. (2021) use a stock–concept graph to separate each stock's shared from its idiosyncratic return components. These models deliver strong predictive performance, but their forecasts stem from opaque parameterizations, making it difficult to isolate the economic rationale behind any individual signal — a property that formulaic alphas inherently preserve by their explicit structural design.

Formulaic Alpha Factors. A formulaic alpha expresses the signal as an explicit formula over operators and raw features; its search space is combinatorially vast, so mining is a program-search problem. Genetic programming was the second theme, encoding candidates as syntax trees refined by mutation and crossover Koza (1994). The Huatai technical reports adapted this to equity signals by extending gplearn with time-series operators and embedding neutralisation and single-factor testing into the evolutionary loop Lin et al. (2019a); a follow-up replaced the RankIC fitness with the mutual information between factor and forward return, recovering nonlinear relationships, and added a validation split against overfitting Lin et al. (2019b). Diversity became the another theme: AutoAlpha grows expressions from a gene pool of effective sub-expressions and screens redundancy via each alpha's leading principal component Zhang et al. (2020), while AlphaEvolve raises the representation to computation graphs, admitting matrix-valued and cross-sectional operators Cui et al. (2021). Later work replaces hand-designed variation with a learned generator. AlphaGen emits alphas as Reverse Polish Notation token sequences and trains a recurrent policy by reinforcement learning, rewarding the downstream combination model rather than any single alpha Yu et al. (2023). AlphaForge makes that reward differentiable through a predictor network approximating factor fitness, then re-weights factors over time as their predictive power decays Shi et al. (2025). AlphaFormer instead learns to generate rather than search: an encoder–decoder Transformer pre-trained on synthetic stock datasets translates raw market data directly into an alpha pool in a single end-to-end inference, with no task-specific training Huang et al. (2026). However, properly leveraging prior knowledge from existing factors while mining novel factors beyond the scope of such prior knowledge remains a challenging open problem.

Symbolic Regression. Symbolic regression seeks a compact mathematical expression that explains observed data, optimising structure and parameters jointly; because its output is a closed-form formula rather than a fitted black box, it is one of the few interpretable machine-learning paradigms. Early work improved the genetic programming search that long dominated the field: La Cava et al. (2016) replaced tournament selection with ϵ-lexicase selection, which handles the continuous error spaces of regression, Schmidt & Lipson (2010) treated age as an explicit Pareto objective to sustain diversity, and Moraglio et al. (2012) introduced geometric semantic operators that act directly on program semantics. Neural approaches later recast the task as learning rather than search. Champion et al. (2019) coupled an autoencoder with sparse regression to recover governing equations in learned coordinates, Biggio et al. (2021) showed that a Transformer pre-trained on procedurally generated equations can infer expressions directly from data, and Landajuela et al. (2022) unified

pre-training, neural-guided search and genetic programming into a single modular framework. Subsequent work strengthens the decoder: Kamienny et al. (2022) predict the full expression, constants included, end to end, and Shojaee et al. (2023) embed Transformer decoding in Monte Carlo tree search so that non-differentiable feedback on accuracy and complexity can guide generation. A more recent thread injects reusable prior knowledge rather than searching from scratch: Zhang et al. (2025a) retrieve expressions and subtrees from a symbolic library to augment a neural generator, and Zhou & Huang (2025) retrieve by syntactic structure so that non-parametric prior information complements the generator's fixed parametric knowledge. Alpha factor mining is the finance instance of this problem, with expressions built over price–volume operators and judged by how well the resulting signal correlates with forward returns.

C LIST OF OPERATORS, FEATURES, AND CONSTANTS

Table 4: Raw features and operators. F: base market features; U/B/T: unary/binary/ternary operators; CS: cross-sectional operation (within-day across assets); TS: time-series rolling operation. The lookback length d denotes the past d trading days.

Name	Type	Description
Open	F	Opening price
Close	F	Closing price
High	F	Daily highest price
Low	F	Daily lowest price
Volume	F	Trading volume (number of shares)
Vwap	F	Daily average price, weighted by the volume of trades at each price
Abs	U	Absolute value of the input
SLog1p	U	Signed log1p transform: sign(input) times $\log(1 + \lvert\text{input}\rvert)$
Inv	U	Reciprocal of the input
Sign	U	Sign of the input, returning -1, 0, or 1
Log	U	Natural logarithm of the input
Rank	U-CS	Cross-sectional rank normalization within a day, mapped to the range $[0, 1]$
Scale	U-CS	Cross-sectional scaling transformation within a day across assets
Add	B	Element-wise addition of two inputs
Sub	B	Element-wise subtraction: first input minus second input
Mul	B	Element-wise multiplication
Div	B	Element-wise division
Pow	B	Element-wise power: raise the first input to the power of the second
Greater	B	Element-wise maximum of two inputs
Less	B	Element-wise minimum of two inputs
CmpLess	B	Element-wise comparison: 1 if first input is less than second, else 0
CmpGreater	B	Element-wise comparison: 1 if first input is greater than second, else 0
Or	B	Logical OR operation between two boolean inputs
GetGreater	B	Element-wise maximum of two inputs
SignedPower	B	Signed power: $\text{sign}(x) \cdot \lvert x\rvert^a$, where a is the second input
Ref	U-TS	Lag operator: the value from d days ago
TsMean	U-TS	Rolling mean over the past d days
TsSum	U-TS	Rolling sum over the past d days
TsStd	U-TS	Rolling standard deviation over the past d days
TsIr	U-TS	Rolling information ratio: rolling mean divided by rolling standard deviation over the past d days
TsMinMaxDiff	U-TS	Rolling range over the past d days (rolling max minus rolling min)
TsMaxDiff	U-TS	Current value minus the rolling max over the past d days
TsMinDiff	U-TS	Current value minus the rolling min over the past d days
TsVar	U-TS	Rolling variance over the past d days
TsSkew	U-TS	Rolling skewness over the past d days
TsKurt	U-TS	Rolling kurtosis over the past d days
TsMax	U-TS	Rolling maximum over the past d days

