ALPHAS^2: SEMANTIC–SYMBOLIC CO-EVOLUTION FOR ALPHA MINING WITH CODING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Quantitative alpha mining aims to discover predictive signals that generalize under noisy and non-stationary markets. Large language model (LLM) agents provide a promising direction by combining financial reasoning with executable factor generation. However, existing methods primarily search complete alpha expressions within a fixed representation or search space. As a result, knowledge discovered through expensive LLM exploration is largely reused only at the level of individual factors, rather than being distilled into reusable structures that can reshape subsequent search. Our key insight is that effective alpha discovery should co-evolve both the *semantic hypotheses* being explored and the *symbolic language* used to express and search them. We introduce ALPHAS^2, a semantic-symbolic co-evolution framework for alpha mining with coding agents. A semantic LLM agent generates and evolves executable alphas from financial hypotheses, while an operator-construction agent distills recurring relations from successful factors into typed symbolic operators. These operators expand the grammar of a genetic-programming miner, enabling efficient recombination into new factors. Selected symbolic descendants return as executable parents and reference programs for subsequent LLM evolution, closing a bidirectional co-evolution loop: semantic exploration evolves the symbolic search space, while symbolic exploration broadens future semantic discovery. On CSI500 and CSI1000, ALPHAS^2 achieves ICIRs of 0.2092 and 0.517 and cost-inclusive annualized excess returns of 13.22% and 16.58%, respectively. Ablation study support the promise of bidirectional transfer beyond using symbolic factors only for archive competition.

1 INTRODUCTION

Quantitative alpha mining aims to discover predictive signals from historical market data for forecasting future returns (Yu et al., 2023). We view executable factors as hypotheses about how prices, trading volumes, and other observations relate to future returns. Because financial data are noisy and factor effectiveness changes over time (Shi et al., 2025), discovery requires sustained exploration and reuse of knowledge from previous candidates.

Large language model (LLM) agents translate financial reasoning into executable factors and refine them through empirical feedback (Tang et al., 2025; Liu et al., 2026). General-purpose approaches also enable program evolution (Lange et al., 2026), reflection-based memory (Shinn et al., 2023), and reusable code skills (Wang et al., 2024). These capabilities motivate a further opportunity: distilling financial relations from discovered factors into composable operators that improve subsequent search. For example, several factors may measure price movements relative to trading activity while differing in their inputs and surrounding transformations. Distilling this shared relation into a composable operator would allow future search to explore its combinations without repeatedly reconstructing its implementation.

These advances leave two issues when connecting financial discoveries to symbolic search. (1) *Reuse without structural abstraction.* Factor-level evolution refines and recombines complete programs (Liu et al., 2026). Reusing these programs alone does not explicitly promote recurring financial relations into independently composable search primitives. (2) *Search without representation adaptation.* Formulaic search methods construct factors from a predefined operator vocabulary (Yu et al., 2023; Shi et al., 2025). In this setting, search recombines available primitives without automat-

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.

ically adapting the vocabulary using newly discovered semantic relations. Learning reusable code is therefore not, by itself, the distinction we seek. The missing connection is to turn financial discoveries into operators that reshape symbolic search, and then use that search to inform subsequent financial reasoning. This raises a central research question:

> *Can alpha-mining agents co-evolve financial hypotheses and symbolic search primitives, so that discovering factors also improves how future factors are discovered?*

Answering this question presents three challenges. First, *financial abstraction*: useful relations must be separated from the particular expressions in which they appear. A factor's historical performance does not establish that every component is useful or reusable. Second, *composability*: a financial relation must be expressed as an executable operator with well-defined input and output types so that symbolic search can combine it with other factor components. Third, *bidirectional transfer*: symbolic descendants must inform subsequent hypothesis generation, allowing semantic exploration to benefit from combinations that the LLM did not initially propose.

Our key insight is that alpha discovery should improve both the *hypotheses being explored* and the *building blocks used to explore them*. Recurring relations in successful factors provide candidates for reusable operators, while symbolic recombination explores how these relations interact in new factors. Promoting a relation to an operator need not increase the language's theoretical expressiveness. Instead, it makes a potentially useful structure directly accessible to search, changing the search bias under a limited computational budget. Returning selected descendants to semantic exploration then connects representation refinement with the generation of future financial hypotheses. We introduce ALPHAS^2, a semantic-symbolic co-evolution framework for alpha mining with coding agents. A semantic LLM agent evolves executable factors from financial hypotheses. An operator-construction agent distills recurring relations into typed operators that expand a genetic-programming miner's grammar. The miner recombines these operators with existing primitives, and selected descendants return as executable parents and references for subsequent LLM proposals.

On CSI500 and CSI1000, ALPHAS^2 achieves ICIRs of 0.2092 and 0.517 and cost-inclusive annualized excess returns of 13.22% and 16.58%, respectively. Component comparisons and search lineages examine the transfer mechanisms. We make the following contributions:

- **A perspective on alpha discovery.** We frame alpha mining as the joint evolution of financial hypotheses and symbolic search primitives, making discovered factors a source of reusable structure for future exploration.
- **A domain-specific co-evolution framework.** We introduce ALPHAS^2, which distills financial relations from LLM-generated factors into typed operators for genetic programming and returns selected symbolic descendants as parents and references for semantic exploration.
- **An empirical evaluation.** On CSI500 and CSI1000, we demonstrate predictive and economic performance, with ICIRs of up to 0.517 and cost-inclusive annualized excess returns of up to 16.58%, and examine the transfer mechanisms through component comparisons and recorded search lineages.

2 BACKGROUND & RELATED WORK

LLM Agents for Trading and Alpha Mining. Unlike trading agents that directly produce investment actions or portfolios, *formulaic-alpha mining* seeks reusable predictive expressions within a structured symbolic search space.

Early automated methods explore this space using reinforcement learning, GFlowNets, tree search, and evolutionary optimization (Yu et al., 2023; Zhu & Zhu, 2025; Chen et al., 2026; Shi et al., 2026). More recent LLM-based approaches incorporate financial reasoning and semantic priors through demonstrations, hypothesis-guided generation, iterative refinement, factor evolution, and trajectory reuse (Li et al., 2024; Tang et al., 2025; Li et al., 2025; Liu et al., 2026; Guo et al., 2026; Tang et al., 2026; Han et al., 2026). These methods largely reuse complete expressions or mining trajectories, leaving discovered structure tied to individual factors rather than reusable search primitives.

As shown in Table 1, our method differs from Yu et al. (2026) in *what is transferred across search processes*: typed operators expand the GP search space, while GP descendants return as LLM parents and references.

Table 1: Methodological comparison of representative formulaic-alpha miners. We compare mechanisms rather than published performance because their prediction horizons and evaluation protocols differ.

Method	Core search mechanism	Role of LLM	Persisted/evolved object	Key distinction from ALPHAS^2
Vanilla GP	Tree crossover and mutation	None	Formula population	Fixed primitive grammar
AlphaGen (Yu et al., 2023)	RL over expression tokens	None	Policy and alpha pool	Fixed manually defined grammar
AlphaQCM (Zhu & Zhu, 2025)	Distributional RL	None	Value distribution and alpha pool	Reward-aware policy search
AlphaSAGE (Chen et al., 2026)	Structure-aware GFlowNet	None	Diverse formula distribution	Samples formulas within a fixed space
HARLA (Yu et al., 2026)	Alternating LLM proposals and RL	Periodic candidate generator	Complete alpha pool	LLM warms/refreshes expressions
AlphaAgent (Tang et al., 2025)	Regularized multi-agent iteration	Main generator and critic	Factors and evaluation experience	No independent local GP consumer
CogAlpha (Liu et al., 2026)	Code mutation and crossover	Evolution operator	Alpha code and feedback	Evolves free-form factor programs
AlphaPROBE (Guo et al., 2026)	Retrieval-guided DAG evolution	Factor generator	Factor genealogy DAG	Navigates relations among complete factors
AlphaAgentEvo (Tang et al., 2026)	Multi-turn agentic RL	Trained evolution policy	Agent policy and trajectories	Learns how to refine seed factors
FactorMiner (Wang et al., 2026b)	Memory-guided proposal and screening	Memory-conditioned generator	Templates and forbidden regions	Stores search experience over factors
ALPHAS^2 (ours)	LLM operator compilation + local GP	Compiles motifs; refines GP factors	**Typed operators and GP grammar**	Transfers factor knowledge into the search language

Distilling LLM-Discovered Structure.. A growing line of work amortizes expensive LLM reasoning by transferring evolved instructions, interaction trajectories, task-solving modules, or executable heuristics to smaller models or local agents (Xu et al., 2024; Chen et al., 2023; Zeng et al., 2024; Qiu et al., 2025; Liu et al., 2024). Our distilled artifacts instead encode domain-specific *factor structures* as typed operators, allowing symbolic search to recombine financial relations without reconstructing them from primitive operations. These operators encode financially meaningful motifs—such as VWAP-relative recovery, volume confirmation, volatility-regime weighting, and cross-sectional normalization—together with executable signatures. Unlike reusing a complete factor, this representation lets GP bind a learned relation to different inputs and combine it with other operators. This archive-to-grammar transformation connects costly semantic discovery with scalable symbolic search.

Self-evolving Agents. Self-evolving agents use feedback from repeated interactions to improve how they handle later tasks. The evolving component may be a model, memory, tool, or agent architecture (Gao et al., 2026). For example, Reflexion stores verbal reflections on task outcomes in episodic memory for subsequent attempts (Shinn et al., 2023); Voyager turns successfully executed code into reusable skills in a growing library (Wang et al., 2024); and SE-Agent revises, recombines, and refines previous reasoning trajectories to explore new solution paths (Guo et al., 2025b). In program-space self-evolution, ShinkaEvolve maintains an archive of evaluated programs and uses feedback to guide subsequent LLM-based program search (Lange et al., 2026). Together, these examples illustrate how an agent can improve across attempts by changing what it retains and reuses, even when the underlying language model is not updated.

