EvoAlpha: Evolutionary Alpha Factor Discovery with Large Language Models

Haochen Luo
City University of Hong Kong
Hong Kong, China
chester.hc.luo@my.cityu.edu.hk

Ho Tin Ko
Yuen Long Merchants Association Secondary School
Hong Kong, China
ltu46166@gmail.com

David Sun
Harrow International School Hong Kong
Hong Kong, China
0220220405@harrowschool.hk

Yuan Zhang
Shanghai University of Finance and Economics
Shanghai, China
zhang.yuan@mail.shufe.edu.cn

Chen Liu
City University of Hong Kong
Hong Kong, China
chen.liu@cityu.edu.hk

Abstract

Alpha factor discovery is a central challenge in quantitative finance, traditionally addressed by human experts or automated search methods such as genetic programming and evolutionary algorithms. These approaches often lack semantic guidance, leading to inefficient search and fragile results. We propose a language-model-guided evolutionary framework, where large language models (LLMs) act as intelligent operators to guide mutation, crossover, and selection of candidate factors. By embedding evolutionary instructions into prompts, the LLM leverages domain knowledge and backtesting feedback to generate interpretable and high-quality signals. We first validate the approach through static factor searching, showing that LLMs can iteratively refine factors in a controlled setting. We then evaluate the framework in sparse portfolio optimization, where LLM-generated factors are used to rank assets and construct portfolios under ℓ_0 constraints. Experiments on multiple real-market datasets demonstrate consistent improvements in portfolio performance over traditional baselines, highlighting the promise of combining LLMs with evolutionary search for systematic factor discovery.

1 Introduction

Alpha factor discovery, the process of formulating interpretable expressions that extract predictive signals from financial market data, has long been a central problem in quantitative research. Traditional approaches rely on domain expertise, manually constructed factor libraries, or symbolic search methods such as genetic programming. Although these methods have produced effective factors in certain contexts, they suffer from three persistent issues: (i) exploring the vast combinatorial factor space is slow and computationally costly, (ii) discovered factors often fail to adapt to evolving market conditions, leading to performance decay, and (iii) most pipelines rely on static, one-shot generation without feedback from prior evaluations. To address these challenges, previous work has explored

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Generative AI in Finance.

automated search paradigms such as random forests and reinforcement-based searching [1, 2], but these methods still provide limited semantic guidance during factor construction.

Large Language Models (LLMs) offer a promising alternative. Modern LLMs, trained on large collections of text and code, can generate syntactically valid and semantically meaningful formulae directly from textual instructions. They have shown broad applications in finance [3, 4], and specifically in alpha mining tasks. Recent studies [5, 6] demonstrate that LLMs can be used to generate novel alpha factors with competitive predictive performance. Human-in-the-loop systems such as AlphaGPT [7, 8] further incorporate expert feedback to guide LLM outputs, highlighting the potential of interactive refinement. More recently, structured search algorithms have been integrated with LLMs, such as Monte Carlo Tree Search (MCTS) for navigating the factor space [9]. Despite these advances, existing methods leave a gap: most either treat factor generation as a single-pass process or rely on highly specific search schemes such as MCTS. We argue that evolutionary algorithms (EAs) provide a natural and more general framework for iterative factor discovery. Unlike tree-based search, EAs balance exploration and exploitation through simple yet powerful operations (mutation and crossover), maintain a diverse candidate pool, and are naturally compatible with feedback loops.

In this paper, we propose **EvoAlpha**, an evolutionary framework that integrates LLMs into iterative factor discovery. The method maintains a pool of candidate factors, refines them through mutation and crossover prompts, and incorporates evaluation feedback to guide subsequent generations. Our contributions are threefold: (1) introducing an LLM-driven evolutionary search process where the model acts as both generator and adaptive operator; (2) designing a closed-loop mechanism that injects factor evaluation results into prompt updates; and (3) demonstrating that EvoAlpha produces diverse, interpretable, and robust factors with strong performance across multiple datasets. By reformulating alpha factor discovery as an LLM-guided evolutionary search problem, this work shows that LLMs can serve as autonomous symbolic optimisers, capable of balancing creativity, interpretability, and empirical performance without relying on manually crafted grammars or brittle search heuristics.

2 Method

2.1 Problem Reformulation

Alpha factor discovery can be naturally cast as a symbolic search problem. An alpha factor is typically represented as a computation tree, where leaves are raw financial features (e.g., price, volume) and internal nodes are operators (e.g., arithmetic, ranking, rolling statistics). Traditional evolutionary algorithms (EAs) generate new factors by mutating subtrees or recombining expressions, but these approaches often suffer from inefficient exploration and invalid outputs.

