ALPHAPSC: A POOL-SUBSPACE CONDITIONED GENERATIVE FRAMEWORK FOR COMPLEMENTARY FORMULAIC ALPHA MINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Formulaic alphas are symbolic predictive expressions widely used in quantitative trading, where newly discovered factors are typically integrated into an existing factor pool and linearly combined into a composite trading signal. Consequently, an effective factor must provide incremental predictive information beyond what the pool already captures. However, existing mining frameworks often generate candidates independently of the evolving factor library, yielding redundant and highly correlated representations. We propose AlphaPSC, a **P**ool-**S**ubspace **C**onditional generative framework for complementary formulaic alpha mining. AlphaPSC conditions generation on a continuous summary of the predictive directions already occupied by the current pool, scores candidates by multiple pool relative rewards. On CSI300, CSI500, and CSI1000 under a shared expanding-window combiner with validation-selected and frozen hyperparameters, AlphaPSC improves combined-factor correlation and frictionless backtest metrics over representative formulaic miners. CSI300 ablations confirm that pool-subspace conditioning is important for these gains.

1 INTRODUCTION

Mining alphas, which are predictive signals that forecast cross-sectional returns, is a central task in quantitative trading (Kakushadze, 2016; Yang et al., 2021). Alphas are usually written as formulaic factors, closed-form expressions of lagged price and volume data. They remain widely adopted in practice because the resulting signals retain interpretability during factor combination (Yu et al., 2023; Shi et al., 2025).

In practice, alphas are rarely used in isolation. Newly mined alphas enter a factor pool, and a downstream combination model produces the final signal used for trading. When screening a single candidate, the standard metric is its Information Coefficient (IC), the mean cross-sectional correlation between its scores and forward returns. However, standalone IC is not enough and each addition should bring predictive information that the pool does not already capture (Feng et al., 2024).

Automated alpha mining has evolved from hand-crafted libraries and genetic programming to Reinforcement Learning (RL) (Yu et al., 2023; Zhu & Zhu, 2025; Schulman et al., 2017) and, more recently, Large Language Models (LLM) (Wang et al., 2024; Li et al., 2024; Chen et al., 2024; Tang et al., 2025). Some of these methods do use the pool, through correlation filters, combination rewards, or novelty penalties. In all of them the pool is consulted only after a candidate has been produced, and the generator itself receives no information about it. The generator therefore does not know which predictive directions are already occupied, and the search keeps rediscovering correlated variants. These variants can all pass the IC screen because factor signals are noisy and alpha returns decay quickly. However, they merely expand the factor pool without adding new information to the combined signal.

To overcome these limitations, we propose AlphaPSC, a **P**ool-**S**ubspace **C**onditional generative framework that builds the pool sequentially by conditioning each new alpha on the pool and scoring it on both its predictive power and its complementarity. To evaluate AlphaPSC, we conduct experiments on CSI300, CSI500, and CSI1000 under a shared expanding-window protocol, and AlphaPSC outperforms representative formulaic miners and end-to-end predictors.

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.

Our contributions are summarized below.

- We value each candidate by the marginal predictive information it adds to the current pool, making complementarity an intrinsic part of the mining objective. This shifts evaluation from individual alphas to the entire pool, so mining and combination share the same goal.
- We propose AlphaPSC, a pool-subspace conditional generative framework that samples complementary candidates conditioned on the current pool.
- We design preference-weighted multiple pool relative rewards that balance predictive power against complementarity and guide exploration during training.

2 Related Work

2.1 Formulaic Alpha Mining and Combination

Formulaic alpha mining began with hand-crafted libraries, in which each expression was designed and tested one at a time (Kakushadze, 2016). Evolutionary methods then automated the search, with AutoAlpha (Zhang et al., 2020) adding hierarchical expansion, AlphaEvolve (Cui et al., 2021) adding a richer operator set, and genetic programming exploring expression trees by mutation and crossover (Chen et al., 2021). All of them evaluate a candidate by its own IC against forward returns, so correlation between alphas is handled by pruning after the search. AlphaGen (Yu et al., 2023) builds each formula token by token and trains the generator with a policy gradient objective (Schulman et al., 2017). Its reward is the IC of the combined pool, so the pool enters the mining objective while formulas are still being generated. This reward arrives only when a formula is complete, which makes the feedback sparse. AlphaQCM (Zhu & Zhu, 2025) applies distributional RL to obtain more reliable value estimates under such feedback. AlphaForge (Shi et al., 2025) takes the opposite route and moves correlation control out of mining, into a time-varying combination model. AlphaSAGE (Chen et al., 2026) replaces reward maximization with a GFlowNet that samples diverse high-reward formulas, and its novelty term penalizes the maximum correlation with the alphas already found. In parallel, LLM pipelines propose or refine factors with language models and multi-agent loops (Wang et al., 2024; Li et al., 2024; Chen et al., 2024; Tang et al., 2025). They change how a candidate is written, but the pool is still read only after generation. A separate line of work gives up the symbolic form altogether. RL trading systems learn portfolio objectives end to end (Wang et al., 2019; Yang et al., 2020; Liu et al., 2021), and deep predictors forecast returns accurately (Gu et al., 2020; Feng et al., 2024), but their outputs cannot be inspected as formulas (Yu et al., 2023; Shi et al., 2025).

Across this progression, the pool has appeared as a post-hoc filter, as a reward term, and as a downstream combiner. In every case it enters after a candidate has been generated, so the generator never knows which predictive directions are already occupied. AlphaPSC conditions generation on a pool occupancy summary of the current pool, so this information is available while the formula is being built.

2.2 GFlowNets and Preference-Conditioned Sampling

GFlowNets sample compositional objects with probability proportional to a non-negative reward, so they cover many modes of the reward landscape that reward maximization would miss (Bengio et al., 2021; 2023). Multi-objective GFlowNets extend this idea by conditioning the policy on a preference weight w that scalarizes a vector reward, so one model covers a range of trade-offs among objectives (Jain et al., 2023; Hayes et al., 2022). Such conditioning signals are fixed descriptions of what the user wants, and they say nothing about what a growing pool already contains. AlphaPSC therefore conditions generation on a pool occupancy summary that changes as the pool grows, so the generator always sees the current state of the pool. A preference weight is still used during training, where it mixes IC, residual IC, and marginal gain, but it carries no information about the pool.

3 METHODOLOGY

3.1 PROBLEM FORMULATION

A formulaic alpha is a closed-form expression of lagged price and volume data that assigns a predictive score to each stock on each trading day. We represent each alpha x as an abstract syntax tree (AST) whose nodes are drawn from the feature and operator vocabulary in Appendix A.2. A generator $P_F(a_t \mid s_t)$ builds the tree top-down. State s_t is the partial AST, and action $a_t \in \mathcal{A}$ picks the next operator or operand. An alpha pool $P = \{x_i\}$ stores accepted alphas together with a lightweight linear combiner that produces a composite prediction $\hat{y}_P \in \mathbb{R}^{T\times N}$ of the future return y we aim to predict.

Under capacity C, we seek a pool whose combined signal $\hat{y}_P$ maximizes predictive quality on the training window.

$$\max_{P:\,|P|\le C} \mathrm{IC}(\hat{y}_P,\, y), \tag{1}$$

where $\mathrm{IC}(\hat{y}_P,\, y)$ denotes the mean daily cross-sectional correlation between $\hat{y}_P$ and y.

Building the pool is not a single optimization problem. It is a sequence of them. When a useful alpha enters the pool, the pool itself changes, and so does the standard for admitting a new alpha. Prior miners score candidates with a fixed standalone IC that ignores the current pool, so they keep producing variants correlated with alphas already discovered. We therefore condition generation on the active pool, so that sampling favors candidates that raise the combined IC of P. Concretely, we define a pool-relative reward $R_w(x \mid P, w)$ and seek a generator whose samples follow

$$P^\star(x \mid P, w) \propto R_w(x \mid P, w), \tag{2}$$

where the preference $w = (w_1, w_2, w_3)$ mixes three components. They are training-window IC, Marginal Gain, and Residual IC, so that one generator can cover different trade-offs between standalone strength and complementarity.

3.2 POOL-SUBSPACE CONTEXT

Because generation should depend on the current pool, the generator must see that pool. A single score such as ensemble IC only tells us how strong a pool is. It does not tell us which predictive directions the pool already covers. Two pools can have similar ensemble IC but different styles such as momentum, reversal, or liquidity, and then need different complementary alphas. Sending a specific set of basis vectors is also unstable, because the same subspace can look different after a rotation. The pool also grows over time, so its size changes and cannot be a fixed network input. We therefore compress the pool into a fixed-dimension context $c_P = c_P(P)$ with three parts, namely occupied directions, the combination residual, and pool statistics. As shown in Figure 1, these three parts are concatenated and passed to the generator.

