

AlphaMemo: Structured Search-Process Memory for Self-Evolving Alpha Mining Agents

Hang Yu

University of Sydney

Zifan Zheng

University of Sydney

Jeff Z. Pan

University of Edinburgh

Tongliang Liu

University of Sydney

Zhiyong Wang

University of Sydney

Fengxiang He

University of Edinburgh

Abstract

LLM agents are promising for alpha mining via combining financial priors, symbolic reasoning, executable factor generation, and feedback-driven refinement. Yet, they face a combinatorial search space, noisy non-stationary feedback, redundant discoveries, and overfitting risks from naively reusing past successes. To address these challenges, we propose AlphaMemo, a self-evolving alpha mining agent with Structured Search-Process Memory. Rather than memorizing only final factors or full trajectories, AlphaMemo records reusable evidence about which edit motifs work or fail under specific parent-factor contexts. It extracts motifs from Abstract Syntax Tree (AST) differences, applies confidence-gated residual memory on top of a search-ledger prior, and uses asymmetric veto control to suppress high-confidence failure patterns. Experiments on CSI 500 and S&P 500 show improved out-of-sample performance and fixed-budget discovery efficiency, with ablations validating the roles of residual learning, confidence gating, AST-diff motifs, and veto memory. Code is at <https://github.com/jarrettyu/AlphaMemo>.

1 Introduction

Alpha factor mining seeks predictive and interpretable signals for future asset returns, typically by representing each formulaic alpha as a symbolic expression over price-volume variables and selecting it into a factor pool through backtesting under quality and diversity constraints. The field has evolved from human-designed anomalies and empirical asset-pricing factors (Fama and French, 1993; Kakushadze, 2016; Gu et al., 2020) to machine-learning predictors and automated symbolic search (Chen et al., 2021; Yu et al., 2023; Shi et al., 2025a; Zhu and Zhu, 2025). This executable-feedback loop makes alpha mining a natural setting for LLM agents, which can use financial priors,

translate hypotheses into formulas, repair invalid expressions, and refine candidates from numerical feedback (Wang et al., 2025; Kou et al., 2025; Li et al., 2024; Tang et al., 2025). More broadly, because self-evolving agents improve by persisting experience across trials (Gao et al., 2025; Fang et al., 2025), alpha-mining agents should accumulate search experience rather than restart each generation from scratch.

However, self-evolving LLM agents face key challenges in alpha mining: symbolic search is combinatorial, feedback is noisy and non-stationary, discoveries are often redundant, and reusing past successes can worsen overfitting. Existing workflows typically store feedback as final-factor scores or unstructured histories, which are too coarse for credit assignment: they do not reveal which local edit caused a child factor to succeed or fail, while full trajectories are costly to retrieve. We therefore argue that memory should target the search process itself, recording which edit motifs help or fail under specific parent-factor contexts. The challenge is to use this process memory as a calibrated correction to the factor-library prior, which already captures quality, redundancy, depth, lineage, and search pressure, rather than as an unconstrained controller.

To address these challenges, we propose AlphaMemo, a self-evolving alpha-mining agent built around *Structured Search-Process Memory* (SSPM). AlphaMemo treats the factor library as an executable search ledger and learns an edit-level residual memory on top of this ledger. For each candidate action, represented by a parent factor and an edit motif, AlphaMemo first scores the parent using the search ledger and then applies a confidence-gated residual correction learned from past parent-context/edit outcomes. When evidence is sparse or unstable, the memory contribution vanishes and the agent falls back to the base search prior. This design makes memory conservative: it can cor-

rect systematic local blind spots, but noisy early observations cannot easily dominate parent-edit selection.

AlphaMemo further introduces an edit representation that matches how LLMs actually modify symbolic formulas. Rather than relying only on manually specified mutation labels, AlphaMemo extracts edit motifs from the abstract syntax tree (AST) difference between a parent expression and its child. This produces a compact symbolic abstraction of LLM-generated edits, enabling credit assignment at the level of reusable search behavior. In addition, AlphaMemo uses an asymmetric process veto: high-confidence negative patterns can veto a candidate action, while positive patterns provide only a soft boost. This asymmetry reflects a practical property of financial search: positive alpha signals are often fragile and regime-dependent, whereas invalid, redundant, or over-mined edit patterns can be more stable failure modes.

We evaluate AlphaMemo on CSI 500 and S&P 500 from Qlib (Yang et al., 2020) under a common 20-trading-day formulaic alpha-mining protocol. The evaluation considers both out-of-sample factor-pool performance and fixed-budget discovery efficiency. Experiments show that AlphaMemo achieves strong predictive and portfolio-level performance compared with representative baselines, and discovers more effective non-redundant factors under the same generation budget. Ablation studies further show that residual learning, confidence gating, AST-diff edit motifs, and asymmetric veto memory are all important: removing these components makes memory less stable, reduces discovery yield, or increases sensitivity to early noisy feedback. Code is at https://github.com/jarrettyu/alpha_memo.

2 Related Work

Self-evolving agents. Self-evolving agents update prompts, tools, memories, policies, or model parameters from interaction feedback (Gao et al., 2025; Fang et al., 2025). Among these targets, memory is appealing because it enables adaptation without parameter updates. Prior agents store reflections (Shinn et al., 2023), use persistent external state for long-horizon interaction (Yao et al., 2023; Park et al., 2023; Packer et al., 2023; Wang et al., 2023), maintain experience buffers or learned memory operations (Liu et al., 2025; Wei et al., 2025; Yan et al., 2025; Yu et al., 2026), and preserve

structured intermediate states in tree search, program search, prompt evolution, or code evolution (Zhou et al., 2024; Antoniadou et al., 2025; Zheng et al., 2025; Guo et al., 2024; Romera-Paredes et al., 2024; Lee et al., 2025; Novikov et al., 2025). These methods show that external state can improve test-time behavior, but also raise the local credit-assignment problem: after a multi-step trajectory, which decision caused success or failure (Zeng et al., 2025).

AlphaMemo studies this issue in symbolic financial search by storing structured edge-level evidence: which edit motif, under which parent-factor context, yields residual gain beyond the base search prior.

LLM agents for alpha mining. Formulaic alpha mining has moved from genetic programming and reinforcement-learning search over symbolic expressions (Zhang et al., 2020; Chen et al., 2021; Yu et al., 2023; Shi et al., 2025a; Zhu and Zhu, 2025; Jiang et al., 2025) to graph-based, generative, and diversity-aware discovery (Li et al., 2025; Zhao et al., 2025; Chen et al., 2025; Ding et al., 2025). LLM-based systems further translate natural-language ideas into executable factors, refine them through feedback, combine generation with evaluation and portfolio construction, or use neural-symbolic and multi-agent workflows (Wang et al., 2025; Kou et al., 2025; Li et al., 2024; Tang et al., 2025; Li et al., 2026). Related refiners use feedback, chain-style reasoning, multi-agent search, or Monte Carlo tree search for local formula improvement (Luo et al., 2025; Shi et al., 2025b). Recent self-evolving alpha-mining systems adapt at different levels, including policy learning (Tang et al., 2026), full hypothesis-expression-code trajectories (Han et al., 2026), and experience memories of successful and forbidden patterns (Wang et al., 2026).

AlphaMemo is complementary: it keeps the interpretable formulaic setting and structured factor-library state, but makes self-evolution an inspectable process memory over parent contexts and AST-diff edit motifs rather than a policy update, terminal-result memory, or full trajectory store.