Inspired by recent advances in applying LLMs to evolutionary computation [10, 11, 12], we reformulate this task as an *LLM-guided evolutionary search problem*. Instead of blindly applying symbolic operations, we let a large language model act as the search operator. The LLM receives descriptions of current candidate factors and their historical performance, and is instructed to perform evolutionary actions such as mutation (local modifications of an expression) or crossover (combining sub-expressions from different factors). This shifts the burden of semantic reasoning from handcrafted rules to the LLM's learned priors, enabling a more efficient and adaptive exploration of the factor space. This reformulation brings several benefits. By leveraging the LLM's prior knowledge of mathematical structures and financial semantics, the search process can avoid many invalid or trivial candidates that typically plague rule-based Evolutionary Algorithm.

2.2 LLM-Guided Factor Generation

The key idea of our framework is to embed evolutionary operations directly into LLM prompts. Each prompt contains three elements: (i) a definition of the search task, specifying the allowed operators and financial constraints; (ii) a description of top-performing factors from the current pool, presented as structured expressions with anonymized performance summaries; and (iii) illustrative evolutionary instructions that demonstrate mutation or crossover actions, for example, "mutate the time window of a rolling mean" or "combine a volatility factor with a momentum factor."

By combining generative ability and reasoning ability, the LLM acts as a structured evolutionary engine. As shown in Figure 1, the EA process iteratively selects the top-N performing factors

and carries them forward to the next generation, where they are modified to create new candidates. Two key operators drive this process: *mutation*, which applies small local changes to an existing factor (e.g., replacing a 10-day moving average with a 20-day version, or altering a normalization operator), and *crossover*, which combines complementary components from multiple parent factors (e.g., embedding a volatility adjustment inside a momentum-based signal). These operations balance local refinement with broader exploration of the search space.

In the LLM-guided design, evolutionary operations can be realized in two ways. One approach is to use dedicated agents for each operator, where mutation and crossover are explicitly separated into different prompt templates. Alternatively, a single unified agent can be employed, with the LLM itself deciding whether to perform mutation or crossover based on the context and provided instructions. Both designs retain the exploratory power of evolutionary search while ensuring that generated factors remain semantically coherent and financially interpretable.

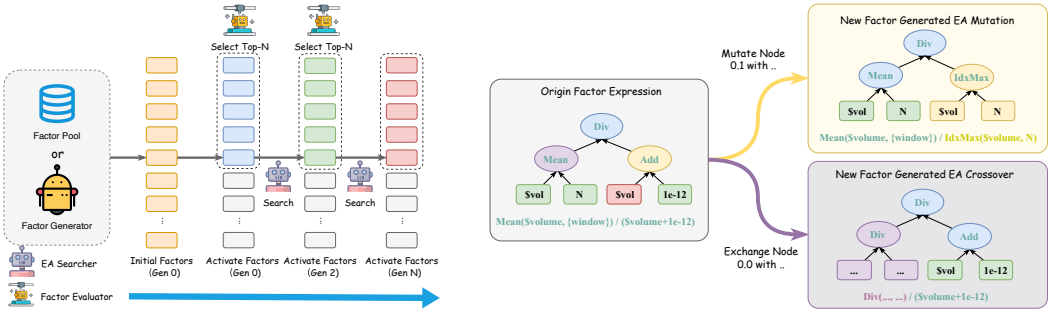


Figure 1: Left: general EA factor-searching framework—at each iteration, select the top-N best factors and carry them forward to generate new candidates for the next round. Right: the generation operators used in each step, including *crossover* and *mutation*.

3 Experiments

3.1 Experiment Settings

To demonstrate the adaptability of the EA+LLM framework, we design two tasks: one for controlled factor-level improvement, and another for dynamic portfolio construction.

(1) Static factor searching. We start from seed factors in Alpha158 from Qlib [13] with in-sample feedback and let the LLM refine it through mutation and crossover. After each round, top candidates are backtested and retained. This tests whether LLMs can improve factor quality, measured by **IC**, **RankIC**, and their **IR**. We run the experiment on CSI300 with Alpha158 as the seed pool, tracking group-level performance in-sample (2023–2024) and out-of-sample (2024–2025).

