ALPHAPADI: FORMULAIC ALPHA DISCOVERY VIA POOL-AWARE HIERARCHICAL DISCRETE DIFFUSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Formulaic alpha discovery seeks symbolic expressions that predict cross-sectional asset returns. In deployment, multiple formulas are combined into an alpha pool, where each formula is valued through the complementary information it contributes to joint predictive performance. While Reinforcement Learning (RL) and Generative Flow Networks (GFlowNets) have emerged as promising approaches for generating formulaic alphas, existing frameworks face three related challenges. First, generating formulas individually leaves pool context and inter-formula complementarity outside the generative state. Second, formula-wise generation lacks a unified mechanism for preserving and revising structures at different levels. Third, pool-level rewards jointly reflect predictive performance and redundancy but cannot be differentiated directly through symbolic evaluation to train the generator. To overcome these challenges, we introduce **AlphaPADI** (Formulaic **Alpha** Discovery via **P**ool-**A**ware Hierarchical Discrete **DI**ffusion), a novel framework built around three components: (1) grammar-constrained buffer initialization that constructs syntactically valid pool candidates, (2) pool-aware hierarchical diffusion that reconstructs complete pools at multiple structural scales under the current pool context, and (3) reward-guided pool refinement that evaluates joint predictive performance and inner diversity, updates the elite buffer, and trains the reverse model through reconstruction and preference learning. Empirical results on the Chinese and U.S. stock markets demonstrate that AlphaPADI outperforms the evaluated baselines in both predictive and portfolio performance, thereby validating pool-aware generation as an effective framework for automated alpha discovery. Our code is available at `https://anonymous.4open.science/r/alphapadi-code`.

1 INTRODUCTION

Formulaic alpha discovery aims to mine interpretable symbolic signals that predict cross-sectional asset returns and support systematic portfolio construction. These predictive signals are commonly referred to as *alphas* (Kakushadze, 2016). A formulaic alpha represents such a signal as a symbolic expression over market variables, such as $\mathrm{TsMean}(\mathrm{Close}, 5) - \mathrm{Open}$. This explicit structure makes the signal interpretable and its computation auditable (Yang et al., 2020). Discovering effective formulas requires searching the combinatorial space of features, operators, and temporal windows (Zhang et al., 2020). In practice, multiple alphas are deployed as a pool whose joint prediction drives asset ranking and portfolio construction (Yu et al., 2023). The practical goal is therefore not merely to find strong individual formulas, but to identify formulas that contribute complementary predictive information when combined.

Traditionally, formulaic alpha discovery has followed a manual, hypothesis-driven process. Researchers formulate financial or economic hypotheses, translate them into symbolic expressions, and assess their predictive power through statistical tests and backtesting (Fama & French, 1993). This process produces interpretable signals grounded in domain knowledge, but the number of hypotheses that can be examined is limited by human effort and prior experience. As financial data become increasingly high-dimensional, the space of nonlinear interactions among variables, operators, and temporal windows grows beyond the reach of manual enumeration (Gu et al., 2020). Manual discovery also makes it difficult to search systematically for complementary formulas in-

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.

stead of repeatedly expressing similar market effects. These limitations have motivated automated alpha mining, in which learning algorithms search large formula spaces at scale.

Automated alpha mining explores formula space through evolutionary search, reinforcement learning, and generative modeling (Shi et al., 2025a). Genetic algorithms evolve formulas through mutation, crossover, and fitness-based selection (Koza, 1994; Zhang et al., 2020; Chen et al., 2021). Reinforcement learning formulates expression construction as a sequential decision process and can score a formula by its contribution to a downstream combination model (Yu et al., 2023; Zhu & Zhu, 2025; Zhao et al., 2025). Generative flow networks instead learn to sample complete formulas with probabilities proportional to their rewards, encouraging diverse coverage of high-reward expressions (Bengio et al., 2021; Chen et al., 2026). Despite their different search mechanisms, these approaches typically retain the individual formula as the generative object. This formula-wise paradigm creates three related challenges. First, pool context and inter-formula complementarity remain outside the generative state, so they cannot determine which formula structure should be modified (e.g., *replacing a moving-average subtree when the pool already contains similar signals*). Second, sequential neural generators lack a unified mechanism for preserving useful structures while revising them at the leaf, subtree, or whole-formula level (Kusner et al., 2017; Kapur et al., 2025). Third, pool-level rewards obtained through symbolic execution and joint fitting provide no direct gradients for learning structural revisions guided by pool context (Black et al., 2024). These challenges motivate incorporating complete pools into both structural generation and learning.

To overcome these challenges, we introduce **AlphaPADI** (Formulaic **Alpha** Discovery via **P**ool-**A**ware Hierarchical Discrete **DI**ffusion), a framework that treats the complete alpha pool as the common unit of generation, evaluation, and learning. First, grammar-constrained buffer initialization constructs syntactically valid pool candidates. Second, pool-aware hierarchical diffusion (Austin et al., 2021; Sahoo et al., 2024) reconstructs complete pools at the leaf, subtree, and alpha levels under the current pool context, allowing pool members to guide structural revisions. Third, reward-guided pool refinement evaluates joint predictive performance and inner diversity, updates an elite buffer, and trains the reverse model through reconstruction and preference learning (Wallace et al., 2024). Together, these components form a closed loop in which pool-level feedback shapes subsequent candidate generation. Our main contributions are summarized as follows:

- We reformulate alpha discovery as joint pool search and make pool context part of the generative state. This formulation allows inter-formula complementarity to govern structural revisions rather than serving only as an evaluation signal.
- We propose **AlphaPADI**, which combines grammar-constrained buffer initialization, pool-aware hierarchical diffusion, and reward-guided pool refinement. Reconstruction and pool-level preference learning jointly preserve valid structures and steer generation toward higher-reward pools.
- Extensive experiments across the Chinese and U.S. stock markets demonstrate the effectiveness of **AlphaPADI** in discovering high-quality alpha pools. Remarkably, **AlphaPADI** ranks first on every reported predictive metric while achieving the highest annualized return and Sharpe ratio, as well as the lowest maximum drawdown, across all three datasets.

2 Related Work

2.1 Formulaic Alpha Discovery in Quantitative Trading

A formulaic alpha maps historical market variables to a cross-sectional score, whose predictive quality is commonly measured by its information coefficient (IC) with subsequent returns (Kakushadze, 2016; Yu et al., 2023). Alphas are often deployed as a linearly combined pool in which correlated members add little information, so useful alphas must be both predictive and complementary. Evolutionary methods evolve expression trees through mutation, crossover, and selection (Koza, 1994; Cui et al., 2021; Zhang et al., 2020), but they can be inefficient and converge prematurely. Reinforcement learning generates formulas token by token. AlphaGen rewards each formula by its marginal contribution to the combined pool (Yu et al., 2023), and later methods improve its policy optimization (Zhao et al., 2025) and value estimation (Zhu & Zhu, 2025). AlphaForge adds dynamic factor combination (Shi et al., 2025a), AlphaSAGE samples structurally diverse formulas over abstract syntax trees (ASTs) with a generative flow network (Chen et al., 2026), and language-model methods refine formulas with backtest feedback (Tang et al., 2025; Shi et al., 2025b).

Across these methods, the pool enters mainly through marginal-contribution rewards, diversity terms, or adaptive combination, while the generator still constructs one formula at a time. These observations motivate frameworks that use the pool to guide generation as well as evaluation. Our design follows this principle: it revises formula subtrees conditioned on the other pool members during generation, and it evaluates and learns from each candidate pool as a whole through its combined predictive performance and internal redundancy.

2.2 Discrete Diffusion and Preference Optimization

Discrete diffusion models reverse corruption processes over categorical objects. D3PM introduces absorbing-state transitions for discrete sequences (Austin et al., 2021). MDLM and MD4 simplify masked diffusion training (Sahoo et al., 2024), SEDD learns discrete diffusion by estimating ratios of the data distribution (Lou et al., 2023), and LLaDA scales masked diffusion to large language models (Nie et al., 2026). Structured variants corrupt categorical node and edge attributes of graphs, as in DiGress (Vignac et al., 2022), or mutate program syntax trees while preserving grammatical validity (Kapur et al., 2025). These formulations corrupt and reconstruct a single sequence, graph, or program. AlphaPADI instead absorbs complete AST subtrees, from individual leaves to whole formulas, and reconstructs each target conditioned on the other members of the alpha pool.

To align diffusion models with downstream objectives, reinforcement learning fine-tunes denoising trajectories with policy gradients (Black et al., 2024), and discrete counterparts backpropagate differentiable rewards through relaxed sampling (Wang et al., 2025). Direct preference optimization (DPO) converts pairwise comparisons into a reference-relative likelihood objective (Rafailov et al., 2023). Diffusion-DPO applies this principle with variational likelihood estimates (Wallace et al., 2024). Discrete extensions apply it to structured binary sequence generation (Borso et al., 2025) and to masked diffusion language models, where sharing noise between the policy and reference estimates reduces variance (Zhu et al., 2026). Our preference signal compares complete pools by combined predictive performance and internal diversity. We evaluate each pair under matched corruption and score both pools with the reconstruction objective used for training. This connects a non-differentiable pool-level financial score to the reverse generator without per-formula rewards.

3 Preliminary

Pool-aware alpha discovery. Pool-reward reinforcement learning (Yu et al., 2023) can formulate individual-formula search through the expected marginal contribution to the current pool:

$$f^*_{\text{new}} = \arg\max_{f \in \mathcal{X}} \mathbb{E}[R_{\text{marg}}(f \mid F)], \quad R_{\text{marg}}(f \mid F) = U(F \uplus [f]) - U(F), \tag{1}$$

where $\mathcal{X}$ is the formula space, F is the current pool, $\uplus$ inserts a formula into the pool, and $U(F)$ denotes the utility of a jointly fitted pool. A formula-level generative flow network (Chen et al., 2026) learns a reward-proportional distribution over individual formulas: $P_\phi(f) \propto R_{\text{form}}(f)$ for $f \in \mathcal{X}$, where P_ϕ is the learned formula distribution and $R_{\text{form}}(f) > 0$ is the reward assigned to formula f. Thus, both formulations retain an individual formula as the generation object, even when pool performance contributes to its reward.

We reformulate alpha discovery as learning a conditional generative model $P_\theta(F' \mid F)$ over the space $\mathcal{M}_K(\mathcal{X})$ of size-K pools of canonically distinct formulas. The model is trained to favor candidates with stronger predictive performance and inner diversity, while using the current pool context to determine how their formulas are modified:

$$F' \sim P_\theta(\cdot \mid F), \quad F, F' \in \mathcal{M}_K(\mathcal{X}), \tag{2}$$

where F' is generated from F through hierarchical corruption and reverse generation under the current pool context. By sampling from this learned pool-conditional distribution, we extend the role of the pool from evaluating completed candidates to guiding their generation, thereby realizing *pool-aware alpha discovery*.

Problem formulation. Following this pool-aware formulation, given a market panel $\{(X_d, y_d)\}$ divided into training, validation, and test periods and a formula language $\mathcal{X}$, the task is to discover an alpha pool $F = [f_1, \ldots, f_K] \in \mathcal{M}_K(\mathcal{X})$. A ridge combiner (Hoerl & Kennard, 1970), fitted

once on the training split and kept fixed afterwards, jointly fits the ranked member signals:

$$\widehat{w}_F = \arg\min_w n_{\mathrm{tr}}^{-1} \|\widetilde{y}_{\mathrm{tr}} - \widetilde{X}_{F,\mathrm{tr}} w\|_2^2 + \gamma \|w\|_2^2,\ \widehat{y}_{F,S} = \widetilde{X}_{F,S} \widehat{w}_F, \tag{3}$$

where $\widetilde{X}_{F,S}$ and $\widetilde{y}_S$ are the stacked tie-aware daily cross-sectional ranks on split S, n_{tr} is the number of eligible training observations, and γ is the regularization coefficient. The pool utility $U_S(F)$ is the mean daily cross-sectional Spearman correlation between $\widehat{y}_{F,S}$ and $\widetilde{y}_S$.

Because executing an unseen formula over the historical panel dominates the search cost, let $\mathcal{A}$ contain the distinct canonical formulas submitted to the evaluator and $\mathcal{E} \subseteq \mathcal{A}$ those with executable, eligible, and nonconstant signals. The objective is formulated as follows:

$$U^\star(F^\star) = \max\{U^\star(F) : F \in \mathcal{M}_K(\mathcal{E})\},\ |\mathcal{A}| \le B, \tag{4}$$

where $U^\star$ is the test utility and B is the budget on distinct canonical submissions. Training data guide search, validation data select the final pool, and test data are used only for final evaluation. The set $\mathcal{A}$ includes first-time submissions from initialization, reverse generation, and duplicate replacement, including failed executions, while cached repeats incur no additional budget.

4 METHODOLOGY

In this section, we propose AlphaPADI, which shifts the unit of alpha discovery from individual formulas to pools of complementary formulas. The key insight is that a formula's value depends on its contribution to the joint predictive performance of its pool. We therefore treat the pool as the common unit of generation, evaluation, and learning. As illustrated in Figure 1, AlphaPADI comprises **buffer initialization** (Section 4.1), **pool-aware hierarchical diffusion** (Section 4.2), and **reward-guided pool refinement** with a joint training objective (Section 4.3). The notation is summarized in Table 3. We introduce these components below.

4.1 BUFFER INITIALIZATION

The elite buffer $\mathcal{Q}$ stores scored pools that serve as starting points for refinement and as training data for the generator. We initialize it through grammar-constrained sampling in Reverse Polish Notation (RPN) (Yu et al., 2023) over a vocabulary $\mathcal{V}$ of 61 feature, operator, window, and constant tokens (Appendix A.1). The tokens BEG and SEP delimit generation, while MASK and FILL are reserved for diffusion. Starting from $s_0 =$ BEG with an empty stack, features and constants push expressions, unary and binary operators consume one and two expressions, and each temporal operator requires a window token. At state s_t, the next token is sampled uniformly from the valid set $\mathcal{V}_{\mathrm{valid}}(s_t)$, and SEP is valid only when the stack holds one complete expression. For example, `(Close 5 TsMean Open Sub SEP)` produces $\varnothing \to [\texttt{Close}] \to [\texttt{Close}, 5] \to [\texttt{TsMean}(\texttt{Close}, 5)] \to [\texttt{TsMean}(\texttt{Close}, 5), \texttt{Open}] \to [\texttt{Sub}(\texttt{TsMean}(\texttt{Close}, 5), \texttt{Open})]$. Sequences that exceed $L_{\max} = 20$ tokens, fail during execution, or produce ineligible or zero-variance signals are resampled.