3 Problem Formulation

Alpha mining. Let \mathcal{X} denote a panel of market features such as open, high, low, close, volume, and derived price-volume variables. A formulaic alpha factor f is an executable symbolic expression that

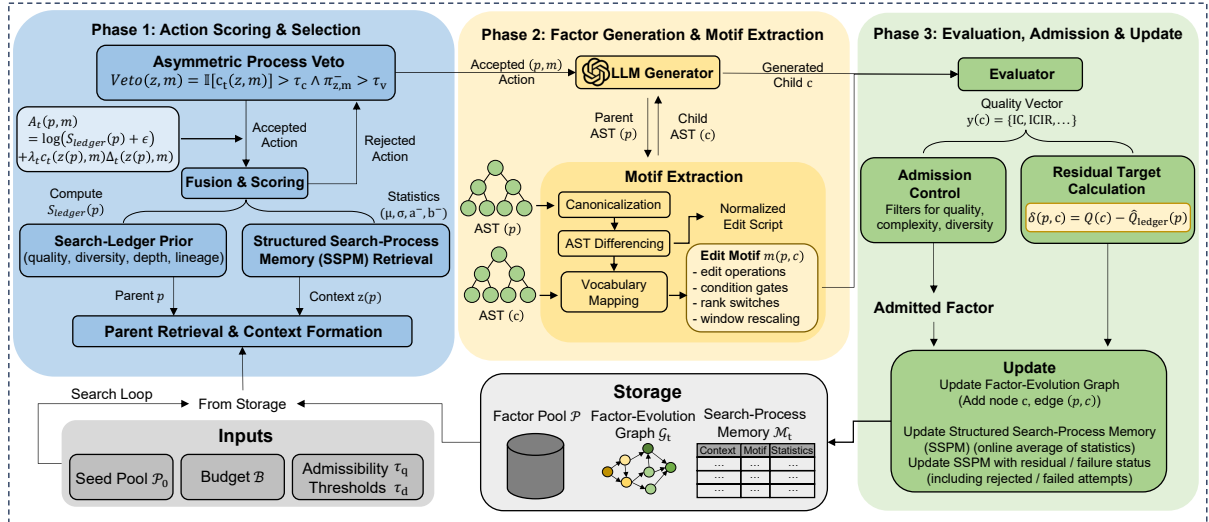


Figure 1: Architecture of AlphaMemo. The framework combines a search-ledger prior with Structured Search-Process Memory (SSPM) over AST-diff edit motifs. Parent-edit actions are scored through confidence-gated residual fusion, filtered by an asymmetric process veto, and updated after evaluation with residual and failure observations.

maps \mathcal{X} to a cross-sectional score $f_{i,t}$ for asset i on date t . Following standard alpha-mining practice, the factor is evaluated against a future return target $r_{i,t+h}$, where h is the prediction horizon. The system searches for a factor pool \mathcal{P} that is both predictive and non-redundant, rather than a single isolated best expression.

Evaluation. Each evaluated factor receives a quality vector

$$y(f) = \{\text{IC}, \text{ICIR}, \text{RankIC}, \text{RankICIR}, \text{AR}, \text{MDD}, \text{SR}\},$$

where IC and RankIC measure Pearson and rank correlations with future returns, ICIR and RankICIR measure temporal stability, and AR, MDD, and SR measure portfolio-level utility. During search, the agent may use training and validation feedback to update its state, but the final test period is held out. We write $Q(f)$ for the scalar quality used by the search algorithm, typically validation or training $|\text{ICIR}|$ under a diversity constraint.

Factor-library search ledger. We model iterative alpha mining as a structured search ledger $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$. Each node $v \in \mathcal{V}_t$ is an evaluated factor. Each directed edge $(p, c) \in \mathcal{E}_t$ records that child factor c was generated by editing parent factor p . A base scorer assigns a parent score $S_{\text{ledger}}(p)$ based on factor quality, diversity, depth, retrieval frequency, and lineage information. This captures a library-state prior: the factor pool is not an unstructured bag of formulas but an auditable record of

evaluated attempts, parent-child edits, and repeated search pressure.

Structured search-process memory. The central object in AlphaMemo is a memory item attached to a factor edge, not only to a factor node. For a generated edge (p, c) , we extract

$$e = (z(p), m(p, c), \delta(p, c), s(p, c)), \quad (1)$$

where $z(p)$ is the parent context, $m(p, c)$ is the edit motif extracted from the parent-child AST difference, $\delta(p, c)$ is the residual outcome beyond the base expectation, and $s(p, c)$ records whether the attempt was invalid, rejected, admitted, or high quality. The memory table aggregates these edge-level observations so that later search can ask: under a similar parent context, which edit motifs have historically produced reliable residual gains?

The parent context is a compact discretization of the state around p :

$$z(p) = (g(p), b_q(p), b_d(p), b_u(p)), \quad (2)$$

where $g(p)$ is the semantic factor category, $b_q(p)$ buckets validation quality, $b_d(p)$ buckets search depth, and $b_u(p)$ buckets retrieval frequency. This keeps memory more specific than a global edit table while avoiding instance-level memorization of individual formulas.

Residual. The residual signal is defined as

$$\delta(p, c) = Q(c) - \widehat{Q}_{\text{ledger}}(p), \quad (3)$$

where $\hat{Q}_{\text{ledger}}(p)$ is the base scorer’s expected child quality for parent p . We estimate this baseline from historical children generated from similar parent contexts:

$$\hat{Q}_{\text{ledger}}(p) = \frac{\sum_{(p_i, c_i) \in \mathcal{H}_t} w_i(p) Q(c_i)}{\sum_{(p_i, c_i) \in \mathcal{H}_t} w_i(p) + \epsilon}, \quad (4)$$

where \mathcal{H}_t is the set of evaluated parent-child edges before iteration t and $w_i(p)$ is nonzero only when p_i falls in the same parent-context bucket as p . If no such history exists, the baseline is initialized by the parent’s own quality $Q(p)$. This zero-centered target is important. The memory does not try to relearn the whole search policy. It only learns when a local edit systematically outperforms or underperforms what the base search prior already expected.

4 AlphaMemo

AlphaMemo is designed around a conservative principle: a self-evolving memory should alter the search policy only when its evidence is reliable. The framework therefore keeps a strong library-state prior for parent selection and adds a calibrated process-memory layer that scores parent-edit actions. At iteration t , the agent maintains a factor-library ledger \mathcal{G}_t , a factor pool \mathcal{P}_t , and a process memory \mathcal{M}_t . It repeatedly selects a parent-edit action, asks the LLM generator to produce a child factor, evaluates the child, updates the ledger, and writes the observed edge into memory. Figure 2 gives the high-level workflow, and Algorithm 1 in Appendix A provides the corresponding pseudocode and derivation details.

4.1 Parent-Edit Action Space

The action space is defined over pairs (p, m) , where p is a parent factor in the search ledger and m is an edit motif. This differs from a standard parent-only retriever. The parent determines where to search in the current library state, while the motif determines how to modify the parent. Examples of high-level motifs include adding a conditional gate, rescaling a temporal window, replacing a ranking operator, adding an interaction term, smoothing a noisy signal, or reversing a sign. In the reported configuration, we use ten motif labels: nine named edit types plus an “other” bucket for rare or ambiguous edits.

For each parent p , we first compute the base search score $S_{\text{ledger}}(p)$ from the ledger state. For each possible edit motif m , we retrieve the memory estimate associated with the context-motif pair

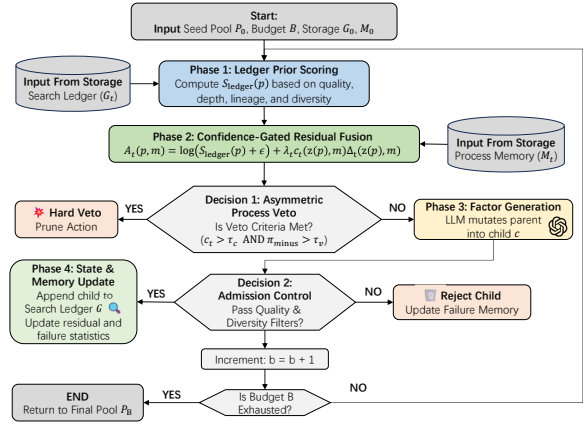


Figure 2: Algorithmic flow of AlphaMemo. At each iteration, AlphaMemo scores parent-edit actions using a search-ledger prior and confidence-gated SSPM residuals, applies asymmetric vetoes to unreliable actions, generates child factors with the LLM, and updates the ledger and memory with evaluation feedback.

$(z(p), m)$. The action score is:

$$A_t(p, m) = \log(S_{\text{ledger}}(p) + \epsilon) + \lambda_t c_t(z(p), m) \Delta_t(z(p), m). \quad (5)$$

Here $\Delta_t(z, m)$ is the posterior mean residual in memory, $c_t(z, m) \in [0, 1]$ is confidence, and λ_t is a scheduled memory weight. The logarithm makes the ledger contribution additive and reduces the chance that memory overwhelms base search evidence through scale mismatch.

4.2 Structured Search-Process Memory

Structured Search-Process Memory (SSPM) is the confidence-gated residual memory that records how local edit motifs behave under parent-factor contexts. For each context-motif pair (z, m) , it stores a compact set of sufficient statistics:

$$\mathcal{M}_t[z, m] = \{n_{z,m}, \mu_{z,m}, \sigma_{z,m}^2, a_{z,m}^-, b_{z,m}^-\}, \quad (6)$$

where n is the number of observations, μ and σ^2 summarize residual outcomes, and (a^-, b^-) parameterize a failure posterior for asymmetric vetoing. When a child c is evaluated, we compute the residual $\delta(p, c)$ from Equation 3 and update the corresponding entry by online averaging.