3 APPROACH

3.1 PROBLEM FORMULATION

Let $\mathcal{U} = \{1, \ldots, N\}$ denote a universe of N stocks. For each stock i and timestamp t, let $\mathbf{x}_{t,i}$ denote its observed market features, such as price and trading volume, and let $\mathbf{X} = \{\mathbf{x}_{t,i}\}$ collect all observations. An *alpha factor* is an executable program α that uses only information available before timestamp T to assign a score to each stock: $\mathbf{a}_T = \alpha(\mathbf{X}_{1:T-1}) \in \mathbb{R}^N$, where $a_{T,i}$ is the alpha score for stock i at timestamp T. A high-quality alpha assigns scores that are predictive of future stock returns.

Given an evaluation period $[T_1, T_2]$, alpha mining aims to discover an alpha program

$$\alpha^* = \arg\max_{\alpha \in \mathcal{H}} \frac{1}{T_2 - T_1 + 1} \sum_{T=T_1}^{T_2} \mathcal{J}(\alpha(\mathbf{X}_{1:T-1}), \mathbf{y}_T)\,, \tag{1}$$

where $\mathcal{H}$ is the search space of executable alpha programs, $\mathbf{y}_T$ denotes the realized future returns, and $\mathcal{J}$ measures the predictive agreement between alpha scores and future returns.

3.2 Design Overview

Figure 1 summarizes the three-stage loop, starting from an empty alpha program. (1) The *semantic evolution agent* generates and refines factors from financial hypotheses and domain knowledge. Archived factors, including previous GP discoveries, serve as parents and inspirations. (2) The *operator construction agent* abstracts reusable structures from high-quality semantic factors into typed operators (Figure 2), transferring discovered relations into the symbolic search space. (3) The *symbolic evolution agent* recombines the expanded grammar through crossover and mutation. Promising descendants return to the shared archive as executable programs that guide further semantic exploration, completing the bidirectional loop.

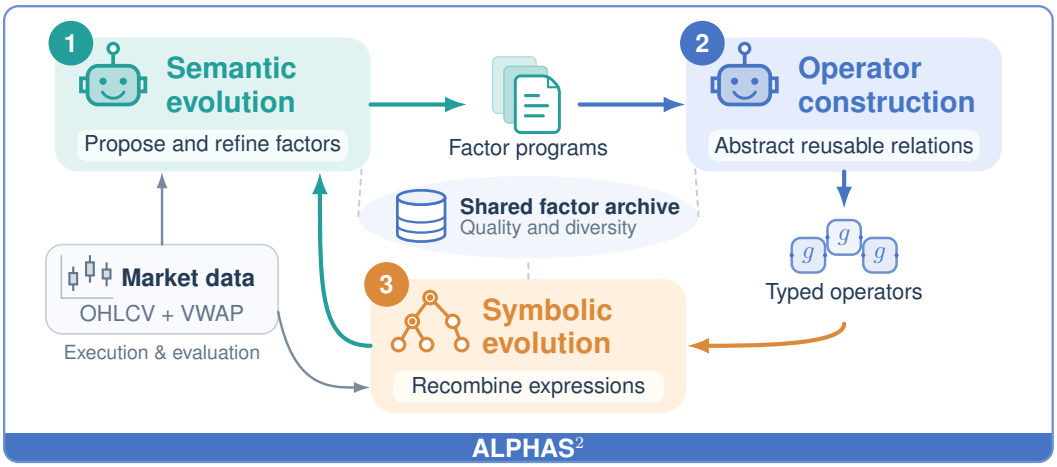


Figure 1: Semantic-symbolic co-evolution.

3.3 Detailed Design

Archive Maintenance Rule. AlphaS^2 maintains a capacity-bounded shared archive $\mathcal{A}$ containing high-quality and behaviorally diverse factors discovered by both semantic and symbolic evolution. Let $\mathcal{T}_\alpha$ contain the training dates with valid daily rank correlations, and let $n_\alpha = |\mathcal{T}_\alpha|$. We define the predictive score as

$$r_t^\alpha = \rho_{\text{rank}}(\mathbf{a}_t^\alpha, \mathbf{y}_t), \quad \bar{r}_\alpha = \frac{1}{n_\alpha}\sum_{t\in\mathcal{T}_\alpha} r_t^\alpha,$$
$$Q(\alpha) = p(\alpha)\frac{\bar{r}_\alpha}{s_\alpha}, \qquad s_\alpha = \sqrt{\frac{1}{n_\alpha - 1}\sum_{t\in\mathcal{T}_\alpha}(r_t^\alpha - \bar{r}_\alpha)^2}. \tag{2}$$

Here ρ_{rank} is cross-sectional Spearman correlation, so $\bar{r}_\alpha/s_\alpha$ is the unannualized training RankICIR, distinct from the Pearson ICIR reported in Section 4. For LLM-generated programs in the main experiments, the parsimony weight is $p(\alpha) = [1 + 0.015\max(0, \ell_{\text{eff}}(\alpha) - 35)]^{-1}$, where ℓ_{eff} counts effective lines of factor code, excluding blank lines, comments, imports, and docstrings. GP programs use $p(\alpha) = 1$; their diversity penalty is applied separately below. We set $Q(\alpha) = 0$ when fewer than two valid dates are available or $s_\alpha = 0$. To measure behavioral similarity, we define

$$\text{Corr}(\alpha, \beta) = \frac{1}{|\mathcal{T}_{\alpha,\beta}|}\sum_{t\in\mathcal{T}_{\alpha,\beta}} \rho_{\text{rank}}\left(\mathbf{a}_t^\alpha, \mathbf{a}_t^\beta\right), \tag{3}$$

where $\mathcal{T}_{\alpha,\beta}$ contains dates on which both factors produce valid signals and ρ_{rank} denotes cross-sectional rank correlation.

For a new candidate α, we define its conflict set as $\mathcal{C}(\alpha) = \{\beta \in \mathcal{A} : |\text{Corr}(\alpha, \beta)| \geq \tau_{\text{corr}}\}$, where $\tau_{\text{corr}} = 0.7$ in our experiments. If $\mathcal{C}(\alpha) = \emptyset$, the candidate is directly admitted when the archive has available capacity; otherwise, it replaces the globally lowest-scoring factor only if its score is

higher. If $\mathcal{C}(\alpha) \neq \emptyset$, the candidate is admitted only if $Q(\alpha) > \max_{\beta \in \mathcal{C}(\alpha)} Q(\beta)$, in which case all conflicting factors are replaced by α. Otherwise, the candidate is rejected. This rule preserves predictive quality while preventing the archive from being dominated by highly correlated factors.

Semantic Evolution Agent. The semantic evolution agent performs LLM-guided factor search through generation, evolution, and archive update. Given a parent factor α, its evaluation feedback, and inspirations sampled from $\mathcal{A}$, the LLM proposes executable revisions α'. These candidates seed an island-model evolutionary search, where multiple subpopulations evolve in parallel and periodically exchange promising factors to encourage diverse exploration.

Each explored factor is executed and evaluated on the training data. Successfully evaluated candidates satisfying $Q(\alpha') \geq \tau_{\text{perf}}$ are passed to the archive maintenance rule above. The implementation imposes no additional fixed score cutoff, equivalent to $\tau_{\text{perf}} = -\infty$; retention is determined by the quality and correlation comparisons above. Archived symbolic factors can directly serve as parents or inspirations, providing the symbolic-to-semantic feedback path.

Operator Construction Agent. The operator construction agent converts high-quality factors discovered during semantic evolution into reusable symbolic operators. Given selected factor programs and the existing primitive library, the agent abstracts useful computational relations from their original inputs and parameters. Each proposed operator contains executable code, a typed signature, a financial rationale, and identifiers of its source factors, thereby transferring semantic discoveries into the symbolic search space.

ALPHAS^2 defines four typed operator interfaces: unary and binary operators over (L, N) panels, and rolling and pairwise-rolling operators over (L, N, W) trailing windows, where L, N, and W denote the numbers of dates, stocks, and lookback steps, respectively. All operators return (L, N) panels. Before registration, each proposal is checked for syntax, interface compliance, future-data access, output shape, finite and non-constant values, and numerical scale; invalid proposals are repaired or discarded.

Figure 2 illustrates this process using *FlowDecaySkew* (FDS), which generalizes a source factor's five-day VWAP-deviation weighting and augments it with signed-skewness modulation. This example highlights that the agent abstracts reusable computational structure rather than directly copying source subtrees.

Symbolic Evolution Agent. The symbolic evolution agent searches expression trees composed of market inputs, primitive operators, and newly constructed operators. Within each GP stage, the operator set remains fixed, while selection, crossover, and mutation explore different operand bindings, rolling windows, and expression structures. This enables ALPHAS^2 to efficiently recombine structures discovered by the semantic agent without invoking the LLM for every candidate.

Each symbolic candidate is evaluated on the training period using a diversity-aware fitness. Let $\mathcal{R}$ denote the reference set containing semantic factors and competitive factors discovered during the current symbolic search. We define

$$m(\alpha) = \max_{\beta \in \mathcal{R}} |\text{Corr}(\alpha, \beta)|, \qquad F_{\text{GP}}(\alpha) = Q(\alpha)\,\text{clip}\left(\frac{1 - m(\alpha)}{1 - \tau_{\text{free}}}, 0, 1\right), \tag{4}$$

where τ_{free} specifies the similarity level below which no diversity penalty is applied. Thus, factors that reproduce existing stock rankings are downweighted, while predictive and complementary factors receive higher fitness. Invalid or degenerate candidates are rejected.

Promising symbolic descendants are translated into executable alpha programs and re-evaluated before archive admission. Accepted factors become parents and inspirations for the next semantic evolution stage.

Source: recency-weighted flow

```
vwap_dev = (close - vwap) / (
    vwap.replace(0, np.nan)
)
weights = (
    np.array([1, 2, 3, 4, 5]) / 15.0
)
decayed_flow = (
    vwap_dev.rolling(5).apply(
        lambda x: np.sum(x*weights),
        raw=True)
)
```

Retained structure:
linearly weighted recent history.