Initial pools are sampled independently of rewards and external factor libraries. We group K canonically distinct formulas per pool and construct $M \le C_{\mathcal{Q}}$ distinct pools:

$$F_0^{(m)} = \left(f_1^{(m)}, \ldots, f_K^{(m)}\right), \mathcal{Q}_0 = \left\{\left(F_0^{(m)}, R(F_0^{(m)})\right)\right\}_{m=1}^{M}. \tag{5}$$

Each pool is scored by the reward in Eq. 9. All canonical formulas submitted during initialization, including resampled ones, enter $\mathcal{A}$ in Eq. 4, and the scored pools initialize both $\mathcal{Q}$ and the validation archive $\mathcal{C}$. Before its first proposal, the generator performs S_0 updates of Eq. 11 on $\mathcal{Q}_0$.

4.2 POOL-AWARE HIERARCHICAL DIFFUSION

Section 4.1 provides scored pools. Here, we introduce how to revise them in the context of the other pool members: hierarchical corruption masks subtrees at multiple structural scales, and pool-aware reverse generation reconstructs them conditioned on the current pool. Applied to a proposal batch $\mathcal{B}_{\mathrm{c}}$ of N_{c} buffered pools, this process produces candidates for evaluation.

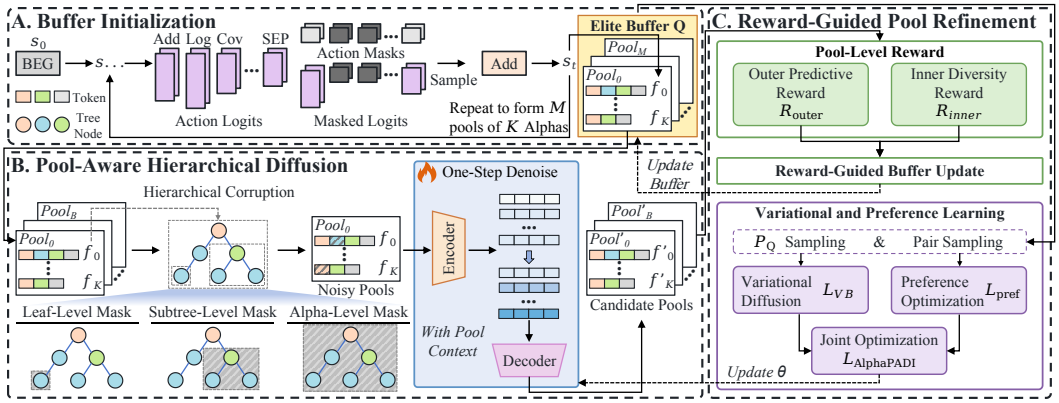

Figure 1: An overview of AlphaPADI. A. Buffer initialization: grammar-constrained RPN sampling forms M scored pools of K formulas in the elite buffer. B. Pool-aware hierarchical diffusion: buffered pools are corrupted at the leaf, subtree, and alpha levels and reconstructed with the current pool as context. C. Reward-guided pool refinement: Candidates scored by R_{outer} and R_{inner} update the buffer, which supplies reconstruction samples and preference pairs for $\mathcal{L}_{\text{AlphaPADI}}$.

Hierarchical corruption. Let $F_0 = (f_1, \ldots, f_K)$ be a clean pool and $\mathbf{m}$ denote MASK. Each formula is parsed into an abstract syntax tree (AST), in which a node is absorbable if its RPN span forms a complete expression. For $f = \text{TsMean}(\text{Close}, 5) - \text{Open}$, serialized as `(Close 5 TsMean Open Sub)`, masking the *leaf* Close, the *subtree* $\text{TsMean}(\text{Close}, 5)$, or the *root* yields $\text{TsMean}(\mathbf{m}, 5) - \text{Open}$, $\mathbf{m} - \text{Open}$, or $\mathbf{m}$, corrupting f at the *leaf*, *subtree*, and *alpha* levels, respectively (Component B in Figure 1). Each absorbable node u receives an independent clock $U_u \sim \text{Uniform}(0, 1)$ shared across levels, and Φ_ρ masks the highest subtrees with $U_u \le \rho$, ancestors taking precedence over descendants. The corrupted pools are formulated as follows:

$$F_t = \Phi_{\rho_t}(F_0), \quad \rho_t = 1 - \cos\big(\pi t/(2T)\big), \quad t \in \{0, \ldots, T\}, \quad F_T = (\mathbf{m}, \ldots, \mathbf{m}). \tag{6}$$

Let H_t denote the exposed masks in F_t and Y_u the state of $u \in H_t$ in F_{t-1}. The shared clocks induce the posterior $q(Y_u \mid F_t, F_0) = a_t \delta_{\mathbf{m}}(Y_u) + w_t \mu_{t-1}^{u,F_0}(Y_u)$ (Proposition B.1), where $a_t = \rho_{t-1}/\rho_t$ and $w_t = (\rho_t - \rho_{t-1})/\rho_t$ are the mask-retention and reveal probabilities, and μ_{t-1}^{u,F_0} is the level-$(t-1)$ distribution of the subtree at u with u visible and its descendants corrupted.

Pool-aware reverse generation. At transition $t \to t-1$, each position in H_t enters the reveal set J_t independently with probability w_t. Selected positions keep their RPN order within each formula and are uniformly interleaved across formulas into $(u_{t,1}, \ldots, u_{t,|J_t|})$, and descendant masks generated at level t are processed at the next level. For the r-th position $u = u_{t,r}$, let $G = G^{(r-1)} = (g_1, \ldots, g_K)$ be the current pool. A shared Transformer ψ mean-pools each formula, a FILL copy of row $i(u)$ marks the queried position, and a set encoder aggregates them:

$$\mathbf{c}_{G,u,t} = \text{Mean}\Big[\text{Enc}_{\text{s}}\Big(\{\psi(g_i)\}_{i=1}^{K}, \psi(g_{i(u)}^{\text{fill}(u)})\Big)\Big] + E_t(t), \tag{7}$$

where $E_t(t)$ embeds the diffusion level, and the context $c_{G,u,t}$ summarizes the current pool G for position u. The autoregressive decoder uses this context to sample a grammar-valid $\hat{Y}_u \sim r_\theta(\cdot \mid G, u, t)$. We update $G^{(r)} = \text{Splice}(G, u, \hat{Y}_u)$ before filling the next mask. This lets each reconstruction use earlier edits and capture dependencies across formulas (Proposition B.2). Because $w_1 = 1$, every remaining mask is reconstructed at $t = 1$, yielding complete and syntactically valid pools.

Candidate generation. Each pool in $\mathcal{B}_{\text{c}}$ is corrupted to $t_0 = \max\{1, \text{round}(\alpha T)\}$, where $\alpha \in (0, 1]$ controls the edit strength, and reconstructed from t_0 to 0. After canonicalization, repeated members are replaced by grammar-valid random formulas so that each pool contains K distinct formulas. Members not yet in $\mathcal{A}$ are submitted to the evaluator, and a candidate whose unseen members would exceed the remaining budget is discarded. Eligible candidates (Section 4.3) are scored and added, together with the initial pools, to the archive $\mathcal{C}$. Search stops when the budget is exhausted or N_{stall} consecutive rounds add no formula to $\mathcal{A}$, and the final pool is $\widehat{F} = \arg\max_{F \in \mathcal{C}} U_{\text{val}}(F)$.

4.3 Reward-Guided Pool Refinement and Training Objective

Section 4.2 produces candidate pools. Here, we score them with a pool-level reward and use this reward in two ways: to refine the elite buffer and to train the reverse model through reconstruction and preference objectives.

Pool-level reward. For each date d, member f_i produces a signal $\mathbf{a}_d(f_i)$, whose tie-aware cross-sectional ranks $\widetilde{\mathbf{a}}_{i,d}$ are computed over assets with finite values for all members and the target. The outer reward is the training performance of the ridge combiner in Eq. 3, whose penalty is selected within the training period. Ridge controls the combination weights but does not prevent redundant formulas from occupying the K pool positions. We therefore measure redundancy by $c_{ij} = |\mathcal{T}_{ij}|^{-1} \sum_{d \in \mathcal{T}_{ij}} |\operatorname{corr}(\widetilde{\mathbf{a}}_{i,d}, \widetilde{\mathbf{a}}_{j,d})|$ and let $d_i = \max_{j \neq i} c_{ij}$. The reward components, the pool reward, and the buffer sampling distribution are formulated as follows:

$$R_{\text{outer}}(F) = \frac{1}{|\mathcal{T}_F|} \sum_{d \in \mathcal{T}_F} \text{Spearman}\big(\widehat{\mathbf{y}}_d(F), \mathbf{y}_d\big), \quad R_{\text{inner}}(F) = -\frac{1}{K} \sum_{i=1}^{K} [d_i - \delta]_+^2, \tag{8}$$

$$R(F) = R_{\text{outer}}(F) + w_{\text{inner}} R_{\text{inner}}(F), \tag{9}$$

$$P_{\mathcal{Q}}(F_i) = \frac{\exp(z_i / T_{\text{r}})}{\sum_j \exp(z_j / T_{\text{r}})}, \quad z_i = \frac{R_i - \overline{R}}{\text{sd}_{\mathcal{Q}}(R) + \epsilon}, \tag{10}$$

where $\mathcal{T}_{ij}$ contains the dates with jointly finite, nonconstant ranked signals ($c_{ij} = 0$ if $\mathcal{T}_{ij} = \varnothing$), $\mathcal{T}_F$ contains the dates with nonconstant pool predictions and targets, $[x]_+ = \max(x, 0)$, $\delta \in [0, 1]$ is the tolerated similarity, $w_{\text{inner}} > 0$ weights diversity, $\overline{R}$ and $\text{sd}_{\mathcal{Q}}(R)$ are the mean and standard deviation of the buffered rewards, $\epsilon > 0$ stabilizes reward standardization, and T_{r} controls the sampling concentration. A pool is excluded if it has no eligible date or a member without a finite, nonconstant training signal. The elite buffer retains the highest-reward canonical pools up to capacity $C_{\mathcal{Q}}$ and samples them from $P_{\mathcal{Q}}$.

Reconstruction objective. To keep the reverse model able to reconstruct high-reward pools, we minimize a variational bound on the negative log-likelihood of buffered pools. Since the reveal probabilities and the reconstruction order are shared by the forward posterior and the reverse model, they cancel, and the bound reduces, up to a constant, to T times the expected negative log-likelihood of the revealed targets under a uniformly sampled level (Proposition B.3). We estimate this bound on a minibatch $\mathcal{B}_{\mathcal{Q}} = \{F_0^{(b)}\}_{b=1}^{N}$ sampled from $P_{\mathcal{Q}}$. For each pool, we draw a level $t_b \sim \text{Uniform}(\{1, \ldots, T\})$ and its reveal set J_b, and teacher-force the targets $Y_{b,r}$, the states of the revealed positions $u_{b,r}$ at level $t_b - 1$, into the pool state $G_b^{(r)}$, starting from $G_b^{(0)} = F_{t_b}^{(b)}$. Normalizing each pool by the number and length of its targets gives:

$$\widehat{\mathcal{L}}_{\text{VB}}(\mathcal{B}_{\mathcal{Q}}, \theta) = \frac{T}{N} \sum_{b=1}^{N} \ell_\theta^{(b)}, \quad \ell_\theta^{(b)} = -\frac{1}{n_b} \sum_{r=1}^{|J_b|} \frac{\log r_\theta\big(Y_{b,r} \mid G_b^{(r-1)}, u_{b,r}, t_b\big)}{L_{b,r}}, \tag{11}$$

where $L_{b,r}$ is the length of $Y_{b,r}$ including SEP and $n_b = \max\{1, |J_b|\}$. The normalization makes $-\ell_\theta^{(b)}$ comparable across pools with different numbers and lengths of targets, which the preference objective below relies on.

Pool-level preference objective. We convert pool rewards into preferences by drawing two distinct pools from $P_{\mathcal{Q}}$, denoting the higher-reward pool by F_{w} and the lower-reward pool by F_{l}, and retaining the pair when $R(F_{\text{w}}) - R(F_{\text{l}}) \geq m_{\text{dpo}} \, \text{sd}_{\mathcal{Q}}(R)$, where $m_{\text{dpo}} \geq 0$ sets the reward margin in units of the buffer reward standard deviation. Ties and pairs with empty reconstruction sets are omitted, and $\mathcal{P}_{\mathcal{Q}}$ denotes the distribution of accepted pairs. Both pools of a pair share the diffusion level and the pseudorandom stream for corruption, reveal selection, and ordering, while each consumes the stream according to its own AST and thus keeps its marginal corruption process. The objective is formulated as follows:

$$\mathcal{L}_{\text{pref}}(\theta) = -\mathbb{E}_{\mathcal{P}_{\mathcal{Q}}} \log \sigma\Big(\beta \Big[\Big(\ell_\theta^{\text{l}} - \ell_\theta^{\text{w}}\Big) - \Big(\ell_{\theta_{\text{ref}}}^{\text{l}} - \ell_{\theta_{\text{ref}}}^{\text{w}}\Big)\Big]\Big), \tag{12}$$

where ℓ_ϑ^a is the single-pool loss of Eq. 11 for $a \in \{\text{w}, \text{l}\}$ and $\vartheta \in \{\theta, \theta_{\text{ref}}\}$, evaluated under the same corruption variables, θ_{ref} is a frozen copy of the reverse model after the S_0 initialization updates, σ is the sigmoid function, and $\beta > 0$ controls the preference strength and absorbs the factor T.

Minimizing Eq. 12 increases the reference-relative reconstruction score of the higher-reward pool, and Proposition B.4 derives it from KL-regularized pool-level reward maximization.

Joint training objective. We combine the preference objective with reconstruction on an independently sampled minibatch:

$$\mathcal{L}_{\text{AlphaPADI}}(\theta) = \mathcal{L}_{\text{pref}}(\theta) + \lambda_{\text{VB}}\widehat{\mathcal{L}}_{\text{VB}}(\mathcal{B}_{\mathcal{Q}}, \theta), \tag{13}$$

where $\lambda_{\text{VB}} > 0$ weights reconstruction, anchoring the generator to the buffered pools while the preference term contrasts them. If no pair is accepted, the update uses reconstruction only. Algorithm 1 in Appendix B.2 summarizes the complete procedure, and Appendix B.3 analyzes its complexity.