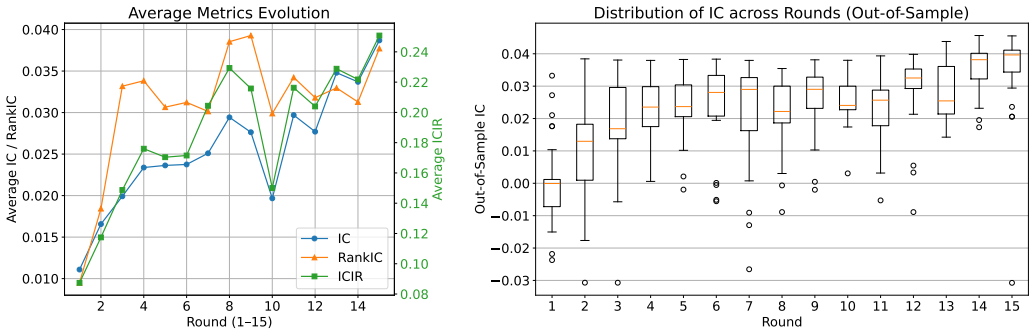
(2) Sparse portfolio optimization. We further evaluate in a dynamic market setting, where LLM-generated factors score assets and the top-m are selected to form a sparse portfolio under ℓ_0 constraints. Datasets include US50, HSI45, and CSI300. Performance is measured by **Cumulative Wealth (CW)**, **Sharpe Ratio (SR)**, and **Maximum Drawdown (MDD)**. Baselines cover two groups: (i) non-sparse strategies — Equal Weighting ($1/N$), Min-CVaR, and Max-Sharpe; (ii) sparse strategies — classical SSPO [14], machine-learning selectors (XGBoost, LGBM), and state-of-the-art methods mSSRM-PGA [15] and ASMCVaR [16].

3.2 Results and Discussion

Static factor searching. Figure 2(a) shows the average values of IC, RankIC, and ICIR within the candidate pool across 15 search rounds. We observe a clear upward trend, indicating that the evolutionary search guided by the LLM can gradually improve the quality of the factor set in-sample. Figure 2(b) illustrates the out-of-sample distribution of IC values across the same rounds. The median IC steadily rises and the interquartile range narrows, suggesting not only enhanced predictive power but also improved robustness of the factor group when evaluated on unseen data. Together, these

results demonstrate that even in a controlled static environment, LLM-driven factor search is able to consistently refine factor quality and generalization.

Sparse portfolio optimization. In Table 3.2, we report the results of sparse portfolio optimization under different settings of sparsity ($m = 10$ and $m = 15$). Across all three markets (US50, HSI45, and CSI300), two LLMs models (GPT-4.1 and Deepseek-V3) consistently achieve the highest cumulative wealth, with improvements that are an order of magnitude larger than traditional baselines such as $1/N$, Min-cVaR, and Max-Sharpe. Compared to machine learning based models (LGBM, XGBoost) and advanced portfolio optimization methods (mSSRM-PGA, ASMCVaR), also delivers superior risk-adjusted performance, reflected in higher Sharpe Ratios and competitive drawdown control. For instance, on US50 with $m = 10$, DeepSeek-V3 reaches a CW of 25.1, far exceeding the best baseline (ASMCVaR, CW=10.3). These results demonstrate that combining LLM-guided factor search with evolutionary exploration yields robust improvements in portfolio-level returns while maintaining risk stability across different markets and sparsity levels.



(a) (b)

Figure 2: (a) Evolution of average in-sample metrics (IC, RankIC, ICIR) across 15 rounds. (b) Distribution of out-of-sample IC values per round. Together, these plots illustrate both the overall trend and the variability of factor performance.

		US50			HSI45			CSI300		
Group	Method	CW↑	SR↑	MDD↓	CW↑	SR↑	MDD↓	CW↑	SR↑	MDD↓
Baseline	1/N	4.562	0.072	0.344	1.333	0.029	0.409	1.087	0.014	0.214
	Min-cVaR	1.779	0.038	0.314	1.628	0.063	0.244	0.992	0.003	0.286
	Max-Sharpe	4.495	0.061	0.461	1.428	0.043	0.300	1.008	0.007	0.333
m=10	LGBM	4.182	0.063	0.332	1.611	0.038	0.367	2.334	0.072	0.225
	XGBoost	6.313	0.077	0.328	1.581	0.035	0.440	1.420	0.032	0.345
	mSSRM-PGA	5.121	0.059	0.569	0.766	-0.003	0.547	0.881	0.002	0.399
	ASMCVaR	10.259	0.073	0.582	2.481	0.052	0.453	1.453	0.030	0.462
	DeepSeek-V3	**25.101**	0.132	0.288	**3.463**	0.080	0.385	3.437	0.079	0.327
	GPT 4.1	22.905	0.130	0.260	2.789	0.067	0.292	**4.962**	0.098	0.301
m=15	LGBM	3.899	0.062	0.328	1.588	0.037	0.387	1.812	0.055	0.250
	XGBoost	5.607	0.076	0.319	1.586	0.036	0.420	1.348	0.029	0.344
	mSSRM-PGA	4.976	0.062	0.477	0.766	-0.003	0.547	0.787	-0.010	0.384
	ASMCVaR	11.124	0.074	0.566	**2.647**	0.054	0.434	1.658	0.035	0.424
	DeepSeek-V3	13.978	0.114	0.298	2.364	0.061	0.406	2.510	0.067	0.298
	GPT 4.1	**14.707**	0.117	0.278	2.277	0.058	0.307	**3.218**	0.082	0.246