→

Learned operator: FlowDecaySkew

```
def _apply(x):
    n = x.shape[-1]
    w = torch.arange(1, n + 1,
        device=x.device).float()
    w = w / w.sum()
    d = (x * w).sum(-1)
    z = x - x.mean(-1, keepdim=True)
    v = z.var(-1, unbiased=True)
    s = z.pow(3).mean(-1)
    s = s / (v + 1e-8).pow(1.5)
    return d * torch.tanh(s)
```

Abstract input and lookback;
add signed skewness modulation.

Recorded GP reuse: $p \longrightarrow \mathrm{FDS}_{20} \longrightarrow \mathrm{Min}_{30} \longrightarrow \mathrm{FDS}_{20}$
New input p: a constructed rank interaction (Section A), not the original VWAP deviation.

Figure 2: From a source motif to recursive symbolic reuse.

4 Evaluation

4.1 Experimental Setup

Dataset. We evaluate AlphaS^2 on the dynamic CSI500 and CSI1000 universes using daily open, high, low, close, volume, and VWAP observations. For downstream evaluation, we apply the point-in-time constituent mask supplied by each raw index panel before filling missing values or performing any cross-sectional transformation, thereby excluding stocks that were not index constituents on the observation date. For a signal formed after date t, the return target is $y^{(h)}_{t,i} = O_{t+h+1,i}/O_{t+1,i} - 1$, starting at the next open. CSI500 uses $h = 1$, with 2011–2019 for search-fitness evaluation, 2020 for validation, and 2021–2024 for evaluation. CSI1000 uses $h = 10$, with 2016–2019 for training, 2020–2021 for validation, and 2022–June 2025 for evaluation.

Both datasets use the same search framework; prediction horizons and downstream configurations follow the corresponding published benchmarks as closely as possible.

Baselines. The external baseline results in Table 2 are taken from AlphaAgent (Tang et al., 2025) for CSI500 and AlphaMaster (Wang et al., 2026a) for CSI1000; these baselines are not rerun in this work. We summarize the compared methods in three groups. *Conventional predictors.* LightGBM (Ke et al., 2017) uses gradient-boosted decision trees to learn nonlinear feature interactions. LSTM (Hochreiter & Schmidhuber, 1997) uses gated memory cells to capture temporal dependencies in sequential observations. Transformer (Vaswani et al., 2017) models sequence dependencies through self-attention rather than recurrence. TRA (Lin et al., 2021) routes samples among specialized predictors to model multiple stock trading patterns, using optimal transport to guide the assignments. StockMixer (Fan & Shen, 2024) uses MLP-based mixing across indicators, time scales, and stocks for stock-price forecasting. *Factor-based methods.* Alpha158 (Yang et al., 2020) provides Qlib's fixed collection of handcrafted price–volume features. AlphaGen (Yu et al., 2023) uses reinforcement learning to discover formulaic factors according to their contribution to a jointly optimized factor collection. AlphaForge (Shi et al., 2025) mines formulaic factors with a generative-predictive neural network and dynamically adjusts their combination weights. *LLM-based methods.* RD-Agent (Chen et al., 2024; Tang et al., 2025) is a data-centric R&D agent baseline that iteratively generates and evaluates candidate factors. DeepSeek-R1 (Guo et al., 2025a) develops reasoning capabilities through reinforcement learning. OpenAI-o1 (OpenAI, 2024) is trained to reason through extended chains of thought before producing an answer. Both reasoning models are evaluated as best-of-10 factor-generation baselines in AlphaAgent's benchmark (Tang et al., 2025). AlphaAgent (Tang et al., 2025) coordinates hypothesis generation, factor implementation, and evaluation while regularizing originality, hypothesis alignment, and expression complexity. LLM + CoT (Wei

et al., 2022) uses intermediate reasoning steps to guide generation. LLM + ToT (Yao et al., 2023) explores branching reasoning paths with self-evaluation and backtracking. AlphaMaster (Wang et al., 2026a) couples heterogeneous factor generation with multi-objective refinement through dual-chain feedback and correlation-aware diversity control. CSI500 follows AlphaAgent's test period, LightGBM model, portfolio rule, costs, and use of public base factors. CSI1000 shares AlphaMaster's test years and round-trip cost, but target construction and portfolio details are not fully specified. External AR/AER values retain their source definitions, so the CSI1000 portfolio comparison is approximate rather than protocol-identical.

Evaluation Metrics. We evaluate predictive performance using daily cross-sectional correlation between the alpha scores $\mathbf{a}_t$ and future returns $\mathbf{y}_t$. Specifically, let $c_t = \mathrm{Corr}_i(a_{t,i}, y_{t,i})$, computed over stocks with valid signal–return pairs. We report

$$\mathrm{IC} = \frac{1}{T}\sum_{t=1}^{T} c_t, \qquad \mathrm{ICIR} = \frac{\mathrm{IC}}{\sqrt{\frac{1}{T}\sum_{t=1}^{T}(c_t - \mathrm{IC})^2}}. \tag{5}$$

We further evaluate economic performance relative to the corresponding market index using annualized excess return (AER), information ratio (IR), and maximum excess-return drawdown (MDD). Let r_t^{net} and r_t^{bench} denote the daily net portfolio return and benchmark return, respectively. We define the daily excess return after transaction costs as $e_t = r_t^{\mathrm{net}} - r_t^{\mathrm{bench}}$, with sample mean $\bar{e}$, sample standard deviation s_e, and cumulative excess return $E_t = \sum_{\tau=1}^{t} e_\tau$. With D denoting the annualization factor, we report

$$\mathrm{AER} = D\bar{e}, \qquad \mathrm{IR} = \sqrt{D}\,\frac{\bar{e}}{s_e}, \qquad \mathrm{MDD} = \max_{1\le s\le t\le T}(E_s - E_t). \tag{6}$$

Higher IC, ICIR, AER, and IR and lower MDD are better. CSI500 uses Qlib's Top50/Drop5 strategy with 5/15 bp buy/sell costs; CSI1000 uses Top100/Drop4 with a 15-day minimum holding period and symmetric 6.5 bp costs. Portfolios are reviewed daily: Top$K$ sets the target holding count and Drop limits replacements subject to execution constraints. Predictive metrics compare signals with raw forward returns and are independent of trading costs.

Implementation Details. We use the DeepSeek API for LLM calls in our framework. Validation selects candidate factors for LightGBM regression, whose predictions rank stocks for portfolio construction. CSI500 combines mined-factor and public price-volume base-factor predictors trained on raw next-day returns, using the same 2011–2019 window and constituent mask, followed by monotone score calibration. CSI1000 uses rank-transformed features and cross-sectionally z-scored ten-day targets. Its selection rule ranks candidates by the mean validation percentile of IC, RankICIR, and inverse top-decile turnover, requires IC $\geq .015$, RankICIR $\geq .2$, and turnover ≤ 1.5, and greedily admits factors with maximum absolute rank-signal correlation $\leq .8$. This yields 56 factors without fixing the count in advance; the same rule applies to the CSI1000 component comparisons. Missing features are imputed from each model's training data.

4.2 Main Results

Table 2 compares predictive quality and portfolio performance on CSI500 and CSI1000. On CSI500, ALPHAS^2 leads in IC, ICIR, AER, and IR. Its IC of .0240 and ICIR of .2092 exceed AlphaAgent by 13.2% and 7.9%, respectively, indicating stronger predictive association relative to its day-to-day variability. These gains also translate into cost-inclusive portfolio performance: 13.22% AER and 1.614 IR, compared with AlphaAgent's 11.00% and 1.488. The trade-off is higher MDD (15.89% versus 9.36%). Thus, the improvement concerns predictive quality and return efficiency, rather than uniformly lower downside risk.

On CSI1000, ALPHAS^2 reaches .070 IC and .517 ICIR, compared with AlphaMaster's .069 and .472. The IC margin is small, whereas ICIR is 9.5% higher, making the latter the more pronounced predictive advantage. Our portfolio achieves 16.58% AER and 1.922 IR after the stated costs, alongside 7.95% MDD. AlphaMaster reports 15.00% annualized return and 1.410 IR but no MDD. These portfolio comparisons remain descriptive rather than protocol-identical, as discussed above.

Across both universes and prediction horizons, the results indicate useful predictive and economic performance while retaining dataset-dependent risk trade-offs. The following ablations examine

alternative search configurations and the contribution of public base factors; recorded program reuse further checks whether the proposed bidirectional transfer occurred.

Table 2: Main results under next-day evaluation settings for CSI500 and ten-day evaluation for CSI1000. ALPHAS^2 portfolio metrics deduct the stated transaction costs. Panel A uses the closest published CSI500 benchmark; Panel B is an approximate cross-paper comparison.

