

ProFiT: Program Search for Financial Trading

Matthew Siper¹, Ahmed Khalifa¹, Lisa Soros¹, Muhammad Umair Nasir¹, Jay Azhang¹, and Julian Togelius¹

¹ Nof1, New York, NY, USA
² m@thenof1.com

Abstract. We present a framework called Program Search for Financial Trading (ProFiT), a large-language-model-driven evolutionary search for automated discovery and continual improvement of algorithmic trading strategies in financial markets. These markets are inherently non-stationary and thus resist static modeling or prediction. ProFiT integrates code-level mutation, self-analysis, and walk-forward validation within a closed feedback loop, enabling trading strategies to autonomously evolve in response to changing market conditions. ProFiT consistently outperforms both random and Buy-and-Hold strategies, which are used as baselines, across seven liquid futures assets. Specifically, it surpasses Buy-and-Hold in over 77% of all evolved strategy–asset combinations, outperforms random in 100% of cases, and furthermore achieves improvements over the initial seed strategies in more than 94% of runs. Collectively, these results demonstrate that the ProFiT framework yields robust, statistically significant, and risk-adjusted gains across diverse assets and market regimes, establishing a practical pathway toward open-ended, self-improving algorithmic trading systems.

Keywords: Evolutionary Algorithms · Tree Search · Finance · LLM · Code Generation.

1 Introduction

Designing profitable trading strategies is an enduring challenge in quantitative finance. Despite decades of progress in machine learning and reinforcement learning (RL), most trading systems still rely on human intuition, manual feature engineering, and static optimization. These methods often fail to adapt to evolving market regimes, wherein initially sustainable patterns eventually decay. Moreover, financial risk exposures drift, and structural market changes render fixed strategies obsolete. As a result, the process of discovering and improving trading algorithms typically requires constant human intervention, and the broader problem of automated strategy discovery remains largely unsolved.

In this paper, we introduce ProFiT, an approach to trading that leverages LLM-based rewriting of trading strategies within an evolutionary framework. It can be seen as a form of genetic programming [10], wherein the search space consists of trading algorithms (written in Python) encoded as trees. To discover

new strategies, a large language model (LLM) analyzes existing code and its performance on real-world market data, then proposes new program variants.

Our method departs from systems based on static neural architectures by directly evolving executable trading strategy code. Rather than solely adjusting model weights or fine-tuning parameters, the ProFiT framework iteratively mutates and validates the source-level logic of trading strategies, enabling structural adaptation in response to changing market conditions. This represents a shift from parameter learning to strategy evolution, extending prior work on genetic programming approaches within finance [3]. Rather than retaining only the single best-performing program, ProFiT stores *all* strategies that exceed a predefined minimum acceptable score within a hierarchical archive. This method is preferable to pure greedy search for the purpose of avoiding premature convergence to local optima.

In the rest of this paper, we formalize the ProFiT framework and demonstrate its application to the automated discovery of trading strategies. Section 2 reviews prior research on automated trading systems, evolutionary computation, and large language models for code generation. Section 3 details the architecture of the ProFiT framework, including its initialization, mutation, and evaluation components. Section 4 describes the experimental design, datasets, and baselines used for empirical validation. Section 5 presents our quantitative results across seven futures markets and five seed strategy archetypes. Section 6 discusses broader implications and future research directions. Finally, Section 7 concludes with reflections on how self-improving algorithmic systems such as ProFiT may advance the field of quantitative trading and beyond.

2 Related Work

2.1 Automated and Machine-Learning-Based Trading Systems

Today’s automated trading strategy landscape largely relies on human-designed rules and data features, sometimes combined with machine learning for optimization. Early reinforcement learning frameworks for financial prediction [18] established the feasibility of adaptive decision-making, which was later extended via deep architectures for trading signal generation [7]. Despite improved representational capacity, these approaches optimize within a fixed architecture tuning parameters rather than modifying the underlying algorithmic logic or inputs. Such models often overfit historical data and degrade under regime shifts inherent in non-stationary financial markets.

2.2 Evolutionary Tree Search and AI for Scientific Discovery

Recent work has linked evolutionary algorithms and tree search as complementary methods for open-ended exploration and automated reasoning. Evolutionary Monte Carlo Tree Search [20,2] combines evolutionary operators such as mutation and recombination within the branching structure of MCTS to balance exploitation and exploration. These approaches have been applied mainly

in reinforcement learning and game domains, where the tree represents sequences of actions or policies rather than executable programs.

Google DeepMind’s AI framework for scientific discovery [25] similarly employs a tree-search process in which nodes represent hypotheses or experiments and expansions correspond to new experimental proposals guided by predictive models. While this system demonstrates that structured search over symbolic programs can accelerate scientific progress, both EvoMCTS and AI-discovery frameworks operate over abstract symbolic representations and fixed objectives.

ProFiT instead applies evolutionary tree search directly to executable trading code. Each node in its population is a runnable strategy, and edges represent LLM-generated code mutations evaluated through empirical backtesting. By coupling symbolic reasoning, code synthesis, and empirical validation, ProFiT evolves both the structure and logic of algorithms within a grounded, real-world domain, extending tree-search-based discovery into self-improving, executable systems.

Our framework is inspired by the Darwin Gödel Machine [29], itself inspired by Gödel Machines [23], which are self-referential systems. In their original conception, this refers to source code that modifies itself only after identifying a provable improvement. Such a requirement has limited the practical applications of Gödel Machines, as formal proofs of improvement are generally intractable in complex environments.