Occupied directions. Occupied directions describe which predictive styles the current pool already covers, such as momentum or reversal. Evaluating x on historical data produces a score panel $S_x \in \mathbb{R}^{T\times N}$ over T trading days and N stocks. We map each pool member's score panel to a fixed-length vector with a sketching map $\Phi : \mathbb{R}^{TN} \to \mathbb{R}^d$, then summarize these vectors by a projection matrix Π_P. We do this because each score panel $S_{x_i} \in \mathbb{R}^{T\times N}$ is too large to feed into the network, and the number of panels changes as the pool grows, so the input size would not be fixed. We fix Φ at initialization and never train it, so every pool member is projected into the same sketch space as the pool changes. In detail, Φ first keeps a random coordinate subset of size s from the flattened panel, then multiplies by a fixed Gaussian matrix $G \in \mathbb{R}^{d\times s}$. For each pool member x_i, let $\tilde{S}_{x_i}$ be its training-window score panel after cross-sectional day-wise z-scoring. Applying Φ gives the sketch embedding

$$e_i = \Phi\big(\mathrm{vec}(\tilde{S}_{x_i})\big) \in \mathbb{R}^d. \tag{3}$$

Stacking these embeddings by row gives $E_P = [e_1; \ldots; e_{|P|}] \in \mathbb{R}^{|P|\times d}$. This set still grows with the pool, so we do not pass $\{e_i\}$ to the network directly. Instead, we standardize each row to unit empirical standard deviation to obtain $\tilde{E}_P$, and form

$$\Pi_P = \tilde{E}_P^\top \big(\tilde{E}_P \tilde{E}_P^\top + \lambda \mathbf{I}\big)^{-1} \tilde{E}_P \in \mathbb{R}^{d\times d}. \tag{4}$$

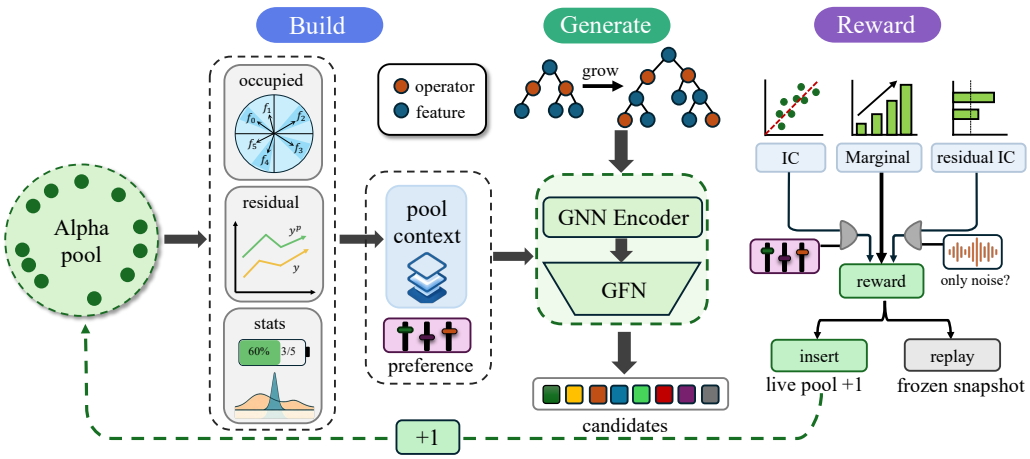


Figure 1: Overview of AlphaPSC. **Build.** The alpha pool is compressed into a fixed context c_P with three parts. Occupied directions record which predictive styles the pool already covers. The combination residual records the part of returns still unexplained by the pooled forecast. Pool statistics record how full the pool is and how concentrated its combination weights are. **Generate.** c_P and a preference weight w condition a GFlowNet that encodes the partial AST and grows a formula token by token. **Reward.** Each candidate is scored by training-window IC, Marginal Gain, and Residual IC. Survivors enter the live pool, after which c_P is rebuilt and the next step starts from the enlarged pool. Trajectory Balance (TB) training instead replays frozen pool snapshots and never writes to the live pool.

When the pool is empty, $\Pi_P = \mathbf{0}$. A small $\lambda > 0$ stabilizes the projection when the pool embeddings are nearly linearly dependent. Details for $\lambda = 0$ and the hyperparameter values are in Appendix A.5.

Combination residual. The combination residual describes how much of the return target the current pool still leaves unexplained. Here $y \in \mathbb{R}^{T\times N}$ is the future return we aim to predict on the training window. A lightweight linear combiner fits the pool members to y and produces a combined forecast $\hat{y}_P \in \mathbb{R}^{T\times N}$. The residual $y - \hat{y}_P$ is what this combined forecast still misses. We compress it with the same map Φ.

$$\Phi\big(\mathrm{vec}(y - \hat{y}_P)\big) \in \mathbb{R}^d. \tag{5}$$

Pool-member panels are day-wise z-scored so that style directions are compared without raw scale differences. We do not apply day-wise z-scoring to $y - \hat{y}_P$, because an extra day-wise z-score would erase how large the unexplained gap still is, which is exactly what this branch should carry. When the pool is empty, $\hat{y}_P = \mathbf{0}$, so the residual branch reduces to $\Phi\big(\mathrm{vec}(y)\big)$.

Pool statistics. Pool statistics describe how full the pool is and how concentrated its combination weights are. These quantities later inform the residual-energy gate. To avoid confusion with the preference vector w, we write the pool combination weights as $\beta_P = (\beta_1, \ldots, \beta_{|P|})$. When the pool is nonempty and $\sum_j |\beta_j| > 0$, define normalized shares $p_i = |\beta_i| / \sum_j |\beta_j|$. Otherwise the weight-based terms are set to zero. From $|P|$ and $\{p_i\}$ we build four scalars in turn.

We first measure capacity utilization,

$$u(P) = |P|/C, \tag{6}$$

which tells the generator how close the pool is to capacity C.

Given that size, we next ask whether the combined forecast leans on one member. The maximum weight share

$$p_{\max}(P) = \max_i p_i \tag{7}$$

is large when a single pool member dominates.

Dominance alone does not show how the remaining weight is spread, so we also compute the Shannon entropy

$$H(P) = -\sum_i p_i \log p_i, \tag{8}$$

with the convention $0 \log 0 = 0$. A larger $H(P)$ means the weights are more evenly dispersed across pool members.

As a second view of the same concentration pattern, we compute the effective number of alphas

$$n_{\text{eff}}(P) = 1 \Big/ \sum_i p_i^2. \tag{9}$$

This value is close to $|P|$ when weights are equal, and close to 1 when one weight dominates.

We collect the four scalars as

$$\text{stats}(P) = \big[u(P),\ p_{\max}(P),\ H(P),\ n_{\text{eff}}(P)\big]. \tag{10}$$

Together they tell the generator how crowded the pool is and how concentrated its combination weights are.

Pool context. We concatenate the three branches into one fixed-dimension vector

$$c_P = \big[\text{vec}(\Pi_P),\ \text{IC}_{\text{ens}},\ \Phi\big(\text{vec}(y - \hat{y}_P)\big),\ \text{stats}(P)\big]. \tag{11}$$

Here IC_{ens} is the training-window IC of the combined forecast $\hat{y}_P$ against y. Together, c_P gives the generator a fixed-size view of the current pool. It records which predictive directions are already covered, how much of the return target remains unexplained, and how full and concentrated the pool is. With this view, the generator can propose alphas that complement the pool and fill what the pool still needs.

3.3 Conditional Generator

We grow each new alpha conditioned on the pool context c_P and a preference weight w. For the current pool, we feed (c_P, w) to the generator and keep them fixed until a new alpha is added to the pool. In this way the generator sees both the current pool and the chosen reward trade-off. We train it as a GFlowNet so that sampling follows pool-relative reward and many strong formulas stay reachable (Bengio et al., 2021; 2023).

Generator architecture. The generator sees the current pool while it builds a new alpha. Building one alpha is a sequential decision process over an AST. At each step the generator picks the next operator or operand until the expression is complete. Three vectors enter that choice. The vector h_s encodes the unfinished formula so far. It records which operators and operands are already on the tree, and which slots are still open. The vector z_{pool} is a fixed-dimension representation of the pool context c_P, so every token choice can use which directions the pool already covers and what remains unexplained. The vector z_w is a fixed-dimension representation of the preference w, so every token choice can also use the current reward trade-off. The next-token distribution combines all three. The partition function depends only on the pool and the preference.

$$P_F(a_t \mid s_t, c_P, w) = \text{softmax}\big(\text{Head}([h_s;\ z_{\text{pool}};\ z_w])\big), \tag{12}$$

$$\log Z(c_P, w) = \text{LogZHead}([z_{\text{pool}};\ z_w]). \tag{13}$$

Invalid actions are masked at each step. Because z_{pool} enters P_F at every step, generation is pool-conditioned from the first token to the last.

Trajectory Balance (TB) is the training rule of the GFlowNet. It asks the probability of generating a finished formula to match that formula's pool-relative reward under the current pool and preference. Concretely, TB minimizes the squared mismatch

$$\mathcal{L}_{\text{TB}}(\tau) = \left(\log Z(c_P, w) + \sum_t \log P_F(a_t \mid s_t, c_P, w) - \log R_w(x \mid P, w) - \sum_t \log P_B(s_{t-1} \mid s_t)\right)^2. \tag{14}$$

The backward model P_B does not use (c_P, w). Taking apart a finished AST follows only the grammar. When the loss is small, formulas that help the current pool receive higher sampling probability under (c_P, w). Seeing the pool through c_P and matching samples to pool-relative reward with TB makes complementarity part of generation itself.