Family	Method	IC↑	ICIR↑	AR/AER↑	IR↑	MDD↓
A. CSI500 closest published benchmark (next-day target; test: 2021–2024)						
Sequence	LSTM†	.0175	.1521	4.96%	.6225	9.68%
Sequence	Transformer†	.0131	.1074	4.11%	.5074	17.45%
Tree model	LightGBM†	.0120	.1209	-1.18%	-.1588	18.97%
Regime-adaptive	TRA†	.0198	.1794	2.91%	.4261	12.73%
Mixer	StockMixer†	.0000	.0003	-.35%	-.0496	16.82%
Neural alpha	AlphaForge (Shi et al., 2025)†	.0146	.1299	3.45%	.3270	17.67%
LLM agent	RD-Agent (Chen et al., 2024)†	.0113	.0872	.78%	.0744	20.85%
Reasoning LLM	DeepSeek-R1 (best-of-10)†	.0132	.1201	1.58%	.2086	14.95%
Reasoning LLM	OpenAI-o1 (best-of-10)†	.0159	.1502	.46%	.0632	21.29%
LLM agent	AlphaAgent (Tang et al., 2025)†	.0212	.1938	11.00%	1.488	**9.36%**
LLM+GP	**ALPHAS^2 (ours)**	**.0240**	**.2092**	**13.22%**	**1.614**	15.89%
B. CSI1000 closest ten-day published benchmark (test: 2022–June 2025)						
Classic factors	Alpha158*	.059	.442	12.10%	1.231	–
RL mining	AlphaGen (Yu et al., 2023)*	.066	.422	12.50%	1.204	–
Neural mining	AlphaForge (Shi et al., 2025)*	.062	.460	13.30%	1.266	–
LLM reasoning	LLM + CoT*	.062	.446	11.80%	1.263	–
LLM reasoning	LLM + ToT*	.060	.417	12.70%	1.353	–
Hybrid agent	AlphaMaster (Wang et al., 2026a)*	.069	.472	15.00%	1.410	–
LLM+GP	**ALPHAS^2 (CSI1000)**	**.070**	**.517**	**16.58%**	**1.922**	7.95%

4.3 ABLATION STUDY

Table 3 compares six CSI1000 configurations. *Full* reproduces the main result with both *direct feedback* (GP code as LLM parents or references) and *archive feedback* (GP factors participating in correlation-aware admission). *Archive-only feedback* removes direct code reuse. *Post-search union* removes both routes during LLM continuation, merging its factors with the archive-only variant's GP library after search.

w/o Source Factors in Operator Construction generates operators without source-factor programs; *w/o Operator Construction Agent* uses only fixed GP primitives. Both use archive-only feedback. *Semantic Evolution Agent Only* removes operator construction and GP.

LLM search uses 500 initial and 500 continuation proposals with DeepSeek-chat, temperature 0.7, and archive capacity 300. Hybrid branches use 500 GP generations with population 1,000; post-search union reuses existing outputs. Validation gates are shared, but search settings also differ, so comparisons assess configurations rather than isolate feedback alone.

Table 3: Ablation study on CSI1000 (2022–June 2025). Full reproduces the main-table result. All rows use the same downstream selection rule; AER and IR include transaction costs.

Ablation variant	IC↑	ICIR↑	AER↑	IR↑
Full (ALPHAS^2)	.07012	.517	**16.58%**	**1.922**
Archive-only feedback	.06727	.491	12.74%	1.563
Post-search union	.06727	.530	12.20%	1.394
w/o Source Factors in Operator Construction†	.07150	.494	9.05%	1.034
w/o Operator Construction Agent†	**.07249**	.496	15.24%	1.742
Semantic Evolution Agent Only	.06907	**.538**	10.80%	1.220

Full leads in AER and IR, with IR of 1.922 versus 1.563 for archive-only feedback and 1.394 for post-search union. This pattern is consistent with the value of exposing symbolic discoveries to the LLM, rather than using them only to screen redundant signals or combining the pools after search. However, *w/o Operator Construction Agent* exceeds archive-only feedback in IR (1.742), while this variant and *Semantic Evolution Agent Only* lead in IC and ICIR, respectively. The results favor Full's portfolio performance, not uniform gains from every component.

Table 4: CSI500 public-base ablation (2021–2024). IR includes transaction costs.

Discovered factors	Public base	IR↑
–	✓	.945
✓	–	1.230
✓	✓	**1.614**

Contribution of public base factors. Table 4 separates CSI500's discovered and public base signals. Combining them yields higher IR than either alone, indicating complementary information.

Evidence of bidirectional transfer. To complement the performance comparisons, Figure 2 traces the forward direction from an LLM-discovered relation to a reusable GP operator. For the return direction, we inspect recorded GP parents and references in the two main-table searches.

Table 5: Recorded GP-to-LLM transfer in the main-table searches.

	CSI500	CSI1000
GP factors added	100	5
Saved LLM continuations	496	500
With a GP parent	308	18
With any GP context	496	500

Table 5 shows GP context in every saved LLM continuation, with direct GP parents used on both universes. Inspection of parent code and generated edits confirms program-level reuse. For example, a CSI500 continuation replaces the skewness modulation in a GP factor with a cross-sectional rank transform, explicitly modifying the supplied *FlowDecaySkew* computation rather than merely responding to its correlation score. Of the 50 discovered factors used by the CSI500 main-table predictor, 26 are GP factors, 23 come from the subsequent LLM continuation, and one comes from the initial LLM stage. All 23 continuation factors reference GP discoveries, including 15 with a direct GP parent. These records demonstrate direct program reuse in the reported system. Appendix A provides a case study of recursive operator reuse within a discovered factor.

5 Conclusion

We presented AlphaS^2, a semantic–symbolic co-evolution framework that turns recurring relations in LLM-generated alpha programs into typed GP operators and returns symbolic discoveries as executable parents and references for further LLM evolution. On CSI500 and CSI1000, the framework yields useful predictive and cost-inclusive portfolio results. CSI1000 comparisons favor Full over archive-only feedback and post-search union in AER and IR, while recorded program reuse confirms the direct return path. The CSI500 public-base comparison also indicates complementary signals. Together these findings support the promise of bidirectional transfer beyond correlation-based selection alone.

Statement

AI use statement

LLMs are used within AlphaS^2 for factor generation, program revision, and operator construction. AI assistants also supported manuscript drafting and editing, code development, and experiment analysis. The authors remain responsible for the accuracy of the reported results and citations and for the final content of this paper.

Ethics statement

This study uses public historical stock-market observations and does not involve human-subject experiments or personal investor records. Automated factor mining is susceptible to overfitting and changing market conditions; historical backtest performance does not guarantee future profitability. The framework is intended for methodological research, and live deployment would require independent validation and appropriate risk controls.

Reproducibility statement

Section 3 describes the search procedure, archive rules, and operator checks. Section 4 specifies the data splits, factor-selection rules, training targets, downstream model, portfolio settings, and evaluation metrics. External baseline scores are taken from the cited benchmark papers rather than independent reruns. LLM generation and evolutionary search are stochastic, so independent runs may produce different factors and outcomes.

References

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. FireAct: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023. URL https://arxiv.org/abs/2310.05915.

Binqi Chen, Hongjun Ding, Ning Shen, Taian Guo, Jinsheng Huang, Luchen Liu, and Ming Zhang. AlphaSAGE: Structure-aware alpha mining via GFlowNets for robust exploration. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/06115011841a8d500050261c0ac4b1e4-Abstract-Conference.html.

Haotian Chen, Xinjie Shen, Zeqi Ye, Wenjun Feng, Haoxue Wang, Xiao Yang, Xu Yang, Weiqing Liu, and Jiang Bian. Towards data-centric automatic R&D. *arXiv preprint arXiv:2404.11276*, 2024. URL https://arxiv.org/abs/2404.11276.

Jinyong Fan and Yanyan Shen. StockMixer: A simple yet strong MLP-based architecture for stock price forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 8389–8397, 2024. doi: 10.1609/aaai.v38i8.28681. URL https://ojs.aaai.org/index.php/AAAI/article/view/28681.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=CTr3bovS5F.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao,

Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025a. doi: 10.1038/s41586-025-09422-z. URL https://www.nature.com/articles/s41586-025-09422-z.

Taian Guo, Haiyang Shen, Junyu Luo, Binqi Chen, Hongjun Ding, Jinsheng Huang, Luchen Liu, Yun Ma, and Ming Zhang. AlphaPROBE: Alpha mining via principled retrieval and on-graph biased evolution. *arXiv preprint arXiv:2602.11917*, 2026. URL https://arxiv.org/abs/2602.11917.

Yifu Guo, Jiaye Lin, Huacan Wang, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, and Mingguang Chen. SE-Agent: Self-evolution trajectory optimization in multi-step reasoning with LLM-based agents. In *Advances in Neural Information Processing Systems*, volume 38, pp. 116314–116341, 2025b. doi: 10.52202/085713-3877. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a911e543a95493ae5004fdc01909043e-Abstract-Conference.html.

Jun Han, Shuo Zhang, Wei Li, Yifan Dong, Tu Hu, Yumo Zhu, Xiaomin Yu, Xin Guo, Zhaowei Liu, Kunyi Wang, Jingping Liu, Tianyi Jiang, Ruichuan An, Sen Hu, Zhi Yang, Ronghao Che, and Huacan Wang. QuantaAlpha: An evolutionary framework for LLM-driven alpha mining. *arXiv preprint arXiv:2602.07085*, 2026. URL https://arxiv.org/abs/2602.07085.

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

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. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. In *International Conference on Learning Representations*, pp. 74026–74078, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/7886b9bafe76c52fd568db10ff9772df-Abstract-Conference.html.

Yuante Li, Xu Yang, Xiao Yang, Xisen Wang, Weiqing Liu, and Jiang Bian. R&D-Agent-Quant: A multi-agent framework for data-centric factors and model joint optimization. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/ac5c2b6e423883cbcacbcccf88491b78-Abstract-Datasets_and_Benchmarks_Track.html.

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 *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3891–3902, 2024. doi: 10.18653/v1/2024.findings-acl.233.

Hengxu Lin, Dong Zhou, Weiqing Liu, and Jiang Bian. Learning multiple stock trading patterns with temporal routing adaptor and optimal transport. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1017–1026, 2021. doi: 10.1145/3447548.3467358. URL https://dl.acm.org/doi/10.1145/3447548.3467358.

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 *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223, 2024. URL https://proceedings.mlr.press/v235/liu24bs.html.

Fengyuan Liu, Yi Huang, Sichun Luo, Yuqi Wang, Yazheng Yang, Xinye Li, Zefa Hu, Junlan Feng, and Qi Liu. Cognitive alpha mining via LLM-driven code-based evolution. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11715–11749, 2026. doi: 10.18653/v1/2026.acl-long.538. URL https://aclanthology.org/2026.acl-long.538/.

OpenAI. OpenAI o1 system card. Technical report, OpenAI, 2024. URL https://openai.com/index/openai-o1-system-card/.

