CAN LLMS DESIGN STRATEGIES FOR INVESTOR GOALS? GOAL-ORIENTED QUANTITATIVE STRATEGY GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Developing quantitative strategies requires searching over factors, trading rules, and parameters through iterative backtesting and refinement to meet investment objectives. LLMs could assist this process using financial knowledge and backtesting feedback, but their ability to find strategies satisfying specified goals remains insufficiently evaluated. To address this gap, we introduce **AlphaGoal**, a benchmark for goal-directed strategy search with limited backtest evaluations. It provides 175 validated atomic strategies and 2,221 strategy-construction tasks, each specifying an investment goal in natural language. Grounded in up to 11.5 years of daily data across 5,941 stocks, these tasks cover strategy generation and parameter optimization. Each goal is empirically reachable, with success verified programmatically under a unified historical backtesting protocol. We also introduce **AlphaLoop**, a reference baseline combining factor information and backtesting feedback to construct and refine strategies. Across six LLMs, AlphaLoop improves overall pass rates over Vanilla, while hard strategy-generation tasks remain challenging. Code and data: `https://anonymous.4open.science/r/AlphaGoal`.

1 INTRODUCTION

Quantitative strategy development begins with investment objectives that specify desired returns, acceptable risks, or required risk-adjusted performance. Meeting these objectives requires selecting factors derived from historical market data and combining them into trading rules. Experts evaluate candidate strategies through historical backtesting and revise their choices using the resulting evidence Wang et al. (2026). Because performance depends jointly on factor selection, trading conditions, and parameter settings, finding a suitable strategy requires iterative search.

This process draws on financial knowledge and empirical analysis, two areas in which LLMs have shown promising capabilities Wu et al. (2023); Cheng et al. (2026); Yao et al. (2026); Lai et al. (2023). We therefore ask: *can an LLM construct a strategy satisfying a specified investment goal with a limited number of backtest evaluations?* We study this question over a designated historical interval, where performance can be checked against explicit conditions. For example, an investor may seek at least 10% annualized return while keeping maximum drawdown at or below 20% for a given stock. The model must use backtesting feedback to find a strategy satisfying both constraints.

Assessing this ability requires an evaluation centered on the requested goal. Recent benchmarks, including QuantEval Kang et al. (2026), AlphaBench Luo et al. (2026), and AlphaForgeBench Zhang et al. (2026), support executable evaluation of strategy coding, factor mining, and strategy design through criteria such as code executability, agreement with reference backtests, factor improvement, and aggregate performance. These criteria do not directly measure whether a model can find a strategy meeting a particular goal. Specifying a goal alone, however, does not make failure informative about search capability: a candidate may fail because its implementation is incorrect, and the search may fail because the goal itself is unattainable.

These alternative explanations motivate two requirements for evaluating strategy search: reliable candidate execution and evidence that the requested goal is feasible. We build **AlphaGoal** around these requirements. All models select, combine, and parameterize a shared library of 175 expert-

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.

verified atomic strategies, each pairing a factor with a signal condition and trading direction. A unified Engine executes the validated implementations chronologically under a fixed protocol, and programmatic checks determine whether the requested conditions hold. To establish feasibility, we derive targets from historical backtests and retain a successful strategy as hidden evidence of attainability. Although directly collected user requests better reflect investment needs, they provide no such guarantee within the allowed strategy space. Natural-language generation therefore serves only to express the structured goals without revealing the reference; any valid strategy satisfying the conditions is accepted. This construction prioritizes controlled capability evaluation over representing real-world demand.

Applying this construction to more than 155 million candidate backtests across 5,941 stocks, using up to approximately 11.5 years of daily data, yields 2,221 tasks. They cover *strategy generation*, in which models search rule structures and parameters, and *parameter optimization*, in which the structure is supplied. Each task imposes one or two metric constraints. Strategy-generation tasks span easy, medium, and hard levels, with more demanding targets drawn from more selective regions of the sampled performance distributions. Targets must also meet minimum performance requirements and, for return-oriented metrics, exceed Buy-and-Hold performance, ensuring that the benchmark includes economically meaningful goals as well as attainable ones.

To establish a baseline for these tasks, we introduce **AlphaLoop**, an agent combining historical factor ICs, atomic-strategy specifications, and backtesting feedback. ICs guide factor prioritization, strategy specifications support rule construction and parameter configuration, and diagnostic feedback guides subsequent revisions. The agent searches with a limited number of backtest evaluations, preserving the supplied rule structure in parameter optimization.

Our contributions are threefold:

- **Standardized evaluation.** We build an expert-verified space of 175 atomic strategies and a unified Engine that applies a standardized execution protocol, computes 12 common financial metrics, and returns detailed trades, portfolio-value series, and daily signal data for reproducible evaluation.
- **AlphaGoal.** We construct 2,221 tasks for strategy generation and parameter optimization, deriving empirically reachable investor goals from backtesting more than 155 million candidates across 5,941 stocks, with strategy-generation difficulty calibrated using empirical performance percentiles.
- **AlphaLoop.** We introduce a dedicated baseline for goal-oriented strategy search with a limited number of backtest evaluations. It improves overall pass rates over Vanilla across all six evaluated LLMs and provides a reproducible reference for future work.

2 Related Work

Benchmarks for Quantitative Strategy Generation. Existing benchmarks evaluate different stages of quantitative investing. InvestorBench Li et al. (2025) covers sequential decisions across stocks, cryptocurrencies, and ETFs, while StockBench Chen et al. (2025) assesses daily trading actions through profitability and risk metrics. These settings evaluate investment decisions rather than the construction of reusable trading rules. Moving toward strategy construction, QuantEval Kang et al. (2026) includes executable strategy coding alongside financial knowledge and mathematical reasoning, and AlphaBench Luo et al. (2026) evaluates factor generation, evaluation, and iterative search. Most closely related, AlphaForgeBench Zhang et al. (2026) generates factor and strategy code from investment queries and evaluates it through standardized backtesting. However, code correctness, factor quality, and aggregate performance do not directly establish whether a model can find a strategy satisfying specified quantitative constraints. Model-generated implementations also introduce coding failures, while investment requests alone do not establish target feasibility. AlphaGoal addresses these evaluation gaps through a shared validated library and empirically reachable goals. Programmatic constraint checking and percentile-based difficulty calibration then support controlled comparisons of goal-directed search with limited backtest evaluations.

Methods for Quantitative Research with LLMs. LLM-based trading agents use financial information, memory, and feedback to improve sequential decisions. InvestorBench Li et al. (2025) adopts layered memory, FinAgent Zhang et al. (2024) combines multimodal information with tools and

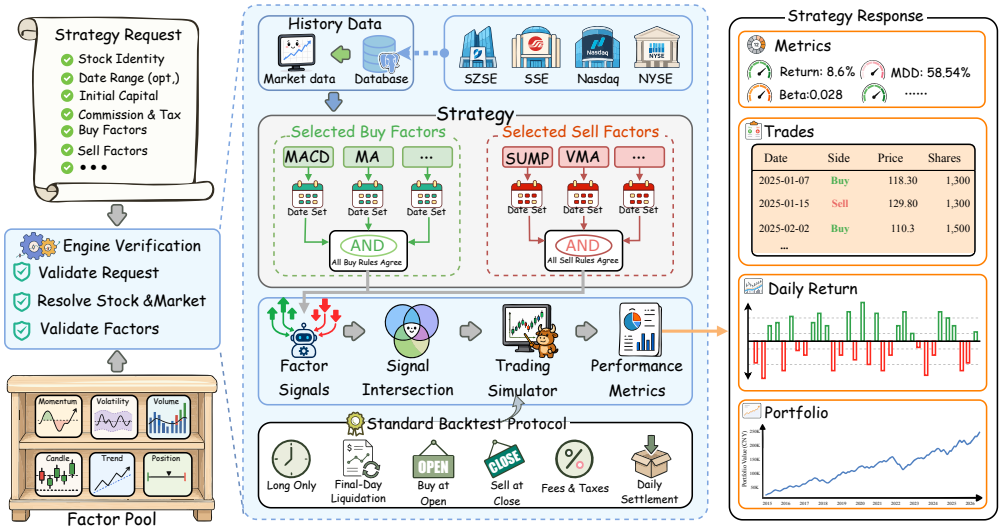


Figure 1: Overview of the Engine's strategy-backtesting.

reflection, and FinCon Yu et al. (2024) coordinates agents through verbal reinforcement and risk control. Their decision-making procedures do not directly produce complete reusable strategies. For factor discovery, AlphaBench Luo et al. (2026) evaluates reasoning and evolutionary search, AlphaAgent Tang et al. (2025) controls similarity and complexity while promoting semantic consistency, and Cognitive Alpha Mining Liu et al. (2026) combines code representations, LLM reasoning, evolutionary operators, and financial feedback. Factor improvement alone, however, does not ensure that the resulting trading strategy satisfies return and risk constraints: performance also depends on thresholds, signal timing, and interactions among buy and sell rules. AlphaLoop provides a reference baseline for this additional construction and revision problem. It uses IC priors and atomic-strategy specifications to select and configure rules, then analyzes backtesting results with code to identify why constraints remain unmet and guide subsequent revisions.

3 AlphaGoal Construction

AlphaGoal evaluates whether models can construct trading strategies or optimize their parameters within a shared rule library to satisfy explicit investor goals. Each task presents a natural-language request specifying a stock, backtesting start date, randomly sampled initial capital, and performance goal, such as achieving a target return while limiting drawdown. The evaluator supplies the end date from the fixed data snapshot and standardized protocol. Standardized backtesting then verifies whether all requested constraints are met.

To isolate strategy-search capability, each task must admit a feasible solution. Although real user requests better reflect investment needs, they do not establish feasibility within a fixed strategy space and execution protocol; failure could therefore reflect either limited search capability or an unattainable goal. We instead derive goals from backtests on real market data, retaining a successful strategy as feasibility evidence. Language generation only expresses the resulting structured goals. Empirical performance percentiles define easy, medium, and hard tiers for strategy-generation tasks, while minimum performance requirements and relevant Buy-and-Hold comparisons exclude economically trivial targets. This design prioritizes controlled capability evaluation over representing real-world demand (Appendix D.12).

Construction therefore proceeds in four stages: defining the shared strategy space (Section 3.1); standardizing backtesting (Section 3.2); deriving feasible goals and calibrating difficulty; and expressing these goals as natural-language requests. The final two stages form the task-construction pipeline (Section 3.3).

3.1 CONTROLLED STRATEGY SPACE

We first define a shared library of validated atomic strategies, each pairing a quantitative factor with a signal condition and trading direction. Models select, combine, and parameterize these strategies within a common search space without implementing new factors.

A *factor* maps historical market data to a quantitative feature:

$$F_{s,t}(f, \boldsymbol{\theta}) = f(\mathbf{D}_{s,\leq t}; \boldsymbol{\theta}) \,, \tag{1}$$

where s denotes a stock, t a trading day, $\mathbf{D}_{s,\leq t}$ the data observable through day t, and $\boldsymbol{\theta}$ the computation parameters, including lookback windows. Continuous factors take real values; pattern-based factors are binary. Values are lagged as required by the execution protocol so that signals use only information available before the corresponding trade.

An *atomic strategy* converts a factor value into a directional signal. A configured instance $(f, \boldsymbol{\theta}, g, \boldsymbol{\phi}, d)$ has a fixed direction $d \in \{\text{buy}, \text{sell}\}$ and signal

$$S_{s,t}(f, \boldsymbol{\theta}, g, \boldsymbol{\phi}) = g(F_{s,t}(f, \boldsymbol{\theta}); \boldsymbol{\phi}) \,, \tag{2}$$

where $g : \mathbb{R} \times \Phi \to \{0, 1\}$ applies trigger parameters $\boldsymbol{\phi} \in \Phi$, including thresholds and tolerances. A value of one indicates that the condition for direction d is met. Thus, $(\boldsymbol{\theta}, \boldsymbol{\phi})$ jointly control factor computation and signal activation. For example, changing an RSI buy rule's lookback window or oversold threshold yields another configuration of the same atomic strategy. Each strategy provides default parameter values.

The library contains 175 parameterized atomic strategies (87 buy-side and 88 sell-side). It covers trend, momentum, reversal, volatility, price-volume, and chart-pattern rules: 70 derived from Alpha158 Yang et al. (2020), 32 from BacktestBench Wang et al. (2026), and 73 from technical-analysis and candlestick factors. Three experts independently implemented every strategy and cross-validated their implementations. The library is fixed within each benchmark version for comparability, while the Engine's registry supports extensions. Appendix C provides the complete inventory, parameter specifications, and validation procedures.

3.2 UNIFIED BACKTESTING ENVIRONMENT

The atomic strategies defined above are combined and evaluated through a unified backtesting engine, ensuring comparable performance metrics and consistent feasibility verification Liu et al. (2022). A complete strategy contains at least one buy-side and one sell-side atomic strategy, each specified by its registered name and parameters. Rules on the same side are combined by conjunction: a buy or sell signal activates only when all corresponding rules fire on the same day.

The Engine takes the stock code, backtesting interval, initial capital, transaction costs, and structured buy/sell rules. It validates inputs, loads market data, computes atomic signals, and passes their intersections to a common account simulator. Execution is long-only, with alternating buys and sells; fixed rules are evaluated chronologically using only information available at each decision time. The Engine computes 12 metrics covering return, risk, risk-adjusted performance, trade quality, and market exposure. It also returns trade records, portfolio-value and daily-return series, portfolio states, and daily factor/signal traces for iterative diagnosis (Section 4).

The Engine also supports factor selection by computing a time-series information coefficient (IC), which measures the association between factor values and next-day returns and does not enter trade execution:

$$\text{IC}(f, s, \boldsymbol{\theta}) = \rho\Big(\{F_{s,t}(f, \boldsymbol{\theta})\}_{t \in \mathcal{T}_{s,f}}, \{r_{t+1}(s)\}_{t \in \mathcal{T}_{s,f}}\Big) \,, \tag{3}$$

where $r_{t+1}(s)$ is the next-day return, $\mathcal{T}_{s,f}$ contains dates with valid factor values and returns for the specified parameters, and ρ denotes Spearman rank correlation. Positive IC indicates that higher factor values are associated with higher subsequent returns; negative IC indicates the reverse. Appendix C provides the complete execution rules, fee settings, metric definitions, and output formats.

3.3 TASK CONSTRUCTION PIPELINE

We construct tasks using the shared library and unified Engine in four stages: empirical sampling, threshold derivation, structured record construction, and natural-language conversion.

Empirical sampling. We begin with 5,955 Chinese and U.S. security time series comprising 15,953,660 daily observations and retain 5,941 stocks after validation. For each stock, we sample strategy combinations and parameter configurations, backtest them, and record all 12 metrics. During dataset construction, each sampled strategy contains at least one buy and one sell rule, totaling at most five to limit signal sparsity from same-side conjunctions. This yields 155,350,871 candidates and 155,106,377 valid backtests (approximately 26,108 per stock); Buy-and-Hold is evaluated under identical settings. Appendix D.1 details market coverage and data fields.

For stock s with N valid sampled strategies, the empirical cumulative distribution of metric m is

$$\widehat{F}_{m,s,N}(y) = \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}[\mathcal{M}_m(\pi_n, s) \leq y]\,, \tag{4}$$

where π_n is the nth strategy, $\mathcal{M}_m(\pi_n, s)$ its metric value, and $\mathbb{I}[\cdot]$ the indicator function. For example, $\widehat{F}_{\text{Sharpe},s,N}(1.5) = 0.90$ means 10% of sampled strategies achieve a Sharpe ratio above 1.5. These distributions describe the sampled pool; Figure 8 in Appendix D.2 illustrates all 12 metrics.

Threshold derivation. Empirical percentiles define targets at level P_p, denoting a nominal marginal pass proportion of $p\%$. Smaller p indicates a more demanding target in the favorable tail, calibrating difficulty relative to the sampled pool. Single-metric tasks use P_1–P_{10}; dual-metric tasks use selected levels from P_2 to P_{35} for each metric, requiring joint satisfaction. Targets must meet metric-specific viability requirements, such as annualized return $\geq 3\%$ or Sharpe ratio ≥ 0.30, and exceed Buy-and-Hold performance for return-oriented metrics.

A target is retained only if a sampled strategy satisfies every constraint and passes checks for backtesting anomalies, minimum annualized return, and trading activity. Eligible strategies are ranked by completed trade count, then total buy/sell atomic-rule count, then annualized return, all in descending order, with remaining ties resolved by a fixed strategy-ID ordering. The selected hidden reference establishes feasibility but is not used for scoring.

Structured record construction. Each record stores the stock, interval, sampled initial capital used to establish reachability, metric names, comparison operators, thresholds, and hidden reference. Parameter optimization tasks also specify fixed buy/sell rules while withholding reference parameters. Task selection seeks approximately balanced representation across single-metric categories. These records define query content and evaluation conditions.

Natural-language conversion. Five LLMs rewrite records as investment requests, varying wording while preserving the stock, start date, initial capital, and metric constraints. Queries exclude reference solutions and backtesting results; parameter optimization queries include fixed rules but omit parameters. Each query is reviewed by the other four models for semantic accuracy and information leakage, then checked programmatically against its source record. Retention requires unanimous reviewer approval and successful programmatic validation.

At evaluation, the Engine backtests each submitted strategy over the designated interval, and code checks its metrics against the structured constraints. Any valid strategy within the task's allowed search space passes if all constraints hold, without matching the reference or relying on LLM judgment. Appendix D provides construction procedures, selection criteria, record formats, prompts, and validation details.

3.4 Benchmark Statistics

AlphaGoal contains 2,221 programmatically verifiable tasks: 1,360 *strategy generation* tasks requiring rule selection and full parameter configuration, and 861 *parameter optimization* tasks with supplied rule structures and hidden reference parameters. Across these task types, 1,530 tasks impose a single metric constraint and 691 impose two simultaneous constraints. Single-metric targets cover annualized return, alpha, Sharpe ratio, Calmar ratio, Sortino ratio, and recovery factor; dual-metric targets pair constraints drawn from return, risk, market exposure, and trade quality. Each task retains a hidden reference establishing feasibility.