4.4 Discussion of Alpha Combination

We evaluate every pool, including those returned by the baselines, with the ridge combiner in Eq. 3, fitted once on the training period and frozen during validation and testing. This aligns evaluation with the predictive component of AlphaPADI's pool-level reward, holds the combination procedure fixed across pools, and uses the same backtesting protocol for all methods (Appendix C.1.1).

5 Experiments

5.1 Experimental Setup

Datasets. We evaluate CSI300, CSI500, and S&P500, representing large-cap Chinese, mid-cap Chinese, and large-cap U.S. equities, respectively. Following formulaic alpha studies (Shi et al., 2025a; Chen et al., 2026), we use Qlib daily market data (Yang et al., 2020) and predict the 20-trading-day forward return. For consistent cross-market evaluation, three universes share a chronological split: 2016-01-01–2020-12-31 for training, 2021-01-01–2022-12-31 for validation, and 2023-01-01–2025-12-31 for testing. Dates whose return horizon crosses a split boundary are excluded.

Evaluation metrics. We evaluate predictive performance using the information coefficient (IC), rank information coefficient (RIC), and their information ratios (ICIR and RICIR), and portfolio performance using annualized return (AR), Sharpe ratio (SR), and maximum drawdown (MDD). Lower MDD is better, and higher values are better for the other metrics. Metric definitions and the shared backtesting protocol are provided in Appendix C.1.2.

Baselines. We compare AlphaPADI against nine baselines spanning the main paradigms of alpha mining, including the predefined factor library Alpha158 (Yang et al., 2020), the supervised predictors MLP (Murtagh, 1991) and XGBoost (Chen & Guestrin, 2016), genetic programming (GP) (Chen et al., 2021), the reinforcement learning methods AlphaGen (Yu et al., 2023) and AlphaQCM (Zhu & Zhu, 2025), the generative-predictive framework AlphaForge (Shi et al., 2025a), the LLM-based miner AlphaAgent (Tang et al., 2025), and the GFlowNet-based AlphaSAGE (Chen et al., 2026). Appendix C.1.3 details all baselines and their settings.

Implementation details. All experiments run on 8 NVIDIA L40S GPUs. Non-LLM automated alpha mining methods share a budget of 40,000 formula evaluations, converted from native units, e.g., 2×10^5 steps (AlphaGen) or 4×10^4 episodes (AlphaSAGE) (Appendix C.1.3 specifies each method's stopping rule). We tune the learning rate over $\{2 \times 10^{-5}, 3 \times 10^{-4}, 1 \times 10^{-3}\}$, w_{inner} over $\{1.0, 2.0, 5.0, 10.0\}$, and λ_{VB} over $\{0.1, 0.3, 0.5, 0.7\}$ on the validation split, and select the pool size $K \in \{20, 50\}$ likewise for non-LLM automated alpha mining baselines and AlphaPADI. We report the mean performance over five runs and assess significance against the strongest baseline at $p < 0.05$. Appendix C.1.4 lists the architecture and hyperparameters, and Appendix C.2.1 analyzes sensitivity to w_{inner} and λ_{VB}.

5.2 Overall Performance

Table 1 reports predictive and portfolio performance on the three stock universes, and Figure 2 shows the CSI300 cumulative returns over the test period. Backtests on CSI500 and S&P500, initialization comparisons, additional portfolio metrics, and search efficiency under the shared budget are provided in Appendices C.2.2, C.2.3, C.2.4, and C.2.5.

Table 1: Predictive and portfolio performance across three equity universes. RIC and RICIR denote RankIC and RankICIR. Higher values are better except for MDD. Best results are **bolded**, and second-best results are underlined.

Method	Year	Predictive Performance				Portfolio Performance		
		IC↑	ICIR↑	RIC↑	RICIR↑	AR↑	SR↑	MDD↓
CSI300								
Alpha158	2020	0.0266	0.1631	0.0250	0.1389	0.0751	2.1532	0.2274
XGBoost	2016	0.0192	0.1024	0.0273	0.1386	0.0591	1.8645	0.2046
MLP	–	0.0254	0.1458	0.0362	0.2047	0.0723	1.8562	0.2755
GP	2021	0.0229	0.2142	0.0326	0.2788	0.0721	1.6105	0.3247
AlphaGen	2023	0.0290	0.1771	0.0380	0.2065	0.0657	2.1058	0.1829
AlphaQCM	2025	0.0352	0.2291	0.0544	0.3371	0.0733	1.4941	0.3787
AlphaForge	2025	0.0276	0.1391	0.0516	0.2501	0.0940	1.5646	0.3669
AlphaAgent	2025	0.0251	0.1617	0.0510	0.3388	0.0713	1.3969	0.3954
AlphaSAGE	2026	0.0228	0.1791	0.0351	0.2656	0.0752	1.5401	0.3939
AlphaPADI	2026	**0.0439**	**0.3024**	**0.0659**	**0.4147**	**0.1058**	**2.8619**	**0.1412**
CSI500								
Alpha158	2020	0.0108	0.0952	0.0163	0.1327	0.1188	2.5419	0.2565
XGBoost	2016	0.0082	0.0564	0.0251	0.1810	0.0957	2.3123	0.2246
MLP	–	0.0086	0.0648	0.0277	0.2126	0.0898	1.9643	0.3198
GP	2021	0.0111	0.0936	0.0251	0.1892	0.1132	2.6883	0.2366
AlphaGen	2023	0.0203	0.1571	0.0379	0.2641	0.1233	2.9673	0.1830
AlphaQCM	2025	0.0167	0.1232	0.0453	0.2996	0.1154	2.5779	0.2190
AlphaForge	2025	0.0184	0.2583	0.0269	0.3484	0.1175	2.2819	0.3198
AlphaAgent	2025	0.0092	0.0693	0.0278	0.2104	0.1158	2.3852	0.2437
AlphaSAGE	2026	0.0235	0.1413	0.0601	0.3477	0.1372	3.1978	0.1687
AlphaPADI	2026	**0.0444**	**0.2855**	**0.0763**	**0.4578**	**0.1435**	**3.5124**	**0.1350**
S&P500								
Alpha158	2020	0.0221	0.1482	0.0194	0.1291	0.1572	3.8617	0.1204
XGBoost	2016	0.0120	0.1079	0.0073	0.0688	0.1498	4.0652	0.1096
MLP	–	0.0175	0.1893	0.0143	0.1588	0.1517	4.5017	0.1041
GP	2021	0.0142	0.3193	0.0113	0.2700	0.1347	4.3319	0.0958
AlphaGen	2023	0.0174	0.1298	0.0103	0.0786	0.1571	3.9775	0.1102
AlphaQCM	2025	0.0270	0.2620	0.0145	0.1458	0.1704	4.7981	0.1064
AlphaForge	2025	0.0315	0.4414	0.0220	0.3122	0.1624	5.3919	0.0952
AlphaAgent	2025	0.0193	0.2153	0.0138	0.1462	0.1428	4.6385	0.0987
AlphaSAGE	2026	0.0319	0.3099	0.0220	0.2151	0.1679	4.6631	0.0997
AlphaPADI	2026	**0.0515**	**0.5020**	**0.0452**	**0.4122**	**0.2061**	**5.8423**	**0.0860**

Obs 1: AlphaPADI achieves the best performance on all seven metrics in all three universes. Its RIC reaches 0.0659, 0.0763, and 0.0452 on CSI300, CSI500, and S&P500, improving over the strongest baseline by 21.1%, 27.0%, and 105.5%, respectively. These gains are also reflected in portfolio metrics: AlphaPADI achieves the highest AR and SR across all three markets, while reducing MDD relative to the best baseline by 22.8%, 20.0%, and 9.7% on CSI300, CSI500, and S&P500, respectively. Under a shared protocol, searching over complete pools yields more robust combinations than generating formulas individually.

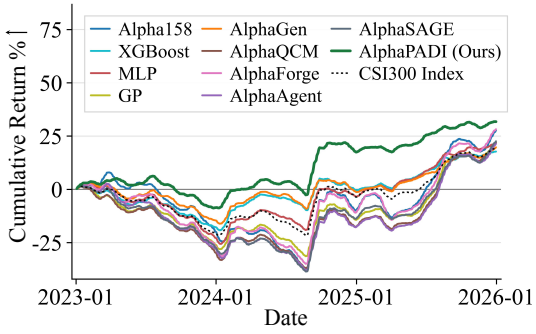

Figure 2: Cumulative returns on CSI300 (2023–2025): AlphaPADI, baseline methods, and the CSI300 index.

Obs 2: No baseline is consistently the strongest. The second-best entries in Table 1 span six baselines, and the leading baseline varies by both universe and metric: AlphaQCM ranks second in IC on CSI300, AlphaSAGE on CSI500, and AlphaForge in ICIR and SR on S&P500.

Obs 3: AlphaPADI leads from the 2024 drawdown through recovery. On CSI300, AlphaPADI has a shallower trough during the decline before September 2024 and finishes above all baselines after the subsequent recovery (Figure 2), consistent with its lower MDD and higher SR.

5.3 Pool-Aware and Component Analysis

Table 2 adds components incrementally on CSI300 under identical budget and evaluation. The base model reconstructs masked subtrees without pool context and, lacking a pool reward, buffers the latest candidates and samples them uniformly. The second row adds pool context $\mathbf{c}_{G,u,t}$. Rows three

Table 2: Ablation study of AlphaPADI on CSI300. The base model is Diffusion w/o pool context. $\mathcal{L}_{\text{VB}}$ denotes pool-aware hierarchical diffusion in Eq. 11. R_{outer} and R_{inner} denote the outer predictive reward and inner diversity reward in Eq. 8, respectively. $\mathcal{L}_{\text{pref}}$ denotes pool-level preference optimization in Eq. 12. Best results are **bolded**, and second-best results are underlined.

Included Components				Predictive Performance				Portfolio Performance		
$\mathcal{L}_{\text{VB}}$	$\mathbf{R}_{\text{outer}}$	$\mathbf{R}_{\text{inner}}$	$\mathcal{L}_{\text{pref}}$	IC↑	ICIR↑	RIC↑	RICIR↑	AR↑	SR↑	MDD↓
✗	✗	✗	✗	0.0215	0.1434	0.0334	0.2060	0.0780	1.9722	0.2708
✓	✗	✗	✗	0.0276	0.2263	0.0386	0.3019	0.0731	1.9039	0.2249
✓	✓	✗	✗	0.0354	0.2684	0.0463	0.3255	0.0881	2.3414	0.2333
✓	✓	✓	✗	0.0396	0.2510	0.0531	0.3130	0.0960	2.6810	0.1915
✓	✓	✓	✓	**0.0439**	**0.3024**	**0.0659**	**0.4147**	**0.1058**	**2.8619**	**0.1412**

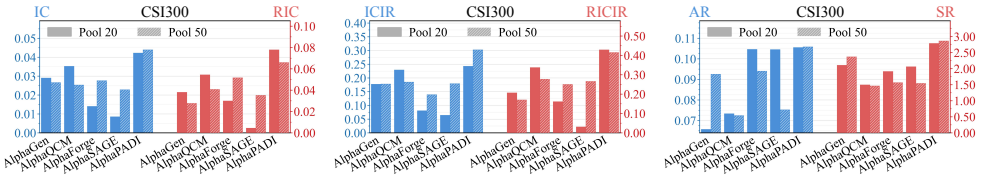

Figure 3: Performance of five alpha-mining methods with pool sizes $K = 20$ (solid) and $K = 50$ (hatched) on CSI300. Blue and red bars use the corresponding left and right metric axes.

and four rank and sample buffered pools by Eq. 9 with $w_{\text{inner}} = 0$ (R_{outer} alone) and $w_{\text{inner}} > 0$. The last row adds $\mathcal{L}_{\text{pref}}$, which needs reward-ordered pairs and therefore follows both rewards.

Obs 4: Every component improves IC and RIC, and preference optimization gives the largest gain in RIC and RICIR. IC and RIC increase monotonically along the sequence, from 0.0215 and 0.0334 to 0.0439 and 0.0659. Adding $\mathcal{L}_{\text{pref}}$ raises RIC from 0.0531 to 0.0659 and RICIR from 0.3130 to 0.4147, the largest single increments of both metrics, which suggests that reward-ordered pool comparisons provide a learning signal beyond reward-guided buffer selection.
Obs 5: The diversity reward mainly improves portfolio risk. Adding R_{inner} reduces MDD from 0.2333 to 0.1915 and raises SR from 2.3414 to 2.6810, while ICIR decreases slightly from 0.2684 to 0.2510. Penalizing redundant members thus trades a small loss in predictive stability for lower drawdown. Pool context alone improves ICIR and RICIR but not AR or SR, indicating that pool-level rewards are needed to turn pool-aware reconstruction into trading gains.

5.4 Effect of Pool Size

Figure 3 compares pools of $K = 20/50$ on CSI300 under the same budget of 40,000 formula evaluations for the five methods that construct alpha pools, whereas Table 1 reports each of them at its validation-selected size. The CSI500 and S&P500 results are provided in Appendix C.2.6.

Obs 6: AlphaPADI leads at both pool sizes. At both $K = 20$ and 50, AlphaPADI attains the highest IC, RIC, ICIR, RICIR, and SR, and its AR is the highest or on par with the strongest baseline.
Obs 7: A larger pool does not uniformly help. From $K = 20$ to 50, predictive and portfolio metrics often move in opposite directions: AlphaForge and AlphaSAGE gain IC and RIC but lose AR and SR, AlphaGen shows the reverse pattern, and AlphaQCM declines on most metrics. For AlphaPADI, IC, ICIR, and SR increase, while RIC and RICIR decrease. Pool capacity should therefore be evaluated through the combined predictor rather than assumed to help, which motivates optimizing the pool-level reward directly.

6 Conclusion

We presented AlphaPADI, a framework that treats the alpha pool as the common unit of generation, evaluation, and learning. Pool-aware hierarchical discrete diffusion modifies formula structures using the current pool context, while predictive and redundancy rewards guide buffer refinement and preference learning. Experiments on three stock universes show higher reported predictive and portfolio performance than the evaluated baselines. Cumulative ablations characterize the incremental effects of the components, and additional analyses examine pool size, initialization, and hyperparameter sensitivity. The findings support pool-level search under the evaluated settings.

AI USE STATEMENT

In this work, we used generative AI tools to obtain feedback on the methodology and experimental evaluation, check the consistency of descriptions of reported results with the tables, and suggest potentially relevant references. We also used these tools to improve the organization and clarity of selected passages, refine English wording, and assist with LaTeX formatting. The authors reviewed all AI-assisted suggestions, decided which to adopt, and finalized the manuscript. The authors take full responsibility for the paper's text, claims, analyses, and results.