3.4 Multiple Pool Relative Rewards

Standalone IC measures how predictive one alpha is by itself. It cannot tell a near-duplicate of an existing pool member from an alpha that adds new return structure. We therefore score each candidate with three pool-relative quantities.

Reward components. **IC.** $R_{\mathrm{IC}}(x)$ is the mean daily cross-sectional correlation between candidate scores S_x and the future return y on the training window. It measures standalone predictive power and acts as the basic quality screen.

$$R_{\mathrm{IC}}(x) = \frac{1}{T}\sum_{d=1}^{T} \mathrm{corr}\big(S_x[d,:],\, y[d,:]\big). \tag{15}$$

Marginal Gain. $\mathrm{MG}_{\mathrm{cv}}(x \mid P)$ asks how much the pool's combined forecast improves after x is added. If we fit the pool combination and score the gain on the same dates, the estimate has a data-leakage problem. To prevent this leakage, we split the training window into K time blocks. For each block B_k, combination weights are fit only on the other dates A_k, and the IC comparison is made on B_k alone. Averaging over blocks gives

$$\mathrm{MG}_{\mathrm{cv}}(x \mid P) = \frac{1}{K}\sum_{k}\Big(\mathrm{IC}_{B_k}(P \cup \{x\}) - \mathrm{IC}_{B_k}(P)\Big). \tag{16}$$

A positive value means that, on held-out training dates, adding x raises the pool's combined IC under the current combiner capacity.

Residual IC. $\mathrm{ResIC}_{\mathrm{cv}}(x \mid P)$ measures how well x predicts the part of y that the pool forecast still misses. Using the same blocks A_k and B_k, we remove from S_x and from y the part already explained by the pool panels, with ridge coefficients fit on A_k and applied on B_k. This gives residuals $r_x^{(B_k)}$ and $r_y^{(B_k)}$, and

$$\mathrm{ResIC}_{\mathrm{cv}}(x \mid P) = \frac{1}{K}\sum_{k}\frac{1}{|B_k|}\sum_{d \in B_k} \mathrm{corr}\big(r_x^{(B_k)}[d,:],\, r_y^{(B_k)}[d,:]\big). \tag{17}$$

Residual IC focuses on leftover predictive signal. Marginal Gain focuses on whether the combiner raises combined IC right away. An alpha with novel structure but modest combination lift can still score high on Residual IC.

These three quantities are the building blocks of the mining objective for a candidate x under the current pool P. We collect them as

$$F(x \mid P) = \big[R_{\mathrm{IC}}(x),\, \mathrm{MG}_{\mathrm{cv}}(x \mid P),\, \mathrm{ResIC}_{\mathrm{cv}}(x \mid P)\big]. \tag{18}$$

A preference weight $w = (w_1, w_2, w_3)$ mixes the three entries. The intended scalar objective is the weighted sum

$$w_1 R_{\mathrm{IC}}(x) + w_2 \mathrm{MG}_{\mathrm{cv}}(x \mid P) + w_3 \mathrm{ResIC}_{\mathrm{cv}}(x \mid P). \tag{19}$$

Larger w_1 favors standalone strength, larger w_2 favors realized pool lift, and larger w_3 favors leftover predictive signal. For training, we do not feed this raw sum to Trajectory Balance directly. Before scoring, a correlation gate rejects candidates too close to an existing pool member, and a residual-energy gate rejects candidates whose leftover mass outside the pool is too weak. We then center and scale F with a frozen training-only normalizer and apply a softplus transform with temperature to obtain the scalar reward $R_w(x \mid P, w)$. Gated-out candidates receive only a small floor value so the TB update stays well-defined. During training, w is sampled from a symmetric Dirichlet distribution so one generator covers many trade-offs. During insertion, a fixed uniform preference is used so no preference is tuned on validation or test data. The exact transform, floor, gate thresholds, and preference settings are in Appendix A.5.

In short, IC checks whether x predicts returns, Marginal Gain checks whether adding x raises the pool's combined IC, and Residual IC checks whether x explains what the pool still misses. Together with w, they define the mining objective. A candidate is valued by what it adds to the current pool under the chosen trade-off.

3.5 Training and Insertion Loops

AlphaPSC trains the generator and grows the live pool in two separate loops.

In the **training loop**, we freeze the current pool and build c_P from it. We sample a preference w from a symmetric Dirichlet distribution, let the generator P_F build one new alpha under (c_P, w), score it with $R_w(x \mid P, w)$, and update the generator with Trajectory Balance. The loop may also reuse earlier generated alphas. It recomputes their generation probabilities under the current generator and recomputes R_w from stored objective values with the frozen normalizer. Random w appears only in this loop. This loop updates the generator only. It does not add alphas to the live pool.

In the **insertion loop**, which runs every few training episodes, we freeze the live pool, rebuild c_P, and generate candidates under the fixed uniform preference used above. Candidates are scored and gated on the training window. The best survivor is admitted if the pool still has capacity, or if its reward exceeds the leave-one-out reward of the weakest current member. After each admission, c_P is recomputed from the updated pool. Pseudocode is in Appendix A.4.

Training therefore learns across many reward trade-offs. Insertion uses one fixed preference to enlarge the live pool and refresh c_P.

4 Experiments

4.1 Experiment Setting

Evaluation Metrics. Following common protocol in prior formulaic-mining work (Yu et al., 2023; Chen et al., 2026; Tang et al., 2025), we report two families of metrics on the combined factor. The first family covers correlation metrics, including IC, IC Information Ratio ICIR, Rank IC RankIC, and Rank ICIR $\mathrm{RankICIR}$. The second covers frictionless backtest metrics, including Annualized Return AR, Maximum Drawdown MDD, and Sharpe Ratio SR, from a shared long-only top-20% portfolio on a 20-session holding horizon, without transaction costs, slippage, or turnover caps. Higher is better except MDD, for which closer to zero is better. These backtest numbers are for relative ranking under one frictionless protocol, not claims of net tradable profit after costs. See Appendices B.1 and B.2.

Datasets. We evaluate on CSI300, CSI500, and CSI1000 (Yang et al., 2021). The chronological split is 2010-01-01–2020-12-31 for training, 2021-01-01–2021-12-31 for validation used only for combiner hyperparameter selection, and a shared test window 2022-01-01–2026-06-30 with 1085 trading days. AlphaPSC uses 10^4 episodes with pool capacity $C{=}50$. Other formulaic miners share the same expanding-window second-stage combiner. Details are in Appendices A.5 and B.2.

Deployment and baselines. Pools are mined on training data through 2020 and frozen for validation and test. Then c_P, F, and admission never use post-2020 labels. This is a static OOS evaluation of a fixed pool. The expanding-window combiner may refit coefficients with labels prior to each date, but does not add factors. We compare against MLP, LightGBM, XGBoost (Murtagh, 1991; Ke et al., 2017; Chen & Guestrin, 2016), GP (Chen et al., 2021), AlphaGen (Yu et al., 2023), AlphaQCM (Zhu & Zhu, 2025), AlphaAgent (Tang et al., 2025), and AlphaSAGE (Chen et al., 2026). Baseline settings are in Appendix B.3.

4.2 Overall Performance

Table 1 reports CSI300/500/1000 under the shared expanding-window evaluation. Across the three universes, AlphaPSC ranks first on IC, ICIR, Rank IC, and Rank ICIR, so the combined signal is stronger and more stable over time. It also ranks first on AR and SR, and records the least severe MDD on every universe. The pool therefore earns higher frictionless returns with a smoother

equity path and smaller peak-to-trough losses. Appendix C.1 reports a five-seed CSI300 check, and Appendix C.2 reports the lowest pool-average pairwise absolute correlation $\overline{|\rho|}$ as a redundancy diagnostic. Figure 2 shows the CSI300 equity curve. CSI500 and CSI1000 curves and a combiner-capacity sweep are in Appendices C.3 and C.4.

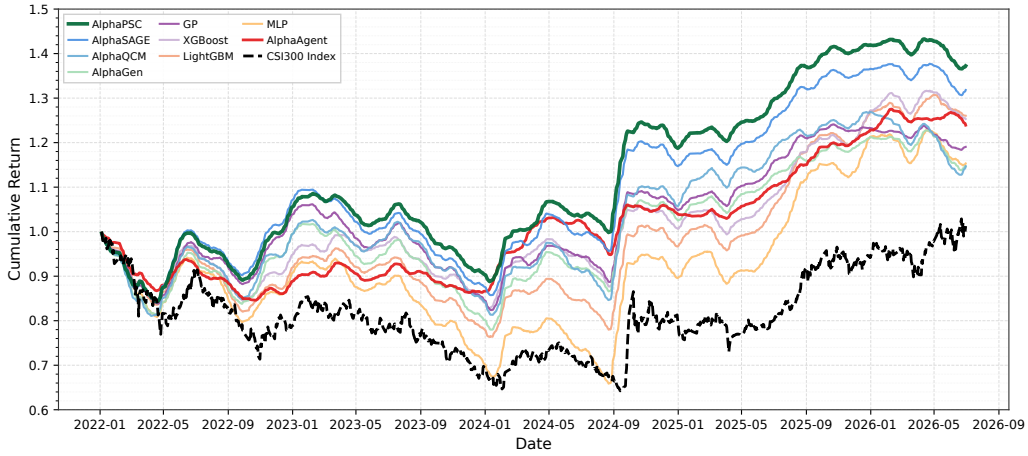


Figure 2: Cumulative return on CSI300, 2022-01–2026-06. Comparison among AlphaPSC, all baselines, and the CSI300 Index benchmark.