Strategy-generation difficulty is calibrated separately for single- and dual-metric tasks. Hard, medium, and easy correspond to P_1–P_3, P_4–P_6, and P_7–P_{10} for single-metric tasks, and selected levels within P_2–P_{10}, P_{15}–P_{25}, and P_{30}–P_{35} for dual-metric tasks. The latter allow broader

marginal pass proportions because both constraints must hold. This yields 533 hard, 447 medium, and 380 easy tasks. Parameter optimization receives no difficulty labels, as percentiles from the full strategy pool do not directly characterize search difficulty under a fixed rule structure.

Appendix D.9 provides representative examples; Appendix D contains complete algorithms, statistics, and audits.

4 ALPHALOOP

We introduce AlphaLoop as a baseline agent for goal-directed strategy search under a finite backtesting budget. The agent receives the natural-language request, while structured metric constraints and the end date are stored separately for programmatic evaluation. The query specifies the stock, backtesting start date, and initial capital. The evaluator supplies the corresponding end date to the Engine under the common protocol. AlphaLoop searches rule structures and parameters in strategy generation, but adjusts only parameters in parameter optimization. Its workflow combines factor prioritization, strategy construction, and iterative revision.

IC-guided factor prioritization. For strategy generation, the Engine computes time-series IC for each registered factor at its catalog-default parameters over the task's full historical interval, ranking valid results by absolute IC. This provides retrospective, in-sample guidance because the same interval is used for evaluation; IC enters neither the submitted strategy nor trade execution. It serves as a search prior Tang et al. (2025), with strategy performance also depending on thresholds, signal density, and rule interactions. IC primarily guides entry selection; exit rules address profit taking, loss control, reversals, and holding periods. Parameter optimization skips this step because its rule structure is fixed.

Strategy construction. The agent retrieves atomic-strategy specifications through `get_factor_info`, which returns six fields: name, side, meaning, formula, rule, and parameters. These specifications support interpretation of shortlisted factors and configuration of entry and exit rules. Since same-side rules are conjunctive, construction balances selectivity against signal sparsity. The Engine validates rule names, directions, and parameters before backtesting. For parameter optimization, the agent uses the same specifications to configure the supplied rules.

Backtest-driven revision. Each valid candidate produces aggregate metrics, constraint-level gaps, and diagnostic data. AlphaLoop examines signal frequency to identify overly restrictive or permissive rules, holding periods and trade outcomes to assess entries and exits, and portfolio paths to locate drawdown episodes. It then adjusts windows, thresholds, or filters and, in strategy generation, may replace entry or exit rules. Parameter optimization uses the same feedback but permits only parameter changes.

The agent retains tested-candidate history to inform revisions Shinn et al. (2023). Search ends when all metric constraints are met, the agent chooses to stop, or a search limit is reached. The latest candidate whose backtest completed without execution errors is re-executed for final programmatic evaluation. Appendix E provides full prompts, tool interfaces, budgets, and failure limits.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Evaluation Design. We ask how difficult AlphaGoal is with limited backtest evaluations and whether AlphaLoop is effective as a reference baseline. Because AlphaGoal derives reachable goals from an empirical strategy pool, Random Search provides a calibration baseline for finding satisfying strategies through uninformed sampling. Since it does not involve an LLM, we further introduce Vanilla, a minimally guided LLM baseline using the same library and backtest limit but lacking AlphaLoop's IC prior, condition-level feedback, and trajectory diagnostics. Comparing AlphaLoop with Vanilla measures the combined benefit of these mechanisms beyond the underlying LLM's basic search capability.

Baseline Implementations. For strategy generation, Random Search uses one buy and one sell rule, corresponding to $K_{\max} = 2$, the best-performing setting in the repeated-run comparison in

Table 1: Overall performance by task category. Main values are pass rates (%); parenthesized values are the average numbers of valid candidate backtests per task.

Model	Method	Overall	Strategy Generation				Parameter Optimization
			Easy	Medium	Hard	All	
Random Search		30.30 (4.24)	40.26 (4.10)	20.81 (4.62)	7.50 (4.91)	21.03 (4.59)	44.95 (3.70)
Qwen3.8-27B	Vanilla	36.65 (3.64)	41.32 (3.36)	25.95 (3.84)	10.51 (4.06)	24.19 (3.79)	56.33 (3.41)
	AlphaLoop	41.60 (3.91)	55.53 (3.55)	31.32 (4.28)	19.51 (4.49)	33.46 (4.16)	54.47 (3.52)
Gemma4-31B	Vanilla	39.44 (3.76)	49.74 (3.51)	**33.11** (3.98)	14.63 (4.39)	30.51 (4.01)	53.54 (3.37)
	AlphaLoop	41.87 (3.92)	53.42 (3.54)	30.65 (4.29)	18.95 (4.57)	32.43 (4.19)	56.79 (3.51)
Hy3	Vanilla	34.85 (3.92)	46.84 (3.60)	29.98 (4.04)	11.63 (4.54)	27.50 (4.11)	46.46 (3.61)
	AlphaLoop	38.77 (3.81)	51.58 (3.39)	28.86 (4.11)	16.89 (4.39)	30.51 (4.02)	51.80 (3.49)
DS-V4-Flash	Vanilla	31.16 (3.83)	48.42 (3.68)	29.08 (4.29)	12.57 (4.66)	28.01 (4.26)	36.12 (3.15)
	AlphaLoop	36.92 (3.36)	44.21 (3.07)	25.06 (3.54)	16.70 (3.69)	27.13 (3.47)	52.38 (3.21)
GLM-5.3-Flash	Vanilla	21.52 (4.13)	36.05 (3.84)	17.00 (4.30)	6.19 (4.62)	18.09 (4.30)	26.95 (3.86)
	AlphaLoop	40.48 (3.96)	46.84 (3.84)	27.52 (4.37)	19.70 (4.53)	29.85 (4.28)	57.26 (3.45)
GPT-5.6-Sol	Vanilla	37.82 (4.02)	44.47 (3.83)	30.43 (4.35)	14.07 (4.69)	27.94 (4.34)	53.43 (3.51)
	AlphaLoop	**47.37** (3.80)	**57.37** (3.52)	**33.11** (4.22)	**21.76** (4.56)	**35.44** (4.16)	**66.20** (3.23)

Appendix G.1. It samples rule identities and parameters from the same library and Engine-defined domains used in benchmark construction. For parameter optimization, it preserves the supplied structure, tests catalog defaults first, and then samples parameters. Vanilla can query atomic-strategy documentation and iteratively submit candidates, receiving only candidate history and the 12 aggregate metrics as backtesting feedback. Appendix F details implementations.

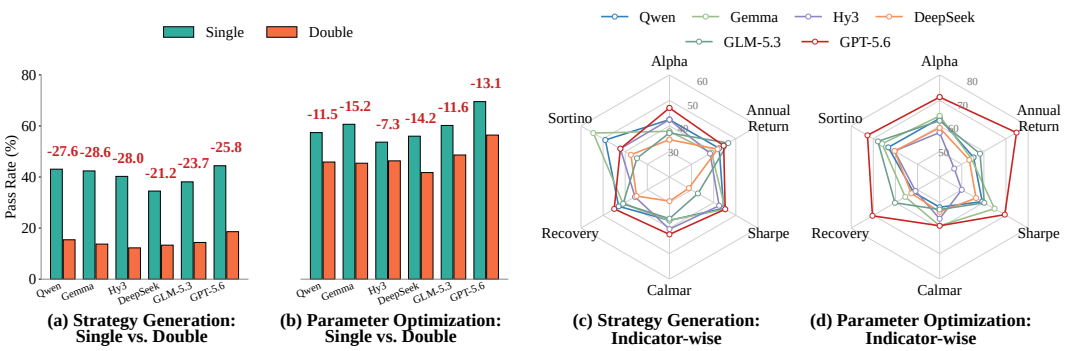
Models and Evaluation Protocol. We evaluate AlphaLoop and Vanilla with six LLMs: Qwen3.8-27B, Gemma-4-31B-it (Gemma4-31B), Hy3, DeepSeek-V4-Flash-0731 (DS-V4-Flash), GLM-5.3-Flash, and GPT-5.6-Sol. All methods share tasks, the strategy library, Engine, task-specified initial capital, transaction settings, and structured success conditions. Unless otherwise stated, each method receives at most five valid backtests per task, with the evaluator automatically stopping search once all target conditions hold. Appendices F and E specify tool-call limits, failure handling, and other stopping rules.

Main-table configurations are run once per task; selected strategy-generation configurations are repeated three times in Appendix G.1. We report task pass rate as the primary metric and mean valid backtests per task, excluding final re-evaluation, as backtesting cost. Appendix G provides deployment details, full configurations, and case studies.

5.2 Main Results

The results in Table 1 show that AlphaLoop improves overall pass rate over Vanilla for all six LLMs, by 2.43–18.96 percentage points. GPT-5.6-Sol with AlphaLoop performs best under default reasoning, reaching 47.37% overall, with 35.44% on strategy generation and 66.20% on parameter optimization. It averages 3.80 valid backtests, compared with 4.02 for Vanilla under the same five-backtest limit. Although AlphaLoop improves performance, its strongest configuration still fails on more than half of the tasks, demonstrating the difficulty of finding satisfying strategies even with validated implementations and verified feasible goals.

The overall gains extend to ten of twelve model–task combinations. The exceptions are Qwen3.8-27B on parameter optimization and DS-V4-Flash on strategy generation, where pass rates decline by 1.86 and 0.88 percentage points, respectively. All AlphaLoop configurations nevertheless exceed Random Search on both task types. Vanilla falls below Random Search on parameter optimization for DS-V4-Flash and GLM-5.3-Flash, and on strategy generation for GLM-5.3-Flash. For these configurations, access to strategy descriptions and aggregate metrics does not translate into better outcomes than uninformed sampling.



(a) Strategy Generation: Single vs. Double **(b) Parameter Optimization: Single vs. Double** **(c) Strategy Generation: Indicator-wise** **(d) Parameter Optimization: Indicator-wise**

Figure 2: AlphaLoop pass rates by constraint cardinality and across six single-metric task categories for strategy generation and parameter optimization. Red annotations in the cardinality panels report the dual-minus-single pass-rate difference in percentage points.

GLM-5.3-Flash illustrates a limitation of parameter revision. Its Vanilla parameter-optimization pass rate is 26.95%, versus 44.95% for Random Search, and rises to 57.26% with AlphaLoop. Among its Vanilla parameter-optimization tasks with multiple valid candidates, 78.3% produce identical backtesting results across iterations. This suggests that parameter changes often fail to alter realized trading behavior, so repeated evaluations consume the limited backtest allowance without providing new feedback to guide the search. Effective parameter search therefore requires identifying changes that affect strategy execution, rather than merely generating different parameter values.

Performance also declines consistently across the predefined difficulty tiers. GPT-5.6-Sol with AlphaLoop achieves 57.37%, 33.11%, and 21.76% on easy, medium, and hard strategy-generation tasks, respectively; Random Search reaches 40.26%, 20.81%, and 7.50%. These results support the empirical difficulty ordering. AlphaLoop outperforms Vanilla on hard tasks for every model, showing that its guidance helps at more demanding target levels. However, the low absolute pass rates indicate that current methods still struggle to locate satisfying strategies within the available backtest evaluations.

As shown in Figure 2(a)–(b), dual-metric pass rates are lower for every model, with reductions of 21.2–28.6 percentage points in strategy generation and 7.3–15.2 in parameter optimization. The larger gap in strategy generation suggests that satisfying simultaneous constraints is particularly difficult when models must also select the rule structure. The metric-level results in panels (c)–(d) show complementary model strengths: GPT-5.6-Sol leads or ties on four strategy-generation metrics, while GLM-5.3-Flash and Gemma4-31B lead on annualized return and Sortino ratio, respectively. With the structure supplied, pass rates increase across corresponding model–metric combinations, and GPT-5.6-Sol leads on five of six metrics. Together, these results suggest that selecting suitable rules and satisfying joint constraints remain important challenges beyond tuning the parameters of a supplied strategy.

5.3 Ablation and Sensitivity Analyses

Effect of Factor IC. To assess factor prioritization, we remove IC while retaining AlphaLoop's feedback and data-analysis loop. Restoring IC raises strategy-generation pass rates by 10.6 points for Qwen3.8-27B, 1.3 for Gemma4-31B, 7.9 for Hy3, and 4.6 for DS-V4-Flash (Figure 3(a)). Thus, IC-guided selection remains important despite diagnostic revision. A Vanilla+IC control further shows that IC accounts for most of Qwen3.8-27B's gain, while detailed feedback provides smaller, model-dependent benefits (Appendix G.3).

Effect of Reasoning Effort. We next examine whether increasing reasoning effort improves AlphaLoop's strategy search. In Figure 3(b), only AlphaLoop's reasoning setting varies; pass rates are expressed as percentage-point differences from each model's fixed Vanilla-Low reference. Qwen3.8-27B performs best at low or medium effort and declines at high effort. DS-V4-Flash exceeds its reference in aggregate strategy-generation performance only at maximum effort. GLM-5.3-Flash

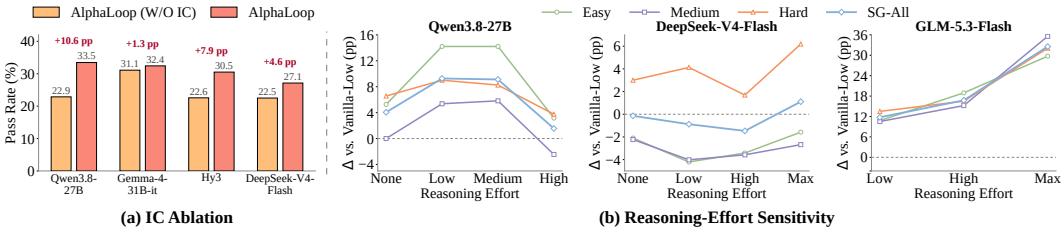


Figure 3: (a) IC ablation on strategy generation. (b) AlphaLoop reasoning-effort sensitivity, reported as strategy generation pass-rate differences from the fixed Vanilla-Low baseline.

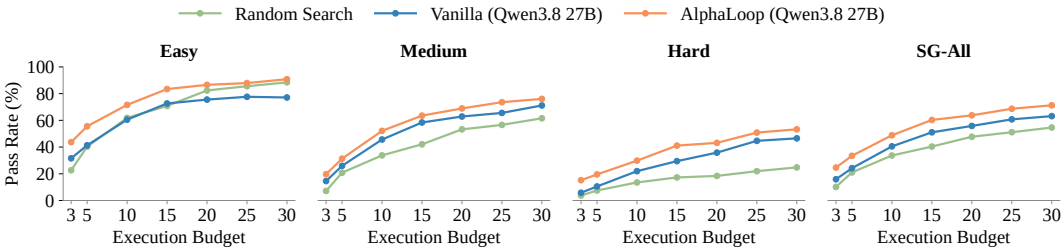


Figure 4: Strategy generation pass rate as the maximum valid-backtest budget increases.

benefits most: its gain over Vanilla-Low increases from 11.76 points at low effort to 32.57 at maximum. Higher reasoning effort therefore does not consistently improve search, and the best tested setting differs across models Cheng et al. (2025).

Effect of the Backtest Limit. To examine whether additional evaluations overcome the limitations observed in the main experiment, we increase the maximum number of valid backtests from 3 to 30. Random Search uses one buy and one sell rule, while Vanilla and AlphaLoop both use Qwen3.8-27B. As shown in Figure 4, aggregate strategy-generation pass rates increase for all three methods, with AlphaLoop leading at every evaluated limit. At 3, 15, and 30 backtests, AlphaLoop achieves 24.63%, 60.29%, and 71.25%, respectively, compared with 10.15%, 40.37%, and 54.63% for Random Search and 15.88%, 51.03%, and 63.16% for Vanilla.

The difficulty breakdown shows where additional attempts are most effective. On easy tasks, Random Search approaches AlphaLoop as the limit increases, although AlphaLoop remains ahead at both 25 and 30 backtests. On medium and hard tasks, AlphaLoop retains a more pronounced advantage. Additional evaluations therefore help uninformed search recover easier solutions, but do not eliminate the benefit of guidance for more demanding targets. AlphaLoop's lead over Vanilla also shows that its advantage persists when both methods have more opportunities to revise candidates.

6 Conclusion

We introduced AlphaGoal to evaluate whether LLMs can construct strategies satisfying explicit investment goals with a limited number of backtest evaluations. Its validated strategy library, empirically reachable goals, and unified backtesting protocol support controlled, programmatically verifiable evaluation over designated historical intervals. We also introduced AlphaLoop as a reference baseline that combines factor information, financial knowledge, and backtesting feedback to guide strategy construction and revision. Across six LLMs, AlphaLoop improves overall pass rates over Vanilla, though gains vary by model and task type. Even its strongest configuration solves fewer than half of all tasks, with hard strategy-generation tasks remaining particularly challenging. These findings indicate substantial room for improving goal-directed strategy search. The benchmark's scope and limitations are discussed in Appendix A.

REPRODUCIBILITY STATEMENT

We provide an anonymous repository containing the 2,221 AlphaGoal tasks, implementations of all 175 atomic strategies, the unified backtesting Engine, AlphaLoop, complete agent prompts, database import utilities, and result-summary code. The repository also provides Docker and Python environment specifications, configuration templates, data preparation instructions, and commands for running the evaluation pipeline. The main paper describes the benchmark construction and evaluation design. Appendices C, D, E, F, and G document the execution protocol, metric definitions, task-construction procedure, tool interfaces, search limits, baseline settings, model configurations, random seeds, and statistical analyses. Reproducing results from externally hosted language models requires access to the corresponding model versions and inference settings.

