Major in International Finance

RESEARCH PAPER

Academic Year 2025-2026

Data Mining of Alpha Formulas Using Reinforcement Learning

Marina BLAZEVIC, Claire LANÇON

SUPERVISOR: Prof. Thomas BARRAU

ABSTRACT

The present research develops a reinforcement learning framework that generates and evaluates large populations of alpha formulas and combines them into a single meta-trading strategy. Candidate formulas are constructed as symbolic expression trees under a context-free grammar that guarantees validity, and the agent learns to favour structures whose information coefficient (IC) against returns residualized on eight common risk factors is high. Evaluated on 544 NASDAQ-listed equities (1995–2025) under a walk-forward protocol, with IC-weighted aggregation across discovered formulas, the resulting meta-strategy achieves a Sharpe ratio of 1.40 over a pseudo 20-years out of sample.

AI use disclosure - Generative AI tools were used only to refine the phrasing and clarity of sentences already written by the authors, and not to generate ideas, conduct the research, or produce any substantive content of this thesis.

SOMMAIRE

1- Literature Review

1.1 Cross-sectional alpha: From academic factors to formulaic programs

Cross-sectional return predictability has been studied extensively since Fama and French (1992) demonstrated that size and book-to-market equity capture substantial variation in average stock returns, and Jegadeesh and Titman (1993) documented the momentum effect. Subsequent work expanded the set of known predictors to include liquidity risk (Pastor and Stambaugh, 2003), statistical arbitrage residuals (Avellaneda and Lee, 2010), and hundreds of other anomalies. However, Harvey, Liu, and Zhu (2016) catalogued over 300 published factors, often called the factor zoo, and warned that conventional significance thresholds are insufficient to account for multiple testing across this expanding universe of candidate predictors. More critically, many published anomalies fail to replicate on new data, raising questions about both the stability of academic factor models and the efficiency of manual factor discovery which can explore only a small fraction of the possible signal space.

The concept of alpha has evolved from the theoretical intercept of a factor regression (Jensen, 1968) to an operational definition: any systematic, rule-based signal that forecasts relative returns. Kakushadze and Tulchinsky (2015) document an empirical analysis of approximately 4,000 real-life trading alphas for U.S. equities. Building on this foundation, Kakushadze (2016) released 101 explicit formulaic alphas, i.e. symbolic expressions composed of operators such as ranks, moving averages, or correlations applied to price and volume data. These 101 formulaic alphas were signals actively traded at WorldQuant (Kakushadze, 2016).

The concept of formulaic alphas connects alpha generation to program synthesis. Each alpha formula is a small program, i.e. a composition of operators (rank, correlation, moving averages, time-series functions, etc.) applied to operands (price series, volume, market capitalization, etc.). The space of valid programs grows combinatorially with expression depth and breadth. For instance, with 40 binary operators and 20 operands, the search space for formulas composed of up to 15 operators exceeds 10^{63} (Xu et al., 2024). This combinatorial explosion renders exhaustive search infeasible and motivates the development of automated discovery methods capable of navigating vast solution spaces efficiently while respecting the constraints of valid symbolic expression.

1.2 Automated discovery: Genetic programming and reinforcement learning

Genetic programming (GP) was one of the first systematic approaches to formulaic alpha construction and remains widely deployed in quantitative finance. GP represents candidate formulas as tree-structured expressions and evolves them through mutation, crossover, and fitness-based selection applied across successive generations. Zhang et al. (2020) demonstrate that GP-based search can produce competitive alpha signals in real markets. Cui et al. (2021) propose AlphaEvolve, an AutoML-based framework that mines weakly correlated alpha sets by rejecting candidates whose portfolio-return correlation with the existing set exceeds 15%, an important requirement for robust portfolio construction.

However, GP faces well-documented limitations. Ren, Qin and Li (2024) empirically validate a fundamental sparsity problem: when 10,000 candidate alphas are randomly generated by classic GP and evaluated on a held-out sample, fewer than 3% reach the standard effectiveness threshold of a mean information coefficient (IC) above 0.03, where the IC of an alpha is defined as the time-average of its daily cross-sectional Pearson correlations with forward returns, while the vast majority exhibit IC values close to zero. Most candidates are ineffective or noisy. Traditional GP also suffers from premature convergence: once a local optimum is found, a dominant gene quickly takes over the population, causing stagnation and high correlation among results, together with an infinite search space arising from unrestricted growth in the depth and length of tree-structured alphas, and the resulting "code bloat" that further hurts search efficiency. The search space is also structurally blind: crossover and mutation operators modify subtrees without understanding their semantic role. Finally, evaluating every candidate in the population is computationally expensive.

Reinforcement learning (RL) can be applied to alpha construction by formulating formula generation as a Markov Decision Process (MDP). An MDP is a mathematical framework for sequential decision-making defined by a tuple (S, A, P, R, γ), where S is the set of possible states, A is the set of available actions, P describes the probability of transitioning from one state to another given an action, R is the reward function, and γ is the discount factor that trades off immediate versus future rewards. The "Markov" property means that the optimal action depends only on the current state, not on the history of how the agent arrived there. In the context of alpha formula construction, the state represents the partially built formula at each step (which operators and operands have been placed so far), the actions are the possible next tokens or production rules to apply, the transition is deterministic (appending a token to the formula yields a unique next state), and the reward is received only at the end of the episode when the completed formula is evaluated for predictive performance. The agent's goal is to learn a policy, i.e. a mapping from states to

actions, that maximizes the expected cumulative reward, which in our setting corresponds to discovering formulas with high IC. Yu et al. (2023) introduced AlphaGen, in which a Proximal Policy Optimization (PPO) agent builds a formula one token at a time, picking operators (e.g., add, log, moving average) and operands (e.g., price, volume) in sequence. Their key innovation was using the IC of the combined alpha set as the reward signal rather than the IC of each individual formula, so the generator is rewarded only for alphas that improve the existing pool's combined predictive performance, enabling the discovery of synergistic alpha sets. Zhao et al. (2025) identify stability issues with PPO in the near-deterministic environment of symbolic generation and propose a variance-bounded REINFORCE alternative (QuantFactor REINFORCE) as more suitable for this setting.

These approaches, however, employ flat token-sequence generation: the agent picks from a vocabulary of tokens at each step without explicit grammar constraints. This creates a fundamental efficiency problem. Most random token sequences are syntactically invalid. The RL agent must learn the grammar of valid expressions while simultaneously learning which valid expressions are predictive. This dual learning objective wastes exploration capacity on invalid sequences that never reach evaluation, reducing sample efficiency. As Ren et al. (2024) demonstrate empirically, the effective alpha density within unconstrained search is low, making implicit grammar learning prohibitively expensive.

1.3 Grammar-constrained generation

Trivedi et al. (2021) show in their LEAPS framework for synthesizing programmatic policies that a RL agent generating program tokens sequentially fails to produce complex structures such as control flow (loops, conditionals). The agent instead learns to emit long sequences of simple action tokens, because these yield partial reward more readily than the coordinated multi-token investment required to compose a syntactically valid loop. This fundamental observation directly motivates grammar-constrained approaches.

Kusner et al. (2017) introduced the Grammar Variational Autoencoder, which constrains a VAE decoder to produce outputs according to a context-free grammar (CFG), guaranteeing validity. Dai et al. (2018) extended this framework with Syntax-Directed VAEs, applying grammar constraints at multiple hierarchical levels. While these methods operate in a VAE framework, the core principle that a context-free grammar can enforce structural validity in generative models by restricting valid actions at each step applies equally to reinforcement learning. In a grammar-guided RL setting, the agent selects production rules rather than raw tokens, and the grammar restricts valid actions at each step to those applicable to the current non-terminal symbol.

1.4 Alpha evaluation, combination, and walk-forward testing

Evaluation of alphas requires metrics that capture predictive power and stability. The IC, defined as the correlation between alpha values and future returns on a given day, averaged across the evaluation period, measures cross-sectional predictive ability (Grinold and Kahn, 2000; Zhang et al., 2020).

Alpha decay, the deterioration of predictive power over time, is a persistent challenge driven by distribution shifts in financial data. Wang et al. (2025) address this through their QuantBench framework, showing that rolling re-estimation mitigates alpha decay in predictive quant models: more frequent model updates (e.g. 3-month rolling) consistently outperform less frequent updates and a no-rolling baseline, as they help address the distribution shifts caused by market evolution. This motivates walk-forward evaluation protocols in which the RL agent is retrained on each successive fold of data, enabling continuous adaptation rather than relying on a single model trained once.