ETHICS STATEMENT

AlphaPADI studies formulaic alpha discovery using historical equity-market data from China and the United States. The experiments consist of offline signal evaluation and backtesting; they involve no human participants, personal data, brokerage accounts, or live market orders. The reported results depend on the studied markets, periods, and backtesting assumptions. They should not be interpreted as investment recommendations or guarantees of future returns. Deploying a discovered strategy would require further assessment of execution costs and financial risk.

REPRODUCIBILITY STATEMENT

Section 4 specifies the pool generator, reward, and training objectives, while Appendix A.1 lists the formula vocabulary and Appendix B.2 gives the complete algorithm. Section 5 describes the datasets, chronological splits, baselines, and metrics. Appendices C.1.1–C.1.4 document the model settings, ridge combiner, backtesting protocol, method-specific formula-evaluation budgets, and hyperparameters. We report mean results over five runs, and the paper links the anonymized code. Because initialization, diffusion sampling, and training are stochastic, repeated runs may select different alpha pools.

REFERENCES

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in neural information processing systems*, 34:27381–27394, 2021.

Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. In *International Conference on Learning Representations*, volume 2024, pp. 4965–4987, 2024.

Umberto Borso, Davide Paglieri, Jude Wells, and Tim Rocktäschel. Preference-based alignment of discrete diffusion models. *arXiv preprint arXiv:2503.08295*, 2025.

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*, volume 2026, pp. 2955–2982, 2026.

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, 2016.

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*, pp. 2208–2216, 2021.

Eugene F Fama and Kenneth R French. Common risk factors in the returns on stocks and bonds. *Journal of financial economics*, 33(1):3–56, 1993.

Shihao Gu, Bryan Kelly, and Dacheng Xiu. Empirical asset pricing via machine learning. *The review of financial studies*, 33(5):2223–2273, 2020.

Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Zura Kakushadze. 101 formulaic alphas. *Wilmott*, 2016(84):72–81, 2016.

Shreyas Kapur, Erik Jenner, and Stuart Russell. Diffusion on syntax trees for program synthesis. In *International Conference on Learning Representations*, volume 2025, pp. 41414–41432, 2025.

John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *International conference on machine learning*, pp. 1945–1954. PMLR, 2017.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

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

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *Advances in Neural Information Processing Systems*, 38:50608–50646, 2026.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

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, 2025a.

Yu Shi, Yitong Duan, and Jian Li. Navigating the alpha jungle: an llm-powered mcts framework for formulaic factor mining. *arXiv preprint arXiv:2505.11122*, 2025b.

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, 2025.

Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734*, 2022.

Bram Wallace, Meihua Dang, Rafael Rafailov, Linqi Zhou, Aaron Lou, Senthil Purushwalkam, Stefano Ermon, Caiming Xiong, Shafiq Joty, and Nikhil Naik. Diffusion model alignment using direct preference optimization. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8228–8238. IEEE, 2024.

Chenyu Wang, Masatoshi Uehara, Yichun He, Amy Wang, Avantika Lal, Tommi Jaakkola, Sergey Levine, Aviv Regev, Hanchen Wang, and Tommaso Biancalani. Fine-tuning discrete diffusion models via reward optimization with applications to dna and protein design. In *International Conference on Learning Representations*, volume 2025, pp. 47871–47899, 2025.

Xiao Yang, Weiqing Liu, Dong Zhou, Jiang Bian, and Tie-Yan Liu. Qlib: An ai-oriented quantitative investment platform. *arXiv preprint arXiv:2009.11189*, 2020.

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.

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

Junjie Zhao, Chengxi Zhang, Min Qin, and Peng Yang. Quantfactor reinforce: mining steady formulaic alpha factors with variance-bounded reinforce. *IEEE Transactions on Signal Processing*, 73:2448–2463, 2025.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11425–11460, 2026.

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

A NOTATION

Table 3 summarizes the principal notation used throughout the paper, and Appendix A.1 lists the formula vocabulary.

Table 3: Summary of the principal notation used in AlphaPADI.

Symbol	Definition
Data, pools, and evaluation	
X_d, y_d	Market panel and forward returns on date d
$\mathcal{X}$, f	Formula space defined by the vocabulary $\mathcal{V}$ and a formula in it
$F = [f_1, \ldots, f_K]$, K	Alpha pool and pool size
$\mathcal{M}_K(\mathcal{X})$	Size-K pools of canonically distinct formulas
$\widehat{w}_F$, γ	Ridge combination weights and penalty (Eq. 3)
$U_S(F)$	Utility of pool F on split S: mean daily Spearman correlation of the combined prediction
$\mathcal{A}$, $\mathcal{E}$, B	Submitted canonical formulas, eligible formulas, and budget on $\lvert\mathcal{A}\rvert$
Hierarchical diffusion	
T, t, ρ_t	Number of diffusion levels, level index, and masking schedule
U_u, Φ_ρ	Clock of absorbable node u and subtree-masking operator
F_t, $\mathbf{m}$, H_t	Pool corrupted to level t, mask token, and exposed masks
a_t, w_t	Mask-retention and reveal probabilities
μ_{t-1}^{u,F_0}	Level-$(t-1)$ distribution of the subtree at u
J_t, $u_{t,r}$, $G^{(r)}$	Reveal set, its r-th position, and the updated pool
ψ, $\mathbf{c}_{G,u,t}$, E_t	Formula encoder, pool context, and level embedding
r_θ, θ_{ref}	Reverse model and frozen reference model
α, t_0	Edit strength and starting level of refinement
Reward and buffer	
c_{ij}, d_i, δ	Pairwise redundancy, nearest-member redundancy, and tolerated similarity
R_{outer}, R_{inner}, R	Outer predictive, inner diversity, and pool rewards
w_{inner}	Weight of the inner diversity reward
$\mathcal{Q}$, $C_{\mathcal{Q}}$, M	Elite buffer, its capacity, and number of initial pools
$P_{\mathcal{Q}}$, T_{r}	Buffer sampling distribution and its temperature
$\mathcal{C}$, $\widehat{F}$	Validation archive and selected final pool
Training and search	
$\widehat{\mathcal{L}}_{\text{VB}}$, ℓ_θ	Reconstruction objective and single-pool loss
$\mathcal{L}_{\text{pref}}$, $\mathcal{P}_{\mathcal{Q}}$	Preference objective and distribution of accepted pairs
β, m_{dpo}	Preference strength and reward margin
λ_{VB}	Weight of the reconstruction objective
N_{c}, N	Proposal batch size and training minibatch size
S_0, S_{u}, N_{stall}	Initialization updates, updates per round, and stall limit

A.1 FEATURES AND OPERATORS

Table 4 lists the features, operators, windows, constants, and special tokens that define the formula space $\mathcal{X}$.

B SUPPLEMENTARY MODEL DETAILS

This section provides the theoretical analysis of AlphaPADI (Appendix B.1), its complete search and training procedure (Appendix B.2), and its computational complexity (Appendix B.3).

B.1 THEORETICAL DERIVATIONS

We analyze the probabilistic structure of the two learned components of AlphaPADI. Appendix B.1.1 derives the posterior of the hierarchical corruption process and the reverse transition

Table 4: Formula vocabulary. The 61 semantic tokens in $\mathcal{V}$ define the formula space $\mathcal{X}$, and the four special tokens control generation and diffusion. In the stack effect, w denotes a window token.

Type (count)	Tokens	Stack effect
Feature (6)	`Open`, `Close`, `High`, `Low`, `Vwap`, `Volume`	push
Constant (14)	±30, ±10, ±5, ±2, ±1, ±0.5, ±0.01	push
Window (5)	1, 5, 10, 20, 40	push window
Unary (6)	`Abs`, `Slog1p`, `Inv`, `Sign`, `Log`, `Rank`	$1 \to 1$
Binary (7)	`Add`, `Sub`, `Mul`, `Div`, `Pow`, `Greater`, `Less`	$2 \to 1$
Temporal (21)	`Ref`, `TsMean`, `TsSum`, `TsStd`, `TsIr`, `TsMinMaxDiff`, `TsMaxDiff`, `TsMinDiff`, `TsVar`, `TsSkew`, `TsKurt`, `TsMax`, `TsMin`, `TsMed`, `TsMad`, `TsRank`, `TsDelta`, `TsDiv`, `TsPctChange`, `TsWMA`, `TsEMA`	$1{+}w \to 1$
Pairwise temporal (2)	`TsCov`, `TsCorr`	$2{+}w \to 1$
Special (4)	BEG, SEP, MASK, FILL	control

it induces, corresponding to Section 4.2. Appendix B.1.2 relates the reconstruction and preference objectives of Section 4.3 to a variational bound and to a KL-regularized pool-level reward maximization, and identifies the approximations involved.

For a clean pool F_0, let $\mathcal{N}(F_0)$ be the set of absorbable AST nodes of its K formulas. Window tokens are attached to their temporal operators and are not absorbable. For $u \in \mathcal{N}(F_0)$, $\mathrm{Sub}(u)$ contains u and its absorbable descendants, and $\mathrm{Desc}(u) = \mathrm{Sub}(u) \setminus \{u\}$. For an antichain $H \subseteq \mathcal{N}(F_0)$, in which no node is an ancestor of another, $\Phi_H(F_0)$ replaces each $\mathrm{Sub}(u)$ with $u \in H$ by $\mathbf{m}$. With the clocks $\{U_u\}$ of Section 4.2, the visible, active, and exposed sets and the schedule quantities are

$$\begin{aligned} V(H) &= \mathcal{N}(F_0) \setminus \textstyle\bigcup_{u\in H} \mathrm{Sub}(u), & A_\rho &= \{u \in \mathcal{N}(F_0) : U_u \le \rho\}, \\ H_\rho &= \{u \in A_\rho : \text{no ancestor of } u \text{ lies in } A_\rho\}, & \Phi_\rho(F_0) &= \Phi_{H_\rho}(F_0), \\ \eta_t &= \frac{\rho_t - \rho_{t-1}}{1-\rho_{t-1}}, & a_t &= \frac{\rho_{t-1}}{\rho_t}, \quad w_t = \frac{\rho_t - \rho_{t-1}}{\rho_t}, \end{aligned} \tag{14}$$

where $0 = \rho_0 < \rho_1 < \cdots < \rho_T = 1$ and $H_t := H_{\rho_t}$ collects the exposed masks of $F_t = \Phi_{\rho_t}(F_0)$. A corrupted pool is treated as a partially masked AST, so H_t is recoverable from F_t. For the r-th reconstruction of a transition, we abbreviate $r_\vartheta^{(r)} = r_\vartheta(Y_{u_r} \mid G^{(r-1)}, u_r, t)$ for $\vartheta \in \{\theta, \theta_{\mathrm{ref}}\}$. We adopt two conventions. (A1) r_ϑ is renormalized over grammar-valid actions, assigns positive probability to every valid target, including targets with MASK at descendant positions, and never outputs the bare target $\mathbf{m}$ at the queried position. (A2) The order π is drawn from a parameter-free distribution $\nu(\pi \mid J)$ over the set $\mathcal{O}(J)$ of interleavings of J that preserve the RPN order within each formula. Canonicalization, duplicate replacement, and executability filtering follow reverse generation and lie outside the analyzed model.

B.1.1 Pool-Aware Hierarchical Diffusion

Proposition B.1 (Hierarchical corruption posterior)**.** *For every $t \in \{0, \dots, T\}$ and every antichain $H \subseteq \mathcal{N}(F_0)$,*

$$q\big(F_t = \Phi_H(F_0) \mid F_0\big) = \rho_t^{|H|}\,(1-\rho_t)^{|V(H)|}. \tag{15}$$

For $t \ge 1$, let $J_t = H_t \setminus H_{t-1}$ and $S_t = H_t \cap H_{t-1}$ be the revealed and retained masks. For $u \in J_t$, let $\mathcal{H}^u = H_{t-1} \cap \mathrm{Desc}(u)$ and $V^u = \mathrm{Sub}(u) \setminus \bigcup_{v\in\mathcal{H}^u} \mathrm{Sub}(v)$. The posterior factorizes over the exposed masks as

$$q(F_{t-1} \mid F_t, F_0) = \prod_{u\in H_t} \Big[a_t\,\delta_{\mathbf{m}}(Y_u) + w_t\,\mu_{t-1}^{u,F_0}(Y_u)\Big], \tag{16}$$

where $\mu_{t-1}^{u,F_0}(Y_u) = \rho_{t-1}^{|\mathcal{H}^u|}(1-\rho_{t-1})^{|V^u|-1}$.

Proof. The exposed set equals H if and only if every node of H is active and every node of $V(H)$ is inactive. Under these conditions, each active node lies below some $u \in H$ whose ancestors

are all inactive. Conversely, an active node in $V(H)$ would have its highest active ancestor in $H_\rho = H$, which contradicts its membership in $V(H)$. Clocks strictly below H do not enter, and the independence of the clocks gives Eq. 15. In particular, $\rho_T = 1$ forces $F_T = (\mathbf{m}, \dots, \mathbf{m})$. Because the clocks are shared and ρ_t increases, $A_{\rho_{t-1}} \subseteq A_{\rho_t}$. Given F_{t-1}, the clocks on $V(H_{t-1})$ are uniform on $(\rho_{t-1}, 1]$, so each visible node is newly masked with probability η_t, independently, while nodes inside masked subtrees cannot affect H_t. The forward process is therefore Markov with $q(F_t \mid F_{t-1}) = \eta_t^{|J_t|}(1-\eta_t)^{|V(H_t)|}$. Each element of H_{t-1} lies either in S_t or below exactly one $u \in J_t$, which gives

$$V(H_{t-1}) = V(H_t) \sqcup \bigsqcup_{u \in J_t} V^u, \qquad |H_{t-1}| = |S_t| + \textstyle\sum_{u \in J_t} |\mathcal{H}^u|, \qquad |H_t| = |S_t| + |J_t|. \tag{17}$$

Applying Bayes' rule with Eqs. 15 and 17, together with $(1-\eta_t)(1-\rho_{t-1}) = 1-\rho_t$ and $\eta_t(1-\rho_{t-1}) = w_t\rho_t$, yields

$$\begin{aligned} q(F_{t-1} \mid F_t, F_0) &= \frac{q(F_t \mid F_{t-1})\, q(F_{t-1} \mid F_0)}{q(F_t \mid F_0)} = \frac{\eta_t^{|J_t|}(1-\eta_t)^{|V(H_t)|}\, \rho_{t-1}^{|H_{t-1}|}(1-\rho_{t-1})^{|V(H_{t-1})|}}{\rho_t^{|H_t|}(1-\rho_t)^{|V(H_t)|}} \\ &= \Big(\frac{\rho_{t-1}}{\rho_t}\Big)^{|S_t|} \prod_{u \in J_t} \frac{\eta_t(1-\rho_{t-1})}{\rho_t}\, \rho_{t-1}^{|\mathcal{H}^u|}(1-\rho_{t-1})^{|V^u|-1} \\ &= a_t^{|S_t|} \prod_{u \in J_t} w_t\, \mu_{t-1}^{u,F_0}(Y_u), \end{aligned} \tag{18}$$

where the factors on $V(H_t)$ cancel and one factor $1-\rho_{t-1}$ from each V^u is absorbed into w_t. The distribution μ_{t-1}^{u,F_0} is Eq. 15 applied to the forest $\mathrm{Desc}(u)$ at level $t-1$ and is therefore normalized. By (A1), it never produces the bare $\mathbf{m}$, so the two terms in Eq. 16 have disjoint supports, and expanding the product recovers Eq. 18. □

Remark. At $t = 1$, we have $a_1 = 0$, $w_1 = 1$, and μ_0^{u,F_0} is the point mass on the clean subtree of u, so the last reverse transition deterministically completes every remaining mask. Drawing one set of clocks and reading off $(\Phi_{\rho_t}(F_0), \Phi_{\rho_{t-1}}(F_0))$ yields an exact sample of $q(F_t, F_{t-1} \mid F_0)$, in which each $u \in H_t$ enters J_t independently with probability w_t and independently of F_0. The reverse process in Section 4.2 uses the same Bernoulli selection, and the pairs $(J_b, Y_{b,r})$ in Eq. 11 are sampled in this way. The hierarchy enters the posterior only through μ_{t-1}^{u,F_0}: a revealed position receives a complete subtree whose descendants remain corrupted at level $t-1$, so a single transition can edit a leaf, a subexpression, or a whole formula.