Continued on next page

Name	Type	Description
TsMin	U-TS	Rolling minimum over the past d days
TsMed	U-TS	Rolling median over the past d days
TsMad	U-TS	Rolling mean absolute deviation over the past d days
TsRank	U-TS	Within-window rank of the input over the past d days
TsDelta	U-TS	Current value minus the first value within the past d days
TsDiv	U-TS	Current value divided by the rolling mean over the past d days
TsPctChange	U-TS	Percentage change over the past d days
TsWMA	U-TS	Rolling weighted moving average over the past d days
TsEMA	U-TS	Rolling exponential moving average over the past d days
DecayLinear	U-TS	Linearly weighted decaying aggregation over the past d days
TsArgMax	U-TS	Index position of the maximum value within the past d days
TsArgMin	U-TS	Index position of the minimum value within the past d days
Product	U-TS	Rolling product over the past d days
TsCov	B-TS	Rolling covariance between two input series over the past d days
TsCorr	B-TS	Rolling Pearson correlation coefficient between two input series over the past d days
If	T	Ternary conditional selection: returns the second input if condition holds, otherwise returns the third input
Time windows: $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 30, 40, 50, 60, 100, 120, 150, 180, 200, 230, 250\}$. **Constants**: $\{10^{-6}, 0.01, 0.5, 1.0, 2.0, 5.0, 10.0, 30.0, 0.0, -0.01, -0.5, -1.0, -2.0, -5.0, -10.0, -30.0\}$.		

D DETAILS OF MODEL

D.1 MODEL STRUCTURE

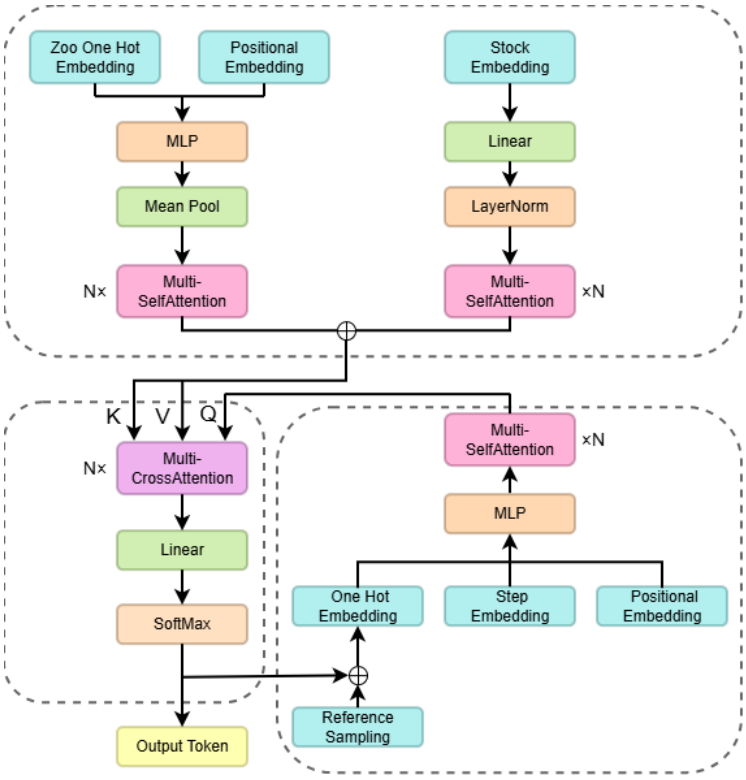


Figure 5: Model architecture.

The detailed architecture of our model is illustrated in Figure 5. For each training batch, we randomly sample 500 stocks and sample a 150-trading-day window for 500 stocks from the training set

as the encoded inputs to the model. We set the size of the factor zoo pool to 20, where each formula in the pool is represented by one-hot encoding, and the maximum sequence length of formulas is set to 64. Regarding the inputs for formula sampling, two types are considered: sampled reference prefixes obtained from previous steps, and reference samples consisting solely of the BEG token with all padding. Both types are encoded via one-hot representation. We elaborate on how the formula references are utilized during training in the subsequent subsection. The encoding pipelines for these three branches have been described in detail in Section 3.2.1.

To model long-range temporal dependencies and inter-correlations among different financial sequential features, we employ the multi-head self-attention (MHSA) mechanism for hierarchical feature enhancement. Compared with single-head attention, multi-head self-attention projects inputs into multiple independent subspaces, capturing diverse semantic and temporal patterns simultaneously, which is designed to capture the representation ability of financial sequence data.

The standard formulation of multi-head self-attention is defined as:

$$\begin{aligned} &\text{MultiHead}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \text{Concat}(\text{Head}_1,\text{Head}_2,\ldots,\text{Head}_8)\boldsymbol{W}_O, \\ &\text{Head}_i = \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_{Q_i},\boldsymbol{K}\boldsymbol{W}_{K_i},\boldsymbol{V}\boldsymbol{W}_{V_i}), \\ &\text{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V}, \end{aligned} \tag{14}$$

where d denotes the dimension of each head, $\boldsymbol{W}_{Q_i}, \boldsymbol{W}_{K_i}, \boldsymbol{W}_{V_i}$ are the learnable projection matrices for query, key and value in the i-th attention head, and $\boldsymbol{W}_O$ is the output projection matrix for feature fusion. The scaling factor $\sqrt{d}$ prevents the gradient saturation problem caused by excessively large inner product values.

In this work, we adopt a **two-layer stacked 8-head self-attention block** with residual connection and layer normalization for all feature branches. The formal expression of the stacked MHSA block is:

$$\begin{aligned} \widetilde{\boldsymbol{X}} &= \text{LayerNorm}\big(\boldsymbol{X} + \text{MultiHead}(\boldsymbol{X},\boldsymbol{X},\boldsymbol{X})\big), \\ \boldsymbol{X}_{\text{out}} &= \text{LayerNorm}\big(\widetilde{\boldsymbol{X}} + \text{FeedForward}(\widetilde{\boldsymbol{X}})\big), \end{aligned} \tag{15}$$

where $\boldsymbol{X}$ denotes the input sequential feature, and FeedForward represents the two-layer linear feed-forward network. Such a stacked structure effectively extracts deep hierarchical features while stabilizing model training via residual connections.