Table 1: Performance comparison on CSI300, CSI500, and CSI1000 over the test window 2022-01-01 to 2026-06-30. Bold and underlined numbers denote the best and second-best results, respectively. For MDD, closer to zero is better. Appendix C.1 reports mean ± std over five mining seeds on CSI300.

		Correlation metrics				Portfolio metrics		
Dataset	Method	IC	ICIR	Rank IC	Rank ICIR	AR	MDD	SR
CSI300	MLP	0.012	0.076	0.028	0.208	3.56%	−33.9%	0.69
	LightGBM	0.014	0.137	0.018	0.181	6.05%	−23.4%	1.39
	XGBoost	0.032	0.227	0.040	0.294	5.88%	−17.0%	1.51
	GP	0.018	0.140	0.029	0.207	4.42%	−22.4%	1.11
	AlphaGen	0.026	0.156	0.039	0.256	3.40%	−23.3%	0.81
	AlphaQCM	0.016	0.091	0.035	0.221	3.41%	−20.8%	0.73
	AlphaAgent	0.034	0.165	0.053	0.229	5.56%	−15.1%	1.96
	AlphaSAGE	0.040	0.265	0.052	0.355	7.39%	−21.7%	1.69
	AlphaPSC (ours)	**0.048**	**0.326**	**0.063**	**0.444**	**8.65%**	**−14.2%**	**2.02**
CSI500	MLP	0.021	0.175	0.028	0.238	7.91%	−36.2%	1.46
	LightGBM	0.021	0.205	0.031	0.340	7.91%	−25.1%	1.66
	XGBoost	0.021	0.162	0.052	0.418	7.60%	−31.6%	1.55
	GP	0.021	0.165	0.040	0.316	5.34%	−26.4%	1.19
	AlphaGen	0.027	0.222	0.053	0.391	6.80%	−18.1%	1.64
	AlphaQCM	0.019	0.136	0.047	0.333	7.92%	−17.1%	1.87
	AlphaAgent	0.033	0.245	0.034	0.290	7.39%	−38.5%	1.36
	AlphaSAGE	0.027	0.269	0.045	0.474	10.16%	−23.0%	2.12
	AlphaPSC (ours)	**0.035**	**0.316**	**0.055**	**0.495**	**11.27%**	**−16.7%**	**2.37**
CSI1000	MLP	0.038	0.334	0.046	0.453	9.14%	−29.1%	1.49
	LightGBM	0.039	0.381	0.059	0.577	9.40%	−25.8%	1.76
	XGBoost	0.045	0.355	0.077	0.603	8.52%	−26.3%	1.63
	GP	0.004	0.078	0.016	0.167	7.99%	−31.3%	1.37
	AlphaGen	0.043	0.390	0.065	0.618	9.74%	−26.8%	1.75
	AlphaQCM	0.041	0.386	0.060	0.646	9.15%	−30.2%	1.61
	AlphaAgent	0.030	0.272	0.051	0.483	8.15%	−30.1%	1.52
	AlphaSAGE	0.049	0.403	0.069	0.550	9.64%	−23.9%	1.81
	AlphaPSC (ours)	**0.055**	**0.414**	**0.081**	**0.696**	**10.61%**	**−20.1%**	**2.12**

4.3 Ablation Study

Tables 2 and 3 ablate AlphaPSC on CSI300. Table 2 varies only the c_P fed to generation. Rewards stay fixed on the true pool. Table 3 then keeps full c_P and varies only the reward.

Table 2 shows that seeing the pool during generation is useful. Reward-only keeps the true-pool reward but hides c_P, and Test IC falls to 0.013. Random-context replaces c_P by noise and is worse still, so an unmatched conditioner hurts generation. The subspace, residual, and stats blocks each help when they are present, and Full AlphaPSC is best on every reported metric. The three parts of c_P are all needed.

Table 3 shows a parallel pattern on the reward side under full c_P. IC-only already beats Reward-only in Table 2. Adding Marginal Gain raises IC further. Adding Residual IC raises it again. Full AlphaPSC remains strongest, so IC, Marginal Gain, and Residual IC each contribute, and the multi-component preference is needed under full c_P.

Table 2: CSI300 pool-context ablations over 2022-01-01–2026-06-30. Blocks of c_P fed to generation use ✓ when supplied, × when zeroed, and ∼ when replaced by noise. All variants keep rewards on the true pool.

	Policy c_P			Test			
Variant	$\mathrm{vec}(\Pi_P)$	resid.	stats	IC	ICIR	AR	SR
Random-context	∼	∼	∼	-0.010	-0.079	1.23%	0.24
Reward-only	×	×	×	0.013	0.095	3.21%	0.69
Subspace-only	✓	×	×	0.022	0.156	2.65%	0.89
Subspace+stats	✓	×	✓	0.027	0.193	3.90%	0.96
No-subspace	×	✓	✓	0.036	0.273	6.04%	1.43
AlphaPSC	✓	✓	✓	$\mathbf{0.048}$	$\mathbf{0.326}$	$\mathbf{8.65\%}$	$\mathbf{2.02}$

Table 3: CSI300 reward-preference ablations under full c_P over 2022-01-01–2026-06-30, with the same stage-2 combiner as Table 2. Checked columns of F receive positive weight in the pinned preference w.

	Preference w			Test			
Variant	IC	MG	ResIC	IC	ICIR	AR	SR
IC-only	✓	×	×	0.020	0.140	3.82%	0.94
IC+MG	✓	✓	×	0.030	0.199	4.80%	1.12
IC+ResIC	✓	×	✓	0.035	0.223	5.86%	1.40
AlphaPSC	✓	✓	✓	$\mathbf{0.048}$	$\mathbf{0.326}$	$\mathbf{8.65\%}$	$\mathbf{2.02}$

5 Conclusion

We propose AlphaPSC for complementary formulaic alpha mining. It conditions generation on a pool occupancy summary c_P that concatenates occupied predictive directions, the combination residual, and pool statistics. The reward scores candidates with IC, residual IC, and marginal combination gain relative to the current pool.

On CSI300, CSI500, and CSI1000, under one expanding-window combiner and frozen validation-selected hyperparameters, AlphaPSC records the strongest combined IC and the strongest frictionless AR/SR against the GP, RL, GFlowNet, LLM, and end-to-end baselines in Table 1.

CSI300 ablations show that seeing a matched pool context during generation is useful, that the subspace, residual, and stats blocks of c_P are all needed, and that the multi-component preference helps under full c_P. Cross-correlation heatmaps in Figure 3 and the combiner-capacity sweep in Figure 5 show that the mined factors are complementary. Pairwise correlation is the lowest among the compared pools, and combined IC keeps rising as more factors enter the combiner.

Together, these results show that pool-conditioned generation of complementary factors, scored by multiple pool relative rewards, improves both combined signal quality and its conversion to frictionless returns. Evaluation therefore moves from individual alphas to the entire pool, so mining and combination share the same goal.

Reproducibility statement

We do not release code with this submission. The pipeline is specified in enough detail to be reimplemented from the paper alone. The data are public. All experiments use the standard Qlib A-share panel (Yang et al., 2021) on CSI300/500/1000 with the chronological split of Section 4.1, and no proprietary source enters any stage. Appendix A gives the full specification of the method. It covers notation in Table 4, the complete feature and operator vocabulary with discrete token ranges in Appendix A.2, the RGCN relation types in Appendix A.3, the training and insertion loops as pseudocode in Algorithm 1 of Appendix A.4, and every default hyperparameter in Table 7, including sketch dimension and seed, ridge values, gate thresholds, fold count and embargo, acquisition period, and learning rate. Appendix B fixes the reporting side. It covers metric definitions in Appendix B.1, the shared expanding-window second-stage combiner with its validation-selected and frozen knobs and daily coefficient refits in Appendix B.2, and the per-baseline settings and search budgets in Appendix B.3 and Table **??**. Objective normalization uses a training-only pilot archive, described in Section 3.4 and Appendix A.5. The ablation variants named in Section 4.3 are defined coordinate-by-coordinate in Appendix B.3. Appendix C reports the multi-seed check in Appendix C.1 and the supplementary figures.

Ethics statement

The study uses historical equity panels and does not involve personal data. Like any work on trading signals, the mined factors could be used in live markets. We do not provide investment advice.

AI use statement

We acknowledge the use of AI assistants during the preparation of this manuscript for language polishing and grammatical correction, in compliance with the ICLR 2027 formatting instructions. All scientific contributions, experimental designs, and interpretations remain entirely our own.