Proposition B.2 (Pool-aware reverse transition)**.** *Let $q(F_0)$ be the distribution of clean pools, such as $P_{\mathcal{Q}}$, and let $h_r = (F_t, u_{1:r}, Y_{u_1}, \dots, Y_{u_{r-1}})$ denote the reconstruction history. For every order $\pi = (u_1, \dots, u_{|J_t|}) \in \mathcal{O}(J_t)$, the true reverse transition and the reverse model in Section 4.2 are*

$$\begin{aligned} q(F_{t-1} \mid F_t) &= a_t^{|S_t|}\, w_t^{|J_t|} \prod_{r=1}^{|J_t|} \mathsf{Q}_t(Y_{u_r} \mid h_r), \\ p_\theta(F_{t-1} \mid F_t) &= a_t^{|S_t|}\, w_t^{|J_t|} \sum_{\pi \in \mathcal{O}(J_t)} \nu(\pi \mid J_t) \prod_{r=1}^{|J_t|} r_\theta^{(r)}, \end{aligned} \tag{19}$$

where $\mathsf{Q}_t(Y_{u_r} \mid h_r) = \mathbb{E}_{F_0 \sim q(\cdot \mid F_t, Y_{u_{<r}})}\big[\mu_{t-1}^{u_r,F_0}(Y_{u_r})\big]$ is the posterior predictive of the r-th target. Let $\mathsf{Q}_t(\cdot \mid F_t)$ and $\mathsf{P}_\theta(\cdot \mid F_t, J_t)$ denote the content distributions over Y_{J_t} in the two lines. The transition divergence then reduces to the reconstructed content,

$$\mathrm{KL}\big(q(F_{t-1} \mid F_t) \,\big\|\, p_\theta(F_{t-1} \mid F_t)\big) = \mathbb{E}_{J_t}\Big[\mathrm{KL}\big(\mathsf{Q}_t(\cdot \mid F_t) \,\big\|\, \mathsf{P}_\theta(\cdot \mid F_t, J_t)\big)\Big], \tag{20}$$

where each $u \in H_t$ enters J_t independently with probability w_t.

Proof. When Eq. 16 is marginalized over $q(F_0 \mid F_t)$, the selection factor does not depend on F_0, and the chain rule applies to the remaining mixture for any order π:

$$\begin{aligned} q(F_{t-1} \mid F_t) &= \sum_{F_0} q(F_0 \mid F_t)\, q(F_{t-1} \mid F_t, F_0) \\ &= a_t^{|S_t|}\, w_t^{|J_t|} \sum_{F_0} q(F_0 \mid F_t) \prod_{u \in J_t} \mu_{t-1}^{u,F_0}(Y_u) \\ &= a_t^{|S_t|}\, w_t^{|J_t|} \prod_{r=1}^{|J_t|} \sum_{F_0} q(F_0 \mid F_t, Y_{u_{<r}})\, \mu_{t-1}^{u_r,F_0}(Y_{u_r}), \end{aligned} \tag{21}$$

where $q(F_0 \mid F_t, Y_{u_{<r}}) \propto q(F_0 \mid F_t) \prod_{s<r} \mu_{t-1}^{u_s,F_0}(Y_{u_s})$. The model transition follows from the reverse procedure. J_t is selected with probability w_t per position, $\pi \sim \nu(\cdot \mid J_t)$, and each target is drawn from $r_\theta^{(r)}$ and spliced into $G^{(r)} = \mathrm{Splice}(G^{(r-1)}, u_r, Y_{u_r})$ before the next prediction, with $G^{(0)} = F_t$. By (A1), J_t is identifiable from F_{t-1}, so both transitions assign the same probability $a_t^{|S_t|} w_t^{|J_t|}$ to the selection. Summing $q \log(q/p_\theta)$ over F_{t-1}, first over the contents and then over the selections, gives Eq. 20. □

Remark. The content distribution $\mathsf{Q}_t(Y_{J_t} \mid F_t) = \mathbb{E}_{F_0 \sim q(\cdot \mid F_t)} \prod_{u \in J_t} \mu_{t-1}^{u,F_0}(Y_u)$ is a mixture of products. It does not factorize across positions whenever $q(F_0 \mid F_t)$ couples the pool members, as it does for buffered pools that are selected jointly by the non-separable reward in Eq. 9. A reverse model that reconstructs each position independently from F_t therefore cannot represent it. The sequential model instead conditions each target on the updated pool $G^{(r-1)}$. If $G^{(r-1)}$ retains h_r and $r_\theta^{(r)} = \mathsf{Q}_t(\cdot \mid h_r)$, every order yields the same product, and $p_\theta(F_{t-1} \mid F_t) = q(F_{t-1} \mid F_t)$ for any ν. This is the formal role of pool-aware reverse generation. Candidate generation composes these kernels from the partial corruption level t_0,

$$P_\theta(F' \mid F) = \sum_{F_{t_0}} q(F_{t_0} \mid F) \sum_{F_1, \ldots, F_{t_0-1}} \prod_{t=1}^{t_0} p_\theta(F_{t-1} \mid F_t) \Big|_{F_0 = F'}, \tag{22}$$

which is the pool-conditional generator in Eq. 2.

B.1.2 Reward-Guided Pool Refinement and Training Objective

Let $\xi = (t, \{U_u\}, \pi)$ collect the diffusion level, the clocks, and the reconstruction order, and let $\Omega = (F_{1:T}, \pi_{1:T})$ denote an augmented reverse path. For $\vartheta \in \{\theta, \theta_{\mathrm{ref}}\}$, the unnormalized and normalized single-pool reconstruction losses are

$$\Lambda_\vartheta(F_0; \xi) = -T \sum_{r=1}^{|J_t|} \log r_\vartheta^{(r)}, \qquad \ell_\vartheta(F_0; \xi) = -\sum_{r=1}^{|J_t|} \frac{\log r_\vartheta^{(r)}}{n\, L_r}, \qquad n = \max\{1, |J_t|\}, \tag{23}$$

so that $\widehat{\mathcal{L}}_{\mathrm{VB}} = (T/N) \sum_{b=1}^{N} \ell_\theta(F_0^{(b)}; \xi_b)$ in Eq. 11. We write $\mathcal{L}_{\mathrm{VB}}(F_0; \vartheta) = \mathbb{E}_\xi[\Lambda_\vartheta(F_0; \xi)]$ and let $\bar{L}$ be the maximal target length, including SEP.

Proposition B.3 (Variational bound and normalized surrogate)**.** *There exists $\mathcal{H}(F_0) \geq 0$, independent of θ, such that*

$$-\log p_\theta(F_0) \leq \mathcal{L}_{\mathrm{VB}}(F_0; \theta) - \mathcal{H}(F_0) \leq T \bar{L}\, \mathbb{E}_\xi\big[n_\xi\, \ell_\theta(F_0; \xi)\big] - \mathcal{H}(F_0). \tag{24}$$

Moreover, for a fixed history h with $\mathsf{Q} = \mathsf{Q}_t(\cdot \mid h)$, the minimizer of $\mathbb{E}_\xi[\ell_\theta]$ over unrestricted conditionals $r(\cdot \mid h)$ is the length-tilted distribution

$$\begin{gathered} r^\star(Y \mid h) = \frac{\mathsf{Q}(Y)}{L(Y)\, Z(h)}, \qquad Z(h) = \mathbb{E}_{\mathsf{Q}}\big[L(Y)^{-1}\big], \\ \max\big\{\mathrm{KL}(\mathsf{Q} \,\|\, r^\star),\, \mathrm{KL}(r^\star \,\|\, \mathsf{Q})\big\} \leq \log \bar{L}. \end{gathered} \tag{25}$$

Proof. The forward path law factorizes as $q(\Omega \mid F_0) = q(F_T \mid F_0) \prod_{t=1}^{T} q(F_{t-1} \mid F_t, F_0)\, \nu(\pi_t \mid J_t)$, and the reverse path law as $p_\theta(F_0, \Omega) = p(F_T) \prod_{t=1}^{T} p_\theta(F_{t-1}, \pi_t \mid F_t)$, where $p_\theta(F_{t-1}, \pi_t \mid$

$F_t)$ is the summand in Eq. 19. Since $\rho_T = 1$, $q(F_T \mid F_0) = p(F_T)$. Jensen's inequality and Eqs. 16 and 19, in which the selection factors and ν cancel, give

$$\begin{aligned} -\log p_\theta(F_0) &\le \mathbb{E}_{q(\Omega \mid F_0)}\Big[\sum_{t=1}^{T} \log \frac{q(F_{t-1} \mid F_t, F_0)\, \nu(\pi_t \mid J_t)}{p_\theta(F_{t-1}, \pi_t \mid F_t)}\Big] \\ &= \sum_{t=1}^{T} \mathbb{E}_q\Big[\sum_{u \in J_t} \log \mu_{t-1}^{u,F_0}(Y_u) - \sum_{r=1}^{|J_t|} \log r_\theta^{(r)}\Big] \\ &= T\, \mathbb{E}_{t \sim \mathrm{Unif}\{1,\dots,T\}}\, \mathbb{E}_q\Big[-\sum_{r=1}^{|J_t|} \log r_\theta^{(r)}\Big] - \mathcal{H}(F_0) = \mathcal{L}_{\mathrm{VB}}(F_0;\theta) - \mathcal{H}(F_0), \end{aligned} \tag{26}$$

where $\mathcal{H}(F_0) = \sum_t \mathbb{E}_q[-\sum_{u \in J_t} \log \mu_{t-1}^{u,F_0}(Y_u)] \ge 0$. Since $-\log r_\theta^{(r)} \ge 0$ and $1 \le L_r \le \bar{L}$, every ξ satisfies $\Lambda_\theta(F_0;\xi) \le T\bar{L}\, n_\xi\, \ell_\theta(F_0;\xi)$, and taking expectations gives the second inequality of Eq. 24. For Eq. 25, the weight $1/n$ depends only on J_t, which is independent of Y_{u_r} given h. It therefore scales the contribution of h by $c(h) = \mathbb{E}[1/n \mid h] > 0$. The weight $1/L(Y)$ depends on the target itself, and

$$\begin{aligned} c(h)\, \mathbb{E}_{\mathsf{Q}}\Big[\frac{-\log r(Y \mid h)}{L(Y)}\Big] &= c(h)\, Z(h)\, \mathbb{E}_{r^\star}\big[-\log r(Y \mid h)\big] \\ &= c(h)\, Z(h)\Big[\mathbb{H}(r^\star) + \mathrm{KL}\big(r^\star \,\|\, r(\cdot \mid h)\big)\Big], \end{aligned} \tag{27}$$

which is minimized at $r = r^\star$, where $\mathbb{H}$ denotes the Shannon entropy. Finally, $\log(r^\star/\mathsf{Q}) = -\log L(Y) - \log Z(h) \in [-\log \bar{L}, \log \bar{L}]$ because $1 \le L(Y) \le \bar{L}$ and $\bar{L}^{-1} \le Z(h) \le 1$, which bounds both divergences. □

Remark. Eq. 24 explains why Eq. 11 is a surrogate rather than an evidence lower bound. The unnormalized loss Λ_θ is an unbiased single-sample estimator of the θ-dependent part of the bound. Teacher forcing is exact because the bound is evaluated along paths drawn from q, and the factor T arises from sampling the level uniformly. The normalized loss controls the same bound up to the known factor $T\bar{L}\, n_\xi$. By Eq. 25, the factor $1/n$ equalizes pools with different numbers of reveals without changing the minimizer. The factor $1/L_r$ equalizes leaf-, subtree-, and alpha-level targets at the cost of a tilt of at most $\log \bar{L}$ nats toward shorter completions. As a result, $-\ell_\theta$ is a per-target, per-token reconstruction score that is comparable across pools of different structure, which is the property used by the preference objective.