Kakushadze (2014) demonstrates that the covariance matrix across thousands of production alphas is nearly singular, requiring structured regularization approaches. Combining many alphas should stabilize performance. However, automated discovery methods often produce alphas with overlapping predictive content, even when their mathematical forms differ. Managing this redundancy is therefore a central concern: Cui et al. (2021) address it by rejecting candidates whose portfolio-return correlation with the existing set exceeds 15%, while Yu et al. (2023) show that pairwise correlation is an unreliable redundancy signal, alphas with mutual IC above 0.97 can still combine synergistically, and instead use the downstream combination model's performance to decide which alphas add value.

The present work sits at the intersection of several research streams. From the academic factor literature, we adopt the operational definition of alpha and IC-based evaluation standards developed in quantitative finance. From RL-based alpha generation (Yu et al., 2023), we take the core MDP formulation and adopt policy gradient training with PPO; however, we depart fundamentally from flat token-sequence generation. Instead, we employ grammar-guided construction of formulas according to a context-free grammar. From the grammar-constrained generation literature (Kusner et al., 2017; Dai et al., 2018), we adopt the principle of using a CFG to enforce syntactic validity, but we apply this principle within a RL framework with action masking rather than within a VAE. This choice allows us to leverage modern deep RL algorithms (PPO with parameter masking, MaskablePPO) while maintaining the sample efficiency and theoretical guarantees of grammar-guided exploration.

Finally, we adopt evaluation practices: IC-based assessment, walk-forward backtesting to prevent overfitting, factor residualization to ensure our signals are not simply rediscovering known factors, and IC-

weighted aggregation. The result is a meta-trading strategy which combines the structural guarantees of grammar-guided generation, the learning efficiency of policy gradient methods with action masking in sparse spaces, and the rigorous out-of-sample evaluation practices.

2. Data

2.1 Data source and sample period

To develop and evaluate systematic formula-based trading strategies, we constructed a comprehensive daily-frequency dataset spanning thirty years (1995–2025) of NASDAQ-listed equities sourced from Bloomberg's database. The full thirty-year period allows us to evaluate strategies across multiple market cycles and regimes.

2.2 Universe definition

To capture historical index composition and its evolution, we extracted the full constituent list from Bloomberg monthly from 1995 to 2025. Using these monthly snapshots, we identified every unique stock that appeared in the NASDAQ index at any point during this period. This approach ensures we include delisted, merged, and historically significant constituents rather than observing only the current index composition, which would introduce survivorship bias into our analysis. For stocks that remain listed and active, we have obtained a complete daily record across this entire window. For historical constituents that were delisted, merged with other companies, or otherwise ceased trading, we collected available data from 1995 or listing to the last published price. The final dataset comprises 544 U.S. equities with daily price history throughout the evaluation period. All securities are denominated in U.S. dollars.

2.3 Data variables

For each security and each trading day, we collect eight core variables representing fundamental market data:

Close Price: The final transaction price at market close. This represents the most referenced daily price and serves as the basis for unadjusted return calculations.

Adjusted Close Price: The closing price after full adjustment for all corporate actions, including stock splits, dividend distributions, and other distributions. Adjusted prices enable accurate historical return calculations and ensure that factor normalizations and return metrics remain consistent across long time periods where cumulative corporate actions may otherwise distort price-based measures.

Open Price: The opening transaction price on the trading day. This variable captures overnight sentiment and opening imbalances that often contain predictive information.

High Price: The highest price at which the security traded during the trading day. High prices combined with low prices enable calculation of intraday range metrics and daily volatility estimates.

Low Price: The lowest price at which the security traded during the trading day.

Volume: The total number of shares traded throughout the trading day, measured in shares.

Volume-Weighted Average Price (VWAP): The average price transacted during the trading day, weighted by the volume executed at each price level. This metric represents a realistic execution price for large institutional orders.

Market Capitalization: The product of outstanding shares and current stock price, expressed in U.S. dollars. Market capitalization captures the size dimension of equities.

2.4 Methodological rationale

Corporate Actions and Historical Data Integrity

We incorporate explicit information on stock splits and dividend distributions from Bloomberg's corporate actions database. This prevents forward-fill errors and ensures that any alpha formulas incorporating price levels or yield metrics operate on corrected historical data. This is particularly important for long-history datasets spanning 1995 to 2025, where cumulative corporate actions can produce distortions in nominal price series.

Adjusted versus Unadjusted Prices

A critical issue concerns the adjustment factor itself (the ratio of unadjusted to adjusted close price). It is mechanically large for stocks that will split or pay dividends in the future, events that typically follow sustained price appreciation, making it a powerful predictor of future returns, but only in hindsight. Using it directly would inject information unavailable to an investor at the time, producing a false signal driven by look-ahead bias. Our analysis employs adjusted prices to avoid look-ahead bias in backtesting. By using adjusted prices throughout our analysis, we ensure that all discovered alpha signals could theoretically have been implemented by traders operating with information available at the time. This choice is fundamental to the validity of our out-of-sample results.

Volume and Liquidity Metrics

The explicit extraction of VWAP alongside raw volume serves two complementary purposes. First, VWAP enables rapid prototyping of volume-weighted factors without requiring reconstruction of intraday order

book data, reducing computational burden while maintaining economic realism. Second, VWAP provides a benchmark price for evaluating execution quality.

Index Composition and Survivorship Bias

We extract monthly snapshots of index composition (the set of securities tradable at each point in time) throughout our sample period. This captures the investment universe available to traders at each historical date, avoiding both survivorship bias, which would omit failed companies and overstate historical returns, and false signal discovery, which would test strategies on stocks not actually tradable in historical periods. By conditioning our analysis on the tradable universe at each point in time, we ensure that our final trading strategy is evaluated on realistic, implementable opportunity sets.

For the purposes of transparency and reproducibility, the complete source code for our framework is publicly available at https://github.com/marinablaz/MasterThesis-DataminingOfAlphaFormulasUsingReinforcementLearning

3. Methodology

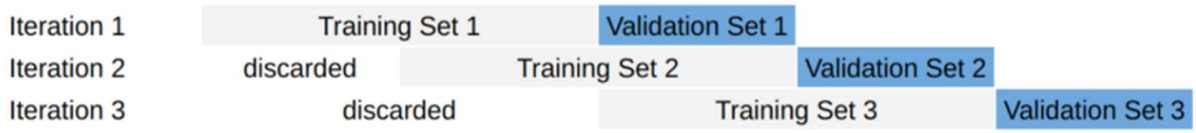
This section describes the complete methodology behind our grammar-guided reinforcement learning framework for an alpha-based trading strategy. We begin with the walk-forward cross-validation design that structures our out-of-sample evaluation, then present the formula representation and grammar, the RL environment and training procedure, the reward design, the alpha evaluation pipeline, and finally the aggregation scheme that combines individual formulas into a composite alpha signal.

3.1 Walk-forward cross-validation

3.1.1 Presentation

Walk-forward cross-validation is the cornerstone of our evaluation methodology. Unlike basic train/test splits or k-fold cross-validation, both of which are inappropriate for financial time series as they violate temporal ordering and induce look-ahead bias, walk-forward validation enforces a strict chronological separation: the model is trained exclusively on past data and evaluated on subsequent, previously unseen periods. This protocol mirrors the constraints faced by a practitioner deploying trading strategies in real time and provides a more faithful estimate of out-of-sample performance.

Figure 1 - Walk-forward cross-validation



We partition the sample period into a sequence of overlapping folds, each comprising a ten-year training window followed by a one-year out-of-sample (OOS) testing window.

Table 1 - Training & Out-of-sample Periods (Available in full in the appendix – See Table A1)

Fold	Training Period	OOS Period
1	1995-01-01 to 2005-01-01	2005-01-01 to 2006-01-01
2	1996-01-01 to 2006-01-01	2006-01-01 to 2007-01-01

...	...	...
20	2014-01-01 to 2024-01-01	2024-01-01 to 2025-01-01
21	2015-01-01 to 2025-01-01	2025-01-01 to 2025-12-25

Several design choices underpin our cross-validation protocol. First, a ten-year training window provides approximately 2,520 trading days, which is long enough for the RL agent to observe a diversity of market regimes (sustained bull markets, corrections, and volatility spikes), yet short enough to remain responsive to evolution in market structure. Second, the one-year OOS window is long enough to assess whether a signal persists beyond the training period, yet short enough to permit frequent re-estimation as new data become available. Third, the one-year rolling step between consecutive folds induces substantial overlap in the training windows, for instance folds 1 and 2 share nine of their ten training years. While this overlap reduces the effective statistical independence of the folds, it reflects the operational reality that a systematic trading strategy would be retrained periodically on the most recent available data.