We first perform deep enhancement on the stock sequential feature $\boldsymbol{O}_s \in \mathbb{R}^{t' \times h}$:

$$\boldsymbol{O}_s^{\text{att}} = \text{MHSA}_8^{\text{2-layer}}(\boldsymbol{O}_s) \in \mathbb{R}^{t' \times h}. \tag{16}$$

The stock feature $\boldsymbol{O}_s$ obtained by linear projection and layer normalization is fed into the two-layer 8-head self-attention module to capture the internal temporal evolution rules of stock time series.

Secondly, the aggregated quantitative formula feature $\boldsymbol{H}_F \in \mathbb{R}^{m \times h}$ is encoded through the identical attention structure:

$$\boldsymbol{H}_F^{\text{att}} = \text{MHSA}_8^{\text{2-layer}}(\boldsymbol{H}_F) \in \mathbb{R}^{m \times h}. \tag{17}$$

This operation models the implicit correlation and complementary information among different quantitative formula indicators.

After obtaining the two enhanced sequences, we concatenate them along the sequence dimension to construct a fused stock-formula representation:

$$\boldsymbol{F}_{\text{fusion}} = \text{Concat}\big(\boldsymbol{O}_s^{\text{att}}, \boldsymbol{H}_F^{\text{att}}\big) \in \mathbb{R}^{(t'+m) \times h}. \tag{18}$$

Furthermore, the formula-prefix representation with position and temporal encoding $\text{REF}_{\text{final}} \in \mathbb{R}^{(L-t+1) \times h}$ is also processed by the two-layer 8-head self-attention block for consistent feature learning:

$$\boldsymbol{F}_{\text{ref}} = \text{MHSA}_8^{\text{2-layer}}\big(\text{REF}_{\text{final}}\big) \in \mathbb{R}^{(L-t+1) \times h}. \tag{19}$$

Finally, we obtain three high-level enhanced representations, including stock temporal feature, quantitative formula fusion feature, and reference formula feature.

After obtaining the enhanced stock-formula fused representation and the reference prefixes representation, we employ the multi-head cross-attention (MHCA) mechanism to align the semantic information between the two branches and generate the final prediction. Different from self-attention where query, key and value are derived from the same input sequence, cross-attention takes query from one sequence and key/value from another, thereby enabling information interaction and semantic alignment across heterogeneous feature spaces.

The standard formulation of multi-head cross-attention is defined as:

$$\begin{aligned}
&\text{MultiHeadCross}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Concat}(\text{Head}_1, \text{Head}_2, \ldots, \text{Head}_8)\boldsymbol{W}_O, \\
&\text{Head}_i = \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_{Q_i}, \boldsymbol{K}\boldsymbol{W}_{K_i}, \boldsymbol{V}\boldsymbol{W}_{V_i}), \\
&\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V},
\end{aligned} \tag{20}$$

where $\boldsymbol{Q} \in \mathbb{R}^{(L-t+1)\times h}$ denotes the query matrix derived from the reference prefixes branch, $\boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{(t'+m)\times h}$ denote the key and value matrices derived from the stock-formula fused branch with sequence length $(t'+m)$, and d is the dimension of each attention head. The scaling factor $\sqrt{d}$ stabilizes the gradient flow by preventing the inner product values from becoming excessively large.

To enable deep cross-modal interaction and hierarchical semantic fusion, we stack a **four-layer 8-head cross-attention block** with residual connection, layer normalization and position-wise feed-forward network. The formal expression of each cross-attention layer is:

$$\begin{aligned}
\widetilde{\boldsymbol{Q}} &= \text{LayerNorm}\big(\boldsymbol{Q} + \text{MultiHeadCross}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})\big), \\
\boldsymbol{Q}_{\text{out}} &= \text{LayerNorm}\big(\widetilde{\boldsymbol{Q}} + \text{FeedForward}(\widetilde{\boldsymbol{Q}})\big),
\end{aligned} \tag{21}$$

where the residual connection preserves the original query information while the cross-attention module injects contextual knowledge from the key-value branch. The feed-forward network, consisting of two linear transformations with a GELU activation in between, further enhances the nonlinear representation capability of each layer. The four-layer stacked structure progressively refines the alignment between the reference pattern and the stock-formula context.

Specifically, the stock-formula fused representation $\boldsymbol{F}_{\text{fusion}}$ serves as the key and value matrices, while the reference prefixes representation $\boldsymbol{F}_{\text{ref}}$ serves as the query matrix. The four-layer cross-attention decoding process is formulated as:

$$\begin{aligned}
&\boldsymbol{Q}^{(0)} = \boldsymbol{F}_{\text{ref}}, \quad \boldsymbol{K} = \boldsymbol{V} = \boldsymbol{F}_{\text{fusion}}, \\
&\boldsymbol{Q}^{(\ell)} = \text{MHCA}_8^{\ell}\big(\boldsymbol{Q}^{(\ell-1)}, \boldsymbol{K}, \boldsymbol{V}\big), \quad \ell = 1, 2, 3, 4, \\
&\boldsymbol{I} = \boldsymbol{Q}^{(4)} \in \mathbb{R}^{(L-t+1)\times h},
\end{aligned} \tag{22}$$

where $\text{MHCA}_8^{\ell}(\cdot)$ denotes the ℓ-th cross-attention layer with 8 heads, and $\boldsymbol{I}$ is the final decoded representation that captures the aligned semantic information between the reference pattern and the stock-formula context.

Finally, the decoded representation is projected through a linear layer followed by a softmax normalization to produce the probability distribution over the formula token space, where D denotes the dimension of one-hot encoding for each formula token (operators, constants and features):

$$\boldsymbol{J} = \boldsymbol{I}\boldsymbol{W}_{\text{out}} + \boldsymbol{b}_{\text{out}} \in \mathbb{R}^{(L-t+1)\times D}, \tag{23}$$

$$\boldsymbol{p} = \text{Softmax}\big(\boldsymbol{J}_{L-t+1,:}\big) \in \mathbb{R}^{D}, \tag{24}$$

where $\boldsymbol{J}i, :$ denotes the i-th row of the logit matrix $\boldsymbol{J}$. Since the decoder emits $L-t+1$ positions, $\boldsymbol{J}L-t+1, :$ picks out the last position, i.e., the logits at the latest decoded step. $\boldsymbol{W}$out $\in \mathbb{R}^{h\times D}$ and $\boldsymbol{b}$out $\in \mathbb{R}^{D}$ are the learnable output projection parameters, D is the dimension of the one-hot encoding of a formula token (operators, constants and features), and $\boldsymbol{p} \in \mathbb{R}^{D}$ is the distribution over the D tokens from which the next token is sampled or selected via argmax.