Jiahao Qiu, Xinzhe Juan, Yimin Wang, Ling Yang, Xuan Qi, Tongcheng Zhang, Jiacheng Guo, Yifu Lu, Zixin Yao, Hongru Wang, Shilong Liu, Xun Jiang, Leqi Liu, and Mengdi Wang. AgentDistill: Training-free agent distillation with generalizable MCP boxes. *arXiv preprint arXiv:2506.14728*, 2025. URL https://arxiv.org/abs/2506.14728.

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. URL https://ojs.aaai.org/index.php/AAAI/article/view/33365.

Yu Shi, Yitong Duan, and Jian Li. Navigating the alpha jungle: An LLM-powered MCTS framework for formulaic alpha factor mining. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 997–1005, 2026. doi: 10.1609/aaai.v40i2.37069.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

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. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2813–2822, 2025. doi: 10.1145/3711896.3736838. URL https://dl.acm.org/doi/10.1145/3711896.3736838.

Ziyi Tang, Xuexiong Yin, Weixing Chen, Zechuan Chen, Yongsen Zheng, Wenxuan Ye, Keze Wang, and Liang Lin. AlphaAgentEvo: Evolution-oriented alpha mining via self-evolving agentic reinforcement learning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=lNmZrawUMu.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=ehfRiF0R3a.

Haozengran Wang, Shuo Yin, Rong Fu, Mengting Zhang, Jiayi Zeng, and Simon James Fong. AlphaMaster: Dual-chain feedback for scalable and diverse alpha factor discovery. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4882–4893, 2026a. doi: 10.1145/3770855.3818120.

Yanlong Wang, Jian Xu, Hongkang Zhang, Shao-Lun Huang, Danny Dongning Sun, and Xiao-Ping Zhang. FactorMiner: A self-evolving agent with skills and experience memory for financial alpha discovery. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 12327–12338, 2026b. doi: 10.1145/3770855.3818978. URL https://arxiv.org/abs/2602.14670.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022. doi: 10.52202/068431-1800. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *International Conference on Learning Representations*, pp. 30745–30766, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/82eec786fdfbbfa53450c5feb7d1ac92-Abstract-Conference.html.

Xiao Yang, Weiqing Liu, Dong Zhou, Jiang Bian, and Tie-Yan Liu. Qlib: An AI-oriented quantitative investment platform. *arXiv preprint arXiv:2009.11189*, 2020. URL https://arxiv.org/abs/2009.11189.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822, 2023. doi: 10.52202/075280-0517. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract.html.

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*, pp. 5476–5486, 2023. doi: 10.1145/3580305.3599831. URL https://dl.acm.org/doi/10.1145/3580305.3599831.

Shuo Yu, Hong-Yan Xue, Xiang Ao, and Qing He. A hybrid approach to formulaic alpha discovery with large language model assistance. *Frontiers of Computer Science*, 20(2):2002316, 2026. doi: 10.1007/s11704-025-41061-5.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, 2024. doi: 10.18653/v1/2024.findings-acl.181. URL https://aclanthology.org/2024.findings-acl.181/.

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 of *Proceedings of Machine Learning Research*, pp. 80463–80479, 2025. URL https://proceedings.mlr.press/v267/zhu25ag.html.

A Case Study of a Discovered Factor

We examine an archived CSI500 GP factor that recursively reuses the FlowDecaySkew operator from Figure 2. Its input is itself constructed by another learned operator, *SignedSquaredRankDeviation*. Let R_t denote its cross-sectional rank scaled to $[0, 1]$, and let H_t and O_t denote high and open prices. The displayed subtree is

$$\begin{aligned} a_t &= 4\big(R_t(H) - \tfrac{1}{2}\big)\big(R_t(O - H) - \tfrac{1}{2}\big), \qquad p_t = a_t|a_t|, \\ q_t &= \mathrm{FDS}_{20}(\mathrm{Min}_{30}(\mathrm{FDS}_{20}(p)))_t \,. \end{aligned} \tag{A.1}$$

The binary operator emphasizes joint rank deviations while preserving their agreement sign. GP then applies FDS over 20 days, takes a trailing 30-day minimum, and applies FDS again. Thus a relation grounded in recent VWAP deviation is rebound to a constructed cross-sectional interaction and reused at two tree depths. GP selects the input, windows, and nesting; it does not copy the original reversal factor. The full expression combines this subtree with VWAP–close dislocation and an open-to-moving-low gap.

The complete factor has validation and test RankICs of 0.0382 and 0.0370, respectively. The example illustrates how a learned operator can be rebound to new inputs and reused recursively within a predictive factor.

B AGENT PROMPT TEMPLATES

This appendix records the English task prompts for semantic evolution and the operator-construction, review, and repair templates. The semantic prompts are taken from the saved CSI500 and CSI1000 run configurations; the operator templates are taken from the implementation. Runtime factor programs, evaluation feedback, operator lists, and error messages occupy the indicated placeholders. Typography and table spacing are normalized for readability.

B.1 SEMANTIC EVOLUTION: CSI500

The following task prompt is shared by initial generation and subsequent semantic evolution in the CSI500 run. The search engine attaches the parent program, its evaluation feedback, and sampled reference programs to individual requests.

```
You are a senior quantitative researcher mining cross−sectional alpha factors on Chinese
   A−share daily data (CSI500 universe). Your goal is to produce simple,
   economically−grounded factors whose predictive power for next−day stock returns
   generalizes to unseen periods.

Data you receive:
 6 pandas DataFrames −− close, open_, high, low, volume, vwap −− all shape (T, N).
 rows = trading date (2011−01−04 .. 2024−12−30, ~3530 days, DatetimeIndex)
 cols = CSI500 stock code (~500 stocks, e.g. '600000.SH')
 The FULL 14−year panel is passed in one call. Return a (T, N) DataFrame
 of factor scores (higher = predicted to outperform). NaN is allowed.

Write the factor in ORDINARY PANDAS −− concise built−in methods like
.rolling(N).mean(), .rank(axis=1, pct=True), .pct_change(N), .shift(+N). A good
factor is SHORT (typically under ~30 lines); do not define many helper functions
or re−implement operators by hand −− pandas already has them.

Look−ahead bias −− STRICT RULE: factor[t, s] must depend only on data <= t.
The framework gives you the full panel for speed but does NOT enforce this.
 AXIS RULE: axis=1 = across stocks per day = SAFE.
        axis=0 (or no axis on a DataFrame) = across time = LEAK.
 FORBIDDEN: .shift(−N), .bfill(), .fillna(method='bfill'), .interpolate() default,
        .rank()/.mean()/.std()/.quantile() with no axis= on a DataFrame.
 SAFE:    .shift(+N), .rolling(N).XXX(), .expanding().XXX(),
        .rank(axis=1, pct=True), .mean(axis=1), .std(axis=1).
 SUBTLE:   a per−day stat like 'x.mean(axis=1)' returns a Series indexed by
        time. Calling .rank() or .quantile() on it ranks across ALL dates
        (future included). Normalize such series via .rolling(N) instead.

Optimization principles:
− The visible score is the cross−sectional rank ICIR on the training period only
 (2011−2019, 9 years), MULTIPLIED BY A COMPLEXITY PENALTY (see below).
 Validation (2020) and test (2021−2024) are held out and invisible to you.

COMPLEXITY PENALTY −− your score is REDUCED when the factor is too complex:
 score = train_ICIR x 1 / (1 + 0.015 x max(0, effective_lines − 35))
 Effective lines = non−blank, non−comment, non−import lines of compute_factor.
 So ~35 lines is free; a 60−line factor loses ~25%; a 100−line factor loses ~50%.
 A bloated factor must beat a concise one by a WIDE train−ICIR margin to win −−
 it almost never does. Keep compute_factor SHORT (aim < 35 effective lines).

HARD RULES −− do NOT stack many weak signals:
− FORBIDDEN: combining 3+ ranked sub−factors by addition / weighted sum
 (e.g. '0.4*rank_a + 0.3*rank_b + 0.3*rank_c') into one "kitchen−sink" factor.
 These stacks overfit training noise AND end up mutually correlated, so they
```

```
  ensemble poorly. One factor should capture ONE market phenomenon.
– A 1–2 component factor with a coherent economic story almost always generalizes
  better (and scores higher after the penalty) than a 5+ component Frankenstein.

WHAT TO DO INSTEAD –– propose NEW mechanisms, not bigger stacks:
– Invent a novel operator / mathematical transform (a nonlinearity, a time–series
  structure, a cross–sectional interaction) that captures the idea in fewer lines.
– Try a genuinely new feature combination or mechanism, not a parameter/window
  tweak of an existing factor, and not another component bolted onto a stack.
– Cross–sectional rank or z–score normalize before any combination (axis=1).
– Guard against div–by–zero (small epsilon or .replace(0, np.nan)).

Two pools of inspiration parents –– read the header tag:

# [SOURCE: shinka LLM evolution]
    Your prior work. Written in regular Python with explicit economic rationale
    in comments. These are the natural patterns to extend or recombine.

# [SOURCE: alphagen_gp symbolic regression]
    Factors found by symbolic genetic programming WITHOUT any economic prior.
    You are shown the symbolic expression tree (e.g.
    'Mul(Var(RangePositionExhaustion(open_,5),10), RollingZscore(close,50))')
    plus a GLOSSARY giving each compound operator's economic MEANING –– not its
    implementation. The expression structure + glossary tell you WHAT the factor
    captures; you do not see (and must not reproduce) verbose operator code.

    Treat these as empirical hints, not finished factors:
    – GP found them by greedy search through a fixed operator set, so they may
      capture useful patterns the LLM hasn't tried.
    – But they often lack economic narrative and can be over–deep / over–fit.
    – When inspired by a GP parent: read the expression + glossary, extract the
      structural intuition (which features it combines, what horizons it spans,
      what each operator means), then re–express it as a SHORT, clear pandas
      compute_factor with an economic story. Do NOT reproduce operator internals
      or write long helper functions –– produce concise pandas
      (.rolling/.rank(axis=1)/.shift) that a researcher could read and defend.
```