Proposition B.4 (Pool-level preference objective)**.** *Consider the KL-regularized pool-level reward maximization over augmented reverse paths,*

$$\max_\theta\ \mathbb{E}_{F_0 \sim p_\theta}\big[R(F_0)\big] - \beta'\ \mathrm{KL}\big(p_\theta(F_0,\Omega)\,\big\|\,p_{\mathrm{ref}}(F_0,\Omega)\big), \qquad \beta' > 0. \tag{28}$$

Assume that its maximizer is attained within the model class, and model preferences as $\Pr(F_{\mathrm{w}} \succ F_{\mathrm{l}}) = \sigma\big(R(F_{\mathrm{w}}) - R(F_{\mathrm{l}})\big)$*. Then the negative log-likelihood of accepted pairs is*

$$\begin{aligned} \mathcal{L}^\star_{\mathrm{pref}}(\theta) &= -\mathbb{E}_{\mathcal{P}_{\mathsf{Q}}} \log \sigma\big(\beta' \Delta^\star_\theta\big), \\ \Delta^\star_\theta &= \big[\mathcal{L}^{\mathrm{l}}_{\mathrm{VB}}(\theta) - \mathcal{L}^{\mathrm{w}}_{\mathrm{VB}}(\theta)\big] - \big[\mathcal{L}^{\mathrm{l}}_{\mathrm{VB}}(\theta_{\mathrm{ref}}) - \mathcal{L}^{\mathrm{w}}_{\mathrm{VB}}(\theta_{\mathrm{ref}})\big], \end{aligned} \tag{29}$$

where $\mathcal{L}^a_{\mathrm{VB}}(\vartheta) = \mathcal{L}_{\mathrm{VB}}(F_a;\vartheta)$ *for* $a \in \{\mathrm{w},\mathrm{l}\}$*. Let* X *be the single-sample estimate of* $\beta'\Delta^\star_\theta$ *obtained from* Λ_ϑ *under a coupling* Γ *of* $(\xi_{\mathrm{w}}, \xi_{\mathrm{l}})$ *that preserves the corruption law of each pool. Then*

$$-\log \sigma\big(\mathbb{E}_\Gamma X\big) \le \mathbb{E}_\Gamma\big[-\log \sigma(X)\big] \le -\log \sigma\big(\mathbb{E}_\Gamma X\big) + \tfrac{1}{8}\,\mathrm{Var}_\Gamma(X), \tag{30}$$

and replacing Λ_ϑ *by* $T\ell_\vartheta$ *with* $\beta = T\beta'$ *yields Eq. 12.*

Proof. Eq. 28 equals $-\beta'\,\mathrm{KL}(p_\theta \,\|\, p_{\mathrm{ref}} e^{R/\beta'}/Z) + \beta' \log Z$ with $Z = \mathbb{E}_{p_{\mathrm{ref}}}[e^{R(F_0)/\beta'}]$, so its maximizer is $p^\star = p_{\mathrm{ref}} e^{R/\beta'}/Z$. The tilt depends only on the endpoint, so $\log(p^\star/p_{\mathrm{ref}}) = R(F_0)/\beta' - \log Z$ on every path ending at F_0, which by (A1) includes the support of $q(\Omega \mid F_0)$. We

parameterize $p^\star$ by p_θ (Rafailov et al., 2023), average under $q(\Omega \mid F_0)$, and apply the cancellations in Eq. 26, which also remove $\mathcal{H}(F_0)$:

$$R(F_0) - \beta' \log Z = \beta' \, \mathbb{E}_{q(\Omega|F_0)} \Big[\log \frac{p_\theta(F_0, \Omega)}{p_{\mathrm{ref}}(F_0, \Omega)} \Big] = \beta' \, \mathbb{E}_{q(\Omega|F_0)} \Big[\sum_{t=1}^{T} \sum_{r=1}^{|J_t|} \log \frac{r_\theta^{(r)}}{r_{\mathrm{ref}}^{(r)}} \Big] \tag{31}$$
$$= -\beta' \big[\mathcal{L}_{\mathrm{VB}}(F_0; \theta) - \mathcal{L}_{\mathrm{VB}}(F_0; \theta_{\mathrm{ref}}) \big].$$

Substituting Eq. 31 for F_{w} and F_{l} into the preference model cancels $\log Z$ and gives Eq. 29. For Eq. 30, let $A_a = \beta'[\Lambda_\theta(F_a; \xi_a) - \Lambda_{\mathrm{ref}}(F_a; \xi_a)]$ for $a \in \{\mathrm{w}, \mathrm{l}\}$ and $X = A_{\mathrm{l}} - A_{\mathrm{w}}$. Because Γ preserves the marginal laws, $\mathbb{E}_\Gamma X = \beta' \Delta_\theta^\star$. The function $g(x) = -\log \sigma(x)$ is convex with $g''(x) = \sigma(x)\sigma(-x) \le 1/4$, so Jensen's inequality gives the lower bound, and a second-order expansion around $\mathbb{E}_\Gamma X$ gives the upper bound. The last claim follows from Eq. 23, since $\beta' T \ell_\vartheta = \beta \, \ell_\vartheta$. □

Remark. Since $\mathrm{Var}_\Gamma(X) = \mathrm{Var}(A_{\mathrm{l}}) + \mathrm{Var}(A_{\mathrm{w}}) - 2\,\mathrm{Cov}_\Gamma(A_{\mathrm{l}}, A_{\mathrm{w}})$, only the covariance depends on the coupling. The matched pseudorandom stream in Section 4.3 therefore tightens Eq. 30 whenever it induces positive covariance, and evaluating θ and θ_{ref} under the same ξ_a makes $X \equiv 0$ at initialization. The normalization in Eq. 12 turns the implicit reward into the per-target, per-token quantity $-\beta(\ell_\theta - \ell_{\mathrm{ref}})$, which prevents pools with more or longer targets from dominating the margin. Proposition B.3 bounds the resulting deviation from Eq. 29.

Although R is computable, the pairwise objective has three advantages over maximizing $\mathbb{E}_{p_\theta}[R]$ directly. First, the expected-reward gradient requires fresh on-policy samples, each consuming distinct-formula evaluations from the budget B in Eq. 4, whereas Eq. 12 reuses scored buffer pools. Second, the pair-conditional likelihood does not involve the distribution from which pairs are drawn, so off-policy elite pairs are admissible. Moreover, R enters only through the acceptance event, which is invariant to positive affine transformations of R. Third, with $\mathcal{B}_\mathcal{Q} \sim P_\mathcal{Q}$, the reconstruction term is a reward-weighted regression toward the Boltzmann distribution over standardized buffer rewards. It reallocates likelihood among retained pools but never contrasts them, and the preference term supplies this contrast. Freezing θ_{ref} fixes the target $p^\star \propto p_{\mathrm{ref}} e^{R/\beta'}$. Writing Δ for the bracket in Eq. 12, the reference enters the gradient only through a per-pair offset that rescales the weight $\sigma(-\beta\Delta)$ without changing the direction $\nabla_\theta \ell_\theta^{\mathrm{l}} - \nabla_\theta \ell_\theta^{\mathrm{w}}$. Resetting the reference after each of k optimal rounds would instead yield $p_{\mathrm{ref}} e^{kR/\beta'}$, a progressively colder target that erodes proposal diversity. The proposition does not claim that finite-step training attains $p^\star$. The fixed selection factors make attainability an assumption, and because labels are deterministic given R, the preference model is recovered only in its zero-temperature limit. Eq. 12 therefore acts as a logistic ranking loss on accepted pairs.

B.2 Algorithm Flow Chart

The algorithm flow of AlphaPADI is shown in Algorithm 1.

B.3 Complexity Analysis

Let h, L, and H denote the hidden width, maximum formula length, and number of reconstructed regions. With fixed Transformer depth, dense attention, full pool re-encoding after each insertion, and key–value caching, reconstructing one region costs $C_{\mathrm{r}} = O((KL^2 + K^2 + L^2)h + (KL + K + L)h^2)$. Thus, forward corruption and reverse generation cost $O(KL)$ and $O(HC_{\mathrm{r}})$, respectively, with the same per-target order during training. For $n = \sum_d N_d$ observations, pool evaluation costs $O(K \sum_d N_d \log N_d + nK^2 + K^3)$. Assuming linear-time DSL operations, the budget B bounds first-time formula execution by $O(BnL)$, while cached signals avoid repeated execution. The elite buffer stores $O(C_\mathcal{Q} KL)$ tokens.

C Supplementary Experiments

This section details the experimental setup (Appendix C.1) and reports additional results (Appendix C.2).

Algorithm 1 AlphaPADI pool refinement and training

Require: buffer capacity $C_{\mathcal{Q}}$, number of initial pools $M \le C_{\mathcal{Q}}$, budget B (Eq. 4), proposal batch size N_{c}, minibatch size N, initialization updates S_0, per-round updates S_{u}, stall limit N_{stall}

Build $\mathcal{Q}_0$ by Eqs. 5 and 9, and add all submitted canonical formulas to $\mathcal{A}$
$\mathcal{Q} \leftarrow \mathcal{Q}_0$, $\mathcal{C} \leftarrow \{F : (F, R(F)) \in \mathcal{Q}_0\}$, $n_{\mathrm{stall}} \leftarrow 0$
Initialize θ and perform S_0 updates of Eq. 11 on minibatches from $\mathcal{Q}_0$
$\theta_{\mathrm{ref}} \leftarrow \theta$ and freeze θ_{ref}
while $|\mathcal{A}| < B$ **and** $n_{\mathrm{stall}} < N_{\mathrm{stall}}$ **do**
    $a \leftarrow |\mathcal{A}|$
    Sample N_{c} pools from $P_{\mathcal{Q}}$ (Eq. 10) to form $\mathcal{B}_{\mathrm{c}}$
    for each $F \in \mathcal{B}_{\mathrm{c}}$ **do**
        Corrupt F to level $t_0 = \max\{1, \mathrm{round}(\alpha T)\}$ by Eq. 6
        Reconstruct F' from t_0 to 0 with r_θ conditioned on Eq. 7
        Canonicalize F' and replace repeated members by random formulas
        $\mathcal{U} \leftarrow$ canonical members of F' that are not in $\mathcal{A}$
        if $|\mathcal{A}| + |\mathcal{U}| \le B$ **then** ▷ otherwise discard F'
            Submit $\mathcal{U}$ to the evaluator and set $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{U}$
            if F' satisfies the eligibility rule of Sec. 4.3 **then**
                Compute $R(F')$ by Eqs. 8 and 9, and set $\mathcal{C} \leftarrow \mathcal{C} \cup \{F'\}$
                Insert $(F', R(F'))$ into $\mathcal{Q}$ and keep the top $C_{\mathcal{Q}}$ canonical pools
            end if
        end if
    end for
    $n_{\mathrm{stall}} \leftarrow 0$ **if** $|\mathcal{A}| > a$ **else** $n_{\mathrm{stall}} + 1$
    for $s = 1, \ldots, S_{\mathrm{u}}$ **do**
        Sample $\mathcal{B}_{\mathcal{Q}}$ of N pools from $P_{\mathcal{Q}}$ and evaluate Eq. 11
        Sample accepted pairs from $\mathcal{P}_{\mathcal{Q}}$ and evaluate Eq. 12, or set $\mathcal{L}_{\mathrm{pref}} = 0$ if none exists
        Update θ by one gradient step on Eq. 13
    end for
end while
return $\widehat{F}$ selected from $\mathcal{C}$ by $\widehat{F} = \arg\max_{F \in \mathcal{C}} U_{\mathrm{val}}(F)$.

C.1 Additional Experimental Details

C.1.1 Implementation Details

Model architecture. The reverse model comprises a formula encoder, a pool encoder, and an autoregressive decoder. The formula encoder is a shared two-layer Transformer that combines token and within-formula positional embeddings and mean-pools the non-padding states. For a queried position, its MASK token is replaced by FILL, and the resulting query copy is encoded by the same formula encoder. The pool encoder applies one set-attention layer to the K formula representations and the query representation without row positional embeddings. Their mean-pooled representation is combined with the diffusion-level embedding $E_t(t)$ to obtain the pool context in Eq. 7. A two-layer causal Transformer decoder conditions every token prediction on this context and generates the reconstruction target as an RPN sequence from BEG to SEP. All Transformer modules share the hidden dimension, number of attention heads, and dropout listed in Table 5. Grammar masking enforces the operator-arity, stack, and maximum-length constraints during decoding.

Training. We optimize the reverse model with AdamW, initializing weight matrices with Xavier uniform initialization and biases with zero. Before candidate generation, the model receives S_0 reconstruction updates on the initialized buffer, after which the reference model is frozen. Each subsequent round generates N_{c} candidate pools and performs S_{u} updates on minibatches of N pools. Search rewards and generator updates use only the 2016–2020 training period, the 2021–2022 validation period selects the configuration, checkpoint, and final pool, and the 2023–2025 test period is used only for the final evaluation. Table 5 lists all values.

Static ridge combination. Every pool, including those returned by the baselines, is evaluated through the same code path. Each member signal is converted on every date to tie-aware cross-sectional ranks scaled to $[-1, 1]$. The combiner solves Eq. 3 in closed form, $\widehat{w}_F = (\widetilde{X}^\top \widetilde{X}/n + \gamma I)^{-1} \widetilde{X}^\top \widetilde{y}/n$, without an intercept and with the penalty normalized by the pool size. The penalty γ is selected on a logarithmic grid by the Spearman correlation on a chronological $80/20$ split of the training period, with 2016–2019 for fitting, 2020 for selection, and a 20-day gap between them, after which the combiner is refitted on the full training period. The weights are then frozen, so every validation and test date uses the same $\widehat{w}_F$ and all K members, without refitting or reselection. We use the static combiner for all automated alpha mining methods for four reasons. It evaluates the same jointly fitted pool that the search reward optimizes. It attributes performance to the pool rather than to an adaptive combiner, which would otherwise perform part of the prediction for every method. Frozen weights cannot compensate for factor decay over the three-year test period, which makes the evaluation stricter. Finally, because all K members contribute to every prediction, the pool size directly affects the combined predictor, which the analysis in Section 5.4 relies on.

C.1.2 Evaluation Metrics and Backtesting Protocol

Let $\widehat{\mathbf{y}}_d$ and $\mathbf{y}_d$ denote the predicted signals and return labels across the eligible assets on date d. We define $\rho_d = \mathrm{Pearson}(\widehat{\mathbf{y}}_d, \mathbf{y}_d)$ and $\rho_d^{\mathrm{r}} = \mathrm{Pearson}(\mathrm{rank}(\widehat{\mathbf{y}}_d), \mathrm{rank}(\mathbf{y}_d))$, where ties are assigned their average ranks. For a daily sequence $\{z_d\}_{d=1}^{D}$, let $\overline{z} = D^{-1} \sum_{d=1}^{D} z_d$ and $s(z)$ denote its sample standard deviation.

- **Information coefficient (IC).** $\mathrm{IC} = |\overline{\rho}|$ measures the average daily cross-sectional linear correlation between predictions and returns.
- **Information coefficient information ratio (ICIR).** $\mathrm{ICIR} = \overline{\rho}/s(\rho)$ measures the temporal stability of IC.
- **Rank information coefficient (RIC).** $\mathrm{RIC} = |\overline{\rho^{\mathrm{r}}}|$ measures the average daily cross-sectional rank correlation.
- **Rank information coefficient information ratio (RICIR).** $\mathrm{RICIR} = \overline{\rho^{\mathrm{r}}}/s(\rho^{\mathrm{r}})$ measures the temporal stability of RIC.