Table 1: Evaluation of Cumulative Wealth (CW↑), Sharpe Ratio (SR↑), and Maximum Drawdown (MDD↓) on real-market datasets (US50, HSI45 and CSI300) for different model variants. Our approach significantly outperforms traditional baselines.

4 Conclusion

This work explored large language models as evolutionary search engines for alpha factor discovery and sparse portfolio optimization. We proposed a framework where LLMs perform mutation and crossover to evolve interpretable factors under quantitative feedback. Experiments revealed two main insights: (1) in controlled static search, LLMs iteratively refine seed factors to achieve stronger and more stable predictive power (IC, RankIC, ICIR); and (2) when applied to sparse portfolio construction, LLM-driven factor search yields substantial gains in cumulative wealth and risk-adjusted performance over traditional baselines.

Our findings highlight language-guided evolution as a promising paradigm for systematic investment research. By combining the generative and reasoning abilities of LLMs with rigorous financial evaluation, factors can be discovered and adapted dynamically under sparsity constraints. Future work will extend this framework with multimodal signals (e.g., news, fundamentals), improved prompt strategies for robustness, and lightweight distillation methods to reduce dependence on large-scale LLMs, paving the way toward interpretable and adaptive factor-driven investment strategies.

References

[1] 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*, pages 5476–5486, 2023.

[2] 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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(12):12524–12532, Apr. 2025.

[3] Yuqi Nie, Yaxuan Kong, Xiaowen Dong, John M Mulvey, H Vincent Poor, Qingsong Wen, and Stefan Zohren. A survey of large language models for financial applications: Progress, prospects and challenges. *arXiv preprint arXiv:2406.11903*, 2024.

[4] Huaqin Zhao, Zhengliang Liu, Zihao Wu, Yiwei Li, Tianze Yang, Peng Shu, Shaochen Xu, Haixing Dai, Lin Zhao, Hanqi Jiang, Yi Pan, Junhao Chen, Yifan Zhou, Gengchen Mai, Ninghao Liu, and Tianming Liu. Revolutionizing finance with llms: An overview of applications and insights, 2024.

[5] Zhiwei Li, Ran Song, Caihong Sun, Wei Xu, Zhengtao Yu, and Ji-Rong Wen. Can large language models mine interpretable financial factors more effectively? a neural-symbolic factor mining agent model. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3891–3902, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[6] Ziyi Tang, Zechuan Chen, Jiarui Yang, Jiayao Mai, Yongsen Zheng, Keze Wang, Jinrui Chen, and Liang Lin. Alphaagent: Llm-driven alpha mining with regularized exploration to counteract alpha decay, 2025.

[7] 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, 2023.

[8] Hang Yuan, Saizhuo Wang, and Jian Guo. Alpha-gpt 2.0: Human-in-the-loop ai for quantitative investment, 2024.

[9] Yu Shi, Yitong Duan, and Jian Li. Navigating the alpha jungle: An llm-powered mcts framework for formulaic factor mining, 2025.

[10] Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32201–32223. PMLR, 21–27 Jul 2024.

[11] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems*, 2024. `https://github.com/ai4co/reevo`.

[12] Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. Multi-objective evolution of heuristic using large language model. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(25):27144–27152, Apr. 2025.

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

[14] Zhao-Rong Lai, Pei-Yi Yang, Liangda Fang, and Xiaotian Wu. Short-term sparse portfolio optimization based on alternating direction method of multipliers. *Journal of Machine Learning Research*, 19(63):1–28, 2018.

[15] Yizun Lin, Zhao-Rong Lai, and Cheng Li. A globally optimal portfolio for m-sparse sharpe ratio maximization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[16] Yizun Lin, Yangyu Zhang, Zhao-Rong Lai, and Cheng Li. Autonomous sparse mean-CVaR portfolio optimization. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 30440–30456. PMLR, 21–27 Jul 2024.