REFERENCES

Yanxu Chen, Zijun Yao, Yantao Liu, Jin Ye, Jianing Yu, Lei Hou, and Juanzi Li. Stockbench: Can LLM agents trade stocks profitably in real-world markets? *CoRR*, abs/2510.02209, 2025. doi: 10.48550/ARXIV.2510.02209. URL https://doi.org/10.48550/arXiv.2510.02209.

Xiang Cheng, Chengyan Pan, Minjun Zhao, Deyang Li, Fangchao Liu, Xinyu Zhang, Xiao Zhang, and Yong Liu. Revisiting chain-of-thought prompting: Zero-shot can be stronger than few-shot. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 13533–13554. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-EMNLP.729. URL https://doi.org/10.18653/v1/2025.findings-emnlp.729.

Xiang Cheng, Yulan Hu, Xiangwen Zhang, Lu Xu, Lide Tan, Zheng Pan, Xin Li, and Yong Liu. Beyond itinerary planning—a real-world benchmark for multi-turn and tool-using travel tasks. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29200–29251, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1347. URL https://aclanthology.org/2026.acl-long.1347/.

Zhaolu Kang, Junhao Gong, Wenqing Hu, Shuo Yin, Kehan Jiang, Zhicheng Fang, Yingjie He, Chunlei Meng, Rong Fu, Dongyang Chen, Leqi Zheng, Eric Hanchen Jiang, Yunfei Feng, Yitong Leng, Junfan Zhu, Xiaoyou Chen, Xi Yang, and Richeng Xuan. Quanteval: A benchmark for financial quantitative tasks in large language models. *CoRR*, abs/2601.08689, 2026. doi: 10.48550/ARXIV.2601.08689. URL https://doi.org/10.48550/arXiv.2601.08689.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-Tau Yih, Daniel Fried, Sida I. Wang, and Tao Yu. DS-1000: A natural and reliable benchmark for data science code generation. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 18319–18345. PMLR, 2023. URL https://proceedings.mlr.press/v202/lai23b.html.

Haohang Li, Yupeng Cao, Yangyang Yu, Shashidhar Reddy Javaji, Zhiyang Deng, Yueru He, Yuechen Jiang, Zining Zhu, K. P. Subbalakshmi, Jimin Huang, Lingfei Qian, Xueqing Peng, Jordan W. Suchow, and Qianqian Xie. INVESTORBENCH: A benchmark for financial decision-making tasks with llm-based agent. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 2509–2525. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.ACL-LONG.126. URL https://doi.org/10.18653/v1/2025.acl-long.126.

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 Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th*

Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026, pp. 11715–11749. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.ACL-LONG.538. URL https://doi.org/10.18653/v1/2026.acl-long.538.

Xiao-Yang Liu, Ziyi Xia, Jingyang Rui, Jiechao Gao, Hongyang Yang, Ming Zhu, Christina Dan Wang, Zhaoran Wang, and Jian Guo. Finrl-meta: Market environments and benchmarks for data-driven financial reinforcement learning. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/0bf54b80686d2c4dc0808c2e98d430f7-Abstract-Datasets_and_Benchmarks.html.

Haochen Luo, Ho Tin Ko, Jiandong Chen, David Sun, Yuan Zhang, and Chen Liu. Alphabench: Benchmarking large language models in formulaic alpha factor mining. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=d97Q8r7ZKZ.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.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 Luiza Antonie, Jian Pei, Xiaohui Yu, Flavio Chierichetti, Hady W. Lauw, Yizhou Sun, and Srinivasan Parthasarathy (eds.), *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining, V.2, KDD 2025, Toronto ON, Canada, August 3-7, 2025*, pp. 2813–2822. ACM, 2025. doi: 10.1145/3711896.3736838. URL https://doi.org/10.1145/3711896.3736838.

Zhensheng Wang, Wenmian Yang, Qingtai Wu, Lequan Ma, Yiquan Zhang, and Weijia Jia. Backtestbench: Benchmarking large language models for automated quantitative strategy backtesting. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '26, pp. 9973–9984, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722592. doi: 10.1145/3770855.3817460. URL https://doi.org/10.1145/3770855.3817460.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David S. Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance. *CoRR*, abs/2303.17564, 2023. doi: 10.48550/ARXIV.2303.17564. URL https://doi.org/10.48550/arXiv.2303.17564.

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

Rui Yao, Qi Chai, Jinhai Yao, Siyuan Li, Junhao Chen, Qi Zhang, and Hao Wang. Interpreting fedspeak with confidence: a llm-based uncertainty-aware framework guided by monetary policy transmission paths. In *Proceedings of the Fortieth AAAI Conference on Artificial Intelligence and Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence and Sixteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'26/IAAI'26/EAAI'26. AAAI Press, 2026. ISBN 978-1-57735-906-7. doi: 10.1609/aaai.v40i40.40739. URL https://doi.org/10.1609/aaai.v40i40.40739.

Yangyang Yu, Zhiyuan Yao, Haohang Li, Zhiyang Deng, Yuechen Jiang, Yupeng Cao, Zhi Chen, Jordan W. Suchow, Zhenyu Cui, Rong Liu, Zhaozhuo Xu, Denghui Zhang, Koduvayur Subbalakshmi, Guojun Xiong, Yueru He, Jimin Huang, Dong Li, and Qianqian Xie. Fincon: A synthesized LLM multi-agent system with conceptual verbal reinforcement for enhanced financial decision making. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/f7ae4fe91d96f50abc2211f09b6a7e49-Abstract-Conference.html.

Wentao Zhang, Lingxuan Zhao, Haochong Xia, Shuo Sun, Jiaze Sun, Molei Qin, Xinyi Li, Yuqing Zhao, Yilei Zhao, Xinyu Cai, Longtao Zheng, Xinrun Wang, and Bo An. A multimodal foundation agent for financial trading: Tool-augmented, diversified, and generalist. In Ricardo Baeza-Yates and Francesco Bonchi (eds.), *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, pp. 4314–4325. ACM, 2024. doi: 10.1145/3637528.3671801. URL https://doi.org/10.1145/3637528.3671801.

Wentao Zhang, Mingxuan Zhao, Jincheng Gao, Jieshun You, Huaiyu Jia, Yilei Zhao, Bo An, and Shuo Sun. Alphaforgebench: Benchmarking end-to-end trading strategy design with large language models. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '26, pp. 10268–10277, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722592. doi: 10.1145/3770855.3817500. URL https://doi.org/10.1145/3770855.3817500.

USAGE OF LLMS

LLMs were used in two roles in this work. First, as part of the research methodology, they were evaluated as strategy-search agents and used to convert structured benchmark records into natural-language investment requests and to review those requests. These uses are described in the corresponding methodology and experimental sections. All strategy performance and pass decisions were produced programmatically by the unified backtesting Engine rather than by LLM judgment.

Second, LLMs assisted with manuscript preparation, including language refinement, terminology and formatting consistency, and the development and layout adjustment of visualization code. The authors reviewed and verified all LLM-assisted text, code, figures, and tables. The authors remain responsible for the research design, experimental analysis, conclusions, and final manuscript.

A LIMITATIONS

AlphaGoal uses a single-stock, long-only protocol with full-position trades, fixed execution timing, market-specific lots and fees, and terminal liquidation. Slippage, market impact, and volume constraints are omitted, potentially affecting performance and target feasibility. The fixed library and same-side conjunction exclude novel factor implementations, disjunctions, conditional branches, dynamic position sizing, and time-varying rules. Search and evaluation share a historical interval, over which AlphaLoop also computes ICs. Although trades use only information available at execution, strategy selection uses full-interval results. Performance therefore measures retrospective goal satisfaction, not out-of-sample generalization or future profitability. Goals and difficulty levels reflect the sampled strategy pool and viability filters rather than real investor demand. Repeated runs cover only two LLMs and strategy generation. Future work can broaden execution assumptions, strategy spaces, and goal coverage, and evaluate unseen periods with broader repeated runs.

A.1 RATIONALE FOR RETROSPECTIVE EVALUATION

AlphaGoal does not reserve a fixed one-year segment as a separate forward test period or require a submitted strategy to satisfy the target conditions in both historical and held-out data. A fixed temporal split introduces substantial variation that is unrelated to the search capability that the benchmark

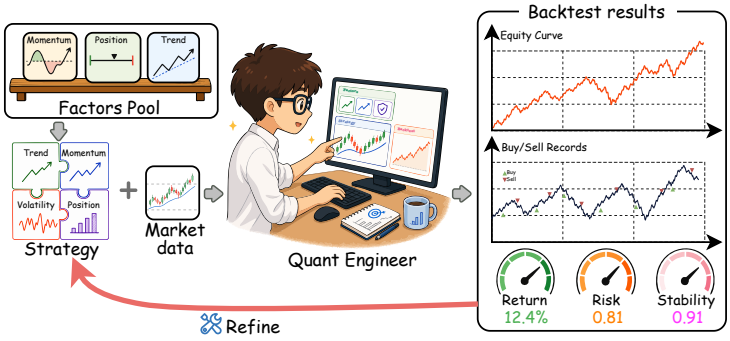


Figure 5: The iterative workflow of quantitative strategy development. Factors and market data form a candidate strategy, whose backtesting results guide subsequent refinement.

is intended to measure. Market-wide conditions can change from a bull regime to a bear or transition regime across the split, and a single stock can experience an idiosyncratic surge, decline, or other price-path event during the held-out year. Consequently, the same strategy can satisfy a historical return or risk target but fail in the forward interval, or fail historically and succeed forward, even when its implementation and search procedure are unchanged.

This issue is particularly pronounced for the single-stock, path-dependent setting of AlphaGoal. Annualized return, maximum drawdown, drawdown duration, Calmar ratio, and related metrics depend on the realized sequence of prices and trades rather than only on a time-invariant property of a factor. A fixed one-year split can therefore make the joint historical–forward criterion measure the particular market regime and stock-specific path realized after the split in addition to strategy-search ability. It also makes target reachability substantially harder to characterize: a strategy that is empirically reachable in the research interval need not remain reachable under a different regime, while a target that is reachable forward may be absent from the historical search space.

We consequently define AlphaGoal as a benchmark for retrospective, in-sample goal-directed strategy search under a fixed historical protocol. All methods receive the same designated interval, strategy library, execution rules, transaction settings, and programmatic success conditions. Targets are derived from the sampled strategy pool and each task retains a hidden reference strategy that establishes reachability within this protocol. This construction isolates whether an agent can use financial information and a limited number of backtest evaluations to find a feasible configuration, rather than mixing that capability with the realized regime transition or idiosyncratic return of a particular held-out year.

This choice does not imply that historical constraint satisfaction establishes predictive validity, robustness, or future investment performance. The execution layer prevents a candidate signal on day t from using data observed after that decision, but the selection layer deliberately reuses the designated historical interval for factor prioritization, candidate backtesting, and goal verification. AlphaGoal therefore measures retrospective goal satisfaction and search efficiency, not out-of-sample generalization. A future benchmark extension could evaluate the strategies selected by AlphaGoal under multiple rolling or regime-aware forward splits, rather than relying on a single one-year period; such an extension would measure robustness as a separate capability while preserving the controlled search evaluation studied here.

B Background: Iterative Quantitative Strategy Development

Quantitative strategy development commonly follows an iterative workflow. A researcher selects factors, combines them into entry and exit rules, and evaluates the resulting strategy on historical market data. Backtesting provides evidence about profitability, risk, trading behavior, and stability, which guides subsequent revisions to the rules or parameters until the investment objective is met or the research budget is exhausted.

Figure 5 illustrates this process. AlphaGoal formalizes its goal-directed search component under a controlled strategy language and a limited backtesting budget, while AlphaLoop operationalizes the refinement loop using factor information and backtesting evidence.

C Atomic-Strategy and Engine Details

C.1 Factor Inventory and Standardization

The factor repository draws on three sources: Alpha158, BacktestBench, and a supplementary collection of widely used technical indicators and price patterns. The source attribution is not used during evaluation; all factors are exposed through the same standardized interface.

Alpha158 contains 158 expanded feature columns, many of which apply the same calculation to different lookback windows. We consolidate these window-specific columns into parameterized factor families. For example, MA5, MA10, MA20, MA30, and MA60 are represented by a single factor, $\mathrm{MA}[N]$. The same treatment is applied to BETA, RSQR, RESI, QTLU, QTLD, ROC, STD, MAX, MIN, RSV, IMAX, IMIN, IMXD, CORR, CORD, CNTP, CNTN, CNTD, SUMP, SUMN, SUMD, VMA, VSTD, WVMA, VSUMP, VSUMN, and VSUMD. After consolidating these variants and retaining factors with a well-defined trading interpretation, we obtain 41 factor families from Alpha158. BacktestBench contributes 19 factor families, and the supplementary collection contributes 62.

Table 2 lists every factor used in the benchmark. The *Type* column indicates whether the factor is available in a buy-side atomic strategy, a sell-side atomic strategy, or both. *Both* denotes two separately registered signal functions with fixed directions, rather than a single function that chooses its direction at runtime.

In total, the repository contains 122 distinct factors without distinguishing trading direction. Among them, 87 are available for constructing buy signals and 88 for constructing sell signals, with factors of type *Both* counted in both directional totals. These direction-specific factor and signal implementations form 175 atomic strategies.

Because published descriptions may leave implementation choices such as rolling-window alignment, missing-value handling, threshold inclusivity, and signal timing unspecified, three quantitative experts implemented each factor according to its conventional interpretation in quantitative practice. They then reconciled these choices into a shared operational definition, preventing incidental implementation differences from changing the meaning or backtesting behavior of a factor.

C.2 Atomic-Strategy Implementation and Verification

Three experts with quantitative research backgrounds independently implement each candidate atomic strategy from a shared factor definition and signal specification. Every implementation separates factor calculation from signal evaluation. It first computes the complete daily factor sequence and then applies the threshold, crossover, or pattern rule that produces buy or sell dates. Continuous indicators retain numerical values, while pattern-based factors may retain Boolean sequences.

The three implementations are executed on identical test cases. We compare the factor values before comparing the signal dates, which prevents two implementations with different intermediate calculations from being accepted merely because they happen to trigger on the same small test sample. When disagreement occurs, the experts identify whether it arises from the mathematical definition, window alignment, missing-value handling, threshold inclusivity, or signal timing. The implementation is revised and retested until all outputs agree. We also check that every value used for a signal on day t is derived only from data observable by that time. Only implementations that pass both consistency and look-ahead checks enter the library.

Figures 6 and 7 show representative implementations of buy-side and sell-side atomic strategies. Each function receives standardized daily market data and a typed parameter set, computes the required factor sequence, applies a predefined signal condition, and returns the trading dates on which that condition is satisfied. The trading direction is fixed by the registered atomic strategy rather than inferred from the factor value at runtime. These examples are illustrative; all other atomic strategies follow the same separation between factor calculation and signal generation.

Table 2: Complete factor inventory grouped by trading direction. Both indicates that separately registered buy-side and sell-side atomic strategies are available.

Type	Count	Factors
Buy only	34	ArbrSentiment, AscendingTriangle, BollingerLowerBand, BullFlag, BullPowerN, BullishAdvancedSoldier, BullishCounterattack, DoubleBottom, DragonEmerging, ExpmaGoldenCross, FallingWedgeReversal, FibonacciRetracement, FlatBaseBreakout, IMAX, InsideBar, InverseHeadAndShouldersBottom, KLOW, KLOW2, LOW0, MIN, MacdGoldenCross, MovingAverageBreakout, MovingAverageSupport, MovingAverageTrend, OpenAboveMovingAverage, PiercingLine, RectangleConsolidation, RisingThreeMethods, RoundingBottom, ThreeWhiteSoldiers, TowerBottom, TweezerBottom, TwoBullsAndABear, VSUMP
Sell only	35	AmountStdN, BearishAdvancedSoldier, BearishSandwich, BollingerUpperBand, BroadeningFormationBreakdown, BrokenNeckKnife, DarkCloudCover, DescendingFlagBreakdown, DescendingTriangleBreakdown, DoubleTopReversal, DownpourPattern, EveningStarLoose, ExpmaDeathCross, FibonacciResistance, HIGH0, HeadAndShouldersTop, HighVolumeBearishShootingStar, IMIN, KUP, KUP2, LinearProjectionResistance, MAX, MacdDeathCross, MultipleMaBreakdown, RectangleBreakdown, ReturnKurtosisN, ReturnVarianceN, RisingWedgeBreakdown, RoundingTopReversal, SpikeTopReversal, SymmetricalTriangleBreakdown, ThreeBlackCrows, TowerTop, TweezerTop, VSUMN
Both	53	ATRN, AmountMAN, AnnualSharpeRatioN, BBI, BETA, CCIN, CNTD, CNTN, CNTP, CORD, CORR, DescendingTrendline, EMAN, EaseOfMovement, IMXD, KLEN, KMID, KMID2, KSFT, KSFT2, KdjStochasticOscillator, LinearProjectionSupport, MA, MikeBaseIndicator, MovingAverageBreakdown, NetMoneyFlowN, OPEN0, ParabolicSar, PbxWaterfall, QTLD, QTLU, RESI, ROC, RSI, RSQR, RSV, ReturnSkewnessN, STD, SUMD, SUMN, SUMP, SingleDayVPT, UltimateOscillator, UniversalLine, VMA, VSTD, VSUMD, VWAP0, VolumeEMAN, VolumeROC, VolumeRatio, VolumeStdN, WVMA

The RSI example in Figure 6 represents a continuous factor followed by a threshold rule. It computes the RSI sequence using a configurable lookback window, stores the factor values in the daily data, and emits a buy signal when the previous trading day's RSI is below the oversold threshold. The one-day shift ensures that the signal assigned to day t depends only on information available before that day.

The MACD example in Figure 7 represents a crossover-based pattern. It computes the MACD and signal lines, compares their relative positions on the two preceding trading days, and emits a sell signal when the MACD line crosses below the signal line. The resulting Boolean sequence is retained in the enriched daily data. This lagged construction prevents the strategy from using same-day closing information to generate a signal retrospectively.