The key property of this design is that no future information ever leaks into signal construction. The RL agent for fold k is trained exclusively on data from the corresponding training period, and the formulas it discovers are subsequently evaluated on the one-year OOS window that follows. Crucially, the ICs used to weight and orient the formulas in the aggregation stage are themselves computed on the training period, never on the testing period. This temporal separation is essential for credible out-of-sample inference and distinguishes our protocol from data-mining exercises in which signal selection and signal evaluation share information.

3.1.2 Why does walk-forward protect against overfitting?

The principal threat to automated alpha discovery is overfitting: the RL agent may learn to exploit idiosyncratic noise in the training data rather than genuine predictive structure. Walk-forward validation mitigates this risk through three complementary mechanisms. First, by retraining the agent independently on each fold, we prevent the accumulation of in-sample fit across the full dataset; a formula that captures a spurious pattern in one training window will fail on the corresponding OOS window and be naturally down-weighted in the aggregation. Second, by evaluating successive one-year OOS windows that span markedly different market regimes (post-GFC recovery, late-cycle expansion, the COVID-19 shock, and the subsequent post-pandemic rotation), we directly test signal robustness to regime change. Third, the IC-

weighted aggregation provides an additional layer of protection by concentrating weight on formulas that exhibit consistent in-sample IC, which are a priori more likely to retain predictive power out of sample.

An alternative would be to train the RL agent on the full 1995–2025 sample and subsequently evaluate the discovered formulas in the same period. Such a procedure would yield impressive in-sample Sharpe ratios but entirely uninformative estimates of out-of-sample performance. Our walk-forward design avoids this pitfall by construction: every observation contributing to our reported performance metrics was out of sample when the corresponding signal was generated. Our reported performance is based on the temporal aggregation of all our OOS windows, as pictured below.

Figure 2 - Temporal aggregation

We must, however, acknowledge a subtler form of in-sample fitting that the walk-forward structure does not fully eliminate: the selection of hyperparameters. Choices such as the learning rate, entropy coefficient, maximum formula depth, and rolling window lengths were made by observing the aggregate OOS performance across all folds. While individual formulas and their IC weights are determined without access to future data, the global configuration under which those formulas are generated was itself implicitly optimized using information from the entire evaluation period. This constitutes an indirect form of look-ahead: not fitting trading signals to future returns but fitting the meta-parameters of the discovery process to the overall out-of-sample outcome. This form of overfitting is considerably less severe than direct in-sample optimization, because the hyperparameters sit behind a thick veil between our choices and the final "PnL performance". Nevertheless, it remains a source of bias. We will address it through robustness tests showing that performance remains positive across a range of hyperparameter settings, demonstrating that our results are not driven by a single carefully tuned configuration.

3.2 Residualization

Residualization is the process of regressing a variable of interest on a set of explanatory factors and retaining only the residual, the component that cannot be explained by any linear combination of those factors. By construction, OLS residuals are orthogonal to every regressor: the correlation between ε_t and each column of F_{t-1} is exactly zero. This means that if we residualize our alpha positions against momentum, size, and volatility, the resulting residualized positions have zero exposure to each of these factors on every day, so any PnL they generate should come from a source of predictive information not captured by the factor set.

In this paper, to assess whether an alpha signal captures genuine predictive power beyond known risk factors, we residualize against eight common factors and the stock's adjustment factor:

- **Momentum**: cumulative return from day 21 to day 252 (Jegadeesh and Titman, 1993).
- **Five reversal factors**: 1-day, 2-day, 5-day, 21-day, and long-term (252–756 day) reversal (Avellaneda and Lee, 2010).
- **Low volatility**: 252-day rolling standard deviation.
- **Size**: log market capitalisation (Fama and French, 1992).
- **Adjustment factor**: the ratio of raw to adjusted close price, proxying for corporate-action exposure.

Formally, the procedure operates as follows. For each trading day t, we run a cross-sectional OLS regression of the target variable on the factor scores observed at $t - 1$.

$$y_t = F_{t-1}\,\beta_t + \varepsilon_t \qquad (1)$$

The OLS estimator is:

$$\widehat{\beta_t} = (F_{t-1}^{\top} F_{t-1})^{-1} F_{t-1}^{\top} y_t \qquad (2)$$

And the residualized target is:

$$\hat{\varepsilon}_t = y_t - F_{t-1}\widehat{\beta_t} \qquad (3)$$

Where:

- $y_t \in R^{Nt}$ is the target vector across the N_t stocks in the universe on day t (either the forward returns or the alpha positions to be residualized);
- $F_{t-1} \in R^{Nt \times 9}$ is the matrix of factor exposures observed one day prior (at $t - 1$), whose nine columns correspond to the eight common risk factors and the adjustment factor;
- $\beta_t \in \mathbb{R}^9$ is the vector of factor loadings, and $\widehat{\beta_t}$ its OLS estimate;
- $^{\top}$ denotes the matrix transpose and $(\cdot)^{-1}$ the matrix inverse;
- $\varepsilon_t \in R^{Nt}$ is the residual vector, and $\hat{\varepsilon}_t \in R^{Nt}$ the estimated residualized target, i.e. the component of y_t orthogonal to the factor exposures.

Using only information available as of close on day $t - 1$ to compute the factor loadings, the residual represents the component of the target unexplained by the factors, and it is this residual that enters our PnL and IC computations.

During the RL training phase, the IC is computed against these residualized returns, ensuring that the agent is rewarded only for discovering signals that predict returns beyond what known factors already explain. This is a deliberately harsh criterion: formulas exhibiting high raw IC because they capture momentum or size effects will receive low residualized IC and correspondingly low reward.

We then evaluate the composite alpha through a residualized evaluation, in which the alpha positions, rather than the returns, are residualized against the risk factors via cross-sectional regression on each trading day, and the residualized positions are then multiplied by raw returns. The Sharpe ratio is computed as the annualized ratio of mean daily PnL to its standard deviation, $Sharpe = \frac{\mu}{\sigma} \times \sqrt{252}$, and we report the residualized Sharpe ratios for each configuration.

The ideal approach to factor-neutral evaluation would be to residualize positions before trading: on each day, regress the alpha positions on the factor exposures and trade only the residual component, producing a portfolio that is by construction orthogonal to all the factors discussed, with a PnL approximately orthogonal to the PnL of each factor. However, this approach is operationally expensive, as it requires solving a cross-sectional regression and rebalancing the entire portfolio every day, with the associated transaction costs and turnover. In practice, we use residualized returns as a proxy: both approaches yield similar PnL profiles, because they project onto the same subspace orthogonal to the factor exposures, but the return-based approach is computationally simpler and avoids the rebalancing overhead. We acknowledge that this is an approximation and did not explore the distinction further within the scope of this thesis.

3.3 Alpha formulas as mathematical expressions

We represent alpha signals as mathematical expressions composed from a domain-specific language of operators and terminals. Each formula takes as input the feature matrices (close, open, high, low, volume, etc.) and produces a single output matrix of the same shape (T × N), where each entry represents the alpha signal for a given security on a given day.

3.3.1 Operators

The operator set is partitioned into three categories:

- **Unary operators** (one argument): CSRank (cross-sectional percentile rank), Abs (absolute value), Log (natural logarithm of the absolute value), and Sign (signum function, returning −1, 0, or +1).
- **Binary operators** (two arguments): Add, Sub, Mul, Div (with division-by-zero protection), and Pow (exponentiation with overflow detection).
- **Time-series operators** (one expression and one window parameter): Mean, EMA (exponential moving average), Std (rolling standard deviation), Var (rolling variance), Delta (first difference at lag k), Lag (value k days prior), Sum (rolling sum), Max (rolling maximum), Min (rolling minimum), Skew, Kurt, and Median.

Formulas are encoded in prefix notation, in which each operator precedes its arguments. For instance, the twenty-day exponential moving average of the cross-sectional rank of closing prices is represented as [BEG, EMA, CSRank, close, 20, END]. This representation maps naturally onto a tree structure in which operators occupy the internal nodes and features or numerical constants form the leaves.

3.3.2 Terminal sets

The terminal feature set comprises eight market variables: close, adjusted close, open, high, low, volume, volume weighted average price, and market capitalization. The terminal numerical set supplies window parameters for the time-series operators and is restricted to {5, 10, 20, 60, 252}, corresponding to trading horizons of one week, two weeks, one month, one quarter, and one year, respectively.