References

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. In *Advances in Neural Information Processing Systems*, volume 34, pp. 27381–27394, 2021.

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J. Hu, Mo Tiwari, and Emmanuel Bengio. GFlowNet foundations. *Journal of Machine Learning Research*, 24(210):1–55, 2023. URL https://jmlr.org/papers/v24/22-0364.html.

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

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

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794. ACM, 2016. doi: 10.1145/2939672.2939785.

Tianxiang Chen, Wei Chen, and Luyao Du. 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)*, pp. 1113–1117. IEEE, 2021.

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

Guanhao Feng, Jingyu He, Nicholas G. Polson, and Jianeng Xu. Deep learning in characteristics-sorted factor models. *Journal of Financial and Quantitative Analysis*, 59(7):3001–3036, 2024. doi: 10.1017/S0022109024000251.

Shihao Gu, Bryan Kelly, and Dacheng Xiu. Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33(5):2223–2273, 2020. doi: 10.1093/rfs/hhz068.

Conor F. Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M. Zintgraf, Richard Dazeley, Fredrik Heintz, Enda Howley, Athirai A. Irissappane, Patrick Mannion, Ann Nowé, Gabriel de Oliveira Ramos, Marcello Restelli, Peter Vamplew, and Diederik M. Roijers. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1):26, 2022. doi: 10.1007/s10458-022-09552-y.

Moksh Jain, Sharath Chandra Raparthy, Alex Hernández-Garcıa, Jarrid Rector-Brooks, Yoshua Bengio, Santiago Miret, and Emmanuel Bengio. Multi-objective GFlowNets. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 14631–14653. PMLR, 2023.

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

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

Zhiwei Li, Ran Song, Caihong Sun, Wei Xu, Zhengtao Yu, and Ji-Rong Wen. Can large language models mine interpretable financial factors more effectively? A neural-symbolic factor mining agent model. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3891–3902. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.233.

Xiao-Yang Liu, Hongyang Yang, Jiechao Gao, and Christina Dan Wang. FinRL: Deep reinforcement learning framework to automate trading in quantitative finance. In *Proceedings of the Second ACM International Conference on AI in Finance*, pp. 1–9. ACM, 2021. doi: 10.1145/3490354.3494366.

Fionn Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5-6):183–197, 1991.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL `https://arxiv.org/abs/1707.06347`.

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

Ziyi Tang, Zechuan Chen, Jiarui Yang, Jiayao Mai, Yongsen Zheng, Keze Wang, Jinrui Chen, and Liang Lin. AlphaAgent: Llm-driven alpha mining with regularized exploration to counteract alpha decay. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 2813–2822. ACM, 2025. doi: 10.1145/3711896.3736838.

Jingyuan Wang, Yang Zhang, Ke Tang, Junjie Wu, and Zhang Xiong. AlphaStock: A buying-winners-and-selling-losers investment strategy using interpretable deep reinforcement attention networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1900–1908. ACM, 2019. doi: 10.1145/3292500.3330647.

Meiyun Wang, Kiyoshi Izumi, and Hiroki Sakaji. LLMFactor: Extracting profitable factors through prompts for explainable stock movement prediction. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3120–3131. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.185.

Hongyang Yang, Xiao-Yang Liu, Shan Zhong, and Anwar Walid. Deep reinforcement learning for automated stock trading: An ensemble strategy. In *Proceedings of the First ACM International Conference on AI in Finance*, pp. 1–8. ACM, 2020. doi: 10.1145/3383455.3422540.

Xiao Yang, Weiqing Liu, Dong Zhou, Jiang Bian, and Tie-Yan Liu. Qlib: An AI-oriented quantitative investment platform. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, pp. 4859–4868, 2021. doi: 10.1145/3459637.3481930.

Shuo Yu, Hongyan Xue, Xiang Ao, Feiyang Pan, Jia He, Dandan Tu, and Qing He. Generating synergistic formulaic alpha collections via reinforcement learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5476–5486, 2023. doi: 10.1145/3580305.3599831.

Tianping Zhang, Yuanqi Li, Yifei Jin, and Jian Li. AutoAlpha: an efficient hierarchical evolutionary algorithm for mining alpha factors in quantitative investment, 2020. URL `https://arxiv.org/abs/2002.08245`.

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

A METHOD SUPPLEMENT

A.1 NOTATION

Table 4 lists symbols used in the paper, lists the key symbols used throughout this paper and gives their definitions.

Table 4: Notation used throughout the paper.

Symbol	Meaning
x	A complete formulaic alpha as an AST
s_t	Partial AST after t generation steps
a_t	Next token among operator, feature, lag, constant, or separator
P_F	Forward generator conditioned on (s_t, c_P, w)
$P = \{x_i\}$	Live factor pool with combination weights β_P
$S_x \in \mathbb{R}^{T\times N}$	Cross-sectional score panel of x over T days and N names
y	Forward cross-sectional return target over the h=20-session holding horizon
$\hat{y}_P$	Linear combination of current pool scores
C	Pool capacity, set to C=50 in the main experiments
$R_w(x \mid P, w)$	Scalar TB reward after preference mixing and gates. It depends on P, not only on c_P
$w \in \Delta^2$	Dirichlet preference over the three coordinates of F
c_P	Pool occupancy summary that concatenates occupied predictive directions, the combination residual, and pool statistics
Φ	Frozen Gaussian sketch mapping flattened panels to $\mathbb{R}^d$. Factor panels are day-wise z-scored first
s	Sketch coordinate-subset size used inside Φ
d	Sketch dimension, set to d=64
$e_i, e_x \in \mathbb{R}^d$	Sketched embeddings of pool member i and candidate x
E_P	Stack of $\{e_i\}$ for the current pool
Π_P	Ridge-stabilized subspace operator summarizing the occupied sketch row span
λ	Ridge coefficient in Π_P
β_P	Pool combination weights used to form $\hat{y}_P$. This is distinct from preference w
$\mathrm{stats}(P)$	Pool-size and weight-concentration features $[u(P),\ p_{\max}(P),\ H(P),\ n_{\mathrm{eff}}(P)]$
$\mathrm{IC}_{\mathrm{ens}}$	Training-window IC of $\hat{y}_P$ against y
h_s	Graph-level embedding of the partial AST
z_{pool}, z_w	Embeddings of c_P and w
$\log Z(c_P, w)$	Conditional log-partition function
P_B	Uniform backward model over syntax. It is not conditioned on c_P
R_{IC}	Standalone training-window IC of candidate x
$\mathrm{MG}_{\mathrm{cv}}$	Cross-fitted marginal combination-IC gain
K	Number of time blocks in embargoed cross-fitting
A_k, B_k	Fit and held-out blocks of fold k, separated by a five-day embargo
$\mathrm{ResIC}_{\mathrm{cv}}$	K-fold cross-fitted residual IC
$F(x \mid P)$	Objective vector $[R_{\mathrm{IC}},\ \mathrm{MG}_{\mathrm{cv}},\ \mathrm{ResIC}_{\mathrm{cv}}]$
$\bar{F}$	F after training-only center/scale normalization
g	Gate flag stored in replay. The value is 1 if pairwise-corr or residual-energy failed
ε	Floor reward 10^{-4} for gated-out trajectories
δ_{e}	Residual-energy stabilizer used by the residual-energy gate
w_{select}	Frozen uniform preference $(1/3, 1/3, 1/3)$ used for live acquisition ranking
P_t	Live pool written by insertion
P_c	Frozen pool snapshot used in TB replay. It may be the live pool, a buffered snapshot, or a random sub-pool
θ	Trainable generator parameters
n_{acq}	Number of candidates sampled per acquisition cycle
n_{pilot}	Size of the empty-pool pilot archive used to freeze the F normalizer
$\mathrm{RankIC}, \mathrm{RankICIR}$	Spearman IC and its information ratio for evaluation and combo eligibility

A.2 Operator Library

The table below summarizes all available features and operators we use in AlphaPSC. The look-back length δ denotes the past δ trading days. Rolling lags δ are restricted to $\{10, 20, 30, 40, 50\}$ trading days. When an expression needs a numeric operand, the constant is chosen from $\{-30, -10, -5, -2, -1, -0.5, -0.01, 0.01, 0.5, 1, 2, 5, 10, 30\}$. Expression length is capped at 20 tokens.

Table 5: Raw features and operators. F means base market features. U/B means unary/binary operators. CS means cross-sectional operation within a day across assets. TS means time-series operation over a rolling window.