C.3 Function Registration and Validation

Every approved atomic strategy follows a common interface. Its inputs are historical daily data and a typed parameter dictionary, and its output is a set of buy or sell signal dates. The function also appends its factor-value sequence to the enriched daily data used by the Engine. At runtime, a registry maps each strategy name to its approved implementation and parameter signature. Before backtesting, the Engine rejects unknown names, missing required parameters, unsupported parameters, invalid types, and values outside declared domains.

The registry is extensible: a new factor can be executed by the same Engine once its factor computation and signal-generation rule are implemented through this interface, its parameter types and

```python
def RSIBuyStrategy(df: pd.DataFrame, N: int = 14, threshold: float = 30.0) -> list:
    """
    Strategy Logic: Buy when T-1 day's RSI < 30 (oversold condition).
    Calculation: Uses Wilder's Smoothed Moving Average (approx via EWM) over N days.
    """
    df.sort_values('trade_date', inplace=True)
    df.reset_index(drop=True, inplace=True)
    delta = df['closing_price'].diff()
    gain = delta.clip(lower=0)
    loss = (-delta).clip(lower=0)
    avg_gain = gain.ewm(alpha=1/N, adjust=False, min_periods=N).mean()
    avg_loss = loss.ewm(alpha=1/N, adjust=False, min_periods=N).mean()
    rs = pd.Series(np.nan, index=df.index)
    valid_loss = avg_loss > 0
    rs[valid_loss] = avg_gain[valid_loss] / avg_loss[valid_loss]
    rsi = 100 - (100 / (1 + rs))
    loss_is_zero = (avg_loss == 0)
    gain_is_positive = (avg_gain > 0)
    rsi[loss_is_zero & gain_is_positive] = 100.0
    rsi[loss_is_zero & ~gain_is_positive] = 50.0
    df[f'RSI@N={N}'] = rsi
    rsi_shifted = rsi.shift(1)
    # Buy condition: RSI < threshold
    signal = (rsi_shifted < threshold) & rsi_shifted.notna()
    return df.loc[signal, 'trade_date'].tolist()
```

Figure 6: Representative buy-side atomic strategy. The function computes a continuous RSI factor and converts the previous day's factor value into a buy signal through an oversold threshold.

domains are declared, and the resulting atomic strategy is registered. No modification to the Engine's execution protocol is required. In this work, however, we fix the registered library throughout evaluation and study strategy generation over approved atomic strategies rather than asking models to implement new factors. Evaluating LLM-generated factor and backtesting code is a distinct task studied by BacktestBench Wang et al. (2026); incorporating such implementations after validation is supported by the Engine but lies outside the scope of this paper.

C.4 Execution Protocol

The account follows a two-state process. When flat, only a buy signal can change the state; when invested, only a sell signal can do so. A buy is executed at the opening price and uses the largest affordable position after fees. Chinese equity orders are rounded down to multiples of 100 shares, while U.S. equity orders use one-share units. A sell is executed at the closing price and liquidates the entire position. No new position may be opened on the final day, and any remaining position is forcibly liquidated at that day's closing price.

For Chinese equities, the default commission rate is 0.03% on both buys and sells, with a minimum commission of CNY 5. A 0.1% stamp duty is additionally charged on sales. The U.S. equity experiments use zero commission and zero stamp duty. Although callers may override these values, all methods within an experiment share exactly the same configuration.

After each market close, portfolio value is computed as

$$V_t = \text{Cash}_t + Q_t \cdot P_t^{\text{close}}, \tag{5}$$

```
def MacdDeathCrossSellStrategy(df: pd.DataFrame, short: int = 12, long: int = 26, signal: int = 9) -> list:
    """
    Moving Average Convergence Divergence (MACD) Death Cross strategy.
    """
    df.sort_values('trade_date', inplace=True)
    df.reset_index(drop=True, inplace=True)
    # Calculate MACD indicator
    short_ema = df['closing_price'].ewm(span=short, adjust=False).mean()
    long_ema = df['closing_price'].ewm(span=long, adjust=False).mean()
    macd_line = short_ema - long_ema
    signal_line = macd_line.ewm(span=signal, adjust=False).mean()
    # Get T-1 and T-2 MACD and signal-line values
    m1, s1 = macd_line.shift(1), signal_line.shift(1)
    m2, s2 = macd_line.shift(2), signal_line.shift(2)
    # Evaluate logical conditions
    death_cross = (m2 > s2) & (m1 < s1)
    df['MacdDeathCrossSell'] = death_cross.fillna(False)
    return df.loc[death_cross.fillna(False), 'trade_date'].tolist()
```

Figure 7: Representative sell-side atomic strategy. The function identifies a MACD death cross from lagged indicator values and returns the corresponding sell-signal dates.

where Q_t is the number of shares held, P_t^{close} is the closing price on day t, and Cash_t is the cash balance. Daily portfolio return is

$$R_t = \frac{V_t}{V_{t-1}} - 1. \tag{6}$$

The Engine records every account transition and produces the full portfolio path under the same capital, trading-unit, fee, tax, and liquidation constraints.

C.5 Backtesting Metrics

All 12 metrics are computed from the same execution result. Let T be the number of trading days, V_0 and V_T the initial and final portfolio values, R_t the daily portfolio return, R_t^m the aligned return of the market index associated with the stock's exchange, and $r_f = 0.0001$ the daily risk-free return used by the Engine. The market reference index is the CSI 300 (`000300.SH`) for Shanghai, the Shenzhen Component Index (`399001.SZ`) for Shenzhen, the Nasdaq Composite (`IXIC.XNAS`) for Nasdaq, and the NYSE Composite (`NYA.XNYS`) for the New York Stock Exchange. Let

$$D_t = \frac{\max_{u \le t} V_u - V_t}{\max_{u \le t} V_u}, \qquad D_{\max} = \max_t D_t, \tag{7}$$

denote the daily drawdown and maximum-drawdown ratio. For completed trade j, let p_j be its realized profit or loss. Table 3 gives the definitions used in the implementation. Percentage-valued outputs are explicitly marked; daily volatility is intentionally not annualized.

The profit-to-loss ratio is undefined when the backtest contains no winning trades or no losing trades. Calmar ratio and recovery factor are undefined when maximum drawdown is numerically zero. Beta and alpha require at least two aligned strategy and benchmark returns and nonzero benchmark-return variance. Metrics are reported only when their required observations and denominators are available; the Engine does not assign an artificial finite value to an undefined case.

Table 3: The 12 metrics computed by the unified Engine. Preferred direction indicates how a target is normally interpreted during task construction.

Metric	Category	Definition	Preferred
Annualized return	Return	$100\left[(V_T/V_0)^{252/T}-1\right]$	Higher
Maximum drawdown	Risk	$100D_{\max}$	Lower
Daily volatility	Risk	Sample standard deviation of $\{R_t\}_{t=1}^{T}$	Lower
Annualized Sharpe ratio	Risk-adjusted	$\sqrt{252}\,(\overline{R}-r_f)/\operatorname{Std}(R)$	Higher
Win rate	Trade quality	$100\,\lvert\{j : p_j > 0\}\rvert/J$, where J is the number of completed trades	Higher
Profit-to-loss ratio	Trade quality	Mean profit of winning trades divided by the absolute mean loss of losing trades	Higher
Calmar ratio	Risk-adjusted	$\left[(V_T/V_0)^{252/T}-1\right]/D_{\max}$	Higher
Sortino ratio	Risk-adjusted	$\sqrt{252}\,(\overline{R}-r_f)/\sqrt{T^{-1}\sum_t \min(R_t-r_f,0)^2}$	Higher
Recovery factor	Risk-adjusted	$[(V_T-V_0)/V_0]/D_{\max}$	Higher
Maximum drawdown duration	Risk	Longest number of trading-day steps from a portfolio peak until recovery to that peak, or until the end of the backtest if unrecovered	Lower
Beta	Market exposure	$\operatorname{Cov}(R,R^m)/\operatorname{Var}(R^m)$ using date-aligned daily returns	Task-dependent
Alpha	Market exposure	$100\times 252\,[\overline{R}-r_f-\beta(\overline{R^m}-r_f)]$	Higher

C.6 Detailed Backtesting Outputs

The detailed output includes the complete trade ledger, daily portfolio value, daily return, and enriched market data. Each completed trade records its entry and exit dates, execution prices, quantity, purchase cost, sale proceeds, and realized profit or loss. Daily records include the account action, holdings, cash, portfolio value, cumulative return, drawdown, and unrealized profit or loss. AlphaLoop uses these outputs to analyze signal frequency, trade outcomes, holding paths, and drawdown periods.

C.7 Time-Series Factor IC

Let $P_t^{\text{close}}(s)$ be the closing price of stock s on trading day t. The next-day return is

$$r_{t+1}(s)=\frac{P_{t+1}^{\text{close}}(s)}{P_t^{\text{close}}(s)}-1. \tag{8}$$

For stock s and factor function f, let $\mathcal{T}_{s,f}$ be the set of dates on which both the factor value $F_{s,t}(f,\boldsymbol{\theta})$ and next-day return $r_{t+1}(s)$ are valid. Let $\widetilde{F}_{s,t}(f,\boldsymbol{\theta})$ and $\widetilde{r}_{t+1}(s)$ denote values winsorized at the 1st and 99th percentiles. The time-series IC is

$$\mathrm{IC}(f,s,\boldsymbol{\theta})=\rho\left(\left\{\widetilde{F}_{s,t}(f,\boldsymbol{\theta})\right\}_{t\in\mathcal{T}_{s,f}},\{\widetilde{r}_{t+1}(s)\}_{t\in\mathcal{T}_{s,f}}\right), \tag{9}$$

where ρ is Spearman's rank correlation. The Engine does not report the value when $|\mathcal{T}_{s,f}|<10$.

IC information boundary. The next-day return r_{t+1} is used only after it has been observed to estimate an aggregate IC over the designated historical research interval. This offline statistic ranks factors during candidate construction but is neither encoded in the submitted strategy nor queried by the trading simulator. At every simulated trading date t, each factor value and signal is computed solely from $\mathbf{D}_{s,\le t}$; the strategy therefore cannot access r_{t+1} or any later observation when making that decision. Reusing the historical interval for IC estimation, iterative candidate backtesting, and goal-based selection is a deliberate design choice that mirrors practical strategy develop-

Table 4: Summary of the daily market data stored in PostgreSQL.

Market	Securities	Daily observations	Trading days
SSE	1,349	3,471,274	2,790
SZSE	1,975	5,114,901	2,790
Nasdaq	1,266	3,508,810	2,889
NYSE	1,365	3,858,675	2,889
Total	5,955	15,953,660	-

ment: researchers learn from historical outcomes and retain strategies whose backtested behavior satisfies their objectives. AlphaGoal thus benchmarks the ability to carry out this retrospective strategy-search workflow under a fixed budget; it does not, by itself, evaluate held-out predictive generalization. Once selected, the executable strategy requires no IC or future return and can be run prospectively using only information available at each decision time.

For comparison, the conventional cross-sectional IC fixes day t and computes the correlation across stocks in universe $\mathcal{U}_t$:

$$\mathrm{IC}_{\mathrm{CS}}(f,t,\boldsymbol{\theta}) = \rho\left(\left\{\widetilde{F}_{s,t}(f,\boldsymbol{\theta})\right\}_{s\in\mathcal{U}_{t,f}}, \{\widetilde{r}_{t+1}(s)\}_{s\in\mathcal{U}_{t,f}}\right), \tag{10}$$

where $\mathcal{U}_{t,f}$ is the set of stocks with valid observations for factor f on day t. Our tasks concern one stock at a time, so AlphaLoop uses time-series IC as auxiliary factor information. IC is not treated as a substitute for complete-strategy performance.

D AlphaGoal Construction Details

D.1 Market Data and PostgreSQL Database

We obtain daily price and trading records for equities listed on the Shanghai Stock Exchange (SSE), Shenzhen Stock Exchange (SZSE), Nasdaq, and New York Stock Exchange (NYSE) from the publicly accessible Yahoo Finance website, together with the benchmark-index series required by the Engine. Yahoo symbols are mapped to the Engine's standardized exchange suffixes, and dates, field names, and numerical formats are normalized before storage in four market-specific PostgreSQL tables. The complete database contains 5,955 security time series and 15,953,660 daily observations. Chinese markets contribute 3,324 securities and 8,586,175 observations, while the U.S. markets contribute 2,631 securities and 7,367,485 observations.

The database covers January 2015 through June 30, 2026 as one historical interval. The first available trading date is January 5, 2015 for the Chinese markets and January 2, 2015 for the U.S. markets. The Chinese tables contain 2,790 distinct trading days, and the U.S. tables contain 2,889. Table 4 summarizes the coverage by market.

Every market table contains the security code, trading date, opening price, closing price, highest price, lowest price, trading volume, trading amount, daily percentage change, and security name. Numerical market fields are stored at daily frequency using a common schema, which allows the Engine to retrieve Chinese and U.S. securities through the same data interface. The import audit found no missing values in the core OHLC, volume, amount, or percentage-change fields, and no records with invalid high-low relationships or negative trading volume or amount.

The PostgreSQL database represents the complete collected market-data repository rather than the final backtesting universe. Before empirical strategy sampling, we remove stocks with insufficient history, incomplete required metadata, or non-executable Engine requests. This produces the final set of 5,941 eligible stocks used to construct the empirical strategy pool.

D.2 Empirical Distribution Visualization and Threshold Selection

Each random candidate contains at least one buy-side and one sell-side atomic strategy. We sample the number of rules, their names, and legal parameter values. Rules on the same side are combined by conjunction. Every candidate is executed by the unified Engine, and invalid configurations or failed executions are excluded from the empirical pool. For each valid strategy, we store its structure, 12 metrics, and Buy-and-Hold reference under the same backtesting configuration.

To construct verifiable targets without manually prescribing performance levels, we approximate the performance distribution induced by the controlled strategy space through large-scale random sampling. For each stock, we sample strategy structures and legal parameters, execute them with the unified Engine, and record all 12 metrics. The resulting empirical distribution shows how common or rare a performance level is under the fixed sampler and supplies executed strategies as evidence that selected targets are reachable; it is not presented as an exhaustive characterization of the full combinatorial space.

This approximation has a direct statistical basis. For a fixed stock s, candidate strategies are drawn independently from the same fixed sampling distribution. Invalid candidates are rejected using the same validity rule, so the retained strategies can be viewed as independent draws from the conditional distribution Q_s^{valid}. Because the Engine is deterministic for a given strategy and backtesting configuration, the metric values $X_n = \mathcal{M}_m(\pi_n, s)$ are also independent and identically distributed. For any value y, the empirical cumulative distribution satisfies

$$\widehat{F}_{s,m,N}(y) = \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}[X_n \le y] \xrightarrow{\text{a.s.}} F_{s,m}(y) = \Pr_{\pi \sim Q_s^{\text{valid}}}[\mathcal{M}_m(\pi, s) \le y] \tag{11}$$

as $N \to \infty$, by the strong law of large numbers. At a fixed y, the estimator has variance $F_{s,m}(y)(1-F_{s,m}(y))/N$, so its sampling error decreases at the standard rate $N^{-1/2}$. Approximately 26,000 valid strategies for this stock therefore provide a substantially more stable estimate than a small pilot sample. The convergence is to the distribution induced by the specified sampler and validity filter. It does not imply exhaustive coverage of every strategy in the combinatorial space.

Figure 8 makes two properties of the search problem visible. First, strategy performance is highly heterogeneous: the metric distributions are skewed, multimodal, or concentrated at discrete values rather than following a common parametric form. This motivates stock- and metric-specific empirical calibration. Second, Buy-and-Hold is a demanding but not dominant reference for this stock. Its dashed line lies toward the favorable side of several return and risk-adjusted-performance distributions, so most randomly sampled active strategies are worse than simply holding the stock. Nevertheless, some sampled strategies lie beyond the reference in the preferred direction. The figure therefore demonstrates both why unconstrained random strategy construction is usually insufficient and why searching for an active strategy that improves on Buy-and-Hold remains empirically feasible.

The Buy-and-Hold win rate of 100% is a mechanical consequence of the evaluation unit, not evidence of repeated trading success. Buy-and-Hold creates exactly one completed round trip. It purchases the stock at the beginning of the interval and liquidates the position at the end. Its win rate can therefore only be 0% or 100%. For NPK-XNYS, that single trade is profitable. Its profit-to-loss ratio is undefined because this statistic requires both winning and losing completed trades, which a single round trip cannot provide.

We derive target thresholds from these empirical distributions, although the P-level thresholds themselves are not drawn in Figure 8. Level P_p has a nominal marginal pass proportion of $p\%$; thus, P_1 corresponds to the 99th percentile for higher-is-better metrics and the 1st percentile for lower-is-better metrics. A P_1 target lies deeper in the desirable tail than P_{10}. We retain a stock–metric threshold only when it is realized by an executed strategy and passes the applicable viability and Buy-and-Hold filters described below. For dual-metric tasks, P_p is applied marginally to both metrics, while success requires their joint satisfaction.