Let $\Delta\Pi_t$ be the realized one-day profit or loss of all cohorts active on trading day t, net of that day's transaction costs, and let V_0 be the fixed initial capital. Define $r_t = \Delta\Pi_t / V_0$ and $W_t = 1 + \sum_{\tau=1}^{t} r_\tau$, with $W_0 = 1$. For the benchmark, define b_t from its contemporaneous one-day profit or loss under the same fixed-capital convention, and let $W_t^{\mathrm{bench}} = 1 + \sum_{\tau=1}^{t} b_\tau$. Write $e_t = r_t - b_t$. There are D test trading days and Y calendar years in the test period. For a daily series z_t, $\bar{z}$ and $s(z)$ denote its sample mean and standard deviation.

- **Annualized return (AR).** $\mathrm{AR} = (W_D - 1)/Y$.
- **Sharpe ratio (SR).** $\mathrm{SR} = \sqrt{252}\,\bar{r}/s(r)$, with the risk-free rate set to zero.
- **Maximum drawdown (MDD).** $\mathrm{MDD} = \max_{1 \le t \le D} \left(1 - \frac{W_t}{\max_{0 \le j \le t} W_j}\right)$.
- **Return information ratio (RETIR).** $\mathrm{RETIR} = \bar{r}/s(r)$.
- **Annualized excess return (AER).** $\mathrm{AER} = (W_D - W_D^{\mathrm{bench}})/Y = Y^{-1} \sum_{t=1}^{D} e_t$.
- **Information ratio (IR).** $\mathrm{IR} = \sqrt{252}\,\bar{e}/s(e)$.

Backtesting protocol. For CSI300, CSI500, and S&P500, stocks are ranked by the predicted signal on each formation date. We form an equal-weighted long cohort from the top 20%, allocate one twentieth of the fixed initial capital V_0 to it, and hold it for 20 trading days. A new cohort is formed each day; thus, the portfolio's realized profit or loss on day t is obtained by marking all cohorts active on that day to market. A cohort begins earning returns only after its formation trade.

Let G_t be the one-day mark-to-market profit or loss of all active cohorts, and let B_t and S_t be the notionals bought and sold on day t. We compute

$$\Delta\Pi_t = G_t - c_{\mathrm{buy}} B_t - c_{\mathrm{sell}} S_t, \qquad r_t = \frac{\Delta\Pi_t}{V_0}, \qquad W_t = 1 + \sum_{\tau \le t} r_\tau,$$

where $c_{\mathrm{buy}} = 0.0005$ and $c_{\mathrm{sell}} = 0.0015$. All return-based portfolio metrics and cumulative-return curves are computed from this realized daily series. The 20-trading-day forward return is used

as the prediction target. The corresponding benchmarks are SH000300, SH000905, and ^GSPC. Benchmark-relative metrics compare strategy and benchmark gains on the same trading dates under the same capital convention. The investable universe is constructed point-in-time: on each date, it contains only constituents effective on that date. Features and portfolio decisions use only information available by the corresponding date. Samples whose 20-day return labels cross a training, validation, or test boundary are excluded.

C.1.3 BASELINE DETAILS

We compare AlphaPADI with nine baselines covering fixed factor libraries, predictive models, and formula-discovery methods:

- **Alpha158** (Yang et al., 2020) is a predefined library of 158 technical factors constructed from normalized OHLCV features over multiple lookback windows. The factor matrix is fitted to the return target using the common combination model.
- **MLP** (Murtagh, 1991) is a feed-forward neural network that maps tabular features to return targets, capturing nonlinear interactions. It provides a generic nonlinear baseline but offers limited interpretability.
- **XGBoost** (Chen & Guestrin, 2016) uses boosted regression trees with second-order optimization, shrinkage, feature subsampling, and explicit regularization to model tabular financial features.
- **GP** (Chen et al., 2021) evolves symbolic expressions through selection, crossover, and subtree mutation. Candidate formulas are evaluated by their predictive performance, and the selected formulas form the final alpha pool.
- **AlphaGen** (Yu et al., 2023) formulates formula construction as a sequential decision process and trains a masked PPO policy. Its reward measures the improvement contributed by a generated formula to a jointly optimized alpha pool.
- **AlphaQCM** (Zhu & Zhu, 2025) extends reinforcement learning-based alpha discovery with distributional value estimation. The estimated return distribution and uncertainty guide exploration under sparse and non-stationary pool rewards.
- **AlphaForge** is a generative–predictive method for formulaic alpha discovery, which couples formula generation with predictive modeling to discover alphas. (Its K selected formulas are evaluated with the shared static ridge combiner.)
- **AlphaAgent** (Tang et al., 2025) uses an LLM-based multi-agent workflow. Its Idea Agent proposes financial hypotheses, the Factor Agent translates them into executable formulas, and the Eval Agent provides empirical feedback for iterative revision.
- **AlphaSAGE** (Chen et al., 2026) represents formula ASTs with a relational graph convolutional network and trains a GFlowNet to sample diverse high-reward formulas. Its reward combines predictive performance, structural alignment, and novelty.

Shared comparison protocol. All baselines use the same point-in-time stock universes, chronological splits, and 20-day forward return targets described in Section 5.1. MLP and XGBoost receive the same Alpha158 features. Formula-discovery methods use the same primitive market fields, formula length limit, and executable-expression constraints. Each formula-mining method returns K formulas, which are evaluated with the static ridge combiner in Appendix C.1.1 and the backtesting protocol in Appendix C.1.2. All methods follow the same validation-based selection.

Search budget. We compare the non-LLM formula-search methods (GP, AlphaGen, AlphaQCM, AlphaForge, AlphaSAGE, and AlphaPADI) under a common target budget of $B = 40{,}000$ formula evaluations. A formula evaluation is a call to a method's evaluator that attempts to compute a candidate formula's signal over the historical panel. Reusing a cached signal without a new evaluator call does not consume this budget. The budget controls formula-evaluation work; it does not equate optimizer updates, generated tokens, or total runtime across methods. It is enforced at each method's evaluation call as follows:

- **AlphaGen and AlphaQCM** share the same environment and alpha pool, and training stops once the pool has evaluated $40{,}000$ formulas.
- **AlphaSAGE** evaluates one formula per GFlowNet episode and runs $40{,}000$ episodes.
- **AlphaForge** evaluates candidates in batches of 256 and stops after the batch that reaches $40{,}000$ evaluations. It may therefore exceed B by fewer than 256 evaluations ($< 0.64\%$).
- **GP** evolves a population of $1{,}000$ formulas for 40 generations and evaluates $1{,}000 \times 40 = 40{,}000$ formulas.

- **AlphaPADI** charges each distinct canonical formula once, on its first submission, including initialization, duplicate replacement, and failed submissions. A candidate whose unseen formulas would exceed the remaining budget is discarded.

For Figure 12, we additionally display $5n_{\text{eval}}$ as an evaluation-equivalent token-action scale, where n_{eval} is the number of formula evaluations. The factor of five corresponds to the median formula length of four tokens plus the terminating SEP action. This is only a rescaling of the figure's horizontal axis; it is not the actual number of training steps performed by every method and is not used as a stopping rule.

For AlphaPADI, each distinct canonical formula is charged on its first submission, including failed submissions, while cached repeats are not charged. The other methods stop according to the evaluator calls or their corresponding native units specified above. Thus, the target budget does not imply equal numbers of distinct formulas, scored pools, validation evaluations, optimizer updates, or equal runtime.

Implementations. For AlphaGen[1], AlphaQCM[2], AlphaForge[3], AlphaAgent[4], and AlphaSAGE[5], we follow the default settings of their open-source projects, apart from the shared protocol and search budget above. For Alpha158, MLP, XGBoost, and GP, we follow the settings in the AlphaForge repository.

C.1.4 HYPERPARAMETER SETTINGS

Baselines use the settings in Appendix C.1.3, with only the pool size $K \in \{20, 50\}$ selected on the validation split. For AlphaPADI, configurations are selected by the mean validation utility across five runs and fixed before test evaluation. Shared hyperparameters are searched over identical ranges wherever applicable, and parameters specific to a model are tuned within that model. All experiments are repeated five times, and we report the mean performance across the five runs. Statistical significance is assessed against the strongest baseline, with $p < 0.05$ considered statistically significant. Table 5 summarizes the settings of AlphaPADI.

Table 5: Hyperparameters of AlphaPADI. Sets are search spaces selected on the validation split.

Hyperparameter	**Symbol**	**Value**
Search		
Formula budget	B	40,000
Maximum formula length	$L_{\max}$	20
Pool size	K	$\{20, 50\}$
Initial pools, buffer capacity	M, $C_{\mathcal{Q}}$	64, 128
Proposal batch size	N_{c}	16
Buffer sampling temperature	T_{r}	0.5
Diffusion and objectives		
Diffusion levels, edit strength, start level	T, α, t_0	32, 0.5, 16
Diversity weight	w_{inner}	$\{1, 2, 5, 10\}$
Reconstruction weight	λ_{VB}	$\{0.1, 0.3, 0.5, 0.7\}$
Preference strength, margin	β, m_{dpo}	1.0, 0.5
Model and optimization		
Hidden dimension, attention heads, dropout	–	128, 4, 0.1
Formula, set, and decoder layers	–	2, 1, 2
Optimizer, weight decay, gradient clipping	–	AdamW, 10^{-4}, 1.0
Learning rate	–	$\{2\times10^{-5}, 3\times10^{-4}, 10^{-3}\}$
Minibatch size, initial updates, updates per round	N, S_0, S_{u}	16, 16, 8

[1] https://github.com/RL-MLDM/alphagen
[2] https://github.com/ZhuZhouFan/AlphaQCM
[3] https://github.com/DulyHao/AlphaForge
[4] https://github.com/RndmVariableQ/AlphaAgent
[5] https://github.com/BerkinChen/AlphaSAGE

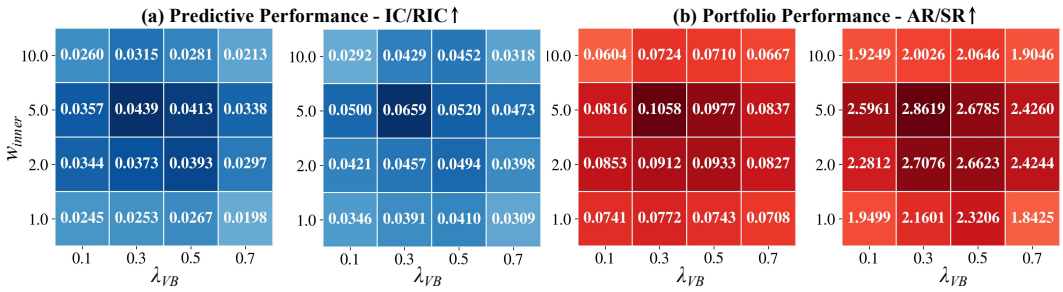

Figure 4: Joint sensitivity to w_{inner} (rows) and λ_{VB} (columns) on CSI300. Panel (a) shows IC and RIC, and panel (b) shows AR and SR. Annotations give metric values, and colors are normalized separately within each metric. Higher values are better for all four metrics.

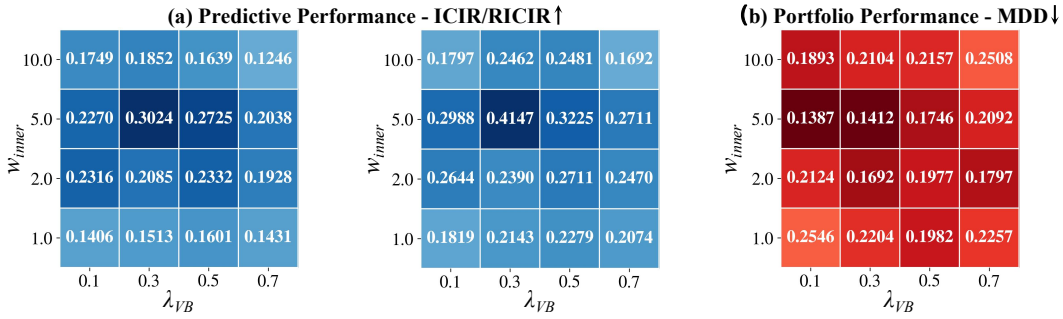

Figure 5: Joint sensitivity to w_{inner} (rows) and λ_{VB} (columns) on CSI300. Panel (a) shows ICIR and RICIR, and panel (b) shows MDD. Annotations give metric values, and colors are normalized separately within each metric. Higher ICIR and RICIR and lower MDD are better.

C.2 Additional Experimental Results

C.2.1 Hyperparameter Analyses

We vary only w_{inner} and λ_{VB}, keeping all other experimental settings and the checkpoint-selection protocol the same as in the main experiments. Figures 4 and 5 show joint sensitivity to the inner diversity reward weight w_{inner} and reconstruction-loss weight λ_{VB} on CSI300. Among the displayed configurations, $(w_{\text{inner}}, \lambda_{\text{VB}}) = (5.0, 0.3)$ has the highest IC, RIC, AR, and SR. Moderate diversity weights generally perform better than the smallest and largest tested values, while increasing the reconstruction weight to 0.7 reduces performance relative to the moderate range. These patterns describe sensitivity within the tested grid.

Figure 4 complements the IC, RIC, AR, and SR grid in Figure 5 with ICIR, RICIR, and MDD on CSI300. The setting $(w_{\text{inner}}, \lambda_{\text{VB}}) = (5.0, 0.3)$ attains the highest ICIR (0.3024) and RICIR (0.4147) in the displayed grid. Holding $w_{\text{inner}} = 5.0$, increasing λ_{VB} to 0.7 lowers these metrics to 0.2038 and 0.2711, respectively, and increases MDD from 0.1412 to 0.2092. The minimum MDD, however, is 0.1387 at $(5.0, 0.1)$, compared with 0.1412 at $(5.0, 0.3)$. Thus, the configuration with the strongest predictive information ratios does not also minimize drawdown. The grid therefore reveals a trade-off between predictive stability and drawdown within the evaluated configurations.