The confidence gate has two roles: it prevents early noisy observations from affecting search, and it downweights unstable residual estimates. One simple instantiation is:

$$c_t(z, m) = \frac{n_{z,m}}{n_{z,m} + \kappa} \cdot \min\left(1, \frac{|\mu_{z,m}|}{\sigma_{z,m} + \epsilon}\right), \quad (7)$$

where κ controls how many observations are needed before memory becomes active. The warmup gate λ_t further keeps the system close to pure ledger search during early iterations:

$$\lambda_t = \lambda_{\max} \min \left(1, \frac{\max(0, t - t_0)}{T_w} \right). \quad (8)$$

Together, these gates ensure that low-confidence memories make Equation 5 collapse back to the base parent-selection policy.

4.3 Edit Motifs from AST Differences

LLM-generated edits are often not well captured by manually designed mutation labels. For example, a model may simultaneously add a volume gate, rescale a rolling window, and change a normalization operator. AlphaMemo therefore extracts motifs from the AST difference between parent and child expressions. We parse each formula with the same typed operator grammar used by the evaluator. Each AST node records the operator name, arity, argument types, and numerical parameters such as rolling-window length. Before differencing, expressions are canonicalized: commutative operators sort child subtrees by their serialized form, constants and window lengths are mapped to coarse parameter bins, and redundant unary wrappers are normalized. Given canonical trees $\text{AST}(p)$ and $\text{AST}(c)$, we compute a normalized edit script:

$$\text{Diff}(p, c) = \{o_1, o_2, \dots, o_k\}, \quad (9)$$

where each operation o_i records an insertion, deletion, replacement, movement, or parameter change over typed expression nodes. The motif $m(p, c)$ is obtained by mapping the canonical edit signature to a compact vocabulary:

$$m(p, c) = \phi(\text{Diff}(p, c)). \quad (10)$$

The current vocabulary covers condition gates, rank switches, interaction additions, window rescaling, operator substitutions, feature swaps, nesting increases, temporal shifts, normalization changes, and other edits. These labels are not hand-written mutation commands for the generator. They are normalized names for recurring edit signatures observed after generation. This representation provides two advantages. First, it assigns credit to the actual symbolic change that the LLM produced. Second, it can transfer across surface-level expression variants that share the same edit behavior.

4.4 Asymmetric Process Veto (APV)

Positive alpha signals are fragile, but negative search evidence can be more stable: invalid expression patterns, over-mined transformations, and high-correlation edits often remain bad across nearby contexts. AlphaMemo therefore uses memory asymmetrically. Positive residual memories only enter as a soft additive term in Equation 5. Negative high-confidence patterns can veto an action before generation or admission. Let $\pi_{z,m}^-$ be the posterior probability that motif m fails under context z . The veto rule $\text{Veto}(z, m)$ equals

$$\mathbb{I} [c_t(z, m) > \tau_c \wedge \pi_{z,m}^- > \tau_v]. \quad (11)$$

If the action is vetoed, the scheduler chooses the next best non-vetoed action. This design deliberately avoids hard-accepting positive memories, reducing the risk that a lucky early pattern dominates the search.

4.5 Search Loop

The full loop is:

1. Score candidate parent-edit actions (p, m) by Equation 5, excluding actions vetoed by Equation 11.
2. Prompt the LLM generator with the selected parent, its lineage trace, the intended motif, and relevant constraints.
3. Parse and validate generated child expressions. Invalid attempts are written as failure observations.
4. Evaluate valid children on the training and validation periods, then admit those that satisfy quality, complexity, and diversity constraints.
5. Add all evaluated children to the search ledger with parent-child edges, and update process memory with the observed residual and status.

This loop makes the agent self-evolving through an external, inspectable state: the ledger records where the agent has searched, and the process memory records how local edits behaved in context.

5 Mechanistic Analysis

This section analyzes the mechanics.

5.1 Residual Memory as Teacher Correction

A naive memory prior can be harmful when paired with a strong search prior. If memory directly

predicts absolute factor quality, then early high-variance observations can dominate parent selection and pull the search toward lucky local regions. Residual memory avoids this failure mode by learning only the difference between the observed child quality and the base prior’s expected child quality. Thus, a positive memory entry does not mean “this motif is good” in isolation. It means “this motif tends to outperform what the base search policy already expected in this context.”

This distinction is important for alpha mining because the base scorer already encodes strong information about factor quality, diversity, depth, and lineage. The memory should explain the remaining local edit effect, not duplicate the global search prior. When $\Delta_t(z, m)$ is centered around zero, it can correct systematic blind spots while preserving the base search policy as the default behavior.

5.2 Bounded Deviation from the Teacher

The confidence-gated residual form provides a simple stability property. Assume memory residuals are clipped to $|\Delta_t(z, m)| \leq B$, with $\lambda_t \leq \lambda_{\max}$ and $c_t(z, m) \in [0, 1]$. Then the memory perturbation in Equation 5 is bounded:

$$|A_t(p, m) - \log(S_{\text{ledger}}(p) + \epsilon)| \leq \lambda_{\max} B.$$

Moreover, during warmup or under insufficient evidence, $\lambda_t c_t(z, m) \rightarrow 0$, so action ranking converges to the base parent-selection policy. This is the core safety valve in AlphaMemo: memory cannot silently replace the search prior before it has accumulated enough evidence.

5.3 Asymmetric Process Veto

In non-stationary financial data, positive patterns are often regime-dependent, while many negative patterns are structural. For example, syntactically invalid transformations, excessive expression complexity, high-correlation edits, and repeated overfitting motifs are likely to remain bad even when market regimes change. The asymmetric veto exploits this difference. It lets negative memory act as a hard constraint only when confidence is high, while positive memory remains a soft preference. This turns process memory into a variance stabilizer rather than a mechanism for aggressive exploitation.

5.4 Interpretability of Self-Evolution

Parametric self-evolution can improve generation behavior, but its learned search knowledge is dif-

ficult to inspect or edit. Full-trajectory memory is inspectable, but expensive to store and retrieve. AlphaMemo exposes a middle ground. Its memory table can be inspected as statements of the form:

Under parent context z , edit motif m has mean residual Δ , confidence c , and failure posterior π^- .

This makes the agent’s self-evolution auditable: one can identify which edit behaviors are being promoted, suppressed, or vetoed.

6 Experiments

We evaluate AlphaMemo from two complementary angles: final factor-pool quality and fixed-budget search behavior. This separates terminal predictive utility from whether calibrated process memory improves discovery beyond the search ledger alone. Final-pool evaluation asks whether selected factors remain predictive after portfolio construction, while search diagnostics ask whether the agent discovers more admissible, non-redundant factors under the same generation budget.

6.1 Experimental Setup

Datasets and splits. The main reported universes are CSI 500 and S&P 500 from Qlib (Yang et al., 2020), covering Chinese A-shares and U.S. large caps with different liquidity, sector composition, and factor stationarity. This two-market setting keeps predictive and portfolio metrics compact while testing memory calibration beyond a single market. Following recent formulaic alpha-mining protocols, we use a 20-trading-day prediction setting with training from 2016-01-01 to 2020-12-31,

Table 1: Component roles in AlphaMemo.

Component	Role
Search-ledger scorer	Provides parent scores from quality, depth, retrieval frequency, lineage, and diversity.
Residual SSPM	Stores context-edit residual statistics beyond the base-score baseline.
AST edit motifs	Assign credit to the symbolic edit actually made from parent to child.
Confidence gates	Suppress low-count or high-variance memories during early search.
APV	Vetoes high-confidence failure patterns while keeping positive memory soft.

Table 2: Performance comparison on CSI 500 and S&P 500 under the 20-trading-day protocol. AR and MDD are reported in percentage points. Best/second-best results are bolded/underlined. MDD closer to zero is better.