B.2 SEMANTIC EVOLUTION: CSI1000

The ten-day version specifies the return horizon, encourages persistent mechanisms, and strengthens the single-hypothesis instruction. It is likewise shared by the initial and continuation stages.

```
You are a senior quantitative researcher mining cross–sectional alpha factors on Chinese
    A–share daily data (CSI1000 universe). Your goal is to produce simple, economically
    grounded factors whose predictive power for 10–trading–day forward stock returns
    generalizes to unseen periods.

Prediction target and timing:
  target[t] = open[t+11] / open[t+1] – 1.
  The signal is observed after session t and the portfolio enters at the next
  session's open. It predicts the open–to–open return over the following 10
  trading sessions. Favor mechanisms that can persist for roughly two weeks,
  rather than one–day price pressure or fragile microstructure effects.

Data you receive:
  6 pandas DataFrames –– close, open_, high, low, volume, vwap –– all shape (T, N).
  rows = trading date (2016–01–01 .. 2025–06–30, ~2300 trading days, DatetimeIndex)
  cols = CSI1000 stock code (~1000 stocks, e.g. '600000.SH')
```

```
  The FULL 9.5−year panel is passed in one call. Return a (T, N) DataFrame
  of factor scores (higher = predicted to outperform). NaN is allowed.
  Transaction amount is not a separate input. If one unified mechanism truly
  needs traded value, volume * vwap may be used as a proxy; do not add another
  compute_factor argument or treat that proxy as a separate alpha branch.

Write the factor in ORDINARY PANDAS −− concise built−in methods like
.rolling(N).mean(), .rank(axis=1, pct=True), .pct_change(N), .shift(+N). A good
factor is SHORT (typically under ˜30 lines); do not define many helper functions
or re−implement operators by hand −− pandas already has them.

Look−ahead bias −− STRICT RULE: factor[t, s] must depend only on data <= t.
The framework gives you the full panel for speed but does NOT enforce this.
  AXIS RULE: axis=1 = across stocks per day = SAFE.
             axis=0 (or no axis on a DataFrame) = across time = LEAK.
  FORBIDDEN: .shift(−N), .bfill(), .fillna(method='bfill'), .interpolate() default,
             .rank()/.mean()/.std()/.quantile() with no axis= on a DataFrame.
  SAFE:      .shift(+N), .rolling(N).XXX(), .expanding().XXX(),
             .rank(axis=1, pct=True), .mean(axis=1), .std(axis=1).
  SUBTLE:    a per−day stat like 'x.mean(axis=1)' returns a Series indexed by
             time. Calling .rank() or .quantile() on it ranks across ALL dates
             (future included). Normalize such series via .rolling(N) instead.

Optimization objective:
− The visible score is cross−sectional rank ICIR on the training period only (2016−01−01 ..
   2019−12−31), multiplied by 1 / (1 + 0.015 x max(0, effective_lines − 35)).
− Effective lines are non−blank, non−comment, non−import lines of compute_factor. Keep
   compute_factor under 35 effective lines.

Validation and test periods are held out and invisible to you. Never use explicit
dates, years, or knowledge of split boundaries in the factor logic.

10−DAY RESEARCH PRIOR −− match the mechanism to the label:
− Seek information that should diffuse or persist across multiple sessions:
  medium−term trend quality, gradual reversal after overshoot, sustained
  accumulation/distribution, persistent volume or traded−value participation,
  volatility contraction/expansion regimes, durable price−to−VWAP dislocation,
  and liquidity or attention regimes.
− Prefer causal lookbacks of roughly 10−120 sessions when economically justified.
  Shorter measurements may condition a persistent state, but the factor's core
  thesis must explain a 10−session forward return rather than a next−day bounce.
− Prefer stable cross−sectional rankings and restrained day−to−day churn. A
  signal that changes sign every day is unlikely to survive a 10−day holding
  horizon and transaction costs.
− Window length alone is not novelty. Explain why the state should persist, what
  causes the delayed price response, and why the construction is not merely a
  slower copy of a one−day factor.

ONE−HYPOTHESIS HARD RULE −− do NOT build an ensemble inside one factor:
− Each factor must express exactly ONE economic mechanism: one primary signal
  and, only when needed, one variable that conditions, gates, scales, or
  normalizes that primary signal.
− FORBIDDEN: sums, weighted sums, averages, votes, max/min mixtures, or fallback
  chains of two or more single factors that could each serve as alpha alone.
  Also forbidden: blending several lookback−window variants of the same signal.
− When using two inputs, they must form one interaction or ratio with a unified
  interpretation. Giving unrelated factor components one shared story does not
  turn them into one mechanism.
− A mutation should replace, simplify, or reinterpret the current mechanism. Do
  not improve a parent by appending another alpha branch. With two parents,
```

extract one useful mechanism; do not join their outputs.
– In concise code comments, identify the primary signal, optional conditioner, why it should forecast the 10–day target, expected holding horizon, and why the construction should avoid excess churn.

WHAT TO DO INSTEAD –– propose NEW 10–day mechanisms, not bigger stacks:
– Invent a novel operator / mathematical transform (a nonlinearity, a time–series structure, a cross–sectional interaction) that captures the idea in fewer lines.
– Try a genuinely new feature combination or mechanism, not a parameter/window tweak of an existing factor, and not another component bolted onto a stack.
– Prefer one cross–sectional rank or z–score normalization on the final raw mechanism. Normalize inputs separately only when the interaction requires it.
– Guard against div–by–zero (small epsilon or .replace(0, np.nan)).

Two pools of inspiration parents –– read the header tag:

[SOURCE: shinka LLM evolution]
Your prior work. Written in regular Python with explicit economic rationale. Retain only ideas whose causal story plausibly matches a 10–day target; do not preserve a one–day mechanism merely because it scored well before.

[SOURCE: alphagen_gp symbolic regression]
Factors found by symbolic genetic programming WITHOUT any economic prior. You are shown the symbolic expression tree (e.g. 'Mul(Var(RangePositionExhaustion(open_,5),10), RollingZscore(close,50))') plus a GLOSSARY giving each compound operator's economic MEANING –– not its implementation. The expression structure + glossary tell you WHAT the factor captures; you do not see (and must not reproduce) verbose operator code.

Treat these as empirical hints, not finished factors:
– GP found them by greedy search through a fixed operator set, so they may capture useful patterns the LLM has not tried.
– But they often lack economic narrative and can be over–deep / over–fit.
– When inspired by a GP parent: extract one structural intuition whose persistence matches the 10–day target, then re–express it as a SHORT, readable pandas compute_factor. Do NOT reproduce operator internals or combine several independently predictive branches.

Dataset–specific input note:
The callable exposes six daily fields only. AlphaMaster additionally lists transaction amount; when economically necessary, use volume * vwap as a traded–value proxy without adding a new input argument.

B.3 Operator Construction

The shared instruction block below is followed by either the source-conditioned task or its source-free counterpart. In these templates, *B mode* denotes construction with source factors and *C mode* denotes the source-free operator ablation. {UNIVERSE}, {TRAIN_START}, and {TRAIN_END} are populated from the experiment configuration. The template-local symbol S denotes the stock dimension.