Through the multi-head cross-attention decoding mechanism, the model effectively leverages the reference formula pattern as a structural prior to guide the generation of quantitative formulas conditioned on the stock-formula fused context, thereby producing candidate formula sequences.

D.1.1 POSITIONAL ENCODING

We use fixed sinusoidal positional encoding to inject intra-sequence position information:

$$\begin{cases} \mathrm{PE}(pos, 2i) = \sin\left(pos / 10000^{2i/d}\right) \\ \mathrm{PE}(pos, 2i+1) = \cos\left(pos / 10000^{2i/d}\right) \end{cases} \tag{25}$$

where $pos \in \{0, 1, \ldots, 63\}$ is the zero-based token index in the RPN sequence (max length 64), d is the embedding dimension, and i is the dimension index. The encoding is pre-computed and remains fixed during training.

D.1.2 STEP ENCODING

Step encoding shares the same sinusoidal form and indicates the current autoregressive generation progress:

$$\begin{cases} \mathrm{TE}(t, 2i) = \sin\left(t / 10000^{2i/d}\right) \\ \mathrm{TE}(t, 2i+1) = \cos\left(t / 10000^{2i/d}\right) \end{cases} \tag{26}$$

where $t \in \{0, 1, \ldots, 63\}$ is the count-down remaining generation step ($t = 63$ for an fully generated, $t = 0$ when fully generated). It is also pre-computed as a non-trainable buffer.

D.1.3 ENCODING FUSION

Token embeddings are summed with position-wise positional encoding and broadcast step encoding, then fed into the decoder:

$$X = \mathrm{LayerNorm}\left(\mathrm{MLP}\left(E(\text{token_ids}) + \mathrm{PE}(pos) + \mathrm{TE}(t)\right)\right) \tag{27}$$

Positional encoding varies per token position, while step encoding is shared across all positions in the same sequence via broadcasting.

D.2 TRAINING DETAILS

D.2.1 DYNAMIC DECAY OF SAMPLED FORMULA REFERENCES

We use sampled formulas as references for model training, where two considerations arise: (1) whether the model can adequate learn formula syntax and prior knowledge embedded within reference formulas; and (2) preventing the model from overfitting to reference formulas and losing the ability to explore novel formulas.

To address these considerations, we design a dynamic decay schedule. The model generates 8 formulas per batch. We adopt full references for auxiliary learning in the first 200 batches. After batch 200, we reduce the reference proportion by $1/8$ every 100 batches, and adopt an all mask containing only the BEG token to gradually shift the model toward reference-free generation. Between batches 800 and 1000, we retain the final $1/8$ of references to help the model adapt to reference-free generation. Since we only update the model using the top-2 factors ranked by reward score, at least one fully reference-free sample is guaranteed to participate in parameter updates during this phase. At batch 1000 and every subsequent 100 batches, we decide whether to keep the $1/8$ reference proportion by evaluating the number of valid formulas generated over the preceding 100 batches. Once the count of valid formulas from the most recent 100 batches reaches the predefined threshold of 100, the model enters a free-exploration stage with full reference-free generation, and no reference formulas are used thereafter.

D.2.2 EXPLANATION OF ALPHA FACTOR REWARD SCORES

We categorize generated formulaic alpha factors into two classes. The first class consists of valid formulas, which are syntactically well-formed and can be evaluated to compute the information coefficient (IC). The second class contains invalid and degenerate formulas. Invalid formulas have ill-posed syntax and cannot be executed. Degenerate formulas are computationally evaluable and can yield metrics such as IC, but they only possess stock selection predictive power for a subset of stocks. For instance, CmpLess(\$open, \$close) outputs a boolean value of either 1 or 0. Such factors only partition stocks into two groups and cannot predict the relative magnitude of future stock returns

within each group. Another example is $\mathrm{Log}(\mathrm{Div}(\mathrm{Sub}(\mathrm{Ref}(\$\mathrm{close}, 1), \$\mathrm{close}), \mathrm{Ref}(\$\mathrm{close}, 1)))$. Here, $\mathrm{Div}(\mathrm{Sub}(\mathrm{Ref}(\$\mathrm{close}, 1), \$\mathrm{close}), \mathrm{Ref}(\$\mathrm{close}, 1))$ computes the rate of change of today's closing price relative to yesterday's closing price. When the rate of change is negative, the formula yields NaN, resulting in missing predictive values for a large subset of stocks. We classify such expressions as degenerate formulas.

D.2.3 MODEL TRAINING DETAILS

The training procedure of AlphaRef is outlined in Algorithm 1. For factor admission into the factor zoo, we establish two selection rules applied to the top-2 highest-scoring factors from each batch:

- **Standard admission rule:** A candidate factor is eligible to enter the zoo if its absolute information coefficient ($|\mathrm{IC}| > 0.01$) and its absolute value of the maximum correlation with existing factors in the zoo ($\mathrm{max_corr} < 0.8$) both satisfy the thresholds.
- **Replacement rule:** If a candidate factor has *exactly one* existing factor in the zoo with an absolute correlation above 0.8, and the candidate achieves a higher $|\mathrm{IC}|$ than that existing factor, the candidate replaces the corresponding existing factor in the zoo.

Eligible factors are added directly to the zoo when the zoo has not reached its maximum capacity. When admitting the top-2 candidates sequentially, the latter candidate is required to compute correlation with the previously admitted candidate during its eligibility judgment. When the zoo is full, a new eligible factor is admitted only if its $|\mathrm{IC}|$ exceeds the smallest $|\mathrm{IC}|$ among existing factors in the zoo; the existing factor with the smallest $|\mathrm{IC}|$ will then be removed to make room for this new factor.