Method	CSI 500							S&P 500						
	IC	ICIR	RankIC	RankICIR	AR(%)	MDD(%)	Sharpe	IC	ICIR	RankIC	RankICIR	AR(%)	MDD(%)	Sharpe
Alpha158	0.0053	0.0634	0.0115	0.1188	7.70	<u>-24.34</u>	0.4055	0.0155	0.1300	0.0081	0.0611	14.36	-21.86	0.6186
GP	0.0226	0.2404	0.0326	0.3403	6.75	-30.28	0.3197	0.0062	0.0494	-0.0013	-0.0096	14.10	-24.58	0.6534
LightGBM	0.0095	0.1124	-0.0115	-0.1085	8.85	-33.83	0.4028	0.0133	0.1009	0.0046	0.0377	15.46	-26.67	0.6040
LSTM	0.0222	0.2384	0.0096	0.0939	<u>9.95</u>	-40.23	0.4047	0.0138	0.0853	0.0068	0.0412	16.36	-28.84	0.5792
AlphaGen	<u>0.0311</u>	0.2988	<u>0.0436</u>	<u>0.4156</u>	8.03	-30.99	0.4040	<u>0.0348</u>	0.3569	0.0101	0.1153	<u>19.44</u>	-24.22	<u>0.9471</u>
AlphaGPT	0.0077	0.0909	0.0011	0.0118	8.08	-30.39	0.3903	0.0163	0.1202	0.0005	0.0038	17.36	-26.14	0.7501
AlphaSAGE	0.0031	0.0335	0.0190	0.2341	5.49	-37.12	0.2541	0.0256	0.2187	0.0079	0.0745	14.26	-29.96	0.6040
AlphaAgent	0.0102	0.1150	-0.0156	-0.1437	4.27	-40.16	0.1818	0.0306	0.2569	0.0133	0.1023	19.40	-24.77	0.8077
AlphaMemo (residual)	0.0101	0.1104	0.0165	0.1808	6.97	-26.08	0.3511	0.0410	<u>0.3434</u>	0.0228	0.1984	23.65	-23.62	1.0672
AlphaMemo (balanced)	0.0401	0.3462	0.0496	0.4597	11.63	-23.43	0.6109	0.0288	0.2406	<u>0.0144</u>	<u>0.1207</u>	17.07	<u>-22.54</u>	0.7743

validation from 2021-01-01 to 2021-12-31, and test/backtest from 2022-01-01 to 2025-12-26. All factor-pool evaluation and backtesting are implemented with the same Qlib pipeline. The prediction target is the 20-trading-day close-to-close forward return. The memory and search ledger are updated only with training and validation feedback. The test period is used only for final reporting. Table 1 summarizes the implemented AlphaMemo components used in the main experiments.

Metrics. We report predictive metrics (IC, ICIR, RankIC, and RankICIR), portfolio metrics (AR, MDD, and Sharpe), and fixed-budget discovery efficiency. Portfolio metrics are computed from realized TopK portfolio returns. Effective factors are executable, high-quality, bounded-complexity, and non-redundant expressions. Appendix B gives the full formulation of exact label and metric definitions, all predictive, portfolio, and search-efficiency metrics.

Baselines. We compare with a compact representative set. Alpha158 (Yang et al., 2020) is the expert factor library, and GP (Chen et al., 2021) is a classical symbolic alpha search baseline. LightGBM (Ke et al., 2017) and LSTM (Hochreiter and Schmidhuber, 1997) are standard predictive-model baselines. Unless otherwise stated, they use the same fixed Qlib feature handler so that the comparison changes the learner rather than the input universe. For formula-generation systems, we include AlphaGen (Yu et al., 2023) and AlphaGPT (Wang et al., 2025), an earlier LLM-based interactive alpha-mining system. We further include AlphaSAGE (Chen et al., 2025) and AlphaAgent (Tang et al., 2025) as recent released alpha-mining baselines. Internal AlphaMemo variants are reported only in the sensitivity study.

Implementation details. All formulaic outputs are evaluated with the same split, label, parser, admission rule, and Qlib backtest protocol when executable. AlphaMemo and the Search-Ledger ablation share the same generator interface, parent scorer, operator set, factor-pool capacity, and evaluation API. The ablation disables SSPM and APV. Additional details of baseline and implementation are provided in Appendix C.

We report two AlphaMemo operating points motivated by an adaptation–stability tradeoff. AlphaMemo (residual) gives residual memory a stronger role after a longer warmup, while AlphaMemo (balanced) keeps SSPM as a weak residual prior on top of the search ledger. Both use the same generator, evaluator, admission rule, and memory mechanism. Appendix C lists their exact parameters. Unless otherwise stated, the pool capacity is 50, the factor length threshold is 40, each selected parent proposes five children, $\tau_q = 0.10$ for $|\text{ICIR}|$, and $\tau_d = 0.70$ for maximum absolute Pearson correlation.

6.2 Performance Comparison

Table 2 reports out-of-sample performance on CSI 500 and S&P 500. We evaluate both factor predictive power and strategy-level performance, following recent alpha-mining evaluations while keeping the two-market comparison compact. This is useful because IC-style metrics measure cross-sectional predictive content, whereas AR, MDD, and Sharpe test whether the admitted factor pool remains useful after portfolio construction. The two AlphaMemo rows should therefore be read as operating points of the same SSPM mechanism rather than separate model families.

AlphaMemo (residual) achieves the best S&P 500 IC, RankIC, RankICIR, annualized return, and Sharpe ratio, while AlphaGen remains slightly

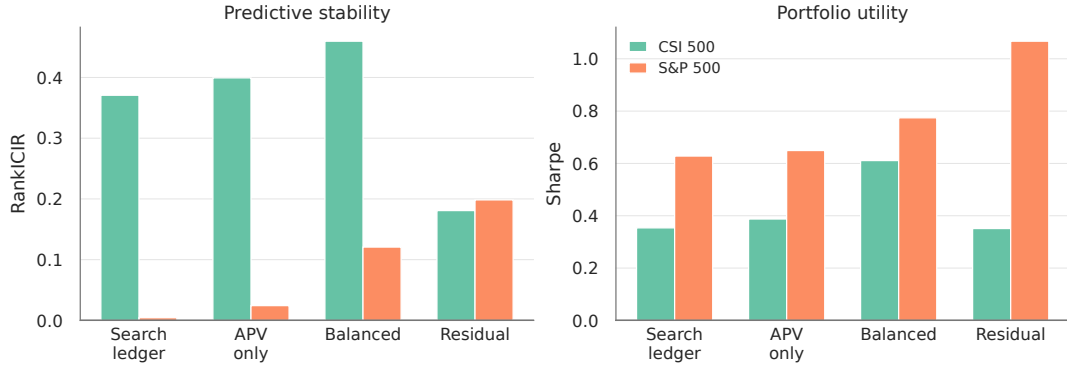


Figure 3: Memory calibration across AlphaMemo operating points and internal diagnostics. Search-ledger only removes process memory. APV-only keeps negative process vetoes. Balanced and residual are the two operating points reported in Table 2. Higher is better for both RankICIR and Sharpe.

Table 3: Sensitivity study for AlphaMemo variants and internal search diagnostics. AR is computed from realized TopK portfolio returns and reported in percentage points.

Variant	CSI500				S&P500			
	ICIR	RankICIR	AR(%)	Sharpe	ICIR	RankICIR	AR(%)	Sharpe
Search-ledger only	0.2313	0.3707	7.00	0.3536	0.1414	0.0044	13.80	0.6282
AlphaMemo (balanced)	0.3462	0.4597	11.63	0.6109	0.2406	0.1207	17.07	0.7743
Weaker memory	0.1455	0.2886	5.80	0.3011	0.1001	-0.0299	17.09	0.7393
Stronger memory	0.2208	0.3454	11.57	0.5646	0.1579	0.0402	16.20	0.7105
APV-only memory	0.2405	0.3993	7.72	0.3875	0.1718	0.0243	15.99	0.6493
AlphaMemo (residual)	0.1104	0.1808	6.97	0.3511	0.3434	0.1984	23.65	1.0672

stronger on ICIR and Alpha158 has the smallest drawdown. On CSI 500, AlphaMemo (balanced) obtains the strongest results across predictive and portfolio metrics, suggesting that weak residual process memory can improve the search-ledger prior without overwhelming it. The two rows therefore expose an adaptation–stability trade-off of the same memory mechanism across markets with different liquidity, sector composition, and factor stationarity.

6.3 Process-Memory Calibration

Table 3 and Figure 3 show that memory strength is a genuine control variable rather than a monotonic knob. The balanced configuration keeps memory weak enough to preserve the search-ledger prior while improving CSI500 predictive and portfolio metrics. The residual configuration performs strongly on S&P500 but is weaker on CSI500, indicating that process memory should be calibrated rather than treated as an unconditional replacement for search. The APV-only variant further separates negative and positive memory: vetoing reliable failure patterns is useful, but it does not match the balanced setting that also uses residual positive evidence. Weaker and stronger memory variants

remain competitive on parts of the table, yet neither gives the same cross-diagnostic balance. These results support exposing calibrated operating points rather than forcing a single unconditioned memory weight. Overall, process memory is useful when treated as a calibrated residual correction, with additional diagnostics reported in Appendix C.