Appendix

A Design of EvoAlpha framework

Our framework follows a closed-loop evolutionary design where a pool of alpha factors is iteratively refined under LLM guidance. At the start, the initial factor pool is evaluated using standard metrics (IC, RankIC, ICIR), and the top-performing factors are selected as seeds. In each round, these seeds are embedded into structured prompts and passed to the LLM to generate new candidates through two operators: *mutation*, which makes small local adjustments (e.g., window size changes or normalization tweaks), and *crossover*, which combines complementary sub-expressions from different factors. The generated candidates are deduplicated, evaluated via backtesting, and merged with the existing pool. A fixed-size pool is maintained by keeping only the top-K factors ranked by IC, ensuring continuous selection pressure. This cycle repeats for a predefined number of rounds, producing an evolution history and a final pool of diverse, interpretable, and high-quality factors.

Algorithm 1 LLM-Guided Evolutionary Factor Searcher (EA_Searcher)

Require: Initial factor pool $P = \{(name, expr)\}$; mutation rate μ; crossover rate γ; candidates per round N; number of rounds R; pool size K
Ensure: Final factor pool and evolution history
Evaluate initial pool P via API $\rightarrow$ assign metrics (IC, RankIC, etc.)
Rank P by IC $\rightarrow$ current_pool
Record baseline mean IC
for $r = 1$ to R **do**
  Select top seeds $S \subset$ current_pool (size $\leq$ seeds_top_k)
  Build JSON block of seeds with {name, expression, metrics}
  Split N into $n_{\text{mut}} = \lfloor N\mu \rfloor$, $n_{\text{cross}} = \lfloor N\gamma \rfloor$ (adjust to sum N)
  Mutation: Send mutation prompt with S to LLM $\rightarrow M_{\text{candidates}}$
  Crossover: Send crossover prompt with S to LLM $\rightarrow C_{\text{candidates}}$
  Combine candidates $C = M_{\text{candidates}} \cup C_{\text{candidates}}$
  Remove duplicates by expression
  Evaluate C via API $\rightarrow$ assign metrics
  Update pool: current_pool $\leftarrow$ Top-K by IC from (current_pool $\cup$ C)
  Record history: prompts, candidates, metrics
  Report round summary (top IC, mean IC)
end for
return Final factor pool (size K), baseline IC, and history

To guide the large language model in generating meaningful factor candidates, we design structured prompts that explicitly encode the evolutionary operations. Each prompt has three components: (i) a definition of the task, including the round index, the number of required outputs, and the constraints on valid Qlib-style expressions; (ii) a seed block in JSON format that lists the top-performing factors from the current pool along with their metrics, providing grounded context for the LLM; and (iii) operator-specific instructions. For *mutation*, the prompt emphasizes local modifications such as adjusting window sizes, altering normalizers, or adding stability terms. For *crossover*, the prompt encourages recombining complementary sub-expressions across factors, such as merging a momentum core with a volatility normalizer. Each prompt also specifies strict output formatting rules (JSON array with name, expression, reason), ensuring the generated factors remain executable, interpretable, and directly comparable during backtesting.