Name	Type	Description
Open	F	Opening price
Close	F	Closing price
High	F	Daily highest price
Low	F	Daily lowest price
Vwap	F	Daily average price, weighted by the volume of trades
Volume	F	Trading volume as number of shares
Abs	U	Absolute value of the input
SLog1p	U	Signed log transform: $\mathrm{sign}(x)\log(1+\lvert x\rvert)$
Inv	U	Reciprocal of the input
Sign	U	Sign of the input, returning -1, 0, or 1
Log	U	Natural logarithm of the input
Rank	U-CS	Cross-sectional percentile rank within a day
Add	B	Element-wise addition of two inputs
Sub	B	Element-wise subtraction
Mul	B	Element-wise multiplication
Div	B	Element-wise division
Pow	B	Element-wise power
Greater	B	Element-wise maximum of two inputs
Less	B	Element-wise minimum of two inputs
Ref	U-TS	Lag operator: the value from δ days ago
TsMean	U-TS	Rolling mean over the past δ days
TsSum	U-TS	Rolling sum over the past δ days
TsStd	U-TS	Rolling standard deviation over the past δ days
TsIr	U-TS	Rolling information ratio over the past δ days
TsMinMaxDiff	U-TS	Rolling range over the past δ days ($\max - \min$)
TsMaxDiff	U-TS	Current value minus the rolling max over the past δ days
TsMinDiff	U-TS	Current value minus the rolling min over the past δ days
TsVar	U-TS	Rolling variance over the past δ days
TsSkew	U-TS	Rolling skewness over the past δ days
TsKurt	U-TS	Rolling kurtosis over the past δ days
TsMax	U-TS	Rolling maximum over the past δ days
TsMin	U-TS	Rolling minimum over the past δ days
TsMed	U-TS	Rolling median over the past δ days
TsMad	U-TS	Rolling mean absolute deviation from the window mean over the past δ days
TsRank	U-TS	Rolling percentile rank of the current value within the past δ days
TsDelta	U-TS	Change over δ days: current value minus the value δ days ago
TsDiv	U-TS	Current value divided by the rolling mean over the past δ days
TsPctChange	U-TS	Percentage change over the window: $(x_t - x_{t-\delta+1})/x_{t-\delta+1}$
TsWMA	U-TS	Linearly decaying weighted moving average over the past δ days
TsEMA	U-TS	Exponential moving average over the past δ days
TsCov	B-TS	Rolling covariance between two inputs over the past δ days
TsCorr	B-TS	Rolling Pearson correlation between two inputs over the past δ days

A.3 RGCN Relation Types

The partial AST is encoded by a 2-layer RGCN with six edge types, matching the typed grammar. An edge always points from an operand node to its parent operator. Table 6 lists the types.

Table 6: RGCN edge types on the formula AST.

Type	Name	When it is used
0	Unary operand	Child of Abs, SLog1p, Inv, Sign, Log, Rank
1	Commutative binary	Either child of Add or Mul
2	Left operand	Left child of Sub, Div, Pow, Greater, Less
3	Right operand	Right child of the same non-commutative operators
4	Feature operand	Non-lag child of a rolling or pair-rolling operator
5	Time operand	The lag δ of a rolling or pair-rolling operator

A.4 Training and Insertion Loops

Algorithm 1 separates network updates from live pool writes. TB trains on frozen pool snapshots P_c and never mutates the live pool P_t. Insertion ranks candidates for P_t without an optimizer step. The validation block is used only to freeze second-stage combiner hyperparameters and factor-selection rules, and to log validation IC.

Algorithm 1 AlphaPSC Training and Insertion Loops

Input: Training panel (X, y), pool capacity C, episodes E, insertion period f, candidates per insertion n_{acq}
Output: Final live pool P_t
Initialization:
1. Set $P_t \leftarrow \emptyset$. Initialize the context buffer, trajectory replay buffer, and generator parameters θ.
2. Fit the training-only normalizer of F on a pilot archive and freeze it.
Main loop: for $e = 1$ **to** E **do**
% Training loop: Trajectory Balance on a frozen pool snapshot
3. Sample preference $w \sim \text{Dirichlet}(\alpha)$.
4. Sample frozen snapshot P_c as P_t, a random sub-pool of P_t, or a buffered snapshot.
5. Build pool occupancy summary $c_P = c_P(P_c)$ on the training window.
6. Sample $x \sim P_F(\cdot \mid s_t, c_P, w)$ under the grammar mask.
7. Evaluate $F(x \mid P_c)$ by K-fold embargoed cross-fitting. Apply the pairwise-correlation and residual-energy gates, set g, and form $R_w(x \mid P_c, w)$.
8. Take one Trajectory Balance step on the generated trajectory.
9. **if** the replay buffer is nonempty and with probability 0.5 **then**
a. Sample a stored trajectory (x, P_c, c_P, w, F, g).
b. Recompute $\log P_F(\cdot \mid s_t, c_P, w)$ under current θ. Rebuild R_w from the stored F, w, frozen normalizer, and g. Do not re-evaluate gates.
c. Take one auxiliary Trajectory Balance step.
10. Store (x, P_c, c_P, w, F, g) in the replay buffer.
% Insertion loop: admit at most one factor; no parameter update
11. **if** $e \bmod f = 0$ **then**
a. Freeze P_t and rebuild $c_P = c_P(P_t)$ on the training window.
b. Sample n_{acq} candidates from $P_F(\cdot \mid s_t, c_P, w_{\text{select}})$ with $w_{\text{select}} = (1/3, 1/3, 1/3)$.
c. Drop candidates that fail the correlation gate or the residual-energy gate.
d. Let $\mathcal{C}_{\text{pass}}$ be the survivors. **if** $\mathcal{C}_{\text{pass}} = \emptyset$ **then** skip insertion.
e. $x^\star \leftarrow \arg\max_{x \in \mathcal{C}_{\text{pass}}} R_w(x \mid P_t, w_{\text{select}})$.
f. **if** $|P_t| < C$ **then** $P_t \leftarrow P_t \cup \{x^\star\}$
g. **else**
i. $x_{i^\star} \leftarrow \arg\min_{x_i \in P_t} R_w(x_i \mid P_t \setminus \{x_i\}, w_{\text{select}})$
ii. **if** $R_w(x^\star \mid P_t, w_{\text{select}}) > R_w(x_{i^\star} \mid P_t \setminus \{x_{i^\star}\}, w_{\text{select}})$
then $P_t \leftarrow (P_t \setminus \{x_{i^\star}\}) \cup \{x^\star\}$
end for
return P_t

The training loop updates only θ on frozen snapshots P_c. The insertion loop writes at most one factor into the live pool P_t and does not update θ. Both loops build c_P and F on the training window only. After E episodes, P_t is held fixed for validation and test.

A.5 HYPERPARAMETERS SETTING

Our default hyperparameter configurations are detailed in Table 7.

Table 7: Default AlphaPSC training hyperparameters for all three universes unless noted.

Quantity	Value
Episodes	10^4
Pool capacity C	50
Encoder	2-layer RGCN, hidden size 128
Sketch dimension d	64
Subset size s	2048
Sketch seed λ	0
Subspace ridge	10^{-3}
Mining ridge for residual and MG	0.1
Dirichlet preference α	$(1, 1, 1)$
Entropy coefficient	0.01
Cross-fitting folds K	5
Embargo	5 days
Acquisition period f / candidates n_{acq}	every 64 episodes / 16
Trajectory replay probability	0.5
Pairwise $\lvert\text{corr}\rvert$ gate for reward / insert	0.5 / 0.3
Residual-energy gate / δ_{e}	0.05 / 10^{-8}
Floor reward ε	10^{-4}
Pilot archive for $\bar{F}$	64 training-only samples
Learning rate	10^{-4}
Train / valid / test	2010–2020 / 2021-01-01–2021-12-31 / 2022-01-01–2026-06-30

Objective normalizer. Before the main loop we sample n_{pilot}=64 training-only candidates from the empty-pool context with independently drawn Dirichlet preferences w, evaluate raw F on the training window, and freeze a robust normalizer. Per-coordinate center is the median, and scale is $1.4826\times$ median absolute deviation MAD, floored at 10^{-4}, then clip standardized coordinates at ± 3. The pilot uses only the empty pool P, not multiple live pool sizes. It does span multiple w. Validation and test never enter this fit.

Computational resources and missing values. Experiments use the standard Qlib A-share pipeline. Daily cross-sectional correlations mask name-days where either series is NaN and compute Pearson on the rest. Days with too few finite pairs contribute 0 after nan_to_num. Ridge fits replace non-finite entries by 0 on the flattened design. Day-wise z-scores use finite-name means and standard deviations. The long-only top-top-20% portfolio requires at least one finite name.

Two combiners. Mining uses a lightweight ridge combiner on flattened training panels to form $\hat{y}_P$, IC_{ens}, residual maps $\Phi(\text{vec}(y - \hat{y}_P))$, ResIC_{cv}, and MG_{cv}. Reported test metrics use a separate expanding-window second-stage least-squares combiner and validation-selected and frozen hyperparameters.

B REPORTING PROTOCOL

B.1 REPORTED METRICS

Prediction target. The prediction label is the h-day forward return, with a fixed horizon $h = 20$ trading days. For stock n on trading day d, the label is defined as

$$y[d, n] = \frac{\text{Close}[d + h,\, n]}{\text{Close}[d,\, n]} - 1. \tag{20}$$

All correlation metrics and the portfolio backtests presented later use this 20-day forward return as the target.

Correlation metrics. We evaluate signal performance with four cross-sectional metrics. Let ρ_d denote the daily Pearson correlation between combined scores and $y[d, \cdot]$ on trading day d, and let ρ_d^{rank} denote the corresponding Spearman rank correlation. The four aggregate metrics are defined as follows.