7 Conclusion

We presented AlphaMemo, a self-evolving LLM agent for alpha mining with Structured Search-Process Memory. Instead of reusing final factors or storing full trajectories, AlphaMemo records which edit motifs help or fail under specific parent-factor contexts. It extracts motifs from AST differences, applies confidence-gated residual corrections over a search-ledger prior, and uses asymmetric veto control to suppress reliable failure modes. Experiments on CSI 500 and S&P 500 show improved out-of-sample performance and fixed-budget discovery efficiency, with ablations validating residual learning, confidence gating, AST-diff motifs, and veto memory.

Limitations

AlphaMemo relies on executable formula parsing, standardized evaluation feedback, and enough repeated search observations for process memory to become informative. Its current design is intended for offline alpha discovery rather than direct deployment in live trading systems, where additional operational checks would be required. Future work may extend the memory to richer market states and broader deployment settings, while the present study focuses on isolating how structured process memory improves self-evolving factor search.

Ethics Considerations

All experiments use publicly available datasets and benchmarks. No human subjects or sensitive data are involved. No direct negative societal impacts are identified.

Acknowledgments

HY was supported in part by a JD Technology Research Scholarship in Artificial Intelligence.

References

- Antonios Antoniadis, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. 2025. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement. In *International Conference on Learning Representations*, volume 2025, pages 64485–64515.
- Binqi Chen, Hongjun Ding, Ning Shen, Jinsheng Huang, Taian Guo, Luchen Liu, and Ming Zhang. 2025. Alphasage: Structure-aware alpha mining via gflownets for robust exploration. *arXiv preprint arXiv:2509.25055*.
- Tianxiang Chen, Wei Chen, and Luyao Du. 2021. An empirical study of financial factor mining based on gene expression programming. In *2021 4th International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE)*, pages 1113–1117. IEEE.
- Hongjun Ding, Binqi Chen, Jinsheng Huang, Taian Guo, Zhengyang Mao, Guoyi Shao, Lutong Zou, Luchen Liu, and Ming Zhang. 2025. Alphaeval: A comprehensive and efficient evaluation framework for formula alpha mining. *arXiv preprint arXiv:2508.13174*.
- Eugene F Fama and Kenneth R French. 1993. Common risk factors in the returns on stocks and bonds. *Journal of financial economics*, 33(1):3–56.
- Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, and 1 others. 2025. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*.
- Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, and 1 others. 2025. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*.
- Shihao Gu, Bryan Kelly, and Dacheng Xiu. 2020. Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33(5):2223–2273.
- Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2024. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*, volume 2024, pages 34133–34156.
- Jun Han, Shuo Zhang, Wei Li, Zhi Yang, Yifan Dong, Tu Hu, Jialuo Yuan, Xiaomin Yu, Yumo Zhu, Fangqi Lou, and 1 others. 2026. Quantaalpha: An evolutionary framework for llm-driven alpha mining. *arXiv preprint arXiv:2602.07085*.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.
- Zuoyou Jiang, Li Zhao, Rui Sun, Ruohan Sun, Zhongjian Li, Jing Li, Daxin Jiang, Zuo Bai, and Cheng Hua. 2025. Alpha-r1: Alpha screening with llm reasoning via reinforcement learning. *arXiv preprint arXiv:2512.23515*.
- Zura Kakushadze. 2016. 101 formulaic alphas. *Wilmott*, 2016(84):72–81.
- Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30.
- Zhizhuo Kou, Holam Yu, Junyu Luo, Jingshu Peng, Xujia Li, Chengzhong Liu, Juntao Dai, Lei Chen, Sirui Han, and Yike Guo. 2025. Automate strategy finding with LLM in quant investment. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 18517–18533.
- Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen. 2025. Evolving deeper llm thinking. *arXiv preprint arXiv:2501.09891*.
- Shicheng Li, Jinshan Zhang, and Feng Wang. 2025. Alphagat: A two-stage learning approach for adaptive portfolio selection. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pages 7500–7508.

- Yuante Li, Xu Yang, Xiao Yang, Xisen Wang, Weiqing Liu, and Jiang Bian. 2026. R&d-agent-quant: a multi-agent framework for data-centric factors and model joint optimization. *Advances in Neural Information Processing Systems*, 38.
- Zhiwei Li, Ran Song, Caihong Sun, Wei Xu, Zhengtao Yu, and Ji-Rong Wen. 2024. Can large language models mine interpretable financial factors more effectively? a neural-symbolic factor mining agent model. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3891–3902.
- Yitao Liu, Chenglei Si, Karthik R Narasimhan, and Shunyu Yao. 2025. Contextual experience replay for self-improvement of language agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14179–14198.
- Haochen Luo, Yuan Zhang, and Chen Liu. 2025. Efs: Evolutionary factor searching for sparse portfolio optimization using large language models. *arXiv preprint arXiv:2507.17211*.
- Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, and 1 others. 2025. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*.
- Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. 2023. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*.
- Joon Sung Park, Joseph O’Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST ’23.
- Bernardino Romera-Paredes, Mohammadamin Barekatin, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, and 1 others. 2024. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475.
- Hao Shi, Weili Song, Xinting Zhang, Jiahe Shi, Cuicui Luo, Xiang Ao, Hamid Arian, and Luis Angel Seco. 2025a. Alphaforge: A framework to mine and dynamically combine formulaic alpha factors. In *Proceedings of the AAAI conference on artificial intelligence*, volume 39, pages 12524–12532.
- Yu Shi, Yitong Duan, and Jian Li. 2025b. Navigating the alpha jungle: an llm-powered mcts framework for formulaic factor mining. *arXiv preprint arXiv:2505.11122*.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.
- Ziyi Tang, Zechuan Chen, Jiarui Yang, Jiayao Mai, Yongsen Zheng, Keze Wang, Jinrui Chen, and Liang Lin. 2025. Alphaagent: Llm-driven alpha mining with regularized exploration to counteract alpha decay. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 2813–2822.
- Ziyi Tang, Xuexiong Yin, Weixing Chen, Zechuan Chen, Yongsen Zheng, Wenxuan Ye, Keze Wang, and Liang Lin. 2026. Alphaagentevo: Evolution-oriented alpha mining via self-evolving agentic reinforcement learning. In *International Conference on Learning Representations*.
- Guangzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.
- Saizhuo Wang, Hang Yuan, Leon Zhou, Lionel Ni, Heung Yeung Shum, and Jian Guo. 2025. Alpha-gpt: Human-ai interactive alpha mining for quantitative investment. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 196–206.
- Yanlong Wang, Jian Xu, Hongkang Zhang, Shao-Lun Huang, Danny Dongning Sun, and Xiao-Ping Zhang. 2026. Factorminer: A self-evolving agent with skills and experience memory for financial alpha discovery. *arXiv preprint arXiv:2602.14670*.
- Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H Chi, and 1 others. 2025. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*.
- Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, and 1 others. 2025. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*.
- Xiao Yang, Weiqing Liu, Dong Zhou, Jiang Bian, and Tie-Yan Liu. 2020. Qlib: An ai-oriented quantitative investment platform. *arXiv preprint arXiv:2009.11189*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Shuo Yu, Hongyan Xue, Xiang Ao, Feiyang Pan, Jia He, Dandan Tu, and Qing He. 2023. Generating synergistic formulaic alpha collections via reinforcement learning. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, pages 5476–5486.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. 2026. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. *arXiv preprint arXiv:2601.01885*.

Siliang Zeng, Quan Wei, William Brown, Oana Frunza, Yuriy Nevmyvaka, Yang Katie Zhao, and Mingyi Hong. 2025. Reinforcing multi-turn reasoning in llm agents via turn-level credit assignment. In *ICML 2025 Workshop on Computer Use Agents*.

Tianping Zhang, Yuanqi Li, Yifei Jin, and Jian Li. 2020. Autoalpha: an efficient hierarchical evolutionary algorithm for mining alpha factors in quantitative investment. *arXiv preprint arXiv:2002.08245*.

Junjie Zhao, Chengxi Zhang, Min Qin, and Peng Yang. 2025. Quantfactor reinforce: mining steady formulaic alpha factors with variance-bounded reinforce. *IEEE Transactions on Signal Processing*.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. 2025. Monte carlo tree search for comprehensive exploration in llm-based automatic heuristic design. *arXiv preprint arXiv:2501.08603*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*.

Zhoufan Zhu and Ke Zhu. 2025. Alphaqcm: Alpha discovery in finance with distributional reinforcement learning. In *Forty-second International Conference on Machine Learning*.

A Algorithmic Details and Derivation

A.1 AlphaMemo Search Procedure

Algorithm 1 summarizes the full search loop. The base scorer proposes promising parent factors from the search ledger, while Structured Search-Process Memory (SSPM) supplies a confidence-gated residual correction over edit motifs observed in previous parent-child transitions. The memory module is deliberately residual and confidence gated, so the search falls back to the base scorer when memory evidence is sparse or unstable.