[MUTATION PROMPT]

```
You are a quantitative researcher. Your task is to **mutate** existing alpha
    factors.

Round: 2
Goal: Propose **exactly 9** mutated candidates that are likely to improve
    the information coefficient (IC) while remaining valid Qlib-style
    expressions.
```

```
Seed factors (JSON; each item has "name", "expression", "metrics"):
[
  {
    "name": "SmoothedMomOverMad_d525fc0f-3e6f-4f6b-99f0-d93f6abc2fe8_5b4790",
    "expression": "Div(Mean(Delta($close, 1), 5), Add(Mad($close, 40),
    1e-12))",
    "metrics": {
      "ic": 0.0257128719240427,
      "rank_ic": 0.012954626788882515,
      "icir": 0.16971179842948914,
      "rank_icir": 0.07512629478580483
    }
  },
  {
    "name":
    "SmoothedMomentumOverRange_8eed97c5-ab31-4bf7-b8ff-d042399f4bea_b4e171",
    "expression": "Div(Mean(Delta($close, 1), 5), Add(Mean(Sub($high, $low),
    20), 1e-12))",
    "metrics": {
      "ic": 0.02426382154226303,
      "rank_ic": 0.00818133429533971,
      "icir": 0.14620926976203918,
      "rank_icir": 0.045271601689904684
    }
  },
  ......
]

What to do (Mutation):
- Tweak window lengths (e.g., 57, 1012, 2018) to control smoothness and
    responsiveness.
- Replace or insert nearby operators while preserving the core signal type
    (momentum / mean-reversion / volatility / liquidity).
- Normalize signals to reduce scale effects (e.g., divide by rolling Std or
    use Rank).
- Add light regularization tricks (e.g., small epsilon in denom, clipping
    via Min/Max) to improve numerical stability.
- Keep expressions parsable and balanced (all parentheses closed), and
    variables limited to: $close, $open, $high, $low, $volume.
- Do NOT invent new variables or unsupported ops.
- Try to use more diverse operators and window sizes than the seeds, don't
    only adjust parameters.

Examples (illustrative only; you must produce new ones):
- From: Mean(Sub($close, Ref($close, 1)), 10)
  To:   Div(Mean(Sub($close, Ref($close, 1)), 12), Add(Std($close, 60),
    1e-12))
- From: Rank(Sub($high, $low))
  To:   Rank(Div(Sub($high, $low), Add(Mean(Sub($close, Ref($close, 1)),
    20), 1e-12)))

Output format:
Return a JSON array of length 9. Each item MUST be an object with:
  - "name": a short unique name (string)
  - "expression": the full Qlib-style expression (string)
  - "reason": 12 sentences explaining the mutation (string)

No extra text. Output ONLY the JSON array.
```

[CROSSOVER PROMPT]

```
You are a quantitative researcher. Your task is to **crossover** existing
    alpha factors.

Round: 2
Goal: Propose **exactly 21** crossover candidates by combining complementary
    parts of the seed expressions to improve robustness and IC.

Seed factors (JSON; each item has "name", "expression", "metrics"):
[
  {
    "name":
    "SmoothedMomentumOverRange_8eed97c5-ab31-4bf7-b8ff-d042399f4bea_b4e171",
    "expression": "Div(Mean(Delta($close, 1), 5), Add(Mean(Sub($high, $low),
    20), 1e-12))",
    "metrics": {
      "ic": 0.02426382154226303,
      "rank_ic": 0.00818133429533971,
      "icir": 0.14620926976203918,
      "rank_icir": 0.045271601689904684
    }
  },
  {
    "name":
    "SmoothedMomOverHybridVol_8e02b4a3-83e9-484e-a0d6-db5f706d4594_ddb2d6",
    "expression": "Div(Mean(Delta($close, 1), 5), Add(Add(Std($close, 30),
    Mean(Sub($high, $low), 20)), 1e-12))",
    "metrics": {
      "ic": 0.02379864640533924,
      "rank_ic": 0.010518853960332362,
      "icir": 0.14901390671730042,
      "rank_icir": 0.05898742807830996
    }
  },
  ...
]

What "crossover" means here
- **Pick good parts from good factors**: identify sub-expressions that
    plausibly drive performance (e.g., momentum cores, volatility/volume
    normalizers, range/volatility proxies, smoothers, gates/filters).
- **Recombine** complementary parts across seeds to form concise, novel
    expressions (not minor edits or concatenations).

How to identify & extract good parts
1) Rank seeds by metrics (prefer higher RankIC/ICIR and stability). Skim top
    seeds first.
2) Decompose expressions into roles:
   - Core signal (e.g., Sub/Delta/Range/Momentum on $close/$high/$low)
   - Normalizer (e.g., Std/Mean/Rank with safe epsilon in denominators)
   - Volume or regime component (e.g., Mean($volume, L), Rank(...))
   - Smoother (e.g., Mean(..., L), Rank(...))
3) Extract the **short, reusable subchains** (24 ops) that carry the
    behavior (trend, mean-revert, breakout) or the stabilizer (vol/volume
    scaling).

Examples (illustrative only; you must produce new ones):
- From A: Mean(Sub($close, Ref($close, 1)), 10)
  From B: Div($volume, Add(Mean($volume, 60), 1e-12))
  To:     Div(Mean(Sub($close, Ref($close, 1)), 12), Add(Mean($volume, 60),
    1e-12))
```

```
- From A: Rank(Sub($high, $low))
  From B: Div(Sub($close, Ref($close, 1)), Add(Std($close, 30), 1e-12))
  To:     Rank(Div(Sub($high, $low), Add(Std($close, 30), 1e-12)))

Output format:
Return a JSON array of length 21. Each item MUST be an object with:
  - "name": a short unique name (string)
  - "expression": the full Qlib-style expression (string)
  - "reason": 12 sentences explaining which traits were combined and why
    (string)

No extra text. Output ONLY the JSON array.
```