IC: time-series average of daily Pearson correlations. It measures the overall linear predictive power of alpha scores relative to future returns.

$$\text{IC} = \mathbb{E}_d\left[\rho_d\right] \tag{21}$$

ICIR: mean IC divided by its time-series standard deviation. Analogous to the Sharpe ratio, it quantifies the temporal stability of predictive performance; higher values indicate more consistent returns prediction.

$$\text{ICIR} = \frac{\mathbb{E}_d\left[\rho_d\right]}{\sqrt{\text{Var}_d\left(\rho_d\right)}} \tag{22}$$

Rank IC: time-series average of daily Spearman rank correlations. It captures the ability to correctly rank stocks by future returns, and is more robust to extreme outliers than Pearson IC.

$$\text{RankIC} = \mathbb{E}_d\left[\rho_d^{\text{rank}}\right] \tag{23}$$

Rank ICIR: mean rank IC divided by its time-series standard deviation. It measures the temporal stability of stock ranking performance.

$$\text{RankICIR} = \frac{\mathbb{E}_d\left[\rho_d^{\text{rank}}\right]}{\sqrt{\text{Var}_d\left(\rho_d^{\text{rank}}\right)}} \tag{24}$$

Portfolio metrics. We construct a long-only portfolio from alpha signals. On each trading day d, we rank stocks by their combined score, remove missing values, and let $\mathcal{T}_d$ denote the set of stocks in the top 20%, with the constraint $|\mathcal{T}_d| \geq 1$.

The equal-weighted daily portfolio return contribution is defined as

$$R_d = \frac{1}{h} \cdot \frac{1}{|\mathcal{T}_d|} \sum_{n \in \mathcal{T}_d} y[d, n], \qquad h = 20. \tag{25}$$

Here $y[d, n]$ is the h-day forward return of stock n. Division by h corresponds to a staggered long-only portfolio that opens one $1/h$ tranche each trading day and holds each tranche for h days. This makes R_d a daily contribution rather than a full h-day holding period return.

We compute three standard portfolio performance metrics.

Annualized return (AR): the annualized level of frictionless portfolio profit.

$$\text{AR} = 252\, \mathbb{E}_d\left[R_d\right] \tag{26}$$

Sharpe ratio (SR): the risk-adjusted return assuming a zero risk-free rate.

$$\text{SR} = \sqrt{252}\, \frac{\mathbb{E}_d\left[R_d\right]}{\sqrt{\text{Var}_d\left(R_d\right)}} \tag{27}$$

Maximum drawdown (MDD): the worst peak-to-trough loss along the cumulative wealth path. Cumulative wealth follows the arithmetic accumulation

$$W_0 = 1, \qquad W_t = 1 + \sum_{d \le t} R_d. \tag{28}$$

Maximum drawdown is then given by

$$\mathrm{MDD} = \min_{u \le v} \frac{W_v - W_u}{W_u}. \tag{29}$$

MDD is always non-positive, and values closer to zero indicate shallower drawdowns and better downside performance.

B.2 Second-Stage Combination

After mining, every formulaic method is scored by one shared second-stage combiner. Hyperparameters and factor-selection rules are frozen on validation. Combination coefficients are re-estimated each evaluation day on an expanding window that uses only labels available before that day. The mined pool P is frozen after training-window mining. Its membership does not change on validation or test dates, and post labels never enter generation, c_P, F, or admission. This second-stage combiner is distinct from the lightweight ridge combiner used inside AlphaPSC to build c_P and mining rewards. MLP, LightGBM, and XGBoost skip combination and emit a return score directly.

Frozen validation settings. We choose the following settings once by validation IC on 2021-01-01–2021-12-31, then freeze them for all later dates: n_{factors}=20, $|\mathrm{RankIC}| \ge 0.015$, $|\mathrm{RankICIR}| \ge 0.15$, $\mathrm{select_by} = \mathrm{pool_order}$, and $\mathrm{combo_features} = \mathrm{raw_cs_rank}$. Here $\mathrm{pool_order}$ keeps eligible factors in their pool admission order before truncating to n_{factors}. And $\mathrm{raw_cs_rank}$ uses each factor's raw score concatenated with its same-day cross-sectional rank as combiner features.

Daily expanding-window fit. On each validation or test day d, we fit ordinary least squares on history strictly before the label horizon for d. The window is shifted back by $h{+}1{=}21$ sessions, so no fitting label overlaps the prediction target. Features are factor raw scores concatenated with same-day cross-sectional ranks. The history may include training days, the validation block, and earlier test days. It never includes returns on or after the target for d. Factors that pass the frozen $|\mathrm{RankIC}|$ and $|\mathrm{RankICIR}|$ gates are kept in pool order and truncated to n_{factors}. Linearly dependent columns are dropped before the solve. The fitted weights produce the combined score used for IC and for the long-only top-20% portfolio.

B.3 Baselines

All methods use the same chronological split, train through 2020, validate 2021-01-01–2021-12-31, and test 2022-01-01–2026-06-30, with seed 0 unless noted. Formulaic miners are scored by the shared second-stage combiner in Appendix B.2. Test labels are used only for the final report metrics in Tables 1–2 and for the correlation plots in Appendix C.2. They never enter mining, checkpointing, or combiner hyperparameter choice.

- **MLP.** A multilayer perceptron that maps the same price–volume features to the 20-session forward return (Murtagh, 1991). It is a standard deep end-to-end predictor: flexible nonlinear fitting, but no symbolic formula and no explicit pool construction.
- **LightGBM.** A gradient-boosted decision-tree model designed for fast histogram-based training on tabular data (Ke et al., 2017). It is a strong non-deep tabular baseline in quantitative prediction, again producing scores.
- **XGBoost.** A regularized boosting tree model widely used for structured financial features (Chen & Guestrin, 2016). Relative to a plain MLP, it emphasizes split-based interactions and strong regularization on noisy cross-sectional panels, still without formulaic outputs.

- **GP.** Genetic programming searches expression trees by mutation and crossover over the shared operator vocabulary (Chen et al., 2021). It is the classical evolutionary formula miner, that symbolic and interpretable, but candidates are scored mainly by standalone fitness, with redundancy handled after search.
- **AlphaGen.** A PPO-based token-level formula miner (Yu et al., 2023; Schulman et al., 2017). Its main idea is to treat formula construction as sequential decision making and to reward the IC of the combined pool. The pool therefore enters the objective, but only as sparse feedback after a complete formula is generated.
- **AlphaQCM.** A distributional RL formula miner (Zhu & Zhu, 2025). It keeps the RL mining setting of AlphaGen, and replaces point value estimates by a distributional critic to stabilize learning under noisy, sparse pool rewards.
- **AlphaAgent.** A multi-agent LLM pipeline for formula discovery (Tang et al., 2025). Agents propose and refine expressions in natural-language loops. The innovation is language-model search and collaboration, while the current pool is still used mainly after proposals are written, not as a conditioner inside generation.
- **AlphaSAGE.** A structure-aware GFlowNet miner for formulaic alphas (Chen et al., 2026). It replaces reward maximization by Trajectory Balance sampling of diverse high-reward formulas, and encodes expression structure with a relational graph network. Novelty and structure-aware rewards encourage diversity.

C Supplementary Examples and Figures

C.1 Random Seeds

Table 8 presents multi-seed results on CSI300 using the shared second-stage combiner. AlphaPSC achieves the highest mean IC, ICIR, AR and SR across repeated random runs. Performance variation across seeds remains small. The multi-seed evidence shows the performance gain is stable and not caused by a single favorable random trial.

Table 8: Multiple seeds check on CSI300. Entries are mean $\pm$ std over multiple seeds.

Method	IC	ICIR	AR	SR
AlphaGen	0.028 ± 0.004	0.166 ± 0.011	$3.50 \pm 0.3\%$	0.83 ± 0.115
AlphaQCM	0.015 ± 0.002	0.10 ± 0.012	$3.39 \pm 0.3\%$	0.74 ± 0.100
AlphaSAGE	0.041 ± 0.003	0.269 ± 0.016	$7.38 \pm 0.4\%$	1.70 ± 0.123
AlphaPSC (ours)	0.049 ± 0.003	0.324 ± 0.024	$8.63 \pm 0.5\%$	1.92 ± 0.125

C.2 Cross-Correlation Heatmaps

Figure 3 visualizes factor redundancy on CSI300 under the shared second-stage selection rule. For each method we form the 20×20 matrix of pairwise day-averaged cross-sectional Spearman correlations on the test window, and display it after hierarchical clustering so correlated blocks are grouped together. As a scalar summary on the same matrices, Table 9 reports the pool-average pairwise absolute correlation

$$\overline{|\rho|} \;=\; \frac{2}{n(n-1)} \sum_{1\le i<j\le n} \big|\rho_{ij}\big|, \tag{30}$$

with n=20. Lower $\overline{|\rho|}$ means the selected factors are less pairwise aligned on average, so the pool is more complementary rather than a set of near-duplicates.