A.2 Residual Process Memory

Let p be a parent formula, m an edit motif, $z(p)$ the parent context, and c the child produced by applying that edit. Its positive counterpart $S_{\text{ledger}}(p)$

Algorithm 1 AlphaMemo factor evolution loop.

Require: seed pool \mathcal{P}_0 , budget B , search ledger \mathcal{G}_0 , memory table \mathcal{M}_0 , thresholds τ_q, τ_d

Ensure: final factor pool \mathcal{P}_B

1: Initialize $\mathcal{P} \leftarrow \mathcal{P}_0$, $\mathcal{G} \leftarrow \mathcal{G}_0$, and $\mathcal{M} \leftarrow \mathcal{M}_0$

2: **for** $b = 1, \dots, B$ **do**

3: Retrieve candidate parent factors p from the search ledger and compute $S_{\text{ledger}}(p)$ using validation quality, depth, retrieval frequency, lineage, and diversity

4: Form parent contexts $z(p)$ and retrieve SSPM statistics for candidate edit motifs m

5: Score each parent-edit action (p, m) by

$$A_t(p, m) = \log(S_{\text{ledger}}(p) + \epsilon) + \lambda_t c_t(z(p), m) \Delta_t(z(p), m)$$

6: Remove actions vetoed by the APV

$$\text{Veto}(z, m) = \mathbb{I}[c_t(z, m) > \tau_c \wedge \pi_{z, m}^- > \tau_v]$$

7: Select the highest-scoring non-vetoed action (p, m)

8: Prompt the LLM generator with parent p , its lineage context, the intended motif m , and generation constraints

9: Parse and validate the generated child c

10: **if** c is invalid **then**

11: Update \mathcal{M} with a failure observation for $(z(p), m)$

12: **continue**

13: **end if**

14: Extract the realized edit motif $m(p, c)$ from the AST difference between p and c

15: Evaluate c on the training/validation periods and compute residual

$$\delta(p, c) = Q(c) - \widehat{Q}_{\text{ledger}}(p)$$

16: Update \mathcal{G} with the evaluated edge (p, c) and update \mathcal{M} with $\delta(p, c)$ and the evaluation status

17: **if** c passes quality, complexity, and diversity filters **then**

18: Admit c into \mathcal{P}

19: **end if**

20: **end for**

21: **return** \mathcal{P}

is the normalized ledger score used in the action scorer. SSPM does not memorize the absolute quality of the child. Instead, it stores the residual effect of the edit beyond the base search prior:

$$\delta(p, c) = Q(c) - \widehat{Q}_{\text{ledger}}(p), \quad (12)$$

where $Q(c)$ is the validation-side utility used by the admission rule. For a memory key $(z(p), m)$, SSPM maintains empirical mean and variance:

$$\mu_{z, m} = \frac{1}{n_{z, m}} \sum_i \delta_i, \quad (13)$$

$$\sigma_{z, m}^2 = \frac{1}{n_{z, m} - 1} \sum_i (\delta_i - \mu_{z, m})^2. \quad (14)$$

The confidence gate follows the main-text definition:

$$c_t(z, m) = \frac{n_{z,m}}{n_{z,m} + \kappa} \cdot \min\left(1, \frac{|\mu_{z,m}|}{\sigma_{z,m} + \epsilon}\right).$$

so early or high-variance memories contribute little to action selection. Let $\Delta_t(z, m) = \mu_{z,m}$ denote the current residual estimate. The final parent-edit score is

$$A_t(p, m) = \log(S_{\text{ledger}}(p) + \epsilon) + \lambda_t c_t(z(p), m) \Delta_t(z(p), m). \quad (15)$$

This form makes memory a bounded correction rather than an alternative controller. When $n_{z,m} < n_{\min}$ or $\sigma_{z,m}$ is large, $c_t(z, m) \approx 0$ and AlphaMemo reduces to ledger-guided evolution.

A.3 Asymmetric Process Veto

Positive memories are used softly through the residual term above, but repeated failures are handled more aggressively. For each context-motif key (z, m) , SSPM also maintains a Beta posterior over failure events with parameters $(a_{z,m}^-, b_{z,m}^-)$:

$$\pi_{z,m}^- = \frac{a_{z,m}^-}{a_{z,m}^- + b_{z,m}^-}. \quad (16)$$

The asymmetric process veto rejects an edit proposal when

$$\text{Veto}(z, m) = \mathbb{I}[c_t(z, m) > \tau_c \wedge \pi_{z,m}^- > \tau_v].$$

This asymmetry reflects a practical property of alpha mining: exploiting a positive motif too strongly can collapse diversity, but avoiding reliable failure motifs saves search budget without forcing all future generations into the same local region.

B Evaluation Metrics

We follow the standard factor-mining convention of reporting both predictive correlations and portfolio-level utility. For a factor f , let $f_{i,t}$ denote the score of asset i at date t , and let $r_{i,t+h}$ be the future holding-period return. The main experiments use $h = 20$ trading days and define the close-to-close forward return as

$$r_{i,t+h} = \frac{C_{i,t+h}}{C_{i,t}} - 1, \quad (17)$$

where $C_{i,t}$ is the adjusted close price. In the Qlib implementation, this corresponds to the next-tradable-close aligned label `Ref($close, -21)` /

`Ref($close, -1) - 1`. The daily Information Coefficient is the cross-sectional Pearson correlation

$$\text{IC}_t(f) = \text{corr}_i(f_{i,t}, r_{i,t+h}).$$

RankIC replaces raw scores and returns with their cross-sectional ranks:

$$\text{RankIC}_t(f) = \text{corr}_i(\text{rank}(f_{i,t}), \text{rank}(r_{i,t+h})).$$

The reported IC and RankIC are temporal means over the test period. Their information ratios measure stability:

$$\text{ICIR}(f) = \frac{\mathbb{E}_t[\text{IC}_t(f)]}{\text{Std}_t[\text{IC}_t(f)] + \epsilon},$$

$$\text{RankICIR}(f) = \frac{\mathbb{E}_t[\text{RankIC}_t(f)]}{\text{Std}_t[\text{RankIC}_t(f)] + \epsilon}.$$

For portfolio metrics, AR is annualized portfolio return, MDD is maximum drawdown, and Sharpe is the annualized return-to-volatility ratio of the backtested strategy. In the main table, AR and MDD are reported in percentage points.

Search efficiency. For fixed-budget diagnostics, we count effective rather than raw generated factors. Given the current admitted pool \mathcal{P} , define the maximum absolute correlation between a candidate f and the existing pool as

$$\rho_{\max}(f, \mathcal{P}) = \max_{g \in \mathcal{P}} |\rho(f, g)|.$$

An effective factor is an executable expression that passes the validation-quality threshold, satisfies the complexity bound, and is not redundant with the current pool:

$$\text{Eff}(f, \mathcal{P}) = \mathbb{I}\{\text{valid}(f) \wedge |\text{ICIR}_{\text{val}}(f)| \geq \tau_q \wedge \ell(f) \leq L \wedge \rho_{\max}(f, \mathcal{P}) \leq \tau_d\}.$$

The fixed-budget discovery yield is

$$N_{\text{eff}}(B) = \sum_{b=1}^B \text{Eff}(f_b, \mathcal{P}_{b-1}),$$

where B is the generation budget.

Alpha decay. To measure whether a factor pool remains useful under non-stationarity, we compute annual IC and RankIC on held-out years. For a factor pool \mathcal{P} , we first form a normalized pool signal

$$s_{i,t} = \frac{1}{|\mathcal{P}|} \sum_{f \in \mathcal{P}} \frac{f_{i,t} - \mu_t(f)}{\sigma_t(f) + \epsilon},$$

where $\mu_t(f)$ and $\sigma_t(f)$ are cross-sectional statistics at date t . Annual alpha decay is then reported as

$$\text{IC}^{(y)}(\mathcal{P}) = \mathbb{E}_{t:\text{year}(t)=y} [\text{corr}_i(s_{i,t}, r_{i,t+h})],$$

with an analogous definition for annual RankIC. This diagnostic asks whether the evolved factor pool retains predictive content across calendar years rather than only on the aggregate test window.