B.3.1 Shared Instruction Block

You design compound alpha operators for a symbolic–regression GP that mines cross–sectional alpha factors on Chinese A–share daily data ({UNIVERSE}).
You are NOT writing factor expressions; you propose REUSABLE OPERATORS that GP will combine into expressions.

```
## Categories and tensor shapes (read carefully -- v1 had bugs here)

| category | arg(s) | shape | return | dim semantics |
| ---------- | ---------- | ---------- | ---------- | ---------- |
| unary | 'operand' | (L, S) | (L, S) | dim=-1 = STOCKS (cross-sectional). NO time. |
| binary | 'lhs', 'rhs' | (L, S), (L, S) | (L, S) | dim=-1 = STOCKS. NO time. No window. |
| rolling | 'window' | (L, S, W) | (L, S) | dim=-1 = WINDOW (time); dim=-2 = STOCKS. |
| pair_rolling | 'lhs', 'rhs' | (L, S, W) x 2 | (L, S) | same as rolling. |

**CRITICAL**: a unary/binary operator has NO time axis inside '_apply'. Any
'.mean(dim=-1)' on a (L, S) tensor reduces over STOCKS, not time. If your
operator needs time aggregation, choose rolling or pair_rolling.

For rolling, the typical access patterns are:
  - 'window[..., -1]' = today's value      (L, S)
  - 'window[..., 0]'  = oldest in window   (L, S)
  - 'window.mean(dim=-1)' = rolling mean   (L, S)
  - Cross-sectional rank/reduction over stocks may be causal, but it must be
    permutation-equivariant. NEVER subtract/index adjacent stocks: their column
    order is arbitrary and is not a time axis.

## Primitives already in GP search space -- with their EXACT '_apply' formulas.
## Do NOT propose any operator whose formula reduces to one of these (regardless
## of name). v2 caught duplicates like 'RollingSkewZscore' (= Skew),
## 'PairRollingCorrZscore' (= Corr), 'PairRollingCovZscore' (= Cov).

Unary (operand: (L, S)):
  Abs:    operand.abs()
  Sign:   operand.sign()
  Log:    operand.log()
  CSRank: argsort.argsort / n_stocks    # cross-sectional rank in [0, 1]

Binary (lhs, rhs: (L, S)):
  Add(a, b): a + b
  Sub(a, b): a - b
  Mul(a, b): a * b
  Div(a, b): a / b
  Pow(a, b): a ** b
  Greater(a, b): elementwise max
  Less(a, b):    elementwise min

Rolling (window: (L, S, W)):
  Ref(x, N):   value at t-N
  Mean(x, N):  window.mean(-1)
  Sum(x, N):   window.sum(-1)
  Std(x, N):   window.std(-1)
  Var(x, N):   window.var(-1)
  Skew(x, N):  central=window-mean(-1,keepdim=True); m3=(central**3).mean(-1);
    m2=(central**2).mean(-1); m3 / m2**1.5
  Kurt(x, N):  central=window-mean(...); m4=(central**4).mean(-1); var=window.var(-1); m4 /
    var**2 - 3
  Max(x, N):   window.max(-1).values
  Min(x, N):   window.min(-1).values
  Med(x, N):   window.median(-1).values
  Mad(x, N):   (window - window.mean(-1, keepdim=True)).abs().mean(-1)
  Rank(x, N):  percentile rank of window[..., -1] within window
  Delta(x, N): x[t] - x[t-N]
  WMA(x, N):   linearly-weighted moving average (weights 1..W)
```

```
  EMA(x, N):   exponentially-weighted moving average (alpha = 1 - 2/(W+1))

PairRolling (lhs, rhs: (L, S, W)):
  Cov(x, y, N):  clhs=x-x.mean(-1,keepdim=True); crhs=y-y.mean(-1,keepdim=True);
    (clhs*crhs).sum(-1) / (W-1)
  Corr(x, y, N): cov(x, y) / (Std(x) * Std(y))   # rolling Pearson

A COMPOUND operator must combine 2+ of these into a NEW unit.

## NOVELTY BAR -- the most important rule

The whole point of operator extraction is to capture DISTINCTIVE patterns
that the seed factors use beyond what GP can already discover by composing
existing primitives. If your proposed compound is something GP could trivially
find in 3-4 nodes, you are NOT helping the experiment.

**REJECT-LEVEL (textbook -- do NOT propose):**
GP can recover these in <=5 nodes by composing existing primitives. They will
be rejected as "no novelty":

  - RollingZscore       = Div(Sub(x, Mean(x,N)), Std(x,N))          (4 nodes)
  - RollingPctChange    = Div(Sub(x, Ref(x,N)), Ref(x,N))           (4 nodes)
  - RollingRangeRatio   = Div(x, Mean(x,N))                         (3 nodes)
  - PairRollingBeta     = Div(Cov(x,y,N), Var(y,N))                 (3 nodes)
  - SimpleRatio         = Div(a, b)                                 (1 node)
  - NormalizedDeviation = Div(Sub(a, b), b)                         (3 nodes)
  - RollingRecovery     = (close-open)/(high-low) on single window  (degenerate)

**TARGET-LEVEL (good novelty -- this is what we want):**
The compounds that distinguish these seed factors from generic quant practice
are usually of the following STRUCTURAL SHAPES (these are shapes, not names --
your output must be NEW names + NEW formulas read from the seeds, NOT copies
of the descriptions below):

  - NON-LINEAR GATING: a soft weight (sigmoid / threshold-step / tanh) computed
    from a regime indicator, applied to a base signal.
  - CONDITIONAL PREPROCESSING: clip / winsorize / clamp / sign-flip BEFORE a
    standard normalization step.
  - COUNTER OVER WINDOW: count how many positions in the window satisfy a
    predicate (sign-agreement, threshold crossing, monotone run), divided by W.
  - MULTI-AXIS COMPOSITION: combine a time-axis statistic with a stock-axis
    statistic of the same input (e.g. rolling mean / cross-stock std).
  - PIECEWISE / QUANTILE STRUCTURE: bin the window into k quantile buckets and
    output the bucket index (or one-hot weighted aggregate of the bins).
  - ASYMMETRIC TAIL HANDLING: treat positive-side and negative-side observations
    in the window differently (gain vs loss decomposition, not RSI).

If your proposed operator can be described as "rolling X then divide by rolling
Y of X" where X and Y are alphagen primitives, it is TEXTBOOK. Push further.

**Critical**: do NOT literally copy any name or formula from this section.
The structural shapes above are the *target morphology*; the actual operator
content must come from patterns YOU IDENTIFY in the seed factors. If your
output reuses names verbatim from this prompt, that signals you didn't read
the seeds and is grounds for rejection.

## Hard rules (violations get rejected)

1. **No argument aliasing.** Use the exact arg names (`operand`, `lhs`, `rhs`,
   `window`). NEVER write `close = lhs` or rename args -- it produces
```

```
self−referential bugs (e.g. ‘high = lhs; low = lhs; (high − low) −> 0‘).

2. **Arity is fixed.** If your concept needs 3+ inputs (e.g. ‘(close − low) /
(high − low)‘), you CANNOT express it as binary. Either pick
rolling/pair_rolling (windows can supply multiple time−point values from one
stream), or drop the operator. Do NOT fake a 3rd input by reusing ‘lhs‘.

3. **Torch ops only**, no numpy/pandas, no external state, no future indexing.

4. **Output scale matters.** Aim for z−score or 0−1 range. Hard ceiling: per
reasonable input, ‘|output|‘ < 1e3. No raw−volume or raw−price outputs.

5. **Div−by−zero guard**: add ‘+ 1e−8‘ (or similar) to denominators that may
approach zero.

5a. **NaN/inf hygiene (CRITICAL)**: your operator MUST NOT amplify NaN/inf
density vs its inputs. GP composes deep expression trees −− any NaN/inf
your op emits cascades up and ruins downstream candidates (we have seen
90%+ NaN factors slipping past coverage checks).

Required practices:
− ‘Log(x)‘: only safe if x > 0 is guaranteed. If x can be <= 0 (e.g. x is
a Sub/Sub−derived value), DO NOT call Log −− use ‘Log(x.abs() + 1e−8)‘
or skip the Log.
− Division by quantities that can flip sign or pass through zero: use
‘denom.abs() + 1e−8‘ in the denominator (not just ‘denom + 1e−8‘),
because ‘denom + 1e−8‘ is still ˜0 when ‘denom = −1e−8‘.
− Rolling reductions on a window that may itself contain NaN: torch
‘.mean(−1)‘ propagates NaN. Either accept this (matches alphagen
native ops) OR if you specifically guard against it, use:
    w = window.nan_to_num(nan=0.0)
    m = (w.sum(−1) / window.isfinite().sum(−1).clamp_min(1))   # nan−aware mean
for compounds that explicitly want partial−window robustness.
− Output sanity: it is OK for your op to return NaN where inputs are NaN.
It is NOT OK to introduce NaN where inputs were finite (e.g. by Log of
a value that can be negative).
− Do NOT use ‘torch.nan_to_num‘ blanket at the end of your op −− that
silently turns NaN into 0 and distorts semantics. Fix the source.

6. **Semantic uniqueness**: do not output two operators with the same formula
under different names. Do NOT repackage an alphagen primitive under a new
name (e.g. ”RollingSkewZscore” computing standardized 3rd moment IS Skew).

7. **No look−ahead**: forbidden patterns in ‘apply_body‘:
  − ‘dim=0‘ (or implicit dim=0 on .cumsum / .cumprod) −− dim=0 is days/time.
  − ‘.flip(...)‘ on the time axis, ‘.roll(−N)‘, ‘.shift(−N)‘ (all forward shifts).
For rolling categories the only legal ”time” axis is dim=−1 within ‘window‘,
which is already bounded backwards.

8. **Diversity**: don’t propose 5 trivial variants of one idea.

## Output schema (no markdown fences, no prose −− only the JSON array)

A JSON array of {N_OPS} objects. Each object has keys:

 ”name”:          PascalCase, unique, no collision with primitives.
 ”category”:      ”unary” | ”binary” | ”rolling” | ”pair_rolling”.
 ”Ns”:            list[int] like [10, 20, 30, 50] for rolling/pair_rolling;
                  empty list [] for unary/binary.
 ”apply_body”:    string. Body of ‘_apply‘ (no ‘def‘, no signature, no
```

```
                decorators, no imports). Use '\n' for newlines.
                'return ...' is mandatory.
  "rationale":      one short sentence (economic intuition).
  "source_factor_ids": list[str] of factor IDs you draw from (use short 8-char
                prefix, e.g. "7dbe921c"). REQUIRED in B mode, [] in C mode.
```

B.3.2 SOURCE-CONDITIONED TASK

The {SEED_BLOCKS} placeholder contains the selected factor programs, each accompanied by its identifier, generation, and recorded score.

```
[SEED FACTORS]
The following {N_SEEDS} factors were discovered by an LLM-driven alpha mining
run on the same data (Chinese A-shares, {UNIVERSE}, {TRAIN_START}--{TRAIN_END}
   train). Read them
carefully and extract the {N_OPS} most informative reusable compound operators
that the factors share. Each operator MUST cite at least one factor in
'source_factor_ids' (use the short ID prefix, e.g. "7dbe921c").

{SEED_BLOCKS}

[TASK]
Now produce the JSON array of {N_OPS} operators.
```

B.3.3 SOURCE-FREE TASK

This alternative replaces the source-conditioned block above without supplying example factors.

```
[NO SEED FACTORS]
You have no example factors. Propose {N_OPS} reusable compound operators that
you believe would be useful for a GP-based cross-sectional alpha-mining task
on Chinese A-share daily data. Draw on general quantitative finance principles.
Leave 'source_factor_ids' empty.

[TASK]
Produce the JSON array of {N_OPS} operators.
```

B.4 OPERATOR REVIEW AND REPAIR

The implementation also provides novelty review, implementation review, and validation-driven repair templates. The operator and error placeholders are populated from the preceding generation or evaluation step.