Together, the distribution, reference, and tail threshold play distinct roles: the empirical percentile controls solution scarcity, an observed satisfying strategy establishes reachability, and the viability and Buy-and-Hold filters exclude rare but financially uninformative targets. Because distributions

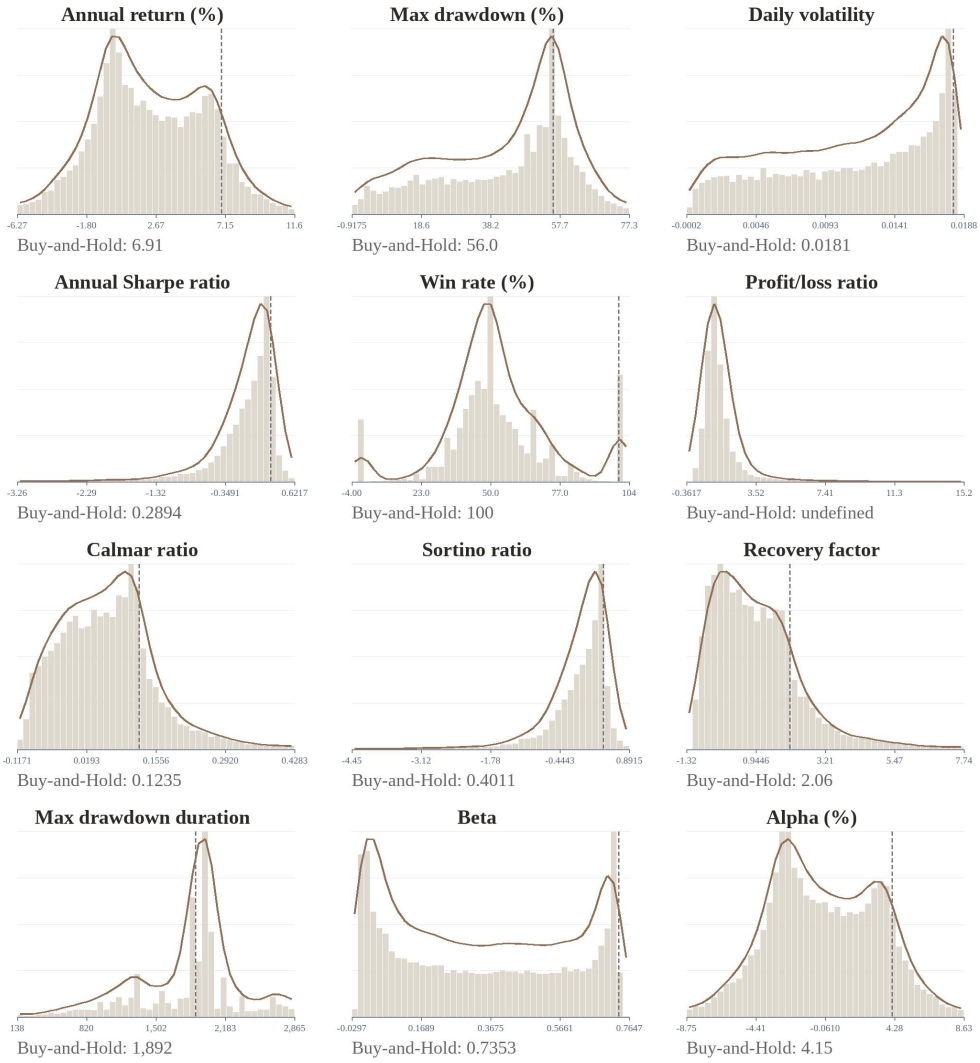


Figure 8: Empirical distributions of 12 backtesting metrics for NPK-XNYS across approximately 26,000 randomly sampled valid strategies. Gray bars show the histogram and the brown curve is a smoothed density estimate. The gray dashed line and the value printed beneath each panel report Buy-and-Hold performance under the same backtesting protocol. For several return and risk-adjusted metrics, most sampled strategies underperform Buy-and-Hold, but a nonempty favorable tail exceeds it.

vary across stocks with their volatility, trend, and tradability, this calibration is performed separately for each stock rather than imposing universal thresholds that may be trivial for one security and unattainable for another.

D.3 Empirical Thresholds and Tie Handling

For a metric m where larger values are preferred, we select an observed threshold $\tau_{\text{high}}(m, s, p)$ from the empirical distribution such that the proportion of strategies satisfying

$$\mathcal{M}_m(\pi, s) \geq \tau_{\text{high}}(m, s, p) \tag{12}$$

is as close as possible to, but does not exceed, the desired proportion $p\%$. For a metric where smaller values are preferred, we analogously select $\tau_{\text{low}}(m, s, p)$ using

$$\mathcal{M}_m(\pi, s) \leq \tau_{\text{low}}(m, s, p). \tag{13}$$

Thresholds are selected only from observed metric values and are never interpolated. When ties make the nominal proportion unattainable, we choose the observed threshold with the closest realized marginal pass proportion not exceeding $p\%$. The realized count and proportion are stored for auditing but do not change the assigned P_p level.

For dual-metric tasks, both directional thresholds are derived from valid paired observations, and a strategy must satisfy both conditions. Their realized joint pass proportion is stored separately and is no greater than either marginal proportion. Because joint satisfaction is rarer, dual-metric tasks use a broader set of P_p levels.

D.4 Business Constraints and Viability Filtering

Raw empirical percentiles can identify rare outcomes, but rarity alone does not guarantee practical value. A strategy with 1% empirical frequency may underperform a passive Buy-and-Hold approach or violate the benchmark's minimum viability criteria. We therefore impose business constraints to ensure that derived targets represent meaningful investor goals under the standardized protocol.

For each metric, we define a minimum viability threshold based on quantitative finance practice. Return-based metrics must meet: annualized return $\geq 3\%$, alpha $\geq 0\%$, annualized Sharpe ratio ≥ 0.30, Calmar ratio ≥ 0.50, Sortino ratio ≥ 0.50, and recovery factor ≥ 0.80. Risk-based metrics are bounded above: maximum drawdown $\leq 30\%$, maximum drawdown duration ≤ 200 days, daily volatility ≤ 0.02, and absolute beta ≤ 0.50. Trade-quality metrics require: win rate $\geq 40\%$ and profit-loss ratio ≥ 1.0.

Additionally, for return-oriented metrics (annualized return, alpha, Sharpe ratio, Calmar ratio, Sortino ratio, recovery factor), we compute the Buy-and-Hold result on the same stock under identical capital, fee, and execution settings. Candidate thresholds must strictly exceed the Buy-and-Hold performance. This eliminates targets that are achievable but offer no advantage over passive investment, ensuring that tasks measure active strategy construction rather than stock selection.

A stock-metric-percentile combination is eligible for task construction only if the candidate threshold satisfies both the business constraint and the Buy-and-Hold requirement. If a stock's empirical distribution at percentile p does not meet these criteria, no task is generated at that difficulty level for that metric. This filtering ensures that all retained targets are both empirically reachable and practically meaningful.

D.5 Reference Strategy Selection

Each accepted task is paired with one executable reference strategy from the empirical pool as evidence that all of its conditions are jointly reachable. The same five-stage selection procedure is used for single- and dual-metric tasks:

1. **Establish goal satisfaction.** Retain all strategies that satisfy the task-specific target conditions. These are the one or two metric constraints stated in the task, each consisting of a metric, a comparison operator, and a stock-specific threshold derived from the empirical distribution (e.g., annualized return $\geq 10\%$ or annualized return $\geq 10\%$ together with maximum drawdown $\leq 20\%$). This step establishes that the recorded reference is a valid witness for the exact goal being evaluated.

2. **Enforce minimum evidence quality.** Apply dataset-wide filters that are independent of the task-specific targets. A candidate must have annualized return strictly above 3%, contain at least 12 completed trades, and exhibit no forced-liquidation artifact or other abnormal backtesting outcome. This step prevents a formally satisfying but economically trivial, weakly evidenced, or execution-dependent result from serving as the reference. These requirements are construction-time filters and are not additional conditions presented to the evaluated model.

3. **Prioritize trading evidence.** Retain the candidates with the largest number of completed trades. A larger trade sample provides broader empirical support for the reported metrics and reduces the likelihood that goal satisfaction is driven by only a few exceptional trades.
4. **Prioritize structural richness.** From these candidates, retain those with the largest total number of buy- and sell-side atomic strategies. When trading evidence is comparable, this step favors references that exercise a richer portion of the compositional strategy space rather than relying on the simplest available rule combination.
5. **Select the final reference.** If more than one candidate remains, choose the one with the highest annualized return. This supplies a financially meaningful final discriminator without overriding the preceding requirements on feasibility, evidence quality, trading activity, and structural richness. The comparison is rarely needed because the earlier filters usually identify a single candidate.

For parameter optimization tasks, the selected reference additionally determines the fixed atomic-strategy structure exposed to evaluated methods. Its factor names, counts, ordering, and buy/sell roles are recorded in the task specification, while the successful parameter values remain hidden. In every setting, the reference is only a feasibility witness: it is not shown to the evaluated model, does not constitute a ground-truth output, and has no role in scoring beyond establishing that the task admits a valid solution.

D.6 Difficulty Levels

Single-metric strategy generation tasks use P_1–P_{10}: P_1–P_3 are hard, P_4–P_6 medium, and P_7–P_{10} easy. Dual-metric tasks use selected levels from P_2 to P_{35}: P_2–P_{10} are hard, P_{15}–P_{25} medium, and P_{30}–P_{35} easy. Tables combine the two task types only after mapping their nominal levels to these common labels; realized marginal and joint proportions are audit statistics rather than relabeling criteria.

parameter optimization tasks are not assigned these labels. Once the atomic-strategy identities and directions are fixed, the search space is defined by the associated parameter domains rather than by the frequency of successful structures in the full strategy space.

D.7 Parameter-Optimization Task Construction

Each parameter optimization task begins with a strategy in the empirical pool that satisfies the target conditions. We expose the names, counts, order, and buy/sell roles of its atomic strategies, while hiding the parameter values that produced the successful backtest. The evaluated method must recover another satisfying parameter configuration without changing the structure. This design provides reachability evidence without revealing the answer.

D.8 Natural-Language Query Generation

The conversion from structured task records to natural-language queries follows a multi-model generation and peer-review protocol to ensure semantic accuracy, completeness, and information security.

D.8.1 Generation Models and Prompt Design

We use five independent generation agents: DeepSeek-V4-Flash-0731, Qwen3.5-397B-A17B, Hy3, GLM-5.1, and MiniMax-M2.7-BF16. Each model rewrites the same structured task record as a natural-language strategy request. The input contains the stock code, start date, initial capital, metric conditions, task type, and, for parameter optimization, the fixed atomic-strategy names. The generation prompt enforces a strict field whitelist that explicitly enumerates which information may appear in the request and which must remain hidden.

For strategy generation tasks, the prompt instructs the model to produce a natural English description that communicates the investor goal, target stock, backtesting start date, initial capital, and the structured metric conditions (metric name, comparison operator, and threshold). The description must

be clear, unambiguous, and suitable for an LLM-based research agent. Crucially, the prompt prohibits exposure of reference strategies, parameter values, backtesting results, benchmark identities, difficulty labels, and any other information not in the whitelist.

For parameter optimization tasks, the prompt additionally allows the model to mention the fixed buy-side and sell-side atomic-strategy names. However, implementation-specific suffixes (e.g., version identifiers or parameter hints embedded in function names) are stripped before exposure to prevent information leakage. All parameter values, the reference configuration that achieved the target, and the actual performance metrics remain hidden.

The prompt also instructs the model to use standard financial terminology, maintain temporal consistency (e.g., correctly interpreting "start date" rather than "evaluation date"), and distinguish strategy-specific logic from Engine-level defaults (e.g., transaction costs, liquidation rules). This guidance reduces ambiguity and minimizes the risk of generating queries that could be misinterpreted during evaluation.

Figures 9 and 10 show the complete rewriting prompts for the two task types.

Both prompts treat the structured JSON as data and require exactly one natural, first-person English request. They preserve the stock, start date, initial capital, metric names, comparison operators, and thresholds verbatim in meaning, while excluding backtesting results, reference solutions, and additional constraints. Under the common evaluation protocol, all backtests end at the database cutoff of June 30, 2026. Their difference follows the evaluated search space. strategy generation requests must not reveal any trading rule or indicator because the method must construct the strategy itself. parameter optimization requests expose the fixed buy and sell factor identities and roles, but prohibit parameter names and values because only their configuration remains to be searched. The prompts also vary surface form without changing task semantics, so linguistic diversity does not alter the programmatically evaluated conditions.

D.8.2 Field Whitelist and Security Protocol

To prevent information leakage that could simplify tasks artificially, we enforce a strict field whitelist during query generation. Table 5 specifies which structured fields may be exposed in the natural-language query and which must remain hidden.

Table 5: Field whitelist for natural-language query generation. A checkmark indicates an exposed field; a cross indicates a hidden field.

Field	**Strategy Gen**	**Parameter Opt**
Stock code	✓	✓
Start date	✓	✓
Initial capital	✓	✓
Metric conditions	✓	✓
Atomic strategy names	×	✓
Parameter values	×	×
Reference strategy	×	×
Backtesting results	×	×
Benchmark identity	×	×
Difficulty label	×	×

This whitelist ensures that the natural-language query conveys only the investor goal and task constraints, without revealing solutions or auxiliary metadata that would reduce task difficulty. Generation models are explicitly instructed that violating the whitelist constitutes a failed generation attempt.

D.8.3 Peer-Review Validation Protocol

To ensure quality and correctness, we implement a rigorous peer-review mechanism. For each candidate query generated by model A, the remaining generation models (e.g., models B, C, D, and E) serve as independent auditors. Each auditor evaluates the structured record and the candidate natural-language query along multiple dimensions:

Prompt for Rewrite (Strategy Generation)

You convert structured investment requirements into exactly one natural English question written in the first person, as if a real user were asking about their own investment needs. Treat the supplied JSON strictly as data.

Requirements:

1. Use a natural personal voice with words such as I, me, my, or for me, and ask the assistant directly for a trading strategy. A first-person question does not need to begin with I.
2. Preserve the exact stock code, starting date, initial-capital value, and every performance condition.
3. Produce one complete English interrogative sentence and nothing else.
4. Preserve comparison operators exactly: > means strictly greater than; >= means greater than or equal to; < means strictly less than; <= means less than or equal to.
5. Render metric names naturally without changing meaning: abs_beta means absolute beta; annual_sharpe_ratio means annual Sharpe ratio; max_drawdown_duration means maximum drawdown duration; profit_loss_ratio means profit-to-loss ratio; daily_volatility means daily volatility; calmar_ratio means Calmar ratio; sortino_ratio means Sortino ratio; recovery_factor means recovery factor; max_drawdown means maximum drawdown; annual_return means annual return; win_rate means win rate.
6. Preserve threshold values exactly. Whole-valued numbers may omit trailing .0. Do not add a unit to a threshold unless the input explicitly contains one. Initial capital may be formatted as equivalent currency, and an ISO date may be rendered naturally.
7. Ask for a strategy without supplying trading rules, indicators, parameters, results, an ending date, or additional constraints.

Diversity:

Keep a natural first-person user voice while varying the opening, information order, request wording, connective phrases, and overall sentence structure. In particular, vary whether the sentence begins with the capital, date, stock, request, goal, or trading context. Use the examples as demonstrations of possible styles rather than fixed templates.

Examples:

Input: {"stock_code":"600550.SH","start_date":"2020-01-02","initial_capital":1600000.0,"conditions": [["alpha",">=",51],["abs_beta","<=",0.3]]}
Output: With 1.6 million available to invest in 600550.SH beginning January 2, 2020, what trading strategy should I use to achieve an alpha of at least 51 while keeping absolute beta at most 0.3?
……

Now convert the following input into one accurate, fluent, and stylistically varied first-person user question:
{{INPUT_JSON}}

Figure 9: Prompt for rewriting a structured strategy generation record as a natural-language strategy request. The placeholder {{INPUT_JSON}} is replaced by the task record.

Prompt for Rewrite (Parameter Optimization)

You convert structured factor-parameter optimization requirements into exactly one natural English question written in the first person, as if a real user were asking how to tune the parameters of chosen buy and sell factors. Treat the supplied JSON strictly as data.

Requirements:

1. Use a natural personal voice with words such as I, me, my, or for my setup, and ask the assistant directly for parameter settings.
2. Preserve the exact stock code, starting date, initial-capital value, every buy-factor name, every sell-factor name, and every performance condition.
3. Produce one complete English interrogative sentence and nothing else.
4. Clearly identify each supplied name as a buy factor or a sell factor. Preserve factor identifiers verbatim, including capitalization and digits. Say factor, buy factor, sell factor, buy signal, or sell signal naturally; never append or invent implementation suffixes such as BuyStrategy or SellStrategy, and never describe the identifiers as class or function names.
5. Make clear that the factors and their buy/sell roles are fixed and only their parameters are to be tuned.
6. Preserve comparison operators exactly: > means strictly greater than; >= means greater than or equal to; < means strictly less than; <= means less than or equal to.
7. Render metric names naturally without changing meaning: abs_beta means absolute beta; annual_sharpe_ratio means annual Sharpe ratio; max_drawdown_duration means maximum drawdown duration; profit_loss_ratio means profit-to-loss ratio; daily_volatility means daily volatility; calmar_ratio means Calmar ratio; sortino_ratio means Sortino ratio; recovery_factor means recovery factor; max_drawdown means maximum drawdown; annual_return means annual return; win_rate means win rate.
8. Preserve thresholds exactly. Whole-valued numbers may omit trailing .0. Do not add a unit to a threshold unless explicitly present. Initial capital may be formatted as equivalent currency, and the date may be rendered naturally.
9. Do not supply parameter names or values, additional factors or rules, an ending date, results, or additional constraints.

Diversity:

Vary the opening, information order, factor presentation, request wording, and sentence structure while keeping the buy/sell roles unmistakable. Use the examples as style demonstrations rather than fixed templates.

Examples:

Input: {"stock_code":"601598.SH","start_date":"2020-01-02","initial_capital":100000.0,"conditions":[["alpha",">=",24]],"buy_factor_names":["DoubleBottom"],"sell_factor_names":["ReturnVarianceN"]}
Output: With DoubleBottom as my buy factor and ReturnVarianceN as my sell factor, what parameter settings should I use to trade 601598.SH from January 2, 2020, with 100,000 and achieve an alpha of at least 24?

Now convert the following input into one accurate, fluent, and stylistically varied first-person user question:
{{INPUT_JSON}}

Figure 10: Prompt for rewriting a structured parameter optimization record as a natural-language parameter-tuning request while preserving the fixed buy and sell factor identities.