Algorithm 1 Factor Mining Pipeline

Input: Stock data $X = \{x_t\}$, target $Y = \{y_t\}$, reference formula set REF
Output: Factor zoo $\mathcal{Z} = \{f_1, f_2, \ldots, f_k\}$

Initialization: $\mathcal{Z} \leftarrow \varnothing$; $\pi_{\mathrm{ref}} \leftarrow \pi_\theta$; batch size $B = 8$

for $e = 1$ **to** n **do**
  $S_e \leftarrow$ random sample of stock features from X
  $\pi_{\mathrm{old}} \leftarrow \pi_\theta$ ▷ snapshot for sampling and ratio
  sample B segments $\{\mathrm{REF}_{e,b}\}_{b=1}^{B}$, each from progressive masking of REF or from BEG (zero reference)
  for $b = 1$ **to** B **do** ▷ autoregressive decoding
    repeat
      $\{\mathrm{REF}_{e,b}, S_e, \mathcal{Z}\} \rightarrow \mathrm{Embedding}$ ▷ Eqs. (3)-(5); $\mathcal{Z}$ omitted when empty
      $\mathrm{token} \leftarrow \mathrm{Model[Embedding]}$
      $\mathrm{REF}_{e,b} \leftarrow \mathrm{REF}_{e,b} \parallel \mathrm{token}$
    until $\mathrm{token} = \mathrm{SEP}$
    $\mathrm{REF}_{e,b} \leftarrow \mathrm{REF}_{e,b} \parallel \mathrm{PAD}$ (pad to length 64)
  end for
  $\{f_{\mathrm{new},b}\}_{b=1}^{B} \leftarrow \{\mathrm{REF}_{e,b}\}_{b=1}^{B}$
  compute $\mathrm{score}_b = \mathrm{Score}\big(f_{\mathrm{new},b}\big)$ via IC and $\mathrm{max_corr}$ ▷ Table 1
  select top-2 factors by score
  if top-2 qualified **then** $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{f_{\mathrm{top}-2}\}$ **end if**
  $\mathcal{F} \leftarrow \{f_{\mathrm{new},b}\}_{b=1}^{B}$ ▷ all B generated formulas
  $L_{\mathrm{GRPO}}(\theta) = L_{\mathrm{policy}}(f_{\mathrm{top}-2}) + \alpha L_{\mathrm{KL}}(\mathcal{F}) - \beta L_{\mathcal{H}}(\mathcal{F})$ ▷ Eq. (13)
  $\theta \leftarrow \theta - lr \cdot \nabla_\theta L_{\mathrm{GRPO}}$
  $\pi_{\mathrm{ref}} \leftarrow 0.995\,\pi_{\mathrm{ref}} + 0.005\,\pi_\theta$ ▷ EMA, only on successful updates
end for
return $\mathcal{Z}$

E DETAILS OF EVALUATION METRICS AND BASELINES MODEL.

E.1 DETAILS OF EVALUATION METRICS

This appendix defines the evaluation metrics adopted for factor assessment, including information coefficient (IC), rank information coefficient (RankIC), information coefficient ratio (ICIR), rank

information coefficient ratio (RankICIR), compound annual growth rate (CAGR), Sharpe ratio (SR), and maximum drawdown (MDD).

Information Coefficient (IC) The information coefficient (IC) quantifies the cross-sectional linear association between factor exposures and future stock returns at each timestamp t. It is defined as the Pearson correlation coefficient between raw factor values and forward h-period returns across all stocks in the cross-section.

$$\mathrm{IC}_t = \frac{\sum_{i=1}^{N_t} \left(f_{i,t} - \bar{f}_t\right)\left(r_{i,t\to t+h} - \bar{r}_t\right)}{\sqrt{\sum_{i=1}^{N_t} \left(f_{i,t} - \bar{f}_t\right)^2}\sqrt{\sum_{i=1}^{N_t} \left(r_{i,t\to t+h} - \bar{r}_t\right)^2}} \tag{28}$$

where N_t denotes the number of available stocks at time t, $f_{i,t}$ is the raw factor value of stock i observed at time t; $r_{i,t\to t+h}$ denotes the forward h-period return of stock i from time t to $t+h$; $\bar{f}_t = \frac{1}{N_t}\sum_{i=1}^{N_t} f_{i,t}$ and $\bar{r}_t = \frac{1}{N_t}\sum_{i=1}^{N_t} r_{i,t\to t+h}$ means of factor values and forward returns at time t, respectively. The range of IC_t lies in $[-1, 1]$. A positive IC_t indicates that stocks with larger factor values tend to achieve higher future cross-sectional returns, while a negative value implies the opposite. Larger absolute values of IC correspond to stronger predictive power of the factor for cross-sectional stock returns.

Rank Information Coefficient (RankIC) RankIC, also known as the Spearman correlation in cross-sectional evaluation, calculates the Pearson correlation between the cross-sectional ranks of factor values and the cross-sectional ranks of forward returns. By replacing raw values with ordinal ranks, RankIC is insensitive to extreme outliers and can capture monotonic relationships rather than only linear correlations.

$$\mathrm{RankIC}_t = \frac{\sum_{i=1}^{N_t} \left(\mathrm{rank}(f_{i,t}) - \overline{\mathrm{rank}(f)}_t\right)\left(\mathrm{rank}(r_{i,t\to t+h}) - \overline{\mathrm{rank}(r)}_t\right)}{\sqrt{\sum_{i=1}^{N_t} \left(\mathrm{rank}(f_{i,t}) - \overline{\mathrm{rank}(f)}_t\right)^2}\sqrt{\sum_{i=1}^{N_t} \left(\mathrm{rank}(r_{i,t\to t+h}) - \overline{\mathrm{rank}(r)}_t\right)^2}} \tag{29}$$

where $\mathrm{rank}(\cdot)$ performs cross-sectional rank transformation at time t: all N_t stocks are sorted and assigned ordinal ranks according to their raw values. $\overline{\mathrm{rank}(f)}_t$ and $\overline{\mathrm{rank}(r)}_t$ are the cross-sectional average ranks of factor values and forward returns at time t, respectively. Similar to IC, RankIC ranges from -1 to 1. Compared with standard IC, RankIC is more robust to outliers and heavy-tailed distributions commonly seen in financial data, making it a more stable metric for evaluating factor predictive efficacy.