3.4 Grammar-guided formula generation

The central methodological contribution of this paper is a grammar-guided reinforcement learning environment in which formula generation is constrained by a context-free grammar (CFG). Rather than emitting tokens sequentially from a flat vocabulary, the agent constructs a formula tree top-down, selecting a production rule at each step. This section explains the grammar, the derivation process, and why this approach resolves the structural problems of unconstrained generation.

3.4.1 The Context-Free Grammar (CFG)

The grammar is built automatically from the available operators, features, and numerical constants. Instead of generating formulas as arbitrary token sequences, the agent constructs them by repeatedly applying valid expansion rules.

At a high level, the agent starts from a single placeholder, an empty expression slot that needs to be filled, and progressively decides how to expand it. At each step, it chooses one of several types of expansion:

- A unary transformation such as CSRank(·), Abs(·), Log(·), or Sign(·), which takes one sub-expression as input.
- A binary combination such as Add(·, ·), Sub(·, ·), Mul(·, ·), or Div(·, ·), which takes two sub-expressions.
- A time-series operator such as Mean(·, w), EMA(·, w), or Std(·, w), which takes one sub-expression and one window parameter.

Each choice creates new slots that must themselves be filled. A unary operator creates one new expression slot; a binary operator creates two; a time-series operator creates one expression slot and one number slot. The process continues recursively until every branch terminates in either a feature variable or a numerical constant.

This grammar-based construction guarantees that every generated formula is syntactically valid by design. The agent never produces malformed expressions, mismatched arguments, or invalid operator nesting. In practice, the grammar derived from our operator library contains approximately 30 production rules: one per operator, plus the terminal rules for features and numbers.

3.4.2 Stack-based top-down derivation

The grammar defines which expansions are valid, but the agent still needs a mechanism to keep track of which slots remain to be filled as it builds a formula. We use a stack, i.e., a last-in, first-out list, that records every unfilled slot in the formula under construction.

The process begins with a single entry on the stack: one expression slot to fill. At each step, the agent looks at the top slot on the stack, decides how to fill it, and if that decision creates new slots (for instance, choosing Add creates two new expression slots), those are pushed onto the stack. When the stack is empty, every slot has been filled and the formula is complete.

To illustrate, we trace the construction of Mean(CSRank(close), 20):

- **Step 1:** the stack contains one expression slot. The agent chooses "rolling mean," which needs a sub-expression and a window size. Two new slots are pushed: one expression slot and one number slot. The expression slot sits on top, so it will be filled next.
- **Step 2:** the top slot is an expression. The agent chooses "cross-sectional rank," which needs one sub-expression. A new expression slot is pushed.

- **Step 3**: the top slot is again an expression. This time, the agent chooses to place a feature here, so a feature slot is pushed.
- **Step 4:** the top slot is a feature. The agent picks "close." Nothing new is pushed. The number slot from Step 1 is now on top.
- **Step 5**: the top slot is a number. The agent picks "20." The stack is empty. The formula Mean(CSRank(close), 20) is complete.

The key property is that at each step, the agent only sees the choices that are valid for the current slot type. When filling an expression slot, the agent chooses among operators and the option to place a feature directly. When filling a feature slot, it sees only the six feature names. When filling a number slot, it sees only the seven window sizes. This means the agent typically faces a few meaningful choices per step, rather than the dozens of undifferentiated tokens of a flat-vocabulary approach, yielding a much more tractable decision problem that enables faster and more directed learning.

3.4.3 Action masking for structural constraints

Beyond the grammar's inherent validity constraints, we impose additional structural constraints through action masking. These include:

- Maximum tree depth (default: 8): prevents excessively deep and overfit formulas. When the current depth approaches the maximum, only the feature leaf rule is allowed, forcing the agent to terminate that branch.
- Maximum node count (default: 40): bounds total formula complexity.
- Minimum tree depth (default: 3): prevents trivially simple formulas like bare features. Below the minimum depth, the feature leaf rule is masked out.

3.4.4 Advantages over flat token-sequence generation

Our first implementation used a flat token-sequence approach where the RL agent generated formula tokens one by one from a vocabulary of approximately 40 symbols. This approach suffered from three fundamental problems that the grammar-guided approach resolves:

- Structural validity: in the flat approach, most random token sequences are syntactically invalid (e.g., Mean(END, CSRank)). The grammar approach guarantees validity by construction, eliminating wasted episodes on malformed formulas.

- Action space size: in the flat approach, the agent sees all 40 tokens at every step. In the grammar approach, the agent sees a few contextually relevant choices. This smaller action space enables faster learning and better exploration.
- Credit assignment: in the flat approach, the reward arrives at the end of a 15-token sequence, providing little information about which individual token choices were good or bad. In the grammar approach, each step involves a semantically meaningful decision (choosing a formula structure, selecting a feature, or picking a window size), making credit assignment more tractable.

3.5 RL environment design

3.5.1 Observation space

The agent's observation at each step is a 23-dimensional vector, encoding the current state of the derivation:

- Current non-terminal type (3 dimensions, one-hot): whether the agent is choosing a formula structure (EXPR), a feature (FEATURE), or a number (NUM).
- Tree statistics (5 dimensions): normalized depth so far, normalized node count, number of features used, normalized stack size and number of pending EXPR non-terminals.
- Rule history (15 dimensions): the IDs of the last 15 production rules chosen, providing sequential context.

3.5.2 Training algorithm

We use Proximal Policy Optimization (PPO) from the stable-baselines3 python library, specifically the MaskablePPO variant from sb3-contrib. The policy network is a standard MLP (multilayer perceptron) with two hidden layers of 64 units each and tanh activations. Key hyperparameters are summarized below:

Table 2 - Key hyperparameters

Parameter	Value	Rationale
Learning rate	5×10^{-4}	Moderate; balances convergence speed and stability
Entropy coefficient	0.1	Encourages exploration of diverse formula structures
Batch size	256	Standard for discrete action spaces

Steps per rollout (n_steps)	2048	About 50 complete formula episodes per rollout
Training timesteps per fold	30,000	Convergence dependent on each problem, but supposed to help achieve significant results
Discount factor (γ)	1.0	Episodic task; only terminal reward matters

3.6 Reward design

The reward function is critical to the quality of generated formulas. We assign a reward of zero for all intermediate steps (each production rule application) and compute the reward only when the formula is complete (stack empty). Invalid or degenerate formulas receive a fixed penalty of -6.0.

For valid formulas, we compute the Spearman information coefficient (IC) between the lagged alpha signal (at time t, we use the alpha computed at t−1) and returns over the training period. The reward is based on the absolute value of the mean IC, following the principle that the agent should discover formulas with strong predictive power regardless of direction; the sign correction is applied at the out-of-sample aggregation stage using only backward-looking information. A strongly negative IC simply implies that the position is shorted rather than held long, and vice versa.

The reward function maps |IC| to a reward value via a log-quadratic function calibrated to three anchor points:

- |IC| = 0.001 maps to reward 5 (weak but detectable signal)
- |IC| = 0.01 maps to reward 50 (strong signal)
- |IC| = 0.1 maps to reward 100 (exceptionally strong signal)
- Formally, letting x = log10(|IC|):

$$reward = 2.5x^2 + 57.5x + 155, \text{ clipped to } [-6, 100]$$

This function is smooth and monotonically increasing, providing clear gradient signal at all IC levels. The logarithmic scaling naturally produces diminishing marginal returns in raw IC: the reward gain from improving a formula's IC from 0.001 to 0.01 is comparable to the gain from 0.01 to 0.1, discouraging the agent from over-optimizing a single formula at the expense of exploration.

3.7 Alpha evaluation pipeline

Each complete formula undergoes the following evaluation pipeline within the RL environment:

- **Step 1** - Compilation: the formula tree is recursively evaluated against the feature matrices to produce a raw alpha DataFrame of shape (T × N).
- **Step 2** - Degenerate formula detection: we check cross-sectional variance (rejecting formulas where more than 30% of dates have near-zero standard deviation across stocks) and minimum uniqueness (rejecting formulas where most dates have fewer than 5 unique values across stocks). These checks catch structurally degenerate formulas such as Sign(Var(...)) which returns +1 for all stocks.
- **Step 3** - Standardization: the raw alpha is cross-sectionally ranked, centered to [-0.5, 0.5] (ensuring dollar neutrality), and L1-normalized so that the absolute values of positions sum to 1 on each date, meaning there is a leverage of 1.
- **Step 4** - Volatility normalization: the standardized alpha is divided by a pre-computed rolling 63-day volatility estimate of stock returns, down-weighting positions in volatile stocks. This step follows from the Markowitz (1952) mean-variance framework: when asset returns are uncorrelated, optimal portfolio weights are inversely proportional to each asset's variance. Estimating the full 544×544 covariance matrix is impractical and numerically unstable, so we use only the diagonal (each stock's own variance), which reduces to dividing positions by rolling volatility. This ensures no single volatile stock dominates portfolio risk. The rolling volatility is computed once at environment initialization and cached.
- **Step 5** - Final standardization: a second round of cross-sectional ranking and normalization after volatility adjustment.
- **Step 6** - IC computation: the lagged alpha (shifted by one day) is correlated (Spearman) against returns for each day, producing a daily IC time series. The mean of this series is the formula's IC score, which is passed to the reward function.