- **Field correctness**: The stock code, start date, initial capital, metric names, comparison operators, and thresholds in the natural-language query exactly match the structured record.
- **Information security**: The query contains no prohibited fields such as reference parameters, backtesting results, benchmark metadata, or difficulty labels.
- **Structural validity**: For parameter optimization tasks, the query correctly lists the fixed atomic-strategy names and their buy/sell roles without exposing parameters.
- **Semantic clarity**: The query is unambiguous, grammatically correct, and interpretable by an LLM without requiring external context.
- **Financial soundness**: The described goal is financially meaningful and the constraints are expressed in standard quantitative finance terminology.

A candidate query is accepted if and only if it receives unanimous approval from all peer auditors on all criteria. This strict consensus rule ensures high precision at the cost of recall. When multiple candidate queries for the same structured record pass validation, one is selected deterministically using a fixed random seed to ensure reproducibility and prevent dataset duplication.

D.8.4 Post-Generation Quality Control

After peer review, every accepted query undergoes schema validation. We programmatically extract the stock code, date, capital, metric conditions, and (if applicable) atomic-strategy names from the natural-language text, then verify that they match the structured record exactly. Queries that fail programmatic parsing or exhibit inconsistencies are rejected.

We also apply diversity constraints during task sampling. We limit the frequency of each stock across the dataset and prevent repeated stock-goal combinations to reduce selection bias. As an additional quality check for this constrained rewriting task, we manually inspect 200 randomly sampled accepted queries against their structured records and find no semantic errors, numerical inconsistencies, or prohibited information. The final dataset contains 2,221 validated tasks with natural-language queries that accurately represent their structured constraints and satisfy unanimous peer approval.

D.8.5 Representative Validation Examples

To illustrate the validation process, we provide examples of queries that pass and fail peer review:

Example 1: Accepted Query (strategy generation)

Structured Record: UUID = 31, Stock = GNTX.XNAS, Start = 2015-01-02, Capital = USD 1,600,000, Conditions = AnnualReturn $\geq$ 3.0 and DailyVolatility $\leq$ 0.0078

Generated Query: "With my $1.6 million starting on January 2, 2015, how should I trade GNTX.XNAS to aim for an annual return of at least 3 while keeping daily volatility no higher than 0.0078?"

Validation Result: PASS. This is the final validated query stored for UUID 31. It is one first-person interrogative sentence and preserves every required field without revealing a strategy.

Example 2: Rejected Query (Information Leakage)

Structured Record: Same as Example 1

Generated Query: "With my $1.6 million starting on January 2, 2015, how should I trade GNTX.XNAS using ROC and RSI to achieve an annual return of at least 3 while keeping daily volatility no higher than 0.0078?"

Validation Result: FAIL. Exposes specific factor names (ROC, RSI) that are not in the whitelist for strategy generation tasks. This leaks solution hints.

Example 3: Rejected Query (Incorrect Threshold)

Structured Record: Same as Example 1

Generated Query: "With my $1.6 million starting on January 2, 2015, how should I trade GNTX.XNAS to aim for an annual return of at least 3.5 while keeping daily volatility no higher than 0.0078?"

Validation Result: FAIL. The annual-return threshold is changed from 3.0 to 3.5, so the query does not exactly preserve the structured condition.

Example 4: Accepted Query (parameter optimization)

Structured Record: UUID = 33, Stock = 300323.SZ, Start = 2015-01-05, Capital = CNY 1,600,000, Fixed factors = [CORR (buy), STD (sell)], Condition = AnnualReturn $\geq$ 14.8

Generated Query: "Given CORR as my fixed buy factor and STD as my fixed sell factor, how should I tune their parameters for 300323.SZ from January 5, 2015, with CNY 1.6 million, to get an annual return of at least 14.8?"

Validation Result: PASS. This is the final validated query stored for UUID 33. It is one first-person interrogative sentence, preserves the fixed factor identities and roles, and omits their parameter values.

These examples demonstrate that the validation process enforces both information security (no prohibited fields) and semantic accuracy (correct values for all exposed fields).

D.9 Representative Task Examples

Figures 11 and 12 illustrate the two benchmark task types. In each example, only the natural-language query is presented to the evaluated model. The fields `candidate_strategy` and `strategy_performance` record a reference strategy sampled during dataset construction and its backtesting results. They serve only as a feasibility witness demonstrating that the target conditions are reachable. They are hidden from the model and are not treated as ground-truth outputs. A generated strategy is evaluated solely by whether it is valid under the task constraints and satisfies the target conditions after execution by the unified Engine.

Figure 11 shows an easy, dual-metric strategy generation task. The query exposes the stock, start date, initial capital, and two quantitative conditions: annualized return must be at least 5.5%, and daily volatility must not exceed 0.008. It does not reveal any factor, atomic-strategy structure, or parameter value. The recorded reference combines four buy-side atomic strategies with one sell-side atomic strategy and achieves an annualized return of 5.5837% with daily volatility of 0.007611. This strategy establishes reachability, but the evaluated model may construct an entirely different strategy. Any valid strategy satisfying both conditions is accepted.

Figure 12 shows a single-metric parameter optimization task. The query exposes CORR as the fixed buy factor and STD as the fixed sell factor, together with the stock, start date, initial capital, and required annualized return. Their parameter values remain hidden. The recorded reference configuration achieves an annualized return of 14.9896%, demonstrating that the 14.8% target is reachable. The evaluated model must preserve the fixed atomic-strategy structure, but it is not required to recover the recorded parameter values. Any legal parameter configuration that satisfies the target is accepted.

D.10 Sampling and Quality Control

Tasks are sampled within metric or metric-pair cells, with additional difficulty stratification for strategy generation. Quotas control market, metric, and task-type coverage. We limit the frequency of each stock and prevent repeated stock-goal combinations to reduce selection bias and improve diversity.

Every final record passes schema validation for the stock, dates, atomic-strategy names, parameter signatures, and Engine request. We remove non-finite targets, invalid requests, duplicate tasks, and references that fail to satisfy all conditions. parameter optimization records receive an additional structural audit. The final dataset contains 2,221 validated tasks.

Strategy generation QA example

"query": "With my 1.6 million starting on January 2, 2015, how should I trade GNTX.XNAS to aim for an annual return of at least 5.5% while keeping daily volatility no higher than 0.008?",
"difficulty": "easy"
"task": "strategy_generation",
"type": "double",
"conditions": [
{"metric": "annual_return", "operator": ">=", "threshold": 5.5},
{"metric": "daily_volatility","operator": "<=","threshold": 0.008}],
"candidate_strategy": {
"stock_code": "GNTX.XNAS",
"start_date": "2015-01-02",
"initial_capital": 1600000.0,
"strategy": {
"buy_rules": [
{"name": "SingleDayVPTBuyStrategy","params": {"threshold": 1.5,"N": 10}},
{"name": "OpenAboveMovingAverageBuyStrategy","params": {"window": 20}},
{"name": "MovingAverageTrendBuyStrategy","params": {"long_window": 15}},
{"name": "ArbrSentimentBuyStrategy","params": {"window": 60,"ar_threshold": 110.0,"br_threshold": 40.0}}],
"sell_rules": [
{"name": "DownpourPatternSellStrategy","params": {}}]}},
"strategy_performance": {"annual_return": 5.5837,"daily_volatility": 0.007611}

Figure 11: strategy generation example. The `candidate_strategy` and `strategy_performance` fields show a hidden feasibility reference rather than a ground-truth answer.

Parameter optimization QA example

"query": "Given CORR as my fixed buy factor and STD as my fixed sell factor, how should I tune their parameters for 300323.SZ from January 5, 2015, with 1.6 million, to get an annual return of at least 14.8%?",
"task": "parameter_optimization",
"type": "single",
"metric": "annual_return",
"conditions": [{"metric": "annual_return","operator": ">=","threshold": 14.8}],
"candidate_strategy": {
"stock_code": "300323.SZ",
"start_date": "2015-01-05",
"end_date": "2026-06-30",
"initial_capital": 1600000.0,
"strategy": {
"buy_rules": [
{"name": "CORRBuyStrategy","params": {"N": 10,"threshold": 0.3}}],
"sell_rules": [
{"name": "STDSellStrategy","params": {"N": 20,"threshold": 0.03}}]}},
"strategy_performance": {"annual_return": 14.9896}

Figure 12: parameter optimization example. The recorded parameters and performance provide a hidden feasibility reference rather than a ground-truth answer; any legal parameter configuration satisfying the goal is accepted.

D.11 Dataset Composition and Scale

The complete AlphaGoal benchmark comprises 2,221 tasks distributed across two task types, two constraint cardinalities, and three difficulty levels. Table 6 summarizes the dataset composition. The benchmark contains 1,360 strategy generation tasks, in which models must select both atomic strategies and parameters, and 861 parameter optimization tasks, in which the atomic-strategy structure is fixed and only parameters may be adjusted.

Table 6: AlphaGoal dataset composition. Difficulty labels apply only to strategy generation tasks.

Category	Number of tasks
Strategy generation tasks	1,360
Parameter optimization tasks	861
Single-metric tasks	1,530
Dual-metric tasks	691
Easy strategy generation tasks	380
Medium strategy generation tasks	447
Hard strategy generation tasks	533
Total tasks	2,221

Table 7: Distribution of single-metric tasks by metric.

Metric	strategy generation	parameter optimization
Alpha	153	108
Annualized return	146	103
Annualized Sharpe ratio	146	108
Calmar ratio	146	110
Recovery factor	149	108
Sortino ratio	147	106
Total	887	643

From a constraint perspective, 1,530 tasks specify a single metric condition (e.g., annualized return $\geq 15\%$), while 691 tasks impose dual-metric constraints that require simultaneous satisfaction of two conditions (e.g., return $\geq 12\%$ and Sharpe ratio ≥ 1.8). Dual-metric tasks are inherently more difficult because the joint pass probability is lower than either marginal probability.

For strategy generation tasks, difficulty labels are derived from the nominal empirical P_p levels defined above. After separately mapping single- and dual-metric levels, the 1,360 tasks comprise 380 easy, 447 medium, and 533 hard tasks. parameter optimization tasks do not receive difficulty labels because their search space is constrained by the fixed atomic-strategy structure rather than the full combinatorial strategy space.

This distribution ensures broad coverage across task types, metrics, and difficulty levels, enabling fine-grained evaluation of model capabilities in goal-oriented strategy optimization.

D.12 Why Goals Are Derived from Empirical Strategy Distributions

Real-user requests capture authentic preferences and language, but they do not show that the requested conditions are feasible under a particular strategy space and execution protocol. A failure on an unverified request could reflect poor search, insufficient budget, or an impossible goal. Even locating one feasible solution would not characterize difficulty, because two reachable goals may have very different proportions of satisfying strategies.

AlphaGoal therefore derives stock-specific goals from observed strategy-performance distributions. An executed reference strategy establishes reachability, while the empirical pass proportion provides a reproducible measure of solution scarcity for the P levels. Domain-specific lower bounds and matched Buy-and-Hold comparisons remove goals that are reachable but not financially meaningful. The empirical proportion is a calibration relative to the fixed sampler, not a claim that the pool exhausts the full strategy space.

Natural-language interaction is retained by converting the verified structured records into audited user-facing queries. This design prioritizes controlled internal comparison; independently collected user requests remain complementary for evaluating external validity.

D.13 Why Reachable Goals Remain Meaningful under Random Search

Reachable goals with positive empirical pass proportions can be found by Random Search. This is a consequence of feasibility and a shared search space, not an evaluation shortcut; forcing the random baseline to have zero probability of success would require giving it a different candidate space.

Let p denote the probability that one candidate independently drawn from a fixed reference distribution satisfies all conditions of a task. Under the idealized independent-trial assumption, the probability that Random Search succeeds at least once within B attempts is

$$\Pr(\text{success within } B) = 1 - (1 - p)^B. \tag{14}$$

For any $p > 0$, this probability approaches one as B grows. AlphaGoal therefore evaluates finite-budget search efficiency rather than asymptotic solvability. The formula is an idealized interpretation; ties, discrete structures, and joint conditions make actual difficulty empirical, so we report the implemented Random Search baseline and Pass@B curves.

Random Search measures how much can be achieved through uninformed repetition alone. As expected, larger budgets make it competitive on easy tasks with dense satisfying regions, while medium and hard tasks better distinguish methods that use prior knowledge and feedback. Reachability makes random success possible; difficulty determines how much budget it requires.

D.14 Why AlphaGoal Uses Audited Atomic Strategies Instead of Model-Generated Factor Code

As demonstrated by AlphaForgeBench Zhang et al. (2026), generating executable factor and signal code is a meaningful end-to-end task. AlphaGoal instead isolates goal-directed search and iterative optimization. A unified Engine standardizes trading after signals are produced, but generated code still determines the signals and may differ in temporal alignment, rolling-window semantics, missing-value handling, or look-ahead validity. Performance would therefore mix strategy research with coding reliability and implementation convention.

Unrestricted code would also make the effective hypothesis space model-dependent. AlphaGoal derives reachability evidence and P levels from a fixed set of atomic strategies and legal parameters; dynamically generated factors would disconnect this construction space from the space searched during evaluation and weaken baseline fairness. Audited primitives ensure that identical names and parameters produce identical signals for every method.

The fixed library does not prescribe the final strategy: methods must still choose and combine entry and exit rules, set parameters, manage signal density, and revise candidates from feedback. We standardize the vocabulary of strategy construction, not the strategy itself. This improves attribution, comparability, auditability, and difficulty calibration, while explicitly leaving novel factor discovery and factor-code implementation outside AlphaGoal's scope.

E AlphaLoop Implementation Details

E.1 Task Inputs and Workflow

AlphaLoop uses a single LLM to construct and revise strategies through four research tools. Following the main text, s denotes the task stock, π a complete strategy, $\mathcal{M}(\pi, s)$ its 12 backtesting metrics, and B the valid-backtest budget. The natural-language query specifies the stock, backtesting start date, and initial capital. The evaluator retains the end date in the structured Engine input and applies it to every candidate for that task. Let q denote this query and $\mathcal{C}$ the original set of one or two structured metric conditions. The model initially receives q, task-specific instructions, and tool descriptions, not $\mathcal{C}$ as a separate input. The conditions are retained for programmatic checking; their comparison results become available to the model after a valid backtest.

The workflow connects factor prioritization and rule interpretation to candidate construction, backtesting, and evidence-guided revision. In strategy generation, `run_factor_ic` supplies a historical factor prior and `get_factor_info` explains candidate rules and parameters. The LLM proposes a

strategy π, which `run_backtest` executes and checks against $\mathcal{C}$. When further revision is needed, `terminal` supports analysis of the resulting trade and portfolio paths. parameter optimization follows the same workflow without IC-based factor selection and preserves the supplied rule structure. The following tool algorithms describe these information exchanges rather than a fixed schedule of tool calls.

E.2 Task-Specific Prompts

The strategy generation prompt specifies the structured strategy format, conjunction among same-side rules, and the available tools. It encourages IC-guided exploration of a small shortlist while distinguishing entry selection from exit design. Each candidate must contain at least one buy rule and one sell rule, with factor and signal parameters $(\boldsymbol{\theta}, \boldsymbol{\phi})$ as defined in the main text. IC guides search but neither determines the trading rules nor enters their execution.

For parameter optimization, the query supplies the fixed buy and sell atomic-strategy names and roles, without the successful reference parameters. The prompt restricts revisions to parameters: rule names, counts, ordering, and sides must remain unchanged. The IC tool is unavailable in this setting. Both prompts encourage the model to relate backtesting outcomes to rule behavior before proposing another candidate.

Figures 13 and 14 present condensed versions of the two prompts. Ellipses omit repeated operational instructions and examples; the complete prompts are released with the implementation.

E.3 Factor Prioritization and Rule Construction

Historical factor prior. For strategy generation, `run_factor_ic` computes the time-series $\mathrm{IC}(f, s, \boldsymbol{\theta})$ defined in Appendix C, using catalog-default factor parameters over the task's full historical interval. The prompt requests the 20 factors with the largest absolute IC values. Denote the returned factor names and signed IC values by $\mathcal{I}$. This summary prioritizes inspection, not admissibility: lower-ranked factors remain available, and exit rules are selected according to their trading roles rather than IC rank alone. The full result is saved once and reused.

Algorithm 1 Historical factor prioritization with `run_factor_ic`

Require: Requested summary size L_I; fixed stock s and historical interval
Ensure: Ranked IC summary $\mathcal{I}$ and a reference to the full result
1: **if** the full IC result has already been saved **then**
2: **return** its reference with an instruction to reuse the previous result
3: **end if**
4: Obtain factor IC values from the Engine at catalog-default parameters
5: Save the complete result, including unavailable values and warnings
6: Rank available values by descending $|\mathrm{IC}|$
7: Form $\mathcal{I}$ from the first L_I factor names and signed IC values
8: **return** $\mathcal{I}$, the saved-result reference, and summary status

Rule interpretation. The model uses `get_factor_info` to retrieve catalog entries for a short list of atomic-strategy names. Each returned entry contains exactly `name`, `side`, `meaning`, `formula`, `rule`, and `parameters`. At most four names are processed per call. Unlike IC ranking, this tool explains how a factor becomes an executable rule and which parameters the model can adjust. In parameter optimization, it serves the same purpose for the supplied rules without any preceding IC call.

Figures 15 and 16 show two representative tool responses. They are selected to illustrate complementary trading directions and signal mechanisms. `RSIBuyStrategy` is a buy-side threshold rule: `formula` defines the continuous RSI value, `rule` applies the oversold condition to the previous trading day's value, and `parameters` specifies the lookback window and threshold. `MacdDeathCrossSellStrategy` is a sell-side crossover rule: its formula constructs the DIF and DEA sequences, while its rule compares their relative positions at $T-2$ and $T-1$ to determine the sell signal assigned to T. The Engine executes that signal at day T's closing price under the common protocol. Thus, the retrieved fields connect financial interpretation to an executable,

Prompt for AlphaLoop (Strategy Generation)

[SYSTEM PROMPT]
You are a professional quantitative researcher skilled at combining alpha factors into trading strategies, analysing backtest results, and using data evidence to improve strategies.
Core task
Design a strategy consisting of buy and sell rules, then evaluate it over the full historical backtest interval.
Required workflow
1. Obtain factor IC references.
- Call run_factor_ic(top=20) once.
- Use the returned Top-20 list only as a ranking reference.
- Shortlist at most four factors using IC, mechanism, objective, and buy/sell role before requesting factor information.