C Additional Experiments

C.1 Baseline Evaluation Details

All factor pools are evaluated with the same 20-trading-day label, market split, parser, and Qlib backtest protocol whenever the released system produces formulaic factors. AlphaGPT is an interactive alpha-mining system and does not expose the exact IC/RankIC/portfolio table used in our experiments, so we evaluate its generated formulas with the same common evaluator used for the other factor-pool methods. For predictive-model baselines, LightGBM and LSTM use the same Qlib feature handler as Alpha158. This keeps the learner comparison fixed while avoiding additional feature engineering. The experimental design separates final factor-pool quality from fixed-budget discovery diagnostics because AlphaMemo is intended to improve the evolution process, not only to output a single high-scoring formula.

Artifact terms and intended use. We use released datasets, libraries, and baseline implementations only for academic evaluation and benchmarking, cite their original creators, and do not redistribute third-party datasets or baseline code. Our anonymous code release is intended for research reproducibility and evaluation, not as financial advice or a live-trading system.

Common implementation protocol. For the Qlib baselines and internal ablations, we use the same data split, label definition, parser, backtest protocol, and feature handler when applicable. External alpha-mining baselines are reproduced with their released code and recommended configurations, and their generated factors are evaluated under the common 20-trading-day protocol whenever their outputs can be parsed as formulaic factors. For symbolic-search methods evaluated in our pipeline, the Qlib backtest uses the admitted effective factor pool produced by the search procedure. To isolate the contribution of search-process memory, Al-

phaMemo and the Search-Ledger ablation share the same parent scorer, operator set, initial seed pool, factor length threshold, factor-pool capacity, and evaluation API. The Search-Ledger ablation disables residual memory and APV. AlphaMemo adds SSPM and APV on top of the same generator interface and admission rule. The shared parent scorer uses validation quality, search depth, retrieval frequency, lineage, and diversity against the current pool. Unless otherwise stated, the pool capacity is 50, the factor length threshold is 40, each selected parent proposes five children, $\tau_q = 0.10$ for |ICIR|, and $\tau_d = 0.70$ for maximum absolute Pearson correlation.

Compute environment. The experiments are run on a workstation with an Intel Core i9-13900 CPU, 64GB RAM, and one 24GB NVIDIA RTX 4090 GPU. Qlib evaluation and symbolic backtesting are CPU-bound, while the GPU is used for neural predictive baselines when applicable.

LLM generator details. LLM-based factor generation is performed through an OpenAI-compatible API endpoint. All reported AlphaMemo variants use the same generator backend unless otherwise stated. The reported AlphaMemo searches use deepseek/deepseek-v4-flash through the OpenRouter-compatible interface, temperature 0.7, maximum generation length 180 tokens, request timeout 45–60 seconds, and automatic retry on transient API failures. The prompt specifies the parent formula, its lineage context, the intended edit motif, parser constraints, and the requirement that the output be a single executable formula.

C.2 AlphaMemo Operating Points

The two AlphaMemo rows in the main table use the same generator, evaluator, admission rule, process-memory implementation, and factor-pool capacity. They differ only in how strongly the memory residual is allowed to intervene after warmup. Table 4 lists the settings used for the reported operating points.

Here w_m denotes the residual-memory weight and p_{rand} denotes the probability of sampling a random edit motif.

C.3 Variant Sensitivity

Table 5 reports additional AlphaMemo variants under the same 20-trading-day protocol used in the main experiments. The results show that the

Table 4: Parameter settings for the AlphaMemo operating points used in the main comparison.

Name	Strategy	Warmup	w_m	Motif	p_{rand}	Role
Balanced	graph-led SSPM	200	0.05	4	0.35	Conservative cross-market setting
Residual	residual SSPM	300	0.05	–	–	Stronger memory intervention

Table 5: Additional AlphaMemo variant results under the 20-trading-day protocol. AR and MDD are reported in percentage points.

Variant	CSI 500							S&P 500						
	IC	ICIR	RankIC	RankICIR	AR(%)	MDD(%)	Sharpe	IC	ICIR	RankIC	RankICIR	AR(%)	MDD(%)	Sharpe
Search-ledger only	0.0218	0.2313	0.0347	0.3707	7.00	-30.63	0.3536	0.0173	0.1414	0.0006	0.0044	13.80	-26.83	0.6282
AlphaMemo (balanced)	0.0401	0.3462	0.0496	0.4597	11.63	-23.43	0.6109	0.0288	0.2406	0.0144	0.1207	17.07	-22.54	0.7743
AlphaMemo (residual)	0.0101	0.1104	0.0165	0.1808	6.97	-26.08	0.3511	0.0410	0.3434	0.0228	0.1984	23.65	-23.62	1.0672
Weaker memory	0.0138	0.1455	0.0295	0.2886	5.80	-29.10	0.3011	0.0130	0.1001	-0.0039	-0.0299	17.09	-22.64	0.7393
Stronger memory	0.0205	0.2208	0.0283	0.3454	11.57	-26.27	0.5646	0.0196	0.1579	0.0048	0.0402	16.20	-25.19	0.7105
Late weak memory	0.0181	0.2231	0.0137	0.1708	8.57	-32.39	0.3917	0.0109	0.0847	-0.0008	-0.0061	15.60	-23.58	0.6814
Late weak memory, seed 2	0.0285	0.2681	0.0195	0.2329	10.39	-28.80	0.4806	0.0223	0.1736	-0.0010	-0.0076	17.15	-25.66	0.7794
APV-only memory	0.0220	0.2405	0.0363	0.3993	7.72	-30.79	0.3875	0.0242	0.1718	0.0035	0.0243	15.99	-26.06	0.6493
Warmup 180, weak memory	0.0339	0.3291	0.0490	0.5074	10.88	-30.82	0.4850	0.0294	0.2266	0.0048	0.0384	19.37	-24.82	0.8605
Warmup 220, weak memory	0.0155	0.2043	0.0089	0.1039	7.94	-29.77	0.3863	0.0308	0.2322	0.0102	0.0778	19.34	-27.28	0.8623
Warmup 240	0.0150	0.1828	0.0213	0.2341	8.78	-24.59	0.4504	0.0210	0.1609	0.0026	0.0192	17.10	-24.57	0.7607
Weak memory, seed 1	0.0041	0.0402	-0.0024	-0.0224	2.50	-45.00	0.1104	0.0330	0.2410	0.0050	0.0374	22.10	-26.82	0.9340
Balanced, seed 3	0.0291	0.3541	0.0350	0.4462	4.08	-38.90	0.1798	0.0246	0.1910	0.0023	0.0179	18.61	-27.98	0.8042

effect of process memory depends on when and how strongly it intervenes. The main configuration is selected because it provides the most balanced performance across CSI 500 and S&P 500, while several variants are competitive on one market but less stable on the other. As an additional no-memory stress test, we rerun the structured search with residual memory and APV disabled, random motif sampling set to 1.00, and the same generation budget. This run discovers 74 effective CSI 500 factors and 93 effective S&P 500 factors, but obtains weaker final-pool performance than the reported AlphaMemo operating points (CSI 500 RankICIR 0.0133, Sharpe 0.2305 and S&P 500 RankICIR -0.0046, Sharpe 0.6696). The result supports the view that the main gains are not explained by search bookkeeping alone.

The additional variants reinforce the calibration view. Memory can improve the search-ledger-only baseline, but overly early or seed-sensitive memory intervention can reduce stability. Conversely, the residual operating point is especially strong on S&P 500, which motivates reporting it separately rather than hiding the cross-market tradeoff inside a single averaged score.

C.4 Alpha Decay Diagnostics

Table 6 reports annual IC and RankIC during the 2022–2025 test window. This diagnostic is intended to probe temporal stability rather than replace the aggregate main-table metrics. The results again show that process memory is not a mono-

tonic knob: APV-only memory is comparatively stable on CSI 500, while the residual operating point is strongest on S&P 500 in later test years. This supports the view that AlphaMemo should expose calibrated operating points instead of forcing a single memory strength across markets.

C.5 Fixed-Budget Discovery Diagnostics

We also report preliminary fixed-budget diagnostics on CSI 500. These experiments measure the number of effective factors found under the same generation budget, where an effective factor must be executable, pass the quality threshold, satisfy the complexity bound, and remain sufficiently diverse from the current pool. Unlike the main table, this diagnostic measures search efficiency rather than final portfolio utility.

The diagnostic supports the main mechanism story: process memory improves discovery yield beyond random search and the search-ledger-only agent, but its gains depend on conservative confidence control and residualized credit assignment. Removing confidence gates or replacing AST-diff motifs with coarse manual labels sharply reduces the number of effective factors, while removing APV mainly weakens the ability to avoid repeated failure patterns.

D Representative Evolved Factors

Table 9 lists representative AlphaMemo factors selected on CSI500. The examples are moved to the appendix because formula expressions are long

Table 6: Annual alpha-decay diagnostics on the held-out test period. Each year reports IC / RankIC.