Information Coefficient Ratio (ICIR) ICIR is defined as the time-series mean of daily IC divided by the time-series standard deviation of IC, quantifying the risk-adjusted predictive consistency of the factor.

$$\mathrm{ICIR} = \frac{\mu_{\mathrm{IC}}}{\sigma_{\mathrm{IC}}} = \frac{\frac{1}{T}\sum_{t=1}^{T} \mathrm{IC}_t}{\sqrt{\frac{1}{T-1}\sum_{t=1}^{T} \left(\mathrm{IC}_t - \mu_{\mathrm{IC}}\right)^2}} \tag{30}$$

where T is the total number of time slices, μ_{IC} is the time-series average of IC_t, and σ_{IC} is the sample standard deviation of IC_t.

Rank Information Coefficient Ratio (RankICIR) RankICIR replaces IC with RankIC in the ICIR formula, measuring the stability of rank-based cross-sectional predictive signals.

$$\mathrm{RankICIR} = \frac{\mu_{\mathrm{RankIC}}}{\sigma_{\mathrm{RankIC}}} = \frac{\frac{1}{T}\sum_{t=1}^{T} \mathrm{RankIC}_t}{\sqrt{\frac{1}{T-1}\sum_{t=1}^{T} \left(\mathrm{RankIC}_t - \mu_{\mathrm{RankIC}}\right)^2}} \tag{31}$$

Compound Annual Growth Rate (CAGR) CAGR describes the annualized geometric growth rate of portfolio net asset value (NAV) over the backtest period.

$$\mathrm{CAGR} = \left(\frac{\mathrm{NAV}_{\mathrm{end}}}{\mathrm{NAV}_{\mathrm{start}}}\right)^{\frac{252}{D}} - 1 \tag{32}$$

where $\mathrm{NAV}_{\mathrm{start}}$ and $\mathrm{NAV}_{\mathrm{end}}$ are the portfolio net asset values at the start and end of the backtest, and D is the total number of trading days, so that $252/D$ converts the holding period into years.

Sharpe Ratio (SR) Sharpe ratio evaluates excess return per unit of volatility. We use daily portfolio returns and annualize the metric.

$$\text{SR} = \frac{\mu_p - r_f}{\sigma_p} \cdot \sqrt{252} \tag{33}$$

where μ_p is the mean of daily portfolio returns, σ_p is the sample standard deviation of daily portfolio returns, r_f is the daily risk-free rate, and 252 denotes the typical number of trading days per year. In our experiments the risk-free rate is negligible, so we set $r_f = 0$.

Maximum Drawdown (MDD) Maximum drawdown quantifies the worst peak-to-trough loss of portfolio NAV, measuring downside risk.

$$\text{MDD} = \max_{t \in [0,T]} \left(1 - \frac{\text{NAV}_t}{\text{peak}_{\leq t}}\right), \quad \text{peak}_{\leq t} = \max_{\tau \leq t} \text{NAV}_\tau \tag{34}$$

where $\text{peak}_{\leq t}$ is the historical maximum NAV up to time t. MDD lies in $[0, 1]$, and a larger MDD indicates more severe historical drawdown risk.

E.2 Baselines Model.

We divide all baseline methods into two categories: machine learning-based methods and symbolic-regression-based methods. Detailed descriptions of each model are provided as follows.

Machine Learning-based Methods These methods directly forecast stock returns without explicit alpha formulas. All models in this group use 250-day raw market features as input and predict 20-day forward returns, following the standard supervised learning paradigm for return prediction tasks.

- **MLP (Multi-Layer Perceptron):** The Multi-Layer Perceptron is the most fundamental feedforward neural network architecture, consisting of stacked fully connected layers with nonlinear activation functions Rumelhart et al. (1986). It maps input feature vectors to output predictions through layer-wise affine transformations and pointwise nonlinearities. In our experiments, the MLP baseline adopts a two-hidden-layer architecture with ReLU activations, trained via the Adam optimizer with mean squared error loss. Despite its simplicity, MLP serves as a standard nonlinear baseline that captures basic feature interactions in stock return prediction.
- **LSTM (Long Short-Term Memory):** Long Short-Term Memory is a specialized variant of recurrent neural networks designed to address the vanishing gradient problem and capture long-range temporal dependencies in sequential data Hochreiter & Schmidhuber (1997). It operates through a memory cell mechanism controlled by input, forget, and output gates, which selectively retain and update information across time steps. For stock return prediction, LSTM processes the 250-day feature sequence step by step, extracting temporal patterns from price and volume dynamics. Its inherent sequential modeling capability makes it widely used for time-series financial forecasting tasks.
- **XGBoost (Extreme Gradient Boosting):** XGBoost is a scalable tree-boosting ensemble algorithm that builds prediction models through sequentially added regression trees, where each new tree corrects the residual errors of the existing ensemble Chen & Guestrin (2016). It implements regularized gradient boosting with both L1 and L2 penalties, column sampling, and tree pruning to prevent overfitting. As a tabular data prediction benchmark, XGBoost excels at capturing nonlinear feature interactions and handling heterogeneous financial features, and is widely recognized for its robustness and computational efficiency in quantitative finance applications.
- **LightGBM (Light Gradient Boosting Machine):** LightGBM is a high-performance gradient boosting framework that introduces gradient-based one-side sampling and exclusive feature bundling to accelerate training while maintaining prediction accuracy Ke et al. (2017). Compared with standard tree boosting methods, it grows trees leaf-wise rather than level-wise, can improve efficiency under suitable settings, with lower computational overhead on large-scale datasets. In our experiments, LightGBM serves as an efficient ensemble baseline for cross-sectional stock return prediction.

Symbolic-regression-based Methods These methods discover alpha factors as interpretable mathematical formulas through symbolic search and optimization. Unlike black-box machine learning models, these methods output explicit mathematical expressions that can be directly evaluated on market data and interpreted by quantitative practitioners.

- **GP (Genetic Programming):** Genetic Programming is an evolutionary algorithm that evolves computer programs or mathematical expressions through selection, crossover, and mutation operations, optimizing for a predefined fitness objective Koza (1992). In the context of alpha factor mining, GP represents each factor as an expression tree composed of arithmetic operators and financial functions, and iteratively evolves the population of formulas to maximize the information coefficient (IC). As the classical symbolic regression baseline, GP provides a standard benchmark for formula-based alpha discovery, though it suffers from low search efficiency and high correlation among generated factors.