2. Design a strategy.
- Treat IC as one reference signal, not a mandatory selection rule.
- Combine IC with factor mechanism, the target metric, signal density, and trading direction.
- High positive IC may help prioritize buy rules, but lower-ranked or unavailable-IC factors remain valid.
- Select sell rules by exit mechanism and transaction-path needs; do not rank sell factors mechanically by absolute IC.

Engine semantics:
- Rules within buy_rules are joined by AND.
- Rules within sell_rules are joined by AND.
- More factors produce sparser signals and fewer trades.
- Begin with one or two factors to avoid no-trade strategies.

Example strategy:
{"buy_rules": [{"name": "...", "params": {...}}],
"sell_rules": [{"name": "...", "params": {...}}]}

3. Submit a backtest.
- Call run_backtest; at least one successful backtest is required.
- If execution fails, inspect the error, revise the strategy, and retry.

4. Analyse the latest result.
- Use terminal to inspect trades.csv, equity_curve.csv, and daily_data.csv.
- Diagnose trade frequency, win/loss behavior, drawdown periods, signal density, and buy--sell interactions.
- Decide from the evidence whether to tune parameters, replace factors, add or remove a filter, or redesign the strategy.
- Preserve the best candidate. Near the target, prefer a focused parameter change to an unrelated structural reset.

5. Optimise or finish.
- Compare condition results, raw metrics, and path-level evidence with the query objective.
- If target_met=true, stop immediately.
- Otherwise, submit the most informative next candidate when another experiment is justified.

Available tools
- run_factor_ic: obtain factor IC rankings (Strategy Generation only)
- get_factor_info: retrieve documentation and valid parameters for at most four shortlisted factors
- run_backtest: execute a candidate over the full historical interval
- terminal: analyse the latest candidate's trades, equity curve, and daily signals

Stop conditions and safeguards
- Stop after five successful strategies or five consecutive failures.
- Do not repeat a strategy or continue after target_met=true.
- Use terminal sparingly and only within the workspace.
- The successful-strategy ceiling is a safety limit, not a quota.

...

[USER PROMPT]
Research a valid single-stock, long-only strategy for the following query:
{natural-language Strategy Generation query}

Figure 13: Condensed AlphaLoop prompt for strategy generation. The agent uses an IC prior, factor knowledge, and backtesting evidence to search both rule structure and parameters.

temporally specified rule and expose only the values that the model may need to configure; they contain neither reference-strategy parameters nor backtesting outcomes.

Prompt for AlphaLoop (Parameter Optimization)

[SYSTEM PROMPT]
You are a professional quantitative researcher skilled at analysing backtest results and optimising alpha-factor parameters.
Core task
Adjust the parameters of the buy and sell factors explicitly specified in the query so that the full-interval historical backtest satisfies the user's conditions.
Hard constraints
- Use only the buy and sell factors specified in the query.
- Modify only their params.
- Never replace, add, or remove a factor.
- Never change a factor's buy/sell assignment.
- Every candidate must preserve the specified factor names, counts, and trading directions.
Required workflow
1. Use get_factor_info to inspect the meaning and valid range of each specified parameter.
2. Preserve the factor structure and call run_backtest with a new parameter configuration.
3. After a successful backtest, use terminal to analyse trades.csv, equity_curve.csv, and daily_data.csv. Use the candidate history and observed parameter--metric response to select the next trial.
4. Compare the raw metrics with the query objective and decide whether to continue or finish.
Search guidance
- Change multiple parameters when supported by evidence.
- Prefer interpretable changes within documented valid ranges.
- Use earlier candidates and path-level evidence to avoid unproductive directions.
- Never submit a duplicate candidate.
- Rules on each side are joined by AND.
Available tools
- get_factor_info: inspect the specified factors and parameter ranges
- run_backtest: evaluate a parameter configuration while preserving structure
- terminal: analyse the latest candidate's trades, equity curve, and daily signals
Do not call run_factor_ic. Factor identities are fixed, so IC ranking is outside this workflow.

Stop conditions and safeguards
- At least one successful backtest is required.
- Stop after five successful strategies or five consecutive failures.
- Every candidate and the final strategy must preserve the user-specified factor structure.
...

[USER PROMPT]
Research a valid single-stock, long-only strategy for the following query:
{natural-language Parameter Optimization query with fixed buy/sell factors}

Figure 14: Condensed AlphaLoop prompt for parameter optimization. The agent preserves the supplied rule structure and uses backtesting evidence to revise parameters only.

Retrieved Factor Knowledge for RSIBuyStrategy

"name": "RSIBuyStrategy",
"side": "buy",
"meaning": "Buy when the T-1 day's N-day RSI is below the threshold (oversold condition). RSI is computed using Wilder's smoothed moving average (approximated via EWM) of gains and losses.",
"formula": "RSI = 100 - 100 / (1 + RS), RS = AvgGain / AvgLoss; Signal_T = RSI_{T-1} < threshold",
"rule": "Generate a buy signal at the T-day open when the T-1 RSI is below the oversold threshold.",
"params": {"N": {"required": false, "type": "int", "default": 14}, "threshold": {"required": false, "type": "float", "default": 30.0}}

Figure 15: Factor knowledge returned for `RSIBuyStrategy`. The entry describes a buy-side threshold rule in which the signal at day T depends on the previous trading day's RSI.

Using q, the retrieved knowledge, and $\mathcal{I}$ when available, the LLM constructs a complete candidate π. Same-side rules are conjunctive, so the model must balance selectivity against signal sparsity. Rule retrieval can recur during later revisions; it is not restricted to initialization.

Retrieved Factor Knowledge for MacdDeathCrossSellStrategy

"name": "MacdDeathCrossSellStrategy",
"side": "sell",
"meaning": "A risk-control and trend-reversal factor designed to capture the breakdown of bullish momentum. A 'Death Cross' occurs when the reactive MACD line (DIF) drops below the smoothed signal line (DEA). This crossover signifies that recent downward price velocity has overpowered the prevailing upward trend, acting as a critical liquidation or stop-loss trigger.",
"formula": "DIF = EMA(Close, short) - EMA(Close, long); DEA = EMA(DIF, signal); Signal = (DIF[T-2] > DEA[T-2]) AND (DIF[T-1] < DEA[T-1])",
"rule": "A sell signal is assigned to day T when the MACD line (DIF) calculated for T-1 crosses strictly below the signal line (DEA), after being strictly above it at T-2. If a position is open, the Engine executes the sell at day T's closing price under the common execution protocol.",
"params": {"short": {"required": false, "type": "int", "default": 12}, "long": {"required": false, "type": "int", "default": 26}, "signal": {"required": false, "type": "int", "default": 9}}

Figure 16: Factor knowledge returned for `MacdDeathCrossSellStrategy`. The entry describes a sell-side crossover rule based on the lagged relative positions of the DIF and DEA sequences.

Algorithm 2 Atomic-strategy interpretation with `get_factor_info`

Require: Ordered list of requested atomic-strategy names; validated catalog
Ensure: Compact rule descriptions and any unprocessed names
1: Select at most the first four requested names
2: Retrieve their definitions from the catalog
3: Retain only `name`, `side`, `meaning`, `formula`, `rule`, and `parameters`
4: **return** the compact entries and any names omitted by the per-call limit

E.4 Backtesting and Goal Verification

The `run_backtest` tool takes the candidate's buy and sell rules as its explicit input. Stock, interval, and initial capital are read from the query, while transaction settings and $\mathcal{C}$ remain fixed for programmatic evaluation. The tool first checks rule validity and, for parameter optimization, preservation of the supplied structure. A valid candidate is then executed chronologically by the Engine.

Let k count successfully completed candidate backtests and π_k denote the corresponding candidate. Its outputs comprise the metric vector $\mathcal{M}(\pi_k, s)$, a report for each condition in $\mathcal{C}$, and detailed records $\mathcal{X}_k$ containing trades, portfolio values, daily returns, and factor and signal traces. Each condition report gives the metric, comparison operator, threshold, observed value, and pass status, with a reason when the condition fails. Overall success is determined by

$$\mathrm{Pass}(\pi_k, s; \mathcal{C}) = \prod_{j=1}^{|\mathcal{C}|} \mathbb{I}[c_j(\mathcal{M}(\pi_k, s))] \,. \tag{15}$$

Here c_j applies the stored comparison to the relevant metric; an unavailable metric fails its condition. Absolute-beta conditions use $|\mathrm{Beta}|$, and equality comparisons use the implementation's numerical tolerance. Neither the metric values nor the pass decision depend on LLM judgment.

An execution failure is distinct from an unmet goal. Invalid or unexecutable candidates return correction feedback and do not consume the valid-backtest budget. A completed backtest consumes one unit even if its strategy fails every metric condition. When all conditions hold, the search stops without further diagnosis.

E.5 Evidence-Guided Revision

If the goal remains unmet and search continues, the model can use `terminal` to investigate $\mathcal{X}_k$ from the latest completed backtest. Its input is an analysis command specifying which evidence to inspect or summarize; its output is the resulting bounded text. The available records include `trades.csv`, `equity_curve.csv`, and `daily_data.csv`. Access is confined to the latest

Algorithm 3 Candidate evaluation with `run_backtest`

Require: Candidate π; fixed task settings and conditions $\mathcal{C}$; completed count $k < B$
Ensure: Metrics, condition reports, overall pass status, and references to detailed evidence
Check nonempty buy and sell rules, registered names, and valid parameters
For parameter optimization, also check the fixed rule names, counts, ordering, and sides
if validation fails **then**
    return a correction message without advancing k
end if
Execute π with the Engine under the fixed task settings, retaining detailed outputs
if Engine execution fails **then**
    return a correction message without advancing k
end if
Set $k \leftarrow k + 1$ and retain the executed candidate as π_k
Save π_k, $\mathcal{M}(\pi_k, s)$, and detailed records $\mathcal{X}_k$
Check every condition in $\mathcal{C}$ and compute $\mathrm{Pass}(\pi_k, s; \mathcal{C})$
Append the candidate, metrics, and condition reports to the search history
return the evaluation summary, evidence references, and a stop or revision instruction

Algorithm 4 Backtesting-evidence analysis with `terminal`

Require: Analysis command; latest detailed records $\mathcal{X}_k$; remaining analysis-call allowance
Ensure: Selected evidence for the next strategy revision
if no completed backtest exists or the analysis-call limit is reached **then**
    return an availability or limit message
end if
Identify the permitted result or analysis directory for π_k
Run the requested analysis within that restricted directory
return bounded analysis output and any execution error

candidate's result directory or its prepared analysis directory, not the full task workspace or factor catalog. Before any successful backtest, the tool returns an instruction to obtain backtesting evidence first.

The resulting evidence links metric shortfalls to strategy behavior: signal frequency can reveal overly restrictive filters, trade outcomes can expose weak entries or exits, and portfolio paths can locate drawdown episodes. The LLM combines these observations with q, the factor documentation, and the retained candidate history to propose π_{k+1}. strategy generation permits both structural and parameter revisions; parameter optimization permits only the latter. The new candidate returns to `run_backtest`, closing the feedback loop. Auxiliary calculations guide revision but do not replace the Engine's official metrics.

Compact tool responses and candidate summaries remain in the conversation, while full documentation and detailed records are stored separately for selective inspection. Thus, prior candidates remain available as contextual evidence even though terminal access follows the latest candidate. Search history records completed experiments, not a requirement to improve monotonically at each step.

E.6 Search and Tool Limits

The main experiments use the following per-task limits. These limits distinguish the valid-backtest budget, which determines the number of evaluated candidates, from bounds on auxiliary tool use.

- **Valid backtests.** At most five successfully completed backtests are permitted. A completed backtest consumes one unit whether or not its candidate satisfies any target condition. Invalid submissions and Engine execution failures do not consume this budget, while a candidate satisfying all conditions terminates the search immediately.
- **Total tool calls.** Under the default five-backtest setting, a task permits at most 30 calls across `run_factor_ic`, `get_factor_info`, `run_backtest`, and `terminal`. In the budget-scaling experiment, the limit for a valid-backtest budget B is $\max(30, 4B)$, giving

30, 30, 40, 60, 80, 100, and 120 calls for $B = 3, 5, 10, 15, 20, 25$, and 30, respectively. Unsuccessful submissions and auxiliary calls count toward this total even when they do not consume the valid-backtest budget.

- **Factor information.** In strategy generation, the prompt requests one `run_factor_ic` call with the top 20 results; the complete IC result is then cached, and any repeated request returns the saved result rather than recomputing it. Each `get_factor_info` call processes at most four shortlisted names. Parameter optimization does not expose `run_factor_ic` because its factor structure is fixed.
- **Trajectory analysis.** The terminal becomes available only after a valid backtest and exposes evidence for the latest candidate. At most eight terminal calls are allowed in each research round. Each call runs for at most 30 seconds and returns at most 8 KB of output; network access and access outside the designated evidence directory are disabled.
- **Failure control.** Five consecutive failed backtest submissions terminate a task in the main experiments. This counter covers invalid or Engine-rejected submissions and is reset whenever a backtest completes successfully.

To verify that terminal access was not used as an alternative path for candidate evaluation, we randomly sampled 100 terminal commands issued by GPT-5.6-Sol in the main AlphaLoop experiment and manually inspected them. None invoked the Engine or performed an additional candidate backtest; the sampled commands were limited to analyzing files produced by an existing valid backtest.

The five-backtest ceiling is the common experimental search budget used to compare methods. The remaining limits prevent unbounded documentation queries, failed submissions, or evidence analysis; they neither change the target conditions nor grant additional candidate evaluations.

E.7 Stopping and Final Output

Search ends when a candidate satisfies all conditions, the LLM elects to stop, or one of the limits above is reached; a model-call failure can also end the run. The valid-backtest ceiling is a maximum rather than a required quota, so the LLM may stop before using all five evaluations when it judges further experiments unjustified.

If $k \geq 1$ candidates have completed backtesting when search ends, the implementation submits the latest candidate, $\pi_{\text{final}} = \pi_k$, rather than asking the LLM to select a best-so-far candidate. The Engine re-executes this strategy under the same task settings without requesting detailed outputs. The reported result is

$$(\pi_{\text{final}},\ \mathcal{M}(\pi_{\text{final}}, s), \text{Pass}(\pi_{\text{final}}, s; \mathcal{C}))\,, \tag{16}$$

with the metrics and condition checks recomputed programmatically. This final verification is separate from the B candidate-search backtests and introduces no new candidate. A run with no successfully backtested candidate, or with a failed final execution, is not counted as passing. Final submission is therefore a programmatic step, not an additional research tool.

For reproducibility, each task retains its candidate strategies, backtesting results, tool interactions, and stopping reason. Model calls, token usage when available, runtime, and failures are also recorded for cost analysis and auditing; these records do not affect the strategy's pass status.

F Baseline Implementation Details

Random Search. For strategy generation, each candidate contains one buy and one sell atomic strategy drawn independently from the corresponding audited libraries. Searchable integer, floating-point, logarithmic, and categorical parameters are sampled from the same finite legal domains used to construct the empirical strategy pool, while non-searchable parameters remain at their catalog defaults. For parameter optimization, the first candidate uses default parameters and subsequent candidates sample from these construction-aligned domains without changing the supplied rule names, counts, order, or sides. Search stops when all conditions are satisfied or the valid-backtest budget is exhausted.

Vanilla. Vanilla uses the same underlying LLM and may query atomic-strategy documentation and call the Engine. The agent retains every tested strategy and its 12 aggregate metrics, but cannot

Table 8: Information available to each method during candidate proposal. "SG" denotes strategy generation.

Method	LLM	Docs	IC	Metrics	Paths
Random Search	×	×	×	×	×
Vanilla	✓	✓	×	✓	×
AlphaLoop	✓	✓	✓ (SG)	✓	✓

Table 9: Strategy generation pass rate (%) over three runs. Values are mean ± sample standard deviation.

Model/Method	Setting	Easy	Medium	Hard	All
Random Search	$K_{\max} = 2$	40.26 ± 0.27	21.18 ± 1.28	6.63 ± 0.88	20.81 ± 0.52
	$K_{\max} = 3$	36.14 ± 1.45	17.60 ± 1.90	5.63 ± 0.56	18.09 ± 0.60
	$K_{\max} = 4$	39.12 ± 1.61	18.64 ± 1.06	4.88 ± 0.68	18.97 ± 0.53
	$K_{\max} = 5$	22.19 ± 1.06	13.42 ± 1.02	4.32 ± 0.57	12.30 ± 0.48
Qwen3.8-27B	Vanilla	40.88 ± 2.27	25.80 ± 1.35	11.13 ± 1.42	24.26 ± 0.55
	AlphaLoop	54.79 ± 0.67	31.76 ± 0.45	20.23 ± 1.32	33.68 ± 0.49
Gemma4-31B	Vanilla	49.14 ± 0.93	31.39 ± 1.79	13.45 ± 1.03	29.36 ± 1.18
	AlphaLoop	52.89 ± 2.67	29.98 ± 0.81	18.45 ± 0.58	31.86 ± 1.18

access factor IC, per-condition results, exposed target status, trade records, portfolio-value series, daily signal data, terminal tools, or workspace files. Detailed outputs may be retained by the runtime for auditing but are not shown to the agent. Only successful Engine executions consume the five-candidate budget, and parameter optimization candidates are programmatically restricted to parameter changes. A programmatic pass check, hidden from the model, controls early stopping; if no candidate passes, a final tool-free LLM call selects one previously tested candidate from its strategy and aggregate-metric history.