GP is an extreme case, $\overline{|\rho|}$ reaches 0.903, and the clustered matrix is dominated by near-uniform high correlations. Evolutionary search easily fills the pool with highly homogeneous expression variants that look different as formulas but move together as signals. AlphaPSC records the lowest $\overline{|\rho|}$ among the six methods, and its heatmap shows fewer large high-correlation blocks after clustering. Together with the higher combined IC, this shows that AlphaPSC mines weakly correlated and complementary factors that add distinct predictive directions to the combined signal.

Table 9: Pool-average pairwise absolute correlation $\overline{|\rho|}$ on CSI300.

Method	$\overline{\lvert\rho\rvert}$
GP	0.903
AlphaGen	0.167
AlphaQCM	0.106
AlphaAgent	0.128
AlphaSAGE	0.123
AlphaPSC (ours)	**0.076**

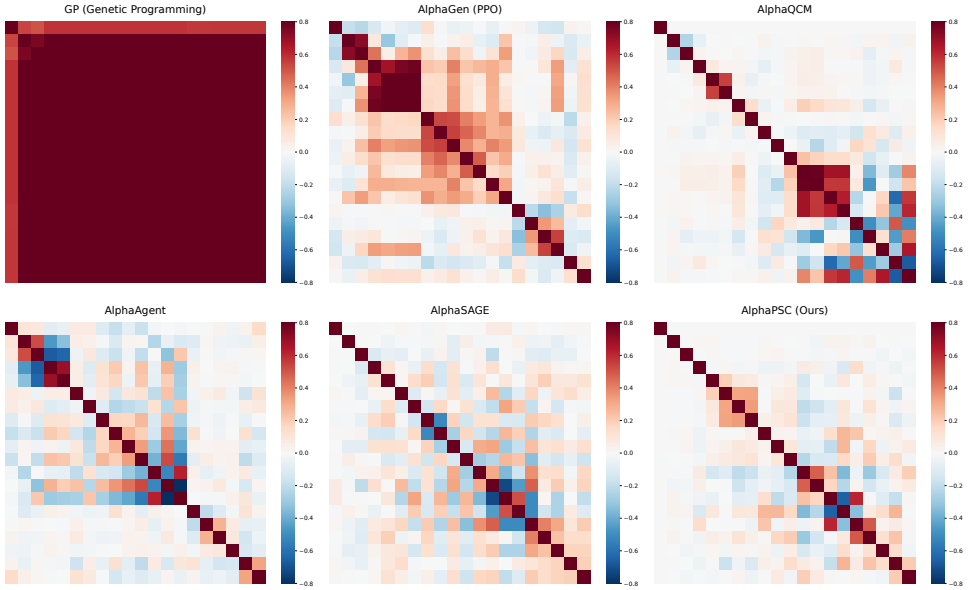
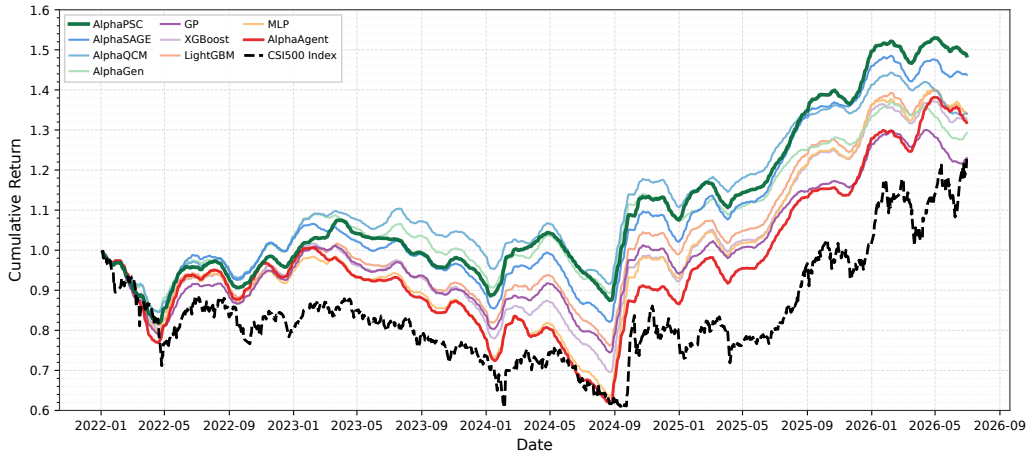


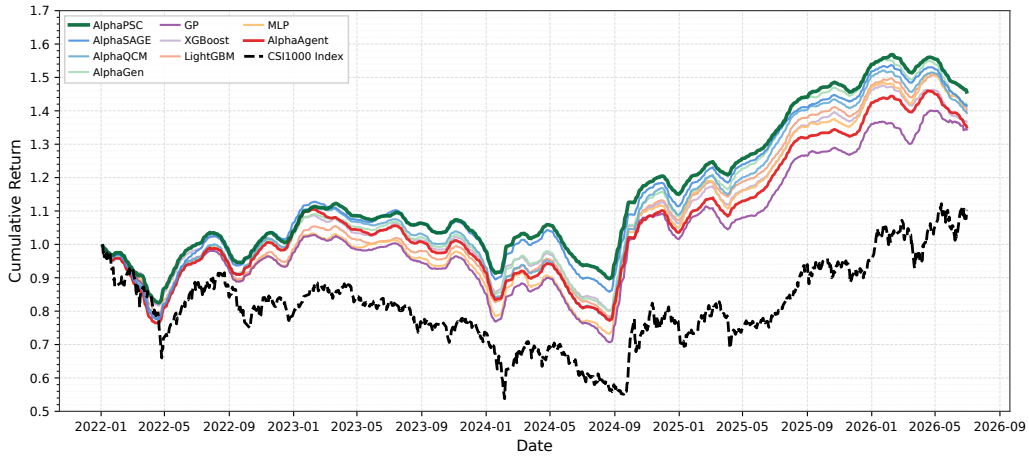
Figure 3: Factor cross-correlation heatmaps on CSI300.

C.3 Cumulative Returns on CSI500 and CSI1000

We repeat the CSI300 equity-curve evaluation of Figure 2 on CSI500 and CSI1000. The test window is 2022-01-01–2026-06-30, and all methods use the same long-only top-20% portfolio. Figure 4 shows the resulting paths. On both universes, AlphaPSC stays ahead of every formulaic and end-to-end baseline over most of the window, with the clearest late-window separation on CSI1000. These out-of-sample gains outside CSI300 indicate that pool-conditioned complementary mining transfers across index universes, and that the method remains effective under broader A-share coverage.



(a) Cumulative return on CSI500.



(b) Cumulative return on CSI1000.

Figure 4: Cumulative returns on CSI500 and CSI1000.

C.4 Combiner Capacity Sweep

Under the shared expanding-window second-stage combiner, we vary capacity $n_{\text{factors}}{=}k$ on the fixed CSI300 pools while keeping the same gates and daily OLS fits. Figure 5 plots combined Test IC against k. AlphaPSC is the only method whose combined Test IC increases steadily with k, and the curve approaches a plateau near the point $k{=}20$. The other methods exhibit non-monotonic changes in combined IC as capacity increases. The continued IC gains under larger k indicate that AlphaPSC admits complementary factors into the pool.

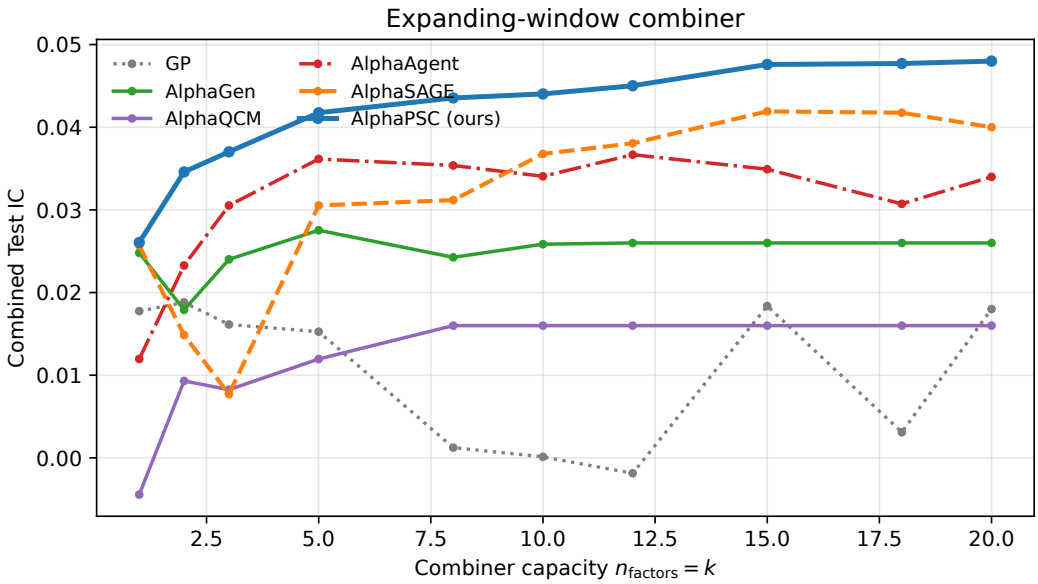


Figure 5: Combined IC under Different Combiner Capacities.