- **AlphaForge:** AlphaForge is a two-stage formulaic alpha generation framework that combines a generative-predictive neural network for factor mining and a dynamic combination module for portfolio construction Shi et al. (2025). The generator network leverages deep learning to explore the formula search space and produce low-correlation alpha factors, while the combination module dynamically adjusts factor weights based on temporal performance. It addresses the inflexibility of fixed-weight factor combinations and improves the adaptability of multi-factor portfolios to changing market conditions.

- **AlphaGen:** AlphaGen formulates the alpha factor generation process as a Markov Decision Process (MDP), and converts mathematical formulas into token sequences via RPN Yu et al. (2023). It adopts the Proximal Policy Optimization (PPO) algorithm with invalid action masking to ensure syntactical validity of generated expressions. Different from traditional methods that optimize individual factor performance, AlphaGen takes the overall performance of the alpha factor pool as the reward signal, explicitly optimizing for synergistic effects among factors to reduce redundancy in the generated factor set.

- **AlphaSAGE:** AlphaSAGE addresses three core limitations of existing RL-based alpha mining methods: reward sparsity, insufficient structural representation of formulas, and limited diversity of generated factors Chen et al. (2026). It introduces three key innovations: a structure-aware encoder based on Relational Graph Convolutional Network (RGCN) that models formulas as abstract syntax trees instead of flat sequences, a Generative Flow Network (GFlowNet) training framework for stable and efficient exploration, and a multi-faceted dense reward function. The framework is designed to discover alpha factors that are simultaneously predictive,weakly correlated, and structurally novel.

- **AlphaQCM:** AlphaQCM is a distributional reinforcement learning method for alpha discovery that addresses the challenges of non-stationarity and reward sparsity in the formula search process Zhu & Zhu (2025). It learns both a Q-function and quantile estimates through separate networks, then applies the quantiled conditional moment method to derive unbiased variance estimation. Guided by the learned value function and uncertainty estimates, AlphaQCM efficiently navigates the vast search space of formulaic alphas and achieves strong performance especially on large datasets with extensive stock universes.

F Examples of Generated Factors

Table 5 presents the alpha factors mined by AlphaRef. During the mining process, the model generates not only factors distinct from the reference formula set, but also factors identical to those in the reference set. This indicates that a subset of the original reference factors remain effective on the new dataset, and we retain these factors accordingly. To illustrate the distinction between the factors generated by our method and the reference formula set, we sample the output results and compare the original reference formulas with their derived variants, as shown in Table 6. We observe that AlphaRef discovers novel alpha factors built upon the original reference factors, and the illustrated variant has a higher test IC than original reference ones on the CSI 300 test set. This demonstrates that our model is capable of mining more effective alpha factors.

Table 5: Generated Alpha Formulas by AlphaRef

#	Alpha
1	Add(-1.0,Add(Rank($volume),Rank($vwap)))
2	Add(-1.0,Log($volume))
3	Add(Rank(TsCorr($vwap,$volume,4)),Add(Rank($low),Rank(TsMean($volume,50))))
4	Add(Mul(Rank(Ref(Div(Sub($high,$low),Div(TsSum($close,5),5.0)),2)),Rank(Rank($volume))), Div(Sub($high,$low),Div(TsSum($close,5),5.0)))
5	Mul(Mul(-1.0,Rank(TsStd($high,10))),TsCorr($high,$volume,10))
6	Add(TsRank(DecayLinear(CmpLess(Add(Div(Add($high,$low),2.0),$close),Add($low,$open)),15),20), Add(Rank($low),Rank(TsMean($volume,30))))
7	DecayLinear(Sub(Add(Rank($open),Rank($low)),Add(Rank($high),Rank($close))),8)
8	Add(Rank($high),TsSum(Rank(TsMean($volume,15)),9))
9	Add($close,Log(Add($volume,1.0)))
10	Add(Rank(DecayLinear(TsCorr($vwap,TsSum(TsMean($volume,5),30),5),7)), Add(Rank($open),Rank(TsMean($volume,15))))
11	TsCorr($close,Log(Add($volume,1.0)),20)
12	Div(Sub($high,$low),Div(TsSum($close,5),5.0))
13	TsCorr($close,Log(Add($volume,1.0)),10)
14	Div(Sub(If(CmpLess($open,$close),$open,$close),$low),$open)
15	Div(TsMean($close,30),$close)
16	Add(Rank(TsCorr($vwap,$volume,4)),Rank($low))
17	TsCorr(Sub(Div($close,Ref($close,1)),1.0), Log(Add(Div($volume,Ref($volume,1)),1.0)),20)
18	TsCorr(Sub(Div($close,Ref($close,1)),1.0), Log(Add(Div($volume,Ref($volume,1)),1.0)),30)
19	TsVar(Div(TsMean($volume,20), $volume), 20)
20	Add(-1.0,Add(Rank(Div(Sub($close,TsMin($low,12)), Sub(TsMax($high,12),TsMin($low,12)))),Rank($volume)))

Table 6: Comparison between Reference Formula and Its Derived Variant

Category	Alpha Formula	IC
Reference	Div(TsMean($volume,20), $volume)	0.0061
Variant	TsVar(Div(TsMean($volume,20), $volume), 20)	0.0154

G Experimental Supplements

To investigate the impact of the factor-zoo size on performance, we further conduct sensitivity experiments. As illustrated in Figure 6, we vary the size of the factor zoo. The results show that a zoo size of 20 achieves the best performance on the CSI300 and CSI1000 benchmarks, whereas a zoo size of 30 performs best on the CSI500 benchmark. This finding is closely related to our portfolio construction strategy: during dynamic factor combination, on each trading day we select the top-10 factors with the best historical performance from the factor zoo and aggregate them via ordinary least squares (OLS). When the zoo is too small, the factor diversity is insufficient; when it is too large, additional noise is introduced. Both cases slightly degrade the performance, so the optimal zoo size concentrates around 20 to 30.

Factor formula sampling methods. We further compare factor formula sampling methods, contrasting our sequential mask sampling with the random subtree masking baseline on syntax trees.