3.8 Formula collection and OOS aggregation

3.8.1 Collection phase

Following training, we collect formulas by executing the trained agent for 500 episodes. Each episode produces a complete formula, which is evaluated through the same pipeline used during training. We retain formulas that pass all validity checks and yield non-NaN IC values, discarding exact duplicates via a seen-

set lookup. For each retained formula, we store the token representation, the in-sample mean IC, and the exponentially-weighted moving average of IC (span 252).

3.8.2 IC-weighted aggregation

The aggregation of multiple formulas into a final alpha is a critical design choice. We adopt IC-weighted aggregation, in which each formula's out-of-sample positions are multiplied by its in-sample IC value prior to averaging. This procedure serves two purposes simultaneously. First, it provides sign correction: a formula with a negative in-sample IC, that is, one whose raw signal is inversely correlated with returns, has its positions reversed, converting a negative signal into a positive one. Second, it implements quality weighting: formulas with higher |IC| receive proportionally greater weight, concentrating the composite signal on the most informative components. The approach requires only a single scalar parameter per formula and provides an intuitive, non-parametric alternative to full portfolio optimization.

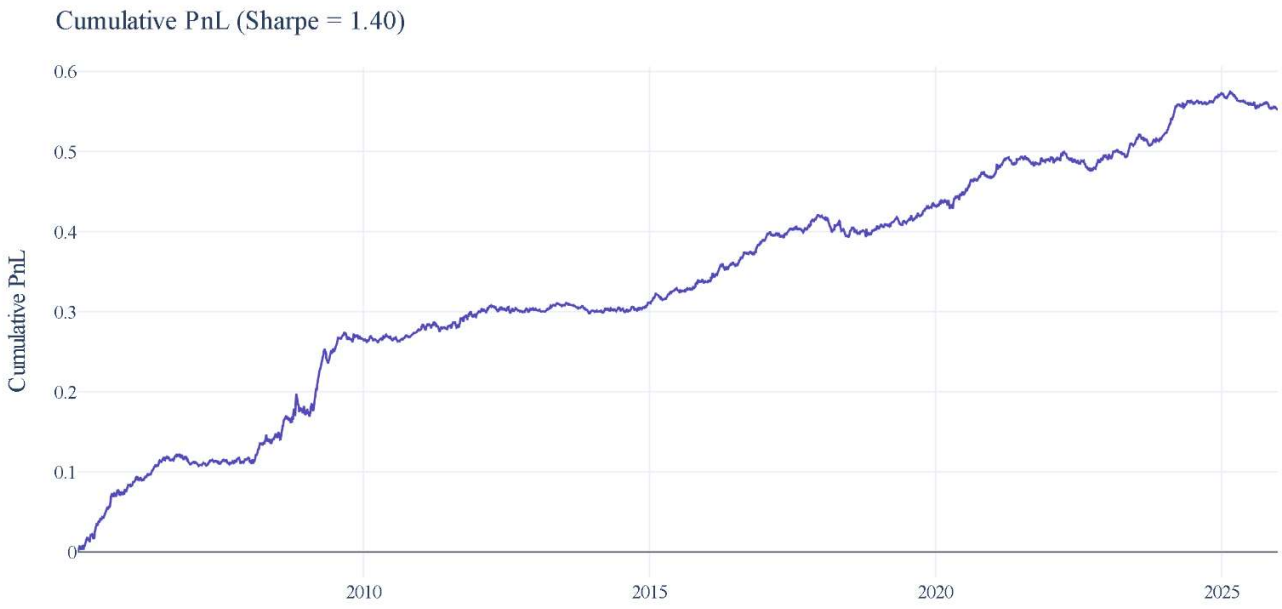
Crucially, this procedure relies solely on backward-looking information: the IC is computed over the training period preceding the OOS window, ensuring that no look-ahead bias is introduced. The IC value thereby functions as a sufficient statistic for both the direction and the strength of the signal.

4. Results

4.1 Main results

Our primary configuration employs 21 walk-forward folds, 500 formulas per fold (aggregated via IC weighting), and 30,000 training timesteps per fold, with evaluation conducted on both raw and residualized returns. The out-of-sample period spans January 2005 to December 2025, covering approximately 5,292 trading days across 544 securities.

Graph 1 - Cumulative PnL for the composite alpha – Backtest on 20 years



The Sharpe ratio of 1.40 over a 20-year pseudo out-of-sample window confirms that the composite alpha signal exhibits strong and persistent predictive power across multiple market regimes. The cumulative PnL trends upward consistently throughout the evaluation period, with no prolonged drawdown erasing prior gains.

4.2 Does RL training add value?

A natural question is whether RL training contributes beyond what random formula generation would achieve. To address this, we compare two formula-generation strategies. The first is the trained RL agent, our proposed method, in which the agent is trained for 30,000 steps and subsequently generates formulas using the learned policy. The second is an untrained RL agent, which generates formulas immediately under a random policy that nonetheless respects the grammar constraints. The only difference is whether the policy has been shaped by RL training.

The most striking difference is in collection efficiency. Without training, only 5.7 to 6.7% of generated formulas pass all numerical validity checks (variance thresholds, non-degenerate output, no overflow). With training, this rate rises to approximately 62%. The grammar guarantees syntactic validity, i.e. every generated expression is a well-formed mathematical formula, but it does not prevent numerical degeneracy. A random policy frequently produces combinations such as Pow(close, volume), which overflows to infinity, or Log(Sign(x)), which is undefined for zero-valued inputs. The trained agent has observed thousands of formula evaluations during training and has learned to avoid these numerically degenerate operator-operand combinations, thus reducing wasted computation during collection.

In terms of aggregate performance, the trained agent achieves a Sharpe ratio of 1.40 compared to 1.27 for the untrained policy**.** The margin is modest because the untrained policy compensates with greater structural diversity: sampling production rules nearly uniformly produces formulas spanning a wider variety of operator combinations, and under IC-weighted aggregation this partially offsets lower per-formula quality.

These results suggest that the primary contribution of our framework lies in the grammar-guided architecture and IC-weighted aggregation rather than in the RL training alone. RL training provides a useful but incremental improvement through computational efficiency (faster collection) and a slight Sharpe gain.

4.3 Hyperparameter sensitivity

To verify that our results are not an artifact of a specific hyperparameter configuration, we vary the key parameters and report the effect on out-of-sample Sharpe. The baseline configuration is as follows: 500 formulas per fold, 30,000 training steps, maximum formula depth set to 8, entropy coefficient of 0.1, and random seed set to 24. For each of these sensitivity tests, we only modify one of the parameters.

4.3.1 Number of formulas per fold

Table 3 - Number of formulas per fold variation

Number of formulas per fold	Sharpe ratio
100	1.04
500	1.43
1,000	1.64

In this first test, we vary the number of formulas collected during each fold (testing window). Increasing the size of the collected alpha pool consistently improves performance, particularly when formulas are weighted by their IC. However, collecting more formulas is computationally expensive, so we decided to choose 500 formulas per fold.

4.3.2 Entropy coefficient

Here, we run tests for the variation of the entropy coefficient; all other things being equal.

Table 4 – Entropy coefficient variation

Entropy coefficient	Sharpe ratio
0.05	1.39
0.10	1.43
0.20	1.51
0.30	1.43

The entropy coefficient controls the degree of exploration in the RL agent's policy: higher values encourage the agent to distribute probability more uniformly across valid production rules, producing diverse formulas, while lower values allow the policy to concentrate on a narrower set of learned templates. Here, performance is stable across the tested range. We adopt the entropy coefficient of 0.1, even though 0.2 yields slightly better results: we prefer this lower entropy setting because it allows the policy to converge

faster during training, concentrating on promising operator combinations while still maintaining enough search space. We also note that we vary only one parameter while holding all other parameters fixed; in practice, interactions between hyperparameters can shift the optimal value. By choosing this moderate setting rather than the extremes of the tested range, we aim for a robust configuration that performs well across different combinations.