C.2.2 Backtesting Results

Figures 6 and 7 extend the CSI300 backtest in the main text to CSI500 and S&P500. On CSI500, AlphaPADI has a shallower cumulative-return trough than the baselines during the decline preceding September 2024, participates in the subsequent recovery, and ends above the competing strategies, with AlphaSAGE providing the closest final comparison. On S&P500, the methods track one an-

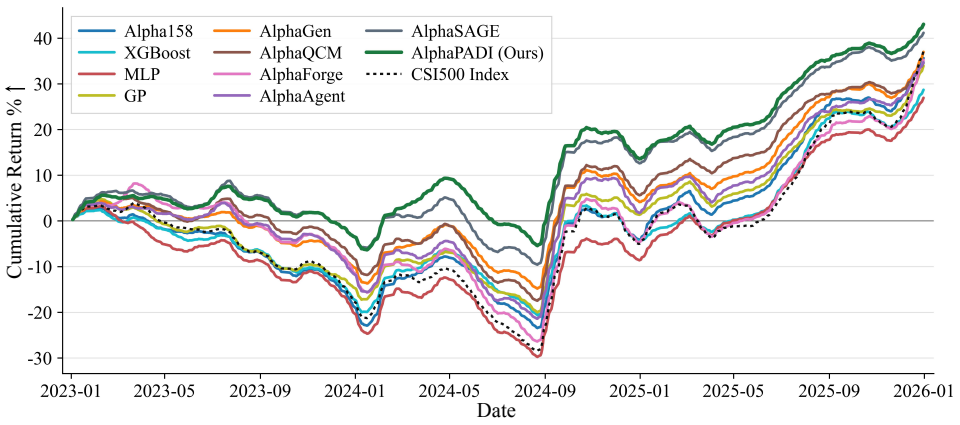

Figure 6: Cumulative returns on CSI500 (2023–2025): AlphaPADI, baseline methods, and the CSI500 index.

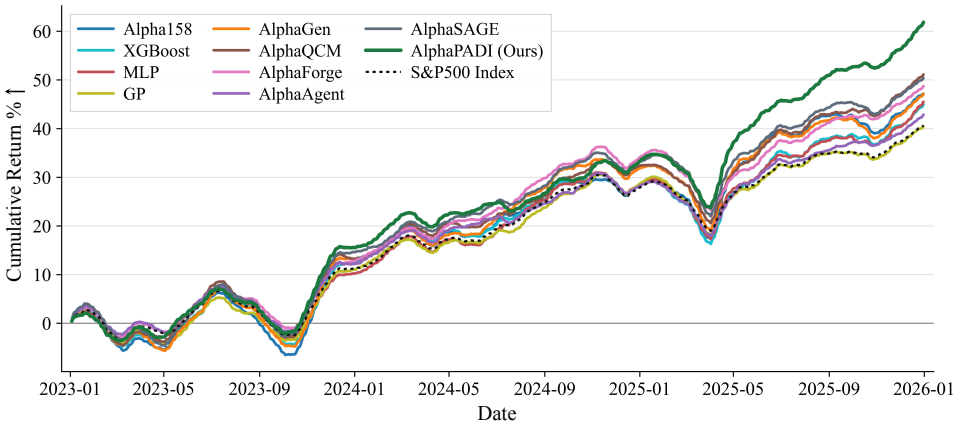

Figure 7: Cumulative returns on S&P500 (2023–2025): AlphaPADI, baseline methods, and the S&P500 index.

other more closely through much of 2023–2024, and the separation becomes pronounced after the decline in early 2025, when AlphaPADI recovers and finishes above the baselines and the index. Together, the two plots show when the differences in Table 1 emerge.

C.2.3 Cold-start Analysis

To evaluate whether AlphaPADI depends on a predefined factor library, we compare initialization with Alpha158 and Alpha360 against training from scratch on CSI300. The from-scratch variant ranks first on five of the seven reported metrics and second on the remaining two. Specifically, it obtains the best ICIR (0.3024), RIC (0.0659), RICIR (0.4147), AR (0.1058), and MDD (0.1412), while achieving the second-best IC (0.0439) and SR (2.8619). Initialization remains beneficial for particular objectives: Alpha158 initialization produces the highest IC of 0.0462 and SR of 3.0287. Refinement also improves the reported metrics relative to the two original factor sets. With Alpha360 initialization, for example, it increases IC from 0.0067 to 0.0401 and AR from 0.0538 to 0.0967, while reducing MDD from 0.4220 to 0.1687. For the two libraries tested, refinement improves performance over the corresponding library baselines. The from-scratch results further show that a predefined factor library is not required to obtain competitive performance on CSI300.

C.2.4 Performance on Other Metrics

Figure 8 presents a complementary evaluation using portfolio-level metrics: return information ratio (RETIR), annualized excess return (AER), and information ratio (IR) on the CSI300 and CSI500

Table 6: Comparison of Alpha158 initialization, Alpha360 initialization, and training from scratch on CSI300, including the original factor-library baselines. Higher values are better except for MDD. Best results are **bolded**, and second-best results are underlined.

Method	Initialization	Predictive Performance				Portfolio Performance		
		IC↑	ICIR↑	RIC↑	RICIR↑	AR↑	SR↑	MDD↓
Alpha158	–	0.0266	0.1631	0.0250	0.1389	0.0751	2.1532	0.2274
AlphaPADI	Alpha158	**0.0462**	0.2497	0.0632	0.3144	0.0952	**3.0287**	0.1582
Alpha360	–	0.0067	0.0326	0.0131	0.0611	0.0538	0.9706	0.4220
AlphaPADI	Alpha360	0.0401	0.2539	0.0541	0.3297	0.0967	2.4779	0.1687
AlphaPADI	From Scratch	0.0439	**0.3024**	**0.0659**	**0.4147**	**0.1058**	2.8619	**0.1412**

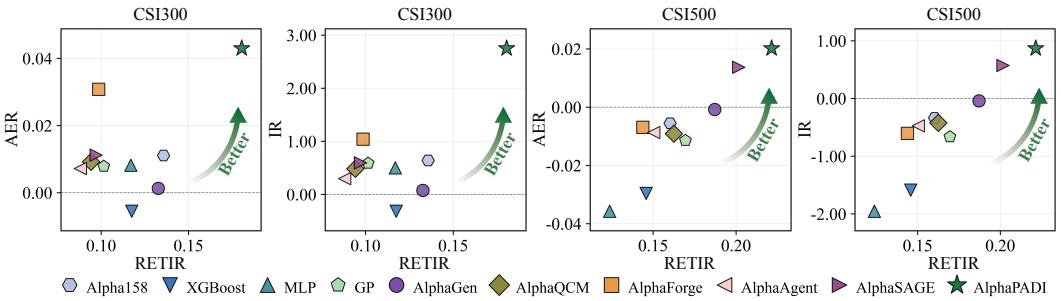

Figure 8: Additional portfolio metrics on CSI300 and CSI500: return information ratio (RETIR), annualized excess return (AER), and information ratio (IR).

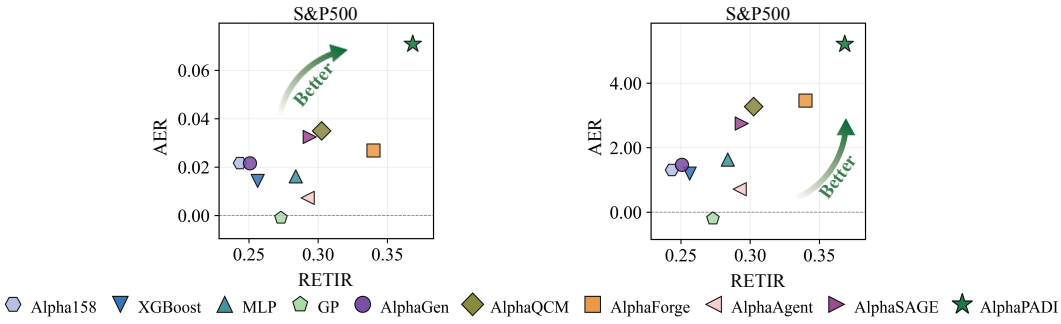

Figure 9: Additional portfolio comparisons on S&P500. The left panel plots AER against RETIR.

stock universes. These indicators complement the main results with additional measures of portfolio returns and benchmark-relative performance. AlphaPADI has the highest plotted values on both axes in each comparison. On CSI300, it achieves a RETIR of 0.1803, an AER of 0.0430, and an IR of 2.7520, outperforming AlphaForge by approximately 0.01 in AER and 1.7 in IR. On CSI500, AlphaPADI obtains a RETIR of 0.2213, an AER of 0.0202, and an IR of 0.8696. Relative to AlphaSAGE, the strongest competing method on this benchmark, these results correspond to improvements of approximately 10.5% in RETIR, 46.8% in AER, and 51.5% in IR. Notably, while several baselines produce near-zero or negative excess performance on CSI500, AlphaPADI remains positive on these reported metrics. These results provide additional evidence of positive benchmark-relative performance on the two evaluated stock universes.

Figure 9 extends the additional portfolio evaluation to S&P500. In the left panel, AlphaPADI occupies the upper-right position, with both a higher RETIR and a higher AER than the displayed baselines. This complements the Chinese-universe results by showing that the return-based and

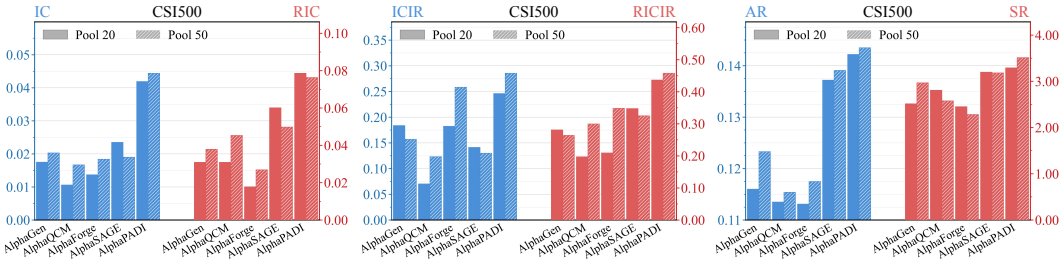

Figure 10: Performance of five alpha-mining methods with pool sizes $K = 20$ (solid) and $K = 50$ (hatched) on CSI500. Blue and red bars use the corresponding left and right metric axes.

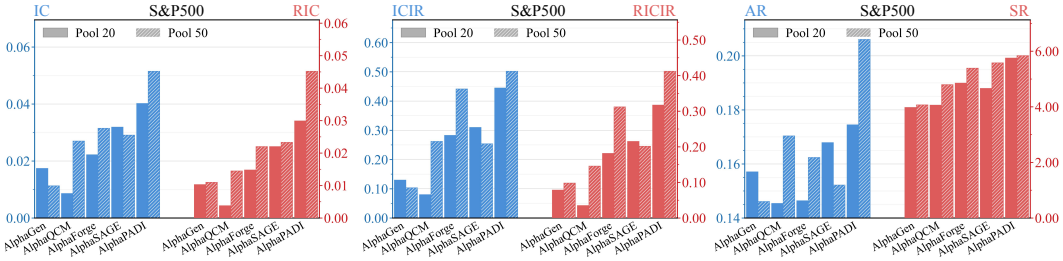

Figure 11: Performance of five alpha-mining methods with pool sizes $K = 20$ (solid) and $K = 50$ (hatched) on S&P500. Blue and red bars use the corresponding left and right metric axes.

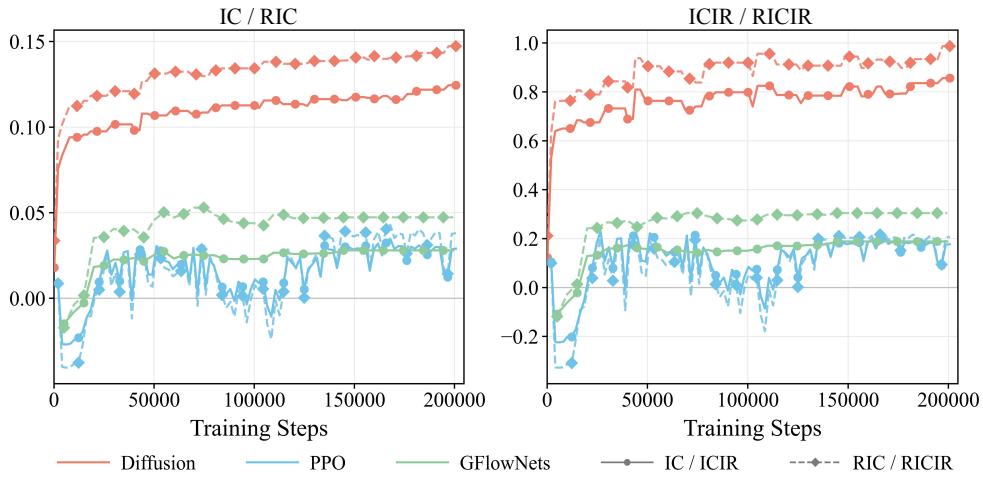

Figure 12: Search dynamics on CSI300: training-split IC/RIC and ICIR/RICIR of the pool each method would return during search. The horizontal coordinate labeled Training Steps in the plot is defined as $5n_{\text{eval}}$, where n_{eval} is the number of formula evaluations. This fixed rescaling is used only for display and does not represent optimizer updates, generated actions, or elapsed training time.

excess-return advantages also appear in the U.S. universe. AlphaForge has a higher RETIR but a lower AER than AlphaQCM, illustrating that rankings can differ across metrics.

C.2.5 Search Dynamics Under a Shared Formula-Evaluation Budget

With the budget fixed at 40,000 formula evaluations, the search-dynamics plot tracks the training-split quality of the pool each method would return at a given evaluation count. Its horizontal coordinate is $5n_{\text{eval}}$ as defined in the caption; comparisons along this axis concern formula evaluations

rather than actual training steps or wall-clock efficiency. Because the curves are measured on the training split, they describe search progress rather than generalization, which is evaluated in Table 1.

AlphaPADI reaches an IC of approximately 0.095 after about $2{,}000$ formula evaluations (displayed at 10^4 on the rescaled horizontal axis). At the end of the budget, its IC and RIC reach 0.1245 and 0.1473, respectively, whereas the PPO and GFlowNet curves end at an IC of approximately 0.03 or lower.

C.2.6 Effect of Pool Size

Figures 10 and 11 extend the pool-size comparison to CSI500 and S&P500. They compare pools of $K = 20$ and 50 for AlphaGen, AlphaQCM, AlphaForge, AlphaSAGE, and AlphaPADI. Increasing pool size does not uniformly improve every method or metric. On CSI500, for example, AlphaPADI improves IC and ICIR with the larger pool, while its RIC decreases slightly. On S&P500, its larger pool improves all six displayed metrics, including IC, RIC, AR, and SR. The differences across methods show that additional pool capacity alone does not guarantee better performance.