To ensure that the factors produced by the LLM are executable and reliable, we employ a multi-stage filtering pipeline. After generation, outputs are first parsed into a unified schema containing name, expression, and optionally a short reasoning. Malformed or empty items are discarded, and duplicate expressions are removed to guarantee uniqueness. Each surviving candidate is then validated through several checks. First, we verify the syntactic structure: only permitted operators and variables (close, open, high, low, volume) are allowed, parentheses must be balanced, rolling window parameters must be positive integers within a reasonable range, and denominators must include small stabilizers (e.g., +1e-12). Second, we enforce diversity by ensuring that factors cover different roles such as momentum, volatility, mean-reversion, and volume-based dynamics, rather than being near-duplicates. Finally, candidates are stress-tested through execution checks: formulas that cannot be evaluated within a fixed timeout, that return invalid outputs, or that produce a large proportion of missing or NaN values are filtered out. This layered mechanism guarantees that only high-quality, interpretable, and robust factors are admitted into the evolving pool.

B Details of Experiment

B.1 Factor Pool Construction

Our experiments use the **Alpha158** library. It comprises three groups: *KBar* (candlestick relations among open/high/low/close), *Price* (simple ratios with 1-step references), and *Rolling* (operators applied over windows to expand variants). Tables 2–3 list the factors and their default expressions used as seeds in our experiments.

Table 2: Alpha158 factors (KBar and Price). Default expressions shown; type cells merged.

Type	Name	Default Expression
KBar	KMID	(close - open) / open
	KLEN	(high - low) / open
	KMID2	(close - open) / (high - low + 1e-12)
	KUP	(high - Greater(open, close)) / open
	KUP2	(high - Greater(open, close)) / (high - low + 1e-12)
	KLOW	(Less(open, close) - low) / open
	KLOW2	(Less(open, close) - low) / (high - low + 1e-12)
	KSFT	(2 * close - high - low) / open
	KSFT2	(2 * close - high - low) / (high - low + 1e-12)
Price	OPEN_REF	Ref(open, 1) / close
	HIGH_REF	Ref(high, 1) / close
	LOW_REF	Ref(low, 1) / close
	VWAP_REF	Ref(vwap, 1) / close
	VOLUME_REF	Ref(volume, 1) / (volume + 1e-12)

In our experiments, we construct a factor pool of 38 factors by selecting the Price and Rolling collections with their default window sizes.

Table 3: Alpha158 factors (Rolling). Windowed operators use default window size of 5 unless otherwise stated.

Type	Name	Default Expression
Rolling	ROC	Ref(close, 5) / close
	MA	Mean(close, 5) / close
	STD	Std(close, 5) / close
	BETA	Slope(close, 5) / close
	RSQR	Rsquare(close, 5)
	RESI	Resi(close, 5) / close
	MAX	Max(high, 5) / close
	LOW	Min(low, 5) / close
	QTLU	Quantile(close, 5, 0.8) / close
	QTLD	Quantile(close, 5, 0.2) / close
	RANK_CLOSE	Rank(close, 5)
	RSV	(close - Min(low, 5)) / (Max(high, 5) - Min(low, 5) + 1e-12)
	IMAX	IdxMax(high, 5) / 5
	IMIN	IdxMin(low, 5) / 5
	IMXD	(IdxMax(high, 5) - IdxMin(low, 5)) / 5
	CORR	Corr(close, Log(volume + 1), 5)
	CORD	Corr(close / Ref(close, 1), Log(volume / Ref(volume, 1) + 1), 5)
	CNTP	Mean(close > Ref(close, 1), 5)
	CNTN	Mean(close < Ref(close, 1), 5)
	CNTD	Mean(close > Ref(close, 1), 5) - Mean(close < Ref(close, 1), 5)
	SUMP	Sum(Greater(close - Ref(close, 1), 0), 5) / (Sum(Abs(close - Ref(close, 1)), 5) + 1e-12)
	SUMN	Sum(Greater(Ref(close, 1) - close, 0), 5) / (Sum(Abs(close - Ref(close, 1)), 5) + 1e-12)
	SUMD	(Sum(Greater(close - Ref(close, 1), 0), 5) - Sum(Greater(Ref(close, 1) - close, 0), 5)) / (Sum(Abs(close - Ref(close, 1)), 5) + 1e-12)
	VMA	Mean(volume, 5) / (volume + 1e-12)
	VSTD	Std(volume, 5) / (volume + 1e-12)
	WVMA	Std(Abs(close / Ref(close, 1) - 1) * volume, 5) / (Mean(Abs(close / Ref(close, 1) - 1) * volume, 5) + 1e-12)
	VSUMP	Sum(Greater(volume - Ref(volume, 1), 0), 5) / (Sum(Abs(volume - Ref(volume, 1)), 5) + 1e-12)
	VSUMN	Sum(Greater(Ref(volume, 1) - volume, 0), 5) / (Sum(Abs(volume - Ref(volume, 1)), 5) + 1e-12)
	VSUMD	(Sum(Greater(volume - Ref(volume, 1), 0), 5) - Sum(Greater(Ref(volume, 1) - volume, 0), 5)) / (Sum(Abs(volume - Ref(volume, 1)), 5) + 1e-12)