4.3.3 Random seed

To ensure that our results are not the product of a fortunate random seed, we re-ran the same pipeline under identical hyperparameters while varying only the random seed. The outcomes remained stable across runs, which provides reassuring evidence of the robustness of our framework.

Table 5 – Random seed variation

Random seed	Sharpe ratio
24	1.43
364	1.29
2047	1.47
2048	1.23
12345	1.34

4.4 Factor exposure analysis

To rigorously assess whether our alpha signal captures genuine new information rather than repackaging known factors, we perform a multivariate regression of the daily alpha PnL (non-cumulative) on the daily PnL of the market and each of the common residualizing factors:

$$PnL\_alpha(t) = \alpha + \beta\_mkt \cdot PnL\_mkt(t) + \beta\_mom \cdot PnL\_mom(t) + \ldots + \beta\_size \cdot PnL\_size(t) + \varepsilon(t)$$

The key test is whether the intercept α (Jensen's alpha) is statistically significantly different from zero. A positive and significant intercept would indicate that our alpha generates returns that cannot be explained by the market or any of the residualizers; in other words, genuinely new predictive information.

Table 6 presents the results of the multivariate regression. Jensen's alpha (the intercept) is positive and highly statistically significant, with a t-statistic of 5.78 and a p-value below 0.0001. This means that after controlling for exposure to all residualizers and the market, our composite alpha generates a positive daily excess return that cannot be attributed to any known risk premium in our factor set.

Three factor loadings are statistically significant at the 5% level: momentum ($\beta = 0.021$, $t = 3.25$), low volatility ($\beta = -0.023$, $t = -3.27$), and the adjustment factor ($\beta = -0.048$, $t = -3.94$). Long-term reversal is marginally significant at the 10% level ($\beta = 0.018$, $t = 1.79$, $p = 0.07$). All remaining factors are statistically insignificant. Crucially, the intercept remains highly significant ($t = 5.78$), confirming that the portion of our alpha's returns explained by these factor exposures is small.

Table 6 — Factor regression of daily alpha PnL

Panel A: OLS regression with Newey-West (HAC) standard errors (10 lags)

	Coeff.	**Std. Err.**	**t-stat**	**p-value**	**Signif.**
Intercept (Jensen's α)	0.0001	0.0000	5.785	<0.0001	***
Momentum	0.0206	0.0063	3.252	0.0011	***
Reversal (21d)	−0.0088	0.0077	−1.155	0.2479	
Reversal (5d)	−0.0007	0.0074	−0.094	0.9250	
Reversal (2d)	−0.0001	0.0082	−0.007	0.9942	
Reversal (1d)	−0.0045	0.0084	−0.539	0.5902	
Long-term reversal	0.0176	0.0098	1.794	0.0727	*
Low volatility	−0.0227	0.0069	−3.273	0.0011	***
Size	−0.0136	0.0133	−1.018	0.3085	
Adjustment factor	−0.0482	0.0122	−3.939	0.0001	***
Market	0.0021	0.0022	0.954	0.3402	

Panel B: Model diagnostics

	Value
R^2	0.038

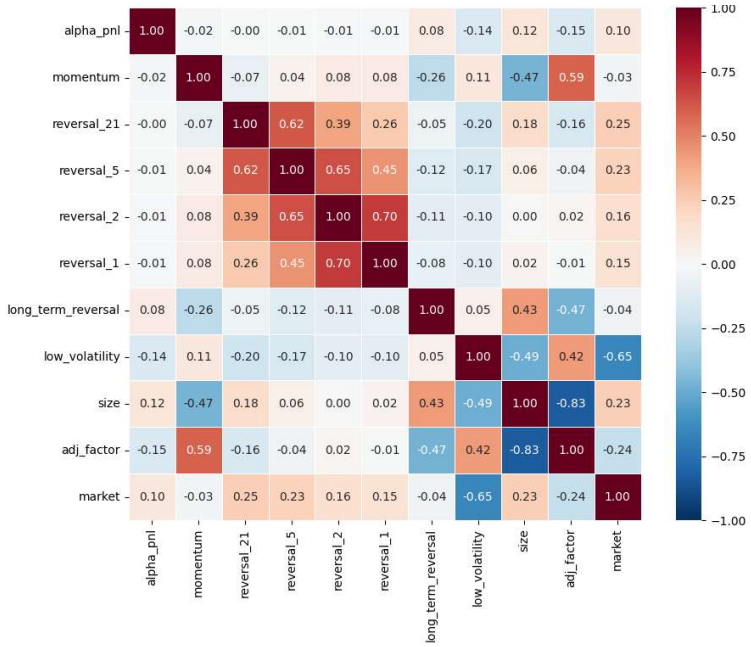
Adjusted R^2	0.037
F-statistic	9.236
Prob (F-statistic)	2.55×10^{-15}
Observations	5,260
Durbin-Watson	1.954

**** $p < 0.01$, ** $p < 0.05$, * $p < 0.10$. Standard errors are heteroscedasticity and autocorrelation robust (Newey-West, 10 lags).*

The correlation matrix further confirms these findings. The first row shows that our alpha PnL exhibits near-zero correlations with all factor PnLs, ranging from −0.15 (adjustment factor) to 0.12 (size). The highest absolute correlation is 0.15, which would not indicate meaningful factor overlap. By contrast, the inter-factor correlations are substantially higher: for example, the short-term reversal factors (reversal_1, reversal_2, reversal_5) are correlated at 0.45 to 0.70 with each other, and low volatility and the market exhibit a correlation of −0.65. Our alpha stands apart from this factor structure, occupying a distinct region of the return space.

Taken together, these results provide evidence that our grammar-guided RL framework discovers predictive patterns in the cross-section of equity returns that do not seem to be attributable to momentum, reversals, size, volatility, or market exposure.

Figure 3: Correlation matrix of Alpha PnL vs Factor PnLs

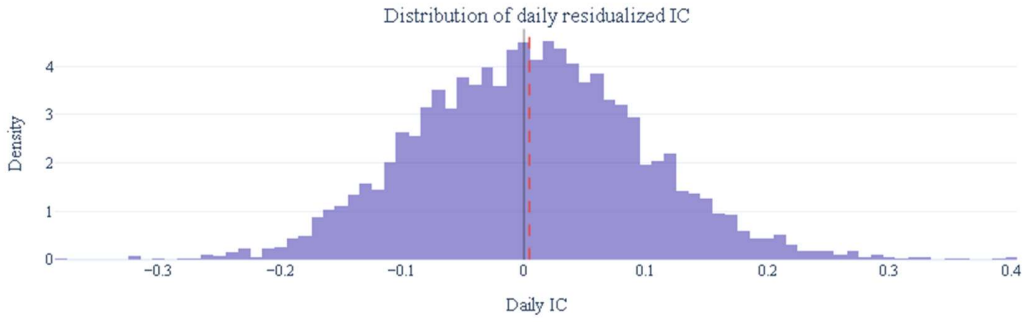


4.5 IC analysis

The information coefficient provides a complementary perspective on the quality and temporal stability of our composite alpha signal. Over the full out-of-sample period, the mean daily IC is 0.0044, which falls within an exploitable range.

We also plot the distribution of daily IC, which is centered above zero and slightly right-skewed. The bulk of the distribution lies between −0.2 and +0.2, with moderate tails on both sides. The small but positive mean is not driven by a few extreme days but rather by a slight and consistent shift of the entire distribution above zero.