Sequential mask sampling (ours). Each reference formula is first parsed into a RPN sequence of the form $[\texttt{BEG}, x_1, \ldots, x_n, \texttt{SEP}, \ldots]$. We then generate progressive masking variants by masking the suffix of the sequence: for a formula of effective length L, variant $k \in \{1, \ldots, L-2\}$ replaces the last k tokens (from the `SEP` token backward) with the `MASK` token, ranging from masking only

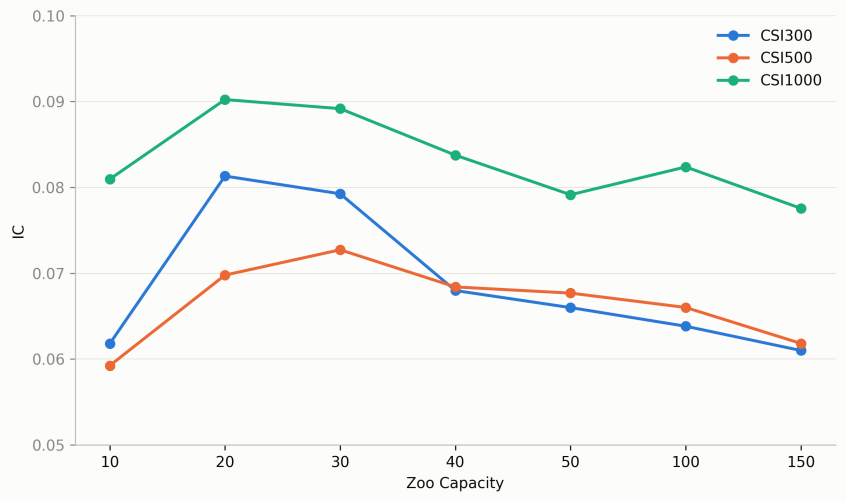


Figure 6: Experiments with different factor zoo sizes.

SEP up to masking nearly the entire suffix while keeping BEG. These variants correspond to the intermediate states of autoregressive left-to-right generation.

Random subtree masking (baseline). Instead of masking suffixes, this baseline masks entire subtrees of the syntax tree. A stack-based pass over the RPN sequence recovers the contiguous span of every sub-expression: each leaf token (a feature, a time window, or a constant) is a single-token subtree, and each operator pops the spans of its operands to form a larger subtree. We then randomly select a subset of these subtrees—including leaf tokens and the root—and replace their tokens with the MASK token, such that the proportion of masked tokens lies in $[0.2, 0.6]$. Each reference formula is expanded into multiple variants, forcing the model to reconstruct complete semantic units (sub-expressions) rather than to continue a suffix.

As shown in Table 7, our method achieves better results on all four metrics (IC, ICIR, RankIC, and RankICIR) on the CSI300 and CSI1000 benchmarks, while on CSI500 it attains a slightly higher IC but slightly lower ICIR, RankIC, and RankICIR. Overall, our sampling method outperforms the random subtree masking baseline on the majority of the metrics. We conjecture that sequential masking is inherently consistent with the autoregressive decoding mechanism: since masking a suffix reproduces the intermediate states of left-to-right generation, the model is trained on exactly the task it performs at inference. In contrast, random subtree masking resembles masked-language-model infilling, which reconstructs an interior subtree from surrounding context and is thus misaligned with autoregressive generation, yielding weaker training signals.

Table 7: Performance comparison on the CSI 300, CSI 500 and CSI 1000.

Method	CSI300				CSI500				CSI1000			
	IC↑(%)	ICIR↑	RankIC↑(%)	RankICIR↑	IC↑(%)	ICIR↑	RankIC↑(%)	RankICIR↑	IC↑(%)	ICIR↑	RankIC↑(%)	RankICIR↑
Random Subtree Mask	7.81	0.480	9.52	0.556	6.97	**0.453**	**11.13**	**0.701**	8.64	0.574	13.66	0.840
Ours	**8.13**	**0.486**	**10.02**	**0.562**	**6.98**	0.446	11.01	0.685	**9.02**	**0.617**	**13.77**	**0.872**

Furthermore, to validate the cross-market applicability of our method, we retrain AlphaRef on the US equity market and evaluate it out-of-sample. Specifically, the model is trained on the full US stock universe from 2015-04-03 to 2022-12-31, and evaluated on the S&P 500 constituents over 2023-01-01 to 2025-12-31. Backtest results are presented in Figure 7, where the strategy achieves a return on par with the S&P 500 index, indicating the broad applicability of our method to a different market.

H HARDWARE SPECIFICATION

Our experiments were conducted on a Linux machine with an Intel Xeon Platinum 8581C CPU and an NVIDIA RTX A6000 GPU with 48 GB VRAM.

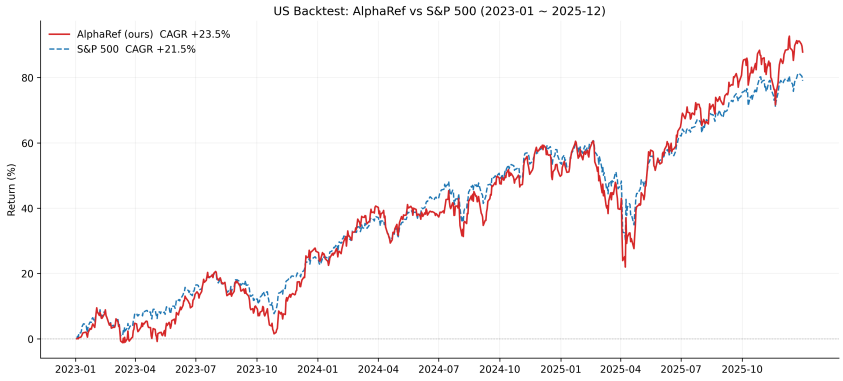


Figure 7: backtest results on the S&P 500 index constituents.

I BROADER IMPACT

This work advances formulaic alpha factor mining by incorporating existing factors as prior knowledge. The mined factors may serve practitioners as candidate signals for constructing and iterating factor libraries; the repair and upgrade of decayed factors is a promising application that we leave for future work. Our findings are based on backtests over large-cap index constituents (CSI300/500/1000 and S&P 500) under idealized execution assumptions. We have not yet validated the approach under realistic transaction costs, slippage, or liquidity constraints, nor across smaller-cap or less-liquid markets, nor through out-of-market transfer beyond the markets studied; these practical limitations remain to be examined in future work.