Variant	Market	2022		2023		2024		2025	
		IC	RankIC	IC	RankIC	IC	RankIC	IC	RankIC
AlphaMemo (balanced)	CSI500	0.0030	0.0048	-0.0090	0.0004	0.0087	0.0209	0.0354	0.0289
AlphaMemo (balanced)	S&P500	0.0186	0.0134	-0.0007	0.0016	0.0036	-0.0131	0.0195	0.0157
Weaker memory	CSI500	0.0152	0.0328	-0.0250	-0.0149	0.0293	0.0399	0.0209	0.0251
Weaker memory	S&P500	0.0201	0.0182	0.0010	0.0000	0.0110	-0.0027	0.0191	0.0125
Stronger memory	CSI500	0.0076	0.0149	-0.0099	-0.0087	0.0275	0.0343	0.0221	0.0245
Stronger memory	S&P500	0.0126	0.0089	0.0041	0.0062	0.0078	-0.0079	0.0172	0.0111
APV-only memory	CSI500	0.0219	0.0385	0.0031	0.0273	0.0320	0.0452	0.0407	0.0596
APV-only memory	S&P500	0.0172	0.0175	0.0083	0.0117	0.0024	-0.0093	0.0164	0.0129
AlphaMemo (residual)	CSI500	0.0066	0.0047	-0.0106	0.0011	0.0111	0.0205	0.0223	0.0181
AlphaMemo (residual)	S&P500	-0.0164	0.0229	0.0651	0.0009	0.0658	0.0335	0.0547	0.0478

Table 7: Fixed-budget discovery efficiency on CSI 500. We report mean effective factors.

Method	Seeds	Effective factors
Random search	5	10.4
Result-level memory	2	34.5
Search-Ledger Agent	5	52.4
AlphaMemo	5	76.0

Table 8: Mechanism ablation under the fixed-budget discovery diagnostic. We report mean effective factors.

Variant	Removed component	Effective factors
AlphaMemo	–	76.0
NoGate	confidence/warmup gates	34.6
AbsOLM	residual target	55.6
ManualMut	AST edit motifs	33.4
NoAPV	asymmetric veto	70.8

and easier to inspect in a wider table. We report absolute predictive metrics because factors with negative IC can be sign-flipped before entering a factor pool.

E More Discussions

▷ **Q1. Why is process-level memory more suitable than final-factor memory for self-evolving alpha mining?**

Final-factor memory retrieves what has worked before, but self-evolution needs to know how useful factors were reached. In alpha mining, two factors with similar validation scores may arise from very different edits, and a historically strong expression may decay after a market regime shift. AlphaMemo therefore stores reusable evidence about parent contexts, AST edit motifs, and edit outcomes. This moves the persistent state closer to the agent’s action space: instead of memorizing only winners, the agent remembers which search moves tend to help or fail.

▷ **Q2. How does AlphaMemo avoid the usual tradeoff between stronger memory and overfitting?**

AlphaMemo is designed so that memory is a calibrated residual correction, not an unconditional policy. The base search ledger still provides parent scores from quality, depth, retrieval frequency, lineage, and diversity. SSPM only adjusts this prior after enough evidence accumulates. Confidence gates suppress low-count or high-variance memories, and APV gives reliable failures veto power while keeping positive evidence soft. This asymmetry favors avoiding repeated mistakes over aggressively exploiting fragile successes.

▷ **Q3. Is AlphaMemo tied to a specific generator, market, or evaluator?**

The memory is attached to the symbolic search process rather than to a particular LLM response format. It observes executable formulas, parent-child edits, validation feedback, and admission decisions. Therefore, the same mechanism can be used with different LLM backbones, heuristic generators, or external formula-generation systems as long as their outputs can be parsed and evaluated. The CSI 500 and S&P 500 experiments provide a compact cross-market test of this design.

▷ **Q4. Are the gains simply caused by a stronger search ledger rather than memory?**

The sensitivity study is constructed to separate these effects. AlphaMemo and the Search-Ledger Agent share the generator interface, parent scorer, operator set, factor-pool capacity, complexity bound, diversity threshold, and admission rule. The Search-Ledger Agent removes process memory. APV-only keeps only negative process vetoes. The balanced and residual settings vary the strength of residual memory. Thus, the mechanism evidence comes from changing the memory

Table 9: Representative AlphaMemo factors selected on CSI500. Metrics are computed on the selection split.

Factor	Formula	Interpretation	IC	ICIR	RankICIR
SSPM_000	CsRank(TsMin(Div(TsSum(Where(Greater(\$close, Delay(\$close, 5)), Delta(Log(Add(\$volume, 1.0)), 5), 0.0), 10), TsStd(TsSum(Where(Greater(\$close, Delay(\$close, 5)), Delta(Log(Add(\$volume, 1.0)), 5), 0.0), 10), 60)), 20))	temporal price change with cross-sectional/temporal ranking	0.0325	0.4461	0.5237
SSPM_017	TsRank(Where(Greater(\$close, Delay(\$close, 5)), TsMean(Log(Add(\$volume, 1.0)), 5), 0.0), 20)	temporal price change with temporal volume level	0.0268	0.2877	0.1979
SSPM_033	CsRank(Add(Where(Greater(\$close, Delay(\$close, 5)), Delta(Log(Add(\$volume, 1.0)), 5), 0.0), Where(Less(\$close, Delay(\$close, 5)), Neg(Delta(Log(Add(\$volume, 1.0)), 5))), 0.0))	signed volume response to price direction	0.0159	0.2310	0.3745
SSPM_038	CsRank(Add(CsRank(Where(Greater(\$close, Delay(\$close, 5)), Delta(Log(Add(\$volume, 1.0)), 5), Neg(Delta(Log(Add(\$volume, 1.0), 5))), CsRank(Mul(Delta(Log(Add(\$volume, 1.0)), 5), Delta(\$close, 5))))))	rank aggregation with price-volume interaction	0.0173	0.2243	0.3743
SSPM_036	CsRank(Add(CsRank(Where(Greater(\$close, Delay(\$close, 10)), Delta(Log(Add(\$volume, 1.0)), 10), 0.0)), CsRank(Mul(Delta(Log(Add(\$volume, 1.0)), 5), Delta(\$close, 5))))))	longer-horizon price condition and volume-price interaction	0.0191	0.2299	0.3881

component while holding the underlying search machinery fixed.

▷ **Q5. Does AlphaMemo require delicate hyperparameter tuning?**

The experiments show that memory strength matters, but the method is not a black-box hyperparameter trick. Weak, strong, APV-only, balanced, and residual variants expose a clear calibration pattern. Balanced memory is stronger on CSI 500, while residual memory is stronger on S&P 500 for most reported metrics. We report both operating points transparently and list their parameters in Appendix C, rather than hiding the tradeoff behind a single averaged score.

▷ **Q6. Why not use policy fine-tuning, reinforcement learning, or full trajectory replay instead?**

Policy fine-tuning and reinforcement learning can be powerful, but they are expensive and can overfit noisy non-stationary financial feedback. Full trajectory replay preserves more information, but it quickly grows large and can inject irrelevant historical context into later decisions. AlphaMemo takes a lighter route: it compresses trajectories into structured parent-edit-outcome statistics and applies them as residual evidence at inference time. This keeps the agent adaptive without requiring a separate policy-training loop.

▷ **Q7. How can AlphaMemo complement existing alpha-mining systems?**

AlphaMemo is most naturally viewed as a memory layer for iterative formula search. It can sit on top of human-designed factor libraries, symbolic search, or LLM-based generators by recording which edits are productive under repeated evaluation. This is especially useful when the generator is strong but feedback is sparse and noisy: the memory layer filters repeated failures, encourages

reusable motifs, and leaves the generator free to propose novel expressions.

▷ **Q8. What remains outside the current scope?**

AlphaMemo still depends on a reliable evaluator, a well-defined symbolic operator grammar, and enough repeated edit observations to estimate memory statistics. Its advantage should be strongest in medium- or long-budget search, where local edit behaviors recur across parent contexts. For very small budgets, the confidence gate naturally suppresses uncertain memory and the method behaves closer to the base search ledger. Future work can extend the memory to richer market states, multi-horizon objectives, and live paper-trading evaluation, but the present design already addresses the core issue studied here: how a self-evolving alpha-mining agent can reuse search experience without storing brittle full trajectories.

F Statements on AI Assistants in Research and Writing

We used LLM assistants for language polishing and code refactoring during this work. All scientific claims, experimental results, and analyses were produced and verified by the authors. No AI-generated text was included without author review.