Graph 2: Distribution of daily residualized information coefficients across the full out-of-sample period

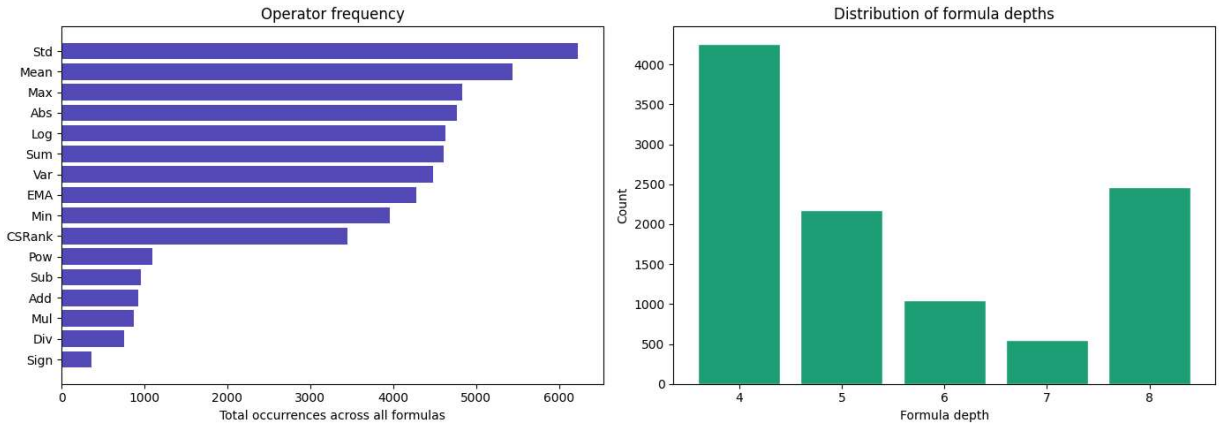


4.6 Formula characteristics

Examining the formulas produced by the system across all folds, we observe distinctive patterns in both operator usage and structural complexity. We report statistics on the 10,500 collected formulas across our 21 folds.

The operator frequency distribution reveals a strong preference for time-series aggregation operators: Std, Mean, Max, Sum, Var, and EMA each appear 4,000 to 6,000 times across all formulas, while binary operators combining two sub-expressions (Add, Sub, Mul, Div) are comparatively rare at 800 to 1,000 occurrences each. The agent has learned to favour deep transformations of individual features over combinations of multiple features.

Graph 3: Operator frequency (left) and distribution of formula depths (right) across the 10,500 collected formulas over the pseudo OOS period



The formula depth distribution is bimodal, with peaks at depth 4 and depth 8 (the maximum allowed). The shallow peak corresponds to simple but effective formulas, while the deep peak reflects the agent's tendency to fill the tree to its maximum permitted depth. The scarcity of intermediate depths suggests that these structures are less stable or less rewarded during training. An example of the formulas created by the agent can be found in the appendix (Table A2).

5. Conclusion and Discussion

This work has proposed a grammar-guided reinforcement learning framework for the automated discovery of formulaic alphas in equity markets. Its central methodological contribution lies in replacing flat token-sequence generation with a context-free grammar over which a MaskablePPO agent constructs formulas top-down. By restricting the action space at each step to those productions that are syntactically applicable to the current non-terminal, this design guarantees the validity of every generated expression, sharpens reward assignment, and concentrates exploration on well-formed candidates rather than on the vast surrounding space of noise.

Evaluated on a thirty-year panel of 544 NASDAQ-listed equities under a twenty-one-fold walk-forward protocol, the framework delivers a pseudo out-of-sample Sharpe ratio of 1.40 after residualization against the common risk factors we control for. The result is obtained without any look-ahead in either signal construction or formula weighting, and robustness analyses across random seeds, entropy coefficients, formula and tree-depth limits indicate that this performance is a stable property of the framework rather than the artifact of a single fortunate configuration.

Our framework occupies a distinct position relative to prior work on alpha discovery. Compared to the academic factor literature (Fama and French, 1992; Jegadeesh and Titman, 1993), which relies on manual factor construction and can explore only a small fraction of the signal space, our approach automates discovery over a combinatorially vast space of symbolic expressions. Relative to genetic programming methods (Zhang et al., 2020; Cui et al., 2021), it avoids the sparsity, premature convergence, and code-bloat problems documented by Ren et al. (2024), since the context-free grammar bounds formula complexity and guarantees validity by construction. Compared to flat token-sequence RL approaches such as AlphaGen (Yu et al., 2023) and QuantFactor REINFORCE (Zhao et al., 2025), our grammar-guided construction eliminates the wasted exploration on syntactically invalid sequences that hampers sample efficiency in those settings. Finally, while the principle of grammar-constrained generation originates in the VAE literature (Kusner et al., 2017; Dai et al., 2018), we embed it within a reinforcement learning framework via action masking (MaskablePPO), thereby combining the structural guarantees of grammar-guided generation with the learning efficiency of modern policy-gradient methods, all under the rigorous walk-forward and factor-residualized evaluation standards of quantitative finance.

Any automated alpha discovery system faces the criticism of data mining: with enough formulas tested, some will appear predictive by chance. We address this concern through several mechanisms. Walk-forward evaluation ensures that every reported metric is out-of-sample, with the RL agent never seeing OOS data during training or formula selection. Residualization against well-known factors reduces the risk that our alpha is simply a repackaged factor premium. IC-weighted aggregation uses in-sample IC as a quality filter, naturally down-weighting noise formulas. A pseudo-twenty-year OOS period (2005–2025) spanning multiple market regimes provides substantial statistical power, and the signal persists across different hyperparameter settings, reducing the risk of configuration-specific overfitting. We nevertheless acknowledge that the risk of data mining cannot be entirely eliminated: the choice of operator set, terminal features, and grammar structure all represent implicit degrees of freedom in the research design.

Several limitations of the present study should be acknowledged. The first concerns transaction costs: our backtest assumes a frictionless market in which trades incur no costs. A realistic implementation would face bid-ask spreads (approximately 5–20 basis points for US large-cap equities), market-impact costs (which scale with portfolio size), and commissions, and the daily rebalancing of a multiple stock long-short portfolio could generate substantial turnover. A second limitation concerns computational cost: a complete twenty-one-fold walk-forward run with 500 formulas per fold and 30,000 training steps requires multiple hours on a workstation with parallelisation; while manageable for research purposes, this would require further optimisation for production deployment with more frequent re-estimation. A third limitation concerns the scope of the universe: our results pertain specifically to NASDAQ-listed US equities, and generalisation to other markets, asset classes, or investment horizons remains untested. A fourth limitation concerns interpretability: although individual formulas are syntactically readable, the financial interpretation of deeply nested expressions such as Pow(Abs(Mean(CSRank(Std(close, 20)), 60)), 5) remains challenging, and the composite alpha is effectively a black box despite the readability of its individual components. Finally, our IC-weighting approach treats each formula independently and does not account for correlations between formulas: if the agent discovers many formulas that are syntactically different but semantically similar, IC-weighting will allocate substantial combined weight to what is effectively a single signal, reducing the diversification benefit of aggregation and concentrating risk.

Several promising extensions follow directly from these limitations. A first direction is the adoption of an IC (Information Coefficient)/IR (Information Ratio)-based reward, in which the agent is rewarded for the

consistency of IC across time rather than for the mean IC alone. The Information Ratio, defined as the mean IC divided by the standard deviation of IC over the evaluation period, captures not only the strength of an alpha's predictive signal but also its stability, penalizing formulas whose performance is erratic even when their average IC is high; the reward function design itself remains a rich area for exploration. A second direction is to implement a decorrelation mechanism before aggregation, to mitigate the redundancy risk inherent in our current IC-weighted framework: forward selection with a pairwise correlation threshold, principal component analysis to extract orthogonal factors, LASSO-penalized regression to select a sparse subset of formulas, or hierarchical clustering to retain one representative per cluster are all candidate approaches. A third direction is the extension of the feature set, incorporating fundamental data (earnings, analyst estimates), alternative data (sentiment scores, web traffic), or options-derived features (implied volatility, put-call ratios) to broaden the informational basis on which formulas are constructed. A final direction is production deployment, which would entail realistic transaction-cost models, portfolio-construction constraints (sector neutrality, turnover limits), and live trading to assess real-world market impact, as well as evaluation on entirely different asset classes (international equities, futures, or cryptocurrencies) to assess the generalizability of the discovered signals.

Taken together, these results suggest that grammar-guided reinforcement learning, combined with rigorous walk-forward evaluation and disciplined aggregation, offers a credible and reproducible path for the systematic discovery of alpha signals beyond those already documented in the academic factor literature.