G Supplementary Experimental Results

G.1 Repeated-Run Analysis and Rule-Count Selection

For each $K_{\max}$, Random Search samples the total rule count uniformly from $\{2, \ldots, K_{\max}\}$ and allocates it between nonempty buy and sell sides. Table 9 reports three-run means and sample standard deviations for rule-count selection and selected LLM configurations under the five-backtest budget.

$K_{\max} = 2$ has the highest mean pass rate in every difficulty tier and is therefore used in the main and budget comparisons. Larger rule counts expand the mixed discrete–continuous space, while same-side conjunction can reduce signal frequency. Across repeated LLM runs, AlphaLoop improves SG-All by 9.42 points for Qwen3.8-27B and 2.50 for Gemma4-31B; the latter declines slightly on medium tasks.

G.2 Model Deployment

Four evaluation models are self-hosted on NVIDIA A800-SXM4-80GB GPUs behind OpenAI-compatible endpoints. As summarized in Table 10, Qwen3.8-27B, Gemma-4-31B-it, and Hy3 use SGLang[1] with tensor parallelism matching their GPU counts, while DeepSeek-V4-Flash-0731 uses distributed vLLM[2] serving. GLM-5.3-Flash is accessed through the official BigModel API,[3] and GPT-5.6-Sol through OpenRouter;[4] their serving hardware is not controlled by us.

[1] https://docs.sglang.io/
[2] https://docs.vllm.ai/
[3] https://open.bigmodel.cn/api/paas/v4
[4] https://openrouter.ai/

Table 10: Deployment of the self-hosted evaluation models.

Model	Serving framework	GPUs
Qwen3.8-27B	SGLang	4
Gemma-4-31B-it	SGLang	4
Hy3	SGLang	16
DeepSeek-V4-Flash-0731	vLLM	8

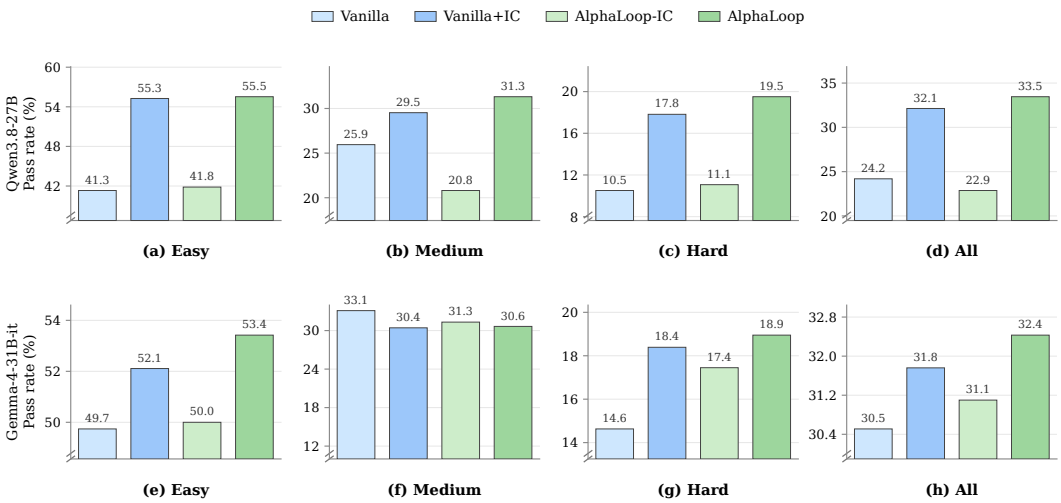


Figure 17: Strategy-generation pass rates for configurations with and without IC guidance and detailed backtesting feedback. The first row reports Qwen3.8-27B and the second Gemma4-31B; columns separate easy, medium, hard, and all tasks. Bar labels give absolute pass rates. Each panel uses an independently truncated vertical axis, marked by an axis break, to make within-panel differences visible.

G.3 Separating IC Guidance from Detailed Backtesting Feedback

The main IC ablation removes IC from AlphaLoop while retaining its detailed feedback loop. To isolate the other direction, we additionally evaluate Vanilla+IC on Qwen3.8-27B and Gemma4-31B. Vanilla+IC receives the same retrospective factor ranking and associated factor-selection guidance as AlphaLoop, but, like Vanilla, each successful backtest exposes only the 12 aggregate metrics. It has no condition-level feedback, exposed target status, terminal, trade records, portfolio paths, or daily signals. Together, Vanilla, Vanilla+IC, AlphaLoop-IC, and AlphaLoop form a controlled comparison of IC guidance and detailed backtesting feedback.

Figure 17 shows a clear pattern for Qwen3.8-27B. Adding IC to Vanilla raises the overall pass rate from 24.19% to 32.13%, while restoring IC to AlphaLoop raises it from 22.87% to 33.46%. The improvement also appears at every difficulty level. Holding IC access fixed, AlphaLoop changes the overall rate by -1.32 points without IC and $+1.33$ points with IC. Thus, for Qwen3.8-27B, most of AlphaLoop's advantage is associated with IC-guided structural selection; detailed feedback provides a smaller additional gain once that prior is available.

Gemma4-31B exhibits smaller component effects. IC improves the easy, hard, and aggregate results under both feedback settings, and the complete AlphaLoop configuration obtains the highest overall rate, 32.43%. Medium tasks are the exception: Vanilla is highest at 33.11%, while Vanilla+IC, AlphaLoop-IC, and AlphaLoop reach 30.43%, 31.32%, and 30.65%, respectively. We do not interpret this isolated reversal as evidence that IC is generally harmful for Gemma4-31B. The same model benefits from IC on the other tiers, and the four aggregate rates lie within a narrow 1.92-point range. Rather, the result suggests a model-specific interaction between IC-guided factor selection and the medium-tier task subset. IC measures next-day factor association rather than alignment with

Table 11: Pass rates by task type. Parenthesized values are average numbers of valid candidate backtests per task.

strategy generation				
Model	**Method**	**Single**	**Dual**	**All**
Gemma-4-31B-it	Vanilla	37.77 (3.97)	16.91 (4.08)	30.51 (4.01)
	AlphaLoop	42.39 (3.92)	13.74 (4.68)	32.43 (4.19)
Qwen3.8-27B	Vanilla	30.10 (3.61)	13.11 (4.13)	24.19 (3.79)
	AlphaLoop	43.07 (3.89)	15.43 (4.65)	33.46 (4.16)
Hy3	Vanilla	33.93 (3.98)	15.43 (4.36)	27.50 (4.11)
	AlphaLoop	40.25 (3.76)	12.26 (4.51)	30.51 (4.02)
DeepSeek-V4-Flash-0731	Vanilla	35.96 (4.12)	13.11 (4.54)	28.01 (4.26)
	AlphaLoop	34.50 (3.20)	13.32 (3.96)	27.13 (3.47)
GLM-5.3-Flash	Vanilla	21.08 (4.23)	12.47 (4.43)	18.09 (4.30)
	AlphaLoop	38.11 (4.09)	14.38 (4.66)	29.85 (4.28)
GPT-5.6-Sol	Vanilla	34.05 (4.18)	16.49 (4.63)	27.94 (4.34)
	AlphaLoop	44.42 (3.93)	18.60 (4.60)	35.44 (4.16)
parameter optimization				
Model	**Method**	**Single**	**Dual**	**All**
Gemma-4-31B-it	Vanilla	58.01 (3.31)	40.37 (3.56)	53.54 (3.37)
	AlphaLoop	60.65 (3.41)	45.41 (3.82)	56.79 (3.51)
Qwen3.8-27B	Vanilla	60.03 (3.31)	45.41 (3.72)	56.33 (3.41)
	AlphaLoop	57.39 (3.44)	45.87 (3.78)	54.47 (3.52)
Hy3	Vanilla	48.99 (3.56)	38.99 (3.75)	46.46 (3.61)
	AlphaLoop	53.65 (3.51)	46.33 (3.44)	51.80 (3.49)
DeepSeek-V4-Flash-0731	Vanilla	39.35 (3.14)	26.61 (3.16)	36.12 (3.15)
	AlphaLoop	55.99 (3.27)	41.74 (3.02)	52.38 (3.21)
GLM-5.3-Flash	Vanilla	28.93 (3.86)	21.10 (3.87)	26.95 (3.86)
	AlphaLoop	60.19 (3.38)	48.62 (3.65)	57.26 (3.45)
GPT-5.6-Sol	Vanilla	56.77 (3.43)	43.58 (3.73)	53.43 (3.51)
	AlphaLoop	69.52 (3.16)	56.42 (3.43)	66.20 (3.23)

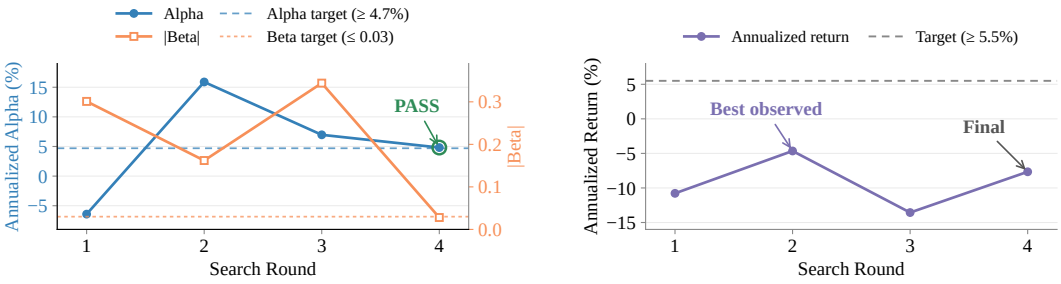
a complete strategy's task constraints; a model that anchors strongly on the ranking may narrow structural exploration even when another factor combination better satisfies a particular goal.

This interpretation is consistent with, but not established by, the repeated-run results in Table 9: Gemma4-31B also shows a small average AlphaLoop decline relative to Vanilla on medium tasks, while improving on easy, hard, and aggregate results. Because Vanilla+IC and AlphaLoop-IC are each run once per task, stochastic variation and unmeasured interactions with the composition of the medium-tier subset cannot be ruled out. The evidence therefore supports a model- and tier-dependent effect rather than a universal monotonic benefit from either component.

More generally, IC and detailed feedback can be complementary, but neither produces uniform gains. IC helps select a structural hypothesis, whereas trade, equity, and signal evidence primarily supports local refinement. Without a useful structural prior, detailed feedback can anchor the model to a weak initial candidate under the five-backtest budget; with IC, that same feedback can refine a more promising structure. The balance depends on how the underlying model uses both sources of evidence.

G.4 Results by Task Cardinality

Table 11 separates single-metric and dual-metric tasks. Values in parentheses are average valid candidate backtest counts, not uncertainty estimates.



(a) Successful Joint-Goal Search **(b) Unsuccessful Return Search**

Figure 18: AlphaLoop search trajectories for (a) the successful joint alpha–beta case and (b) the unsuccessful annualized-return case. Dashed lines mark the requested thresholds.

Table 12: Four-round AlphaLoop trajectory for the joint alpha–beta case. Exposure is the fraction of trading days with an open position.

Round	Buy rule	Sell rule	Alpha (%)	Beta	Trades	Exposure
1	`MikeBaseIndicator`	`MikeBaseIndicator`	−6.42	0.301	21	48.35%
2	`MikeBaseIndicator`	`KUP` (0.03)	15.89	0.162	26	37.41%
3	`LOW0` (0.95)	`KUP` (0.03)	6.97	0.344	64	49.49%
4	`MikeBaseIndicator`	`KUP` (0.01)	4.82	0.028	42	7.69%

Single-metric tasks are consistently easier than dual-metric tasks. In strategy generation, AlphaLoop improves the aggregate result for five of six models, driven mainly by single-metric tasks; dual-metric gains are less consistent because the agent must jointly manage competing constraints and structural search. With structure fixed in parameter optimization, AlphaLoop improves five models overall and dual-metric performance for all six, indicating that detailed feedback is especially useful for parameter refinement under joint metric constraints.

G.5 Case Study: Non-Monotonic Search under Joint Alpha and Beta Constraints

For qualitative analysis, we trace a representative search trajectory of GPT-5.6-Sol at low reasoning effort with a five-backtest budget. We examine a strategy generation task for 600777.SH over the period beginning June 12, 2015, with an initial capital of 100,000. The task requires annualized alpha of at least 4.7% and absolute beta no greater than 0.03. Figure 18(a) shows the non-monotonic trajectory: an initially poor strategy is replaced by a profitable but overly market-exposed candidate, followed by a harmful structural revision, before an exit-rule adjustment satisfies both conditions.

The IC prior assigns relatively large absolute values to the factors in the successful strategy: 0.0754 for the `MikeBaseIndicator` buy factor and −0.0753 for `KUP`. IC nevertheless does not determine a complete solution, because alpha and beta depend strongly on the entry–exit combination and signal threshold.

The first candidate uses `MikeBaseIndicator` on both sides. It performs poorly: annualized return is −13.66%, alpha is −6.42%, and maximum drawdown reaches 92.32%. Its 21 trades have a mean holding period of 97.1 days, and the strategy remains invested on 48.35% of trading days. The candidate therefore fails both target conditions and exhibits prolonged exposure with ineffective exits.

In the second round, AlphaLoop retains the `MikeBaseIndicator` entry but replaces the exit with `KUP` at a threshold of 0.03. Annualized return rises to 16.66%, alpha to 15.89%, and win rate from 33.33% to 61.54%; mean holding time falls to 55.5 days. However, beta remains 0.162, more than five times the permitted value. Improving the return-oriented condition alone is therefore insufficient for this dual-metric goal.

Table 13: Four-round AlphaLoop trajectory for the unsuccessful annualized-return case.

Round	Buy rule	Sell rule	Annual return (%)	Trades
1	KLOW (0.03)	MikeBaseIndicator	−10.78	25
2	DescendingTrendline	MikeBaseIndicator	−4.65	36
3	PiercingLine	MikeBaseIndicator	−13.56	17
4	KLOW2 (0.50)	MikeBaseIndicator	−7.66	30

The third round is a failed revision. Replacing the entry with LOW0 increases the number of trades from 26 to 64 and market exposure from 37.41% to 49.49%. Beta worsens from 0.162 to 0.344, while annualized return falls from 16.66% to 4.18%. Although alpha remains above its threshold, this structural change moves substantially farther from the beta constraint, showing that AlphaLoop's revisions are not monotonically improving.

In the fourth round, the search returns to the MikeBaseIndicator entry and lowers the KUP exit threshold from 0.03 to 0.01. Mean holding time falls to 8.7 days and exposure to 7.69%. Beta decreases to 0.028 while alpha remains 4.82%, satisfying both conditions within four valid backtests and triggering programmatic termination.

This trajectory illustrates that IC provides a useful factor prior but does not specify a viable entry–exit mechanism, while detailed backtesting evidence exposes changes in trading frequency, holding duration, and market exposure. It also reveals a limitation: despite satisfying the requested alpha and beta constraints, the final candidate still has a maximum drawdown of 58.54% and a profit-to-loss ratio of 0.93. AlphaGoal accepts it because these quantities are not requested; goal satisfaction therefore does not imply broad strategy quality or future profitability.

G.6 Case Study: Unsuccessful Search for a Single Return Target

We also examine the unsuccessful trajectory in Figure 18(b). This strategy generation task concerns 300176.SZ over the period beginning January 5, 2015, with an initial capital of 600,000, and requires an annualized return of at least 5.5%. Although the agent may perform up to five valid backtests, it terminates after four without finding a profitable candidate.

The second round improves annualized return by 6.13 percentage points and is the closest of the four candidates to the target. The search nevertheless abandons this entry rule, and both subsequent candidates perform worse. Meanwhile, the sell rule remains unchanged throughout, so the revisions explore alternative entries without testing whether the common exit mechanism contributes to the persistent losses. The agent then stops with one valid backtest still available and returns the latest candidate, whose annualized return is 3.01 points below that of the best candidate observed.

This failure shows that access to IC and detailed backtesting feedback does not guarantee effective use of a limited search budget. The agent neither preserves the most promising search direction nor broadens the strategy structure after repeated losses, leading to premature termination far from the requested return target.

G.7 Experimental Metadata

Table 14 reports the runtime metadata of each method, model, and reasoning-effort configuration. *Avg. BT* is the average number of valid candidate backtests actually executed under the common maximum of five, and *LLM calls* is the average number of model invocations per task. *Input*, *Output*, and *Reason.* report the corresponding average token counts in thousands. A dash indicates unavailable reasoning-token usage in the source data.

Table 14: Experimental configurations and average runtime metadata.

Method	Model	Effort	Avg. BT	LLM calls	Input (k)	Output (k)	Reason. (k)
AlphaLoop	Gemma-4-31B-it	low	3.92	7.75	36.90	0.40	—
	Hy3	no-think	3.94	12.33	78.04	2.36	0
		low	3.81	12.25	77.66	8.37	6.61
		high	3.15	12.68	88.06	17.25	14.99
	Qwen3.8-27B	no-think	3.99	10.16	55.94	0.96	0
		low	3.91	10.44	67.46	7.85	6.30
		medium	3.91	10.54	67.95	8.01	6.44
		xhigh	2.76	14.39	125.27	19.57	16.71
	DS-V4-Flash	none	3.72	13.33	99.49	3.13	—
		low	3.36	13.26	97.73	6.97	—
		high	3.34	13.25	97.89	6.85	—
		max	3.38	13.82	113.23	10.97	—
	GLM-5.3-Flash	low	3.96	9.72	52.31	1.04	0.26
	GPT-5.6-Sol	low	3.80	8.90	47.14	1.76	0.63
Vanilla	Gemma-4-31B-it	low	3.76	7.57	106.66	0.43	—
	Hy3	low	3.92	6.03	51.40	2.89	2.09
	Qwen3.8-27B	low	3.64	8.84	122.95	2.89	2.00
	DS-V4-Flash	low	3.83	8.60	156.57	2.34	—
	GLM-5.3-Flash	low	4.13	9.36	60.87	0.70	0.11
	GPT-5.6-Sol	low	4.02	4.85	22.03	0.88	0.32