B.2 Searching Settings

For the **static factor search**, we adopt fixed evolutionary parameters: the crossover rate is set to 0.7, the mutation rate to 0.3, and each search is conducted for 15 rounds with a pool size of 30 candidates. This configuration allows sufficient exploration while maintaining stability of the candidate pool.

For the **sparse portfolio search**, the horizon of evaluation is much longer and the total number of search iterations becomes prohibitively large. To reduce computational cost and transaction overhead, we limit the search size to 10 candidates per iteration. In this setting, we further allow the LLM to autonomously decide whether to perform mutation or crossover in each round, instead of enforcing a fixed ratio. This design makes the search more efficient and adaptive under real market constraints.

B.3 Additional Experiment Results

Table 4 summarizes the top-5 candidate factors discovered at different rounds of the evolutionary search. For each candidate, we report the expression, provenance (mutation or crossover), and its IC score. We illustrate results from Round 1, Round 5, and Round 15.

Table 4: Top-5 candidate factors across different rounds of evolutionary search. The results demonstrate gradual refinement of factor structures and improved IC scores over iterations.

Round	Candidate	Expression	Provenance	IC
1	#1	`Div(Sum(Greater($close, Ref($close,1)), 7), Add(Mean(Abs(Sub($close, Ref($close,1))), 14), 1e-12))`	Mutation	0.026
	#2	`Div(Slope($close, 5), Add(Sub($high, $low), 1e-12))`	Crossover	0.021
	#3	`Div(Slope($close, 7), Add(Mul($close, Std($close, 20)), 1e-12))`	Mutation	0.017
	#4	`Div(Rsquare($close, 5), Add(Sum(Abs($close-Ref($close,1)), 5), 1e-12))`	Crossover	0.016
	#5	`Div(Sub(IdxMax($high, 7), IdxMin($low, 7)), 7)`	Mutation	0.013
5	#1	`Div(Div(Sum(Greater(Sub($close, Ref($close,1)),0),5), Add(Sum(Abs(Sub($close, Ref($close,1))),10),1e-12)), Add(Sub($high,$low),1e-12))`	Mutation	0.035
	#2	`Div(Sum(Greater($close, Ref($close,1)),21), Add(Mul(Sub($high, $low), WMA($close,20)),1e-12))`	Crossover	0.035
	#3	`Div(Sum(Greater($close, Ref($close,1)),14), Add(Mul(Sub($high,$low), EMA($low,14)),1e-12))`	Crossover	0.035
	#4	`Div(Mean($low,7), Add(Sub($high,$low),1e-12))`	Mutation	0.034
	#5	`Div(Div(Sum(Greater(Sub($close, Ref($close,1)),0),7), Add(Mean(Abs(Sub($close, Ref($close,1))),14),1e-12)), Add(Sub($high,$low),1e-12))`	Mutation	0.033
15	#1	`Div(Mean(Rank(Sum(Greater($close, Ref($close,1)),27),3),5), Add(Sqrt(Sub($high,$low)),1e-12))`	Mutation	0.043
	#2	`Div(Rank(Mean(Greater($close, Ref($close,1)),27),3), Add(Sqrt(Sub($high,$low)),1e-12))`	Crossover	0.043
	#3	`Div(Mean(Rank(Sum(Greater($close, Ref($close,1)),21),3),10), Add(Sqrt(Sub($high,$low)),1e-12))`	Mutation	0.042
	#4	`Div(Rank(WMA(Sum(Greater($close, Ref($close,1)),27),10),3), Add(Sqrt(Sub($high,$low)),1e-12))`	Mutation	0.042
	#5	`Div(Rank(Med(Greater($close, Ref($close,1)),24),3), Add(Sqrt(Sub($high,$low)),1e-12))`	Crossover	0.041