Technical Appendix

Data Fields and Bloomberg Extraction

For each stock, we extracted daily observations from Bloomberg terminal using a standardized BQL (Bloomberg Query Language) query. The query integrates multiple fields required for price-based and volume-based factor construction, as well as corporate action adjustments. Our extraction query took the following form:

```
BQL Query: dropna(px_last) as #closeprice dropna(px_last(ca_adj=full)) as #adjustedclose dropna(px_open) as #open dropna(px_low) as #low dropna(px_high) as #high dropna(px_volume) as #volume dropna(cur_mkt_cap) as #mktcap dropna(vwap) as #vwap
```

Table A1 - Training & Out-of-sample Periods

Fold	Training Period	OOS Period
1	1995-01-01 to 2005-01-01	2005-01-01 to 2006-01-01
2	1996-01-01 to 2006-01-01	2006-01-01 to 2007-01-01
3	1997-01-01 to 2007-01-01	2007-01-01 to 2008-01-01
4	1998-01-01 to 2008-01-01	2008-01-01 to 2009-01-01
5	1999-01-01 to 2009-01-01	2009-01-01 to 2010-01-01
6	2000-01-01 to 2010-01-01	2010-01-01 to 2011-01-01
7	2001-01-01 to 2011-01-01	2011-01-01 to 2012-01-01
8	2002-01-01 to 2012-01-01	2012-01-01 to 2013-01-01
9	2003-01-01 to 2013-01-01	2013-01-01 to 2014-01-01
10	2004-01-01 to 2014-01-01	2014-01-01 to 2015-01-01
11	2005-01-01 to 2015-01-01	2015-01-01 to 2016-01-01

12	2006-01-01 to 2016-01-01	2016-01-01 to 2017-01-01
13	2007-01-01 to 2017-01-01	2017-01-01 to 2018-01-01
14	2008-01-01 to 2018-01-01	2018-01-01 to 2019-01-01
15	2009-01-01 to 2019-01-01	2019-01-01 to 2020-01-01
16	2010-01-01 to 2020-01-01	2020-01-01 to 2021-01-01
17	2011-01-01 to 2021-01-01	2021-01-01 to 2022-01-01
18	2012-01-01 to 2022-01-01	2022-01-01 to 2023-01-01
19	2013-01-01 to 2023-01-01	2023-01-01 to 2024-01-01
20	2014-01-01 to 2024-01-01	2024-01-01 to 2025-01-01
21	2015-01-01 to 2025-01-01	2025-01-01 to 2025-12-25

Table A2 - Representative formulas from each fold

Fold	Tokens	Formula and interpretation
1	30 (15 ops)	Add(Max(Var(Std(EMA(Std(volume,5),10),20),20),60), Sum(Std(Min(Add(Sub(Add(volume,low),Abs(low)),EMA(Min(low,20),10)),252),10),20))
2	23 (11 ops)	Add(Sum(EMA(volume,5),10),Var(Var(EMA(Add(EMA(Sum(volume,252),252), Mean(Div(vwap,high),60)),20),20),252))
3	30 (16 ops)	Mul(Abs(Abs(Mean(Max(volume,10),5))),Max(Add(Min(Min(Abs(Max(low,252)),5),5), Min(Mean(Sub(Mul(mcap,high),Mean(low,20)),252),5)),20))
4	35 (18 ops)	Add(Div(Max(volume,20),Abs(Var(Std(Div(Pow(mcap,5),Log(high)),60),10))), Max(Min(Add(Add(Add(Var(low,20),low),EMA(Mean(mcap,252),252)),Std(open,20)),252),20))
5	35 (18 ops)	Mul(Mean(Add(high,EMA(EMA(Max(Sub(vwap,mcap),20),60),60)),60), Min(Sum(Add(Div(Mul(EMA(mcap,20),Add(low,low)),Div(Sign(high),Max(high,5))),Abs(high)),5),60))
6	21 (12 ops)	Sub(Sum(Max(CSRank(Std(mcap,5)),20),5), Sum(Abs(Log(Sign(Mul(Div(high,high),Min(open,10))))),5))

7	20 (11 ops)	Mean(EMA(Log(Div(Mul(Mean(volume,5),Log(high)),Std(Mul(CSRank(vwap),Pow(volume,20)),5))),252),5)
8	38 (20 ops)	Sub(Max(Abs(Sum(Std(Min(Sub(low,volume),252),252),5)),252), Div(EMA(Std(Div(Sign(Mean(vwap,20)),Var(EMA(low,5),5)),5),252), EMA(CSRank(Pow(Div(Max(volume,252),open),252)),252)))
9	26 (14 ops)	Sub(Abs(EMA(mcap,10)),Var(Add(Std(CSRank(Max(Sub(vwap,mcap),10)),10), Mean(Log(Mul(Sub(vwap,high),Mean(vwap,60))),10)),5))
10	25 (13 ops)	Sum(Add(Mean(Std(Abs(Log(vwap)),10),5), Mean(Std(Div(Sum(Mean(high,5),252),Max(Var(open,10),5)),20),10)),10)

References

- Avellaneda, M. and Lee, J.-H. (2010) 'Statistical arbitrage in the U.S. equities market', *Quantitative Finance*, 10(7), pp. 761–782.
- Cui, C., Wang, W., Zhang, M., Chen, G., Luo, Z. and Ooi, B.C. (2021) 'AlphaEvolve: a learning framework to discover novel alphas in quantitative investment', in *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*. New York: ACM, pp. 2208–2216.
- Dai, H., Tian, Y., Dai, B., Skiena, S. and Song, L. (2018) 'Syntax-directed variational autoencoder for structured data', in *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, Vancouver, Canada, 30 April – 3 May.
- Fama, E.F. and French, K.R. (1992) 'The cross-section of expected stock returns', *Journal of Finance*, 47(2), pp. 427–465.
- Grinold, R.C. and Kahn, R.N. (2000) *Active portfolio management: a quantitative approach for producing superior returns and controlling risk*. 2nd edn. New York: McGraw-Hill.
- Harvey, C.R., Liu, Y. and Zhu, H. (2016) '… and the cross-section of expected returns', *The Review of Financial Studies*, 29(1), pp. 5–68.
- Jegadeesh, N. and Titman, S. (1993) 'Returns to buying winners and selling losers: implications for stock market efficiency', *Journal of Finance*, 48(1), pp. 65–91.
- Jensen, M.C. (1968) 'The performance of mutual funds in the period 1945–1964', *Journal of Finance*, 23(2), pp. 389–416.
- Kakushadze, Z. (2014) 'Factor models for alpha streams', *The Journal of Investment Strategies*, 4(1), pp. 83–109.
- Kakushadze, Z. (2016) '101 formulaic alphas', *Wilmott*, 2016(84), pp. 72–81.
- Kakushadze, Z. and Tulchinsky, I. (2015) 'Performance v. turnover: a story by 4,000 alphas', *Journal of Investment Strategies*, 5(2), pp. 75–89.
- Kusner, M.J., Paige, B. and Hernández-Lobato, J.M. (2017) 'Grammar variational autoencoder', in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, Sydney, Australia, 6–11 August. PMLR, pp. 1945–1954.
- Markowitz, H. (1952) 'Portfolio selection', *Journal of Finance*, 7(1), pp. 77–91
- Pastor, L. and Stambaugh, R.F. (2003) 'Liquidity risk and expected stock returns', *Journal of Political Economy*, 111(3), pp. 642–685.

- Ren, W., Qin, Y. and Li, Y. (2024) *Alpha mining and enhancing via warm start genetic programming for quantitative investment*. arXiv:2412.00896. Available at: https://arxiv.org/abs/2412.00896.
- Trivedi, D., Zhang, J., Sun, S.-H. and Lim, J.J. (2021) 'Learning to synthesize programs as interpretable and generalizable policies', in *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*. Available at: https://arxiv.org/abs/2108.13643.
- Wang, S., Yuan, H., Zhou, L., Ni, L.M., Shum, H.-Y. and Guo, J. (2025) *QuantBench: benchmarking AI methods for quantitative investment*. arXiv:2504.18600. Available at: https://arxiv.org/abs/2504.18600.
- Xu, F., Yin, Y., Zhang, X., Liu, T., Jiang, S. and Zhang, Z. (2024) *Alpha²: discovering logical formulaic alphas using deep reinforcement learning*. arXiv:2406.16505. Available at: https://arxiv.org/abs/2406.16505.
- Yu, S., Xue, H., Ao, X., Pan, F., He, J., Tu, D. and He, Q. (2023) 'Generating synergistic formulaic alpha collections via reinforcement learning', in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23)*. New York: ACM, pp. 5476–5486.
- Zhang, T., Li, Y., Jin, Y. and Li, J. (2020) 'AutoAlpha: an efficient hierarchical evolutionary algorithm for mining alpha factors in quantitative investment', arXiv:2002.08245. Available at: https://arxiv.org/abs/2002.08245.
- Zhao, J., Zhang, C., Qin, M. and Yang, P. (2025) *QuantFactor REINFORCE: mining steady formulaic alpha factors with variance-bounded REINFORCE*. arXiv:2409.05144. Available at: https://arxiv.org/abs/2409.05144.