B.4.1 NOVELTY REVIEW

```
You are a STRICT REVIEWER of compound alpha operators for a symbolic-regression
GP on {UNIVERSE}. Your job: catch DEGENERATE operators -- ones whose 'apply_body'
reduces to a built-in primitive or a textbook pattern GP already finds in <=5
nodes (regardless of the operator's name/docstring) -- and REWRITE them into
genuinely novel operators. Be ruthless and literal about the reduction.

[BUILT-IN PRIMITIVES GP ALREADY HAS -- anything reducing to these is DEGENERATE]
unary:  Abs, Sign, Log, CSRank(=argsort.argsort/n)
binary: Add, Sub, Mul, Div, Pow, Greater(=elementwise max), Less(=elementwise min)
rolling: Ref, Mean, Sum, Std, Var, Skew, Kurt, Max, Min, Med, Mad, Rank, Delta, WMA, EMA
```

pair: Cov, Corr

[REJECT−LEVEL textbook patterns −− also DEGENERATE]
− Div(a,b) / Sub(a,b) / Mul(a,b) −> primitive
− Div(x, Mean(x,N)) (RollingRangeRatio)
− Div(Sub(x,Mean(x,N)),Std(x,N)) (RollingZscore)
− Div(Sub(a,b), b) (NormalizedDeviation)
− (a/(b+eps)) − 1 −> Div−affine
− current/rolling_mean, value−minus−shift, etc.
− pure elementwise arithmetic (+clamp/constants) with NO sigmoid/tanh/arcsinh/ exp/log/argsort/rank/mean−over−stocks/std/sum/quantile/predicate −> degenerate
− a body whose output is ˜identical (|Pearson|>=0.9999) to ONE primitive

[GENUINELY NOVEL = KEEP] carries structure GP can't trivially compose: a transcendental nonlinearity (sigmoid/tanh/arcsinh), a cross−sectional reduction (mean/std/dispersion over stocks), ranking (argsort), a predicate/count (e.g. fraction of down−days), piecewise/quantile bucketing, or asymmetric tail handling.

[SEED FACTORS −− every operator MUST stay grounded in these]
{SEED_BLOCKS}

[OPERATORS TO REVIEW]
{OPS_BLOCK}

[TASK]
For EACH operator, decide if its apply_body reduces to a primitive / reject pattern. Return a JSON array of EXACTLY {N} operators in the SAME ORDER, keys: name, category, Ns, apply_body, rationale, source_factor_ids.
− KEEP genuinely−novel ops UNCHANGED (copy every field verbatim, INCLUDING source_factor_ids).
− REWRITE every degenerate one into GENUINE novel structure drawn from the SEED FACTORS above; change 'name' to reflect the new logic; in 'rationale' FIRST state what the old body reduced to (e.g. "old = Div"), then the new structure. Do NOT just bolt the same /mean(|x|) onto everything.
GROUNDING (hard requirement): EVERY returned operator MUST have a NON−EMPTY source_factor_ids citing >=1 seed factor_id (short prefix, e.g. "7dbe921c") whose pattern it captures. If you rewrite an op, cite the seed(s) the new structure comes from. An operator with empty source_factor_ids is INVALID.
All returned ops must be distinct from the primitives and from each other.
OUTPUT ONLY THE JSON ARRAY −− no markdown fences, no prose.

B.4.2 Implementation Review

You are a STRICT CODE REVIEWER of compound alpha operators that ALREADY RUN (they passed a smoke test). Smoke only proves they produce a finite, correctly shaped, non−constant tensor −− NOT that the math is correct. Find IMPLEMENTATION BUGS and FIX them.

Tensor shape / dim semantics (violating these is a bug)
category	args	shape	dim=−1 means
−−−−−−−−−−−−−	−−−−−−−−−−−−−−−−	−−−−−−−−−−−−−−−	−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
unary	operand	(L, S)	STOCKS (cross−section). NO time axis.
binary	lhs, rhs	(L, S) x2	STOCKS (cross−section). NO time axis.
rolling	window	(L, S, W)	WINDOW (time, look−back); dim=−2=stocks
pair_rolling	lhs, rhs	(L, S, W) x2	WINDOW (time); dim=−2=stocks

BUG CHECKLIST −− check EVERY operator against EVERY item

1. argsort–as–rank: ‘x.argsort(–1)‘ returns SORT INDICES, NOT ranks. A cross–sectional rank MUST be ‘x.argsort(–1).argsort(–1).float()/(N–1)‘ (double argsort). A single argsort divided by N is a SCRAMBLED permutation = a bug.
2. wrong reduction axis: for unary/binary there is NO time axis, so ‘.std(–1)‘, ‘.mean(–1)‘, ‘.sum(–1)‘ reduce over STOCKS (cross–section) –– if the docstring claims a TIME stat (e.g. ”20–day volatility”), that is a bug (mislabelled or wrong category). Cross–sectional operations over stocks are allowed, but they must be permutation–equivariant: reordering stock columns may only reorder the corresponding outputs. Adjacent–column differences or positional stock indexing are bugs because stock order has no economic meaning.
3. input–domain assumption: GP feeds ARBITRARY expressions (often raw prices, always $>$ 0). Code that assumes the operand is a rank in [0,1] (e.g. ‘(operand–0.5)$\star$k‘ then clamp, or ‘sigmoid(operand)‘), or a return/centered series (e.g. ‘(window $>$ 0).sum()‘ to count up–days –– prices are always $>$ 0 so this is constant), DEGENERATES on the real input. Make it self–normalize (z–score / rank / divide by own scale) so it works on any input.
4. dead/ineffective gate: a threshold on an UN–normalized quantity (e.g. ‘(ratio $>$ 0.9)‘ where ratio ˜ 1 almost always) makes the gate ˜constant. Gate on a normalized quantity (rank/z–score), or remove the gate.
5. docstring vs implementation mismatch: if the code does not compute what the rationale claims (e.g. rationale says ”range” but code takes ‘.mean‘ of values; says ”volatility” but takes cross–sectional std), fix the code to match the stated economic intent (preferred) or correct the rationale.
6. look–ahead: any use of data after the last window index is a hard bug. Contemporaneous cross–sectional ranks/statistics are causal, but must satisfy the stock–permutation rule above.

Operators (each already runs)
{OPS_BLOCK}

Task
Return a JSON array of EXACTLY {N} operators in the SAME ORDER, same keys: name, category, Ns, apply_body, rationale, source_factor_ids.
– If an operator is CORRECT, copy every field VERBATIM (including source_factor_ids).
– If it has ANY bug above, FIX the ‘apply_body‘ (keep the economic intent and the category’s dim semantics), and prepend to ‘rationale‘ a short ”[fixed: <what>]”. Keep source_factor_ids non–empty.
Do NOT change correct operators. OUTPUT ONLY THE JSON ARRAY –– no fences, no prose.

B.4.3 VALIDATION-DRIVEN REPAIR

You previously proposed compound operators for the alpha–mining task. The following {N_FAILED} operators FAILED validation at stage ‘{STAGE}‘. For each, the original code and the specific failure reason are shown below.

Your job: produce a FIX for each that resolves the specific error while preserving the operator’s original intent. Common causes:
– bad output shape (missing .squeeze(), wrong dim aggregation, expand_as bug)
– look–ahead leak (dim=0 used as time–axis, or .shift(–N), .bfill())
– runtime crash (division–by–zero, type mismatch, undefined name)
– constant/NaN output (always returns same value or all NaN)
– DEGENERATE: body is a trivial primitive composition (identity pass–through, pure ‘lhs$\star$rhs‘ / ‘lhs/rhs‘, or arithmetic–only + clamp with no real work). To fix, the operator must carry GENUINE novel structure –– a transcendental nonlinearity (sigmoid/tanh/arcsinh), a cross–sectional reduction (mean/std over stocks), ranking (argsort), or a predicate/count. Do NOT just bolt the SAME ‘/mean(|x|)‘ reduction onto every op –– that produces clones (see the distinctness rule below).

[CATEGORY SHAPE RULES –– verbatim from original task]

category	arg(s)	shape	return	dim semantics
–––––––––––––––	–––––––––––––––––––	–––––––––––––––––	–––––––––	–––
unary	‘operand‘	(L, S)	(L, S)	dim=–1 = STOCKS (cross–sectional). NO time.
binary	‘lhs‘, ‘rhs‘	(L, S), (L, S)	(L, S)	dim=–1 = STOCKS. NO time. No window.
rolling	‘window‘	(L, S, W)	(L, S)	dim=–1 = WINDOW (time); dim=–2 = STOCKS.
pair_rolling	‘lhs‘, ‘rhs‘	(L, S, W) x 2	(L, S)	dim=–1 = WINDOW; dim=–2 = STOCKS.

[ALREADY–ACCEPTED OPERATORS –– these are FINAL; do NOT duplicate them]
The following operators already passed validation and WILL be in the output set.
Your replacements MUST be structurally DISTINCT from every one of them (different formula, not just a renamed/clamped variant). Read them so you don’t collide.

{EXISTING_BLOCKS}

[SEED FACTORS –– extract your replacements FROM these]
{SEED_INTRO}

{SEED_BLOCKS}

[FAILED OPERATORS]

{FAILED_BLOCKS}

[TASK]

For each failed operator, produce ONE replacement operator. You have two routes, and you should PREFER route (a):

(a) RE–EXTRACT (preferred): the failed op was a bad attempt to capture some pattern in the SEED FACTORS above. Go back to the seeds, find a DIFFERENT informative compound pattern that is not yet covered by the already–accepted operators, and extract THAT as a fresh operator. This keeps every operator grounded in a real seed factor –– cite the seed(s) in ‘source_factor_ids‘. Give it a new descriptive ‘name‘.

(b) REPAIR: if the failed op’s intent is genuinely valuable and seed–grounded, fix the specific error while preserving intent. Keep ‘source_factor_ids‘ pointing at the seed(s) it came from.

Either way, do NOT invent a generic textbook operator with empty ‘source_factor_ids‘ just to pass the gate –– an operator with no seed grounding is itself grounds for rejection. Every replacement MUST cite at least one seed ‘factor_id‘ (short prefix, e.g. ”7dbe921c”).

Return a JSON array of {N_FAILED} operators in the SAME ORDER as the failed list above. Each object MUST have keys: name, category, Ns, apply_body, rationale, source_factor_ids.

DISTINCTNESS (hard requirement): the fix must differ from every already–accepted operator above in its CORE MECHANISM, not merely in its inputs. If the accepted list already has several ‘rank –> arcsinh‘ ops (or several ‘/mean‘ ratios, or several sigmoid gates), do NOT add another variant of that same theme –– reach for a DIFFERENT mechanism: a counter/predicate over the window, a quantile bucket, a cross–sectional dispersion, an asymmetric tail ratio, a regime blend, etc. Same theme with a different input column is NOT distinct enough.

OUTPUT ONLY THE JSON ARRAY –– no markdown fences, no prose.