2.3 Evolutionary Computation and Genetic Programming in Finance

Evolutionary approaches, though capable of searching over strategy populations, often lack higher-level reasoning mechanisms to guide refinement beyond blind search. Genetic Programming (GP) [9,21,1] demonstrated that symbolic programs could evolve via selection and mutation, while neuroevolution [12,28] methods such as NEAT [24] evolve neural weights and topologies. More recently, Quality-Diversity (QD) [22] algorithms such as MAP-Elites [19] and novelty search [13,14] introduced mechanisms that maintain behavioral diversity while promoting performance.

Recent advances in GP for trading, specifically, have shifted from fixed rule induction to hybrid and data-driven approaches. Methods such as grammatical evolution [6] and multi-objective genetic programming [15] have been used to jointly optimize return, drawdown, and interpretability. More recent systems integrate deep learning components within GP trees [17], allowing symbolic trading rules to evolve alongside neural feature extractors. Despite these advances, most approaches remain confined to static evolutionary cycles that operate offline and require manual retraining as market conditions change. ProFiT differs by embedding genetic-programming-style evolution within an online, self-improving loop guided by large language models and empirical validation, enabling continuous adaptation rather than periodic re-optimization.

Most of the mentioned methods operate primarily over syntactic or parametric spaces. ProFiT instead operates in the space of Python programs, and em-

ploys semantically informed mutation guided by LLM reasoning, where changes are driven by textual analysis and empirical feedback rather than random perturbations. This combines the explorative benefits of evolutionary algorithms with the power of language-based reasoning.

2.4 Large Language Models for Code Generation and Reasoning

Large language models trained on code have expanded the frontier of automated reasoning and program synthesis. Codex [4] showed that language models can generate executable Python from natural language, while subsequent systems such as AlphaCode and Code Llama advanced autonomous problem solving. Yet, these systems remain largely static: they generate code once but do not engage in iterative empirical improvement.

ProFiT transforms this paradigm by embedding LLMs within an evolutionary feedback loop. Our ProFiT framework consists of LLM-generated mutations that are not only syntactically evaluated but also empirically tested through an out-of-sample (OOS) walk-forward backtesting scheme. The coupling of reasoning with empirical feedback bridges the gap between symbolic code generation and performance-grounded evolution, echoing the iterative dynamics observed in open-ended LLM agents such as Voyager [26].

2.5 Open-Ended Learning and Self-Evolving Systems

DGM belongs to the lineage of self-improving systems that aim for continual strategy innovation, sometimes referred to as open-ended algorithms. Works such as POET [27] and Go-Explore [8] demonstrate that some degree of open-ended co-evolution can yield creative solutions in complex domains. The AI-GA paradigm [5] further envisions meta-evolutionary systems capable of inventing new learning algorithms. Recent efforts [11,29] revisit these ideas through modern LLM-driven architectures, exploring open-ended evolution of agent policies.

Within this broader context, our ProFiT framework operationalizes the open-ended approach in a quantitative financial setting, where empirical market feedback serves as the evolutionary fitness function. By combining evolutionary search, Gödelian self-modification, and LLM reasoning, our ProFiT framework represents a pragmatic bridge between open-ended learning systems and algorithmic trading.

3 ProFiT for Algorithmic Trading

ProFiT is a meta-learning framework designed to explore the space of algorithmic trading strategies through an open-ended evolutionary loop guided by LLMs. Similar to biological evolution, each trading strategy is represented as an independent program that undergoes mutation, evaluation, and selection (Figure 2).

The system maintains an population of strategies that surpass a minimum performance criterion. The system will select a strategy from the population

<p>You are an expert quantitative strategist.</p> <p>Analyze the following trading strategy code and its recent backtest results.</p> <p>Your task:</p> <ol style="list-style-type: none"> Identify the most significant weaknesses, inefficiencies, or sources of poor performance in the strategy. Propose no more than 2-3 concrete, high-impact improvements that could realistically improve out-of-sample returns and robustness. <p>Guidelines:</p> <ul style="list-style-type: none"> - Be specific and actionable (for example: "replace SMA with EMA to improve responsiveness", "add a volatility filter to reduce false signals"). - Focus on core logic changes, not cosmetic or minor tweaks. - Avoid generic advice (for example: "tune parameters", "improve risk management") unless you specify exactly how. - Keep the response concise, in plain English, suitable for direct implementation in the next rewrite step . - Do not output code here; only short textual suggestions. <p>Backtest summary: <code>{bt_results}</code></p> <p>Strategy code: <code>{strategy_code}</code></p>	<p>You are a quantitative trading developer.</p> <p>Using the original strategy code and the improvement proposals provided below, rewrite the strategy to incorporate the requested enhancements.</p> <p>Requirements:</p> <ul style="list-style-type: none"> - Output only valid Python code. - Do not include any explanations, comments, or extra text. - The rewritten class must: <ul style="list-style-type: none"> - Keep the same class name. - Remain compatible with the backtesting.py library. - Compile and run inside a call such as: <pre>Backtest(data, StrategyClass, cash=10000, commission=0.002, exclusive_orders =True).run(**params)</pre> - Retain the existing structure unless a change is explicitly needed for the improvements. - Include all tunable hyperparameters as class variables with sensible defaults. <p>Improvement proposals: <code>{improvement_proposals}</code></p> <p>Strategy code: <code>{strategy_code}</code></p>
---	---

Fig. 1: LLM prompts used in the evolutionary loop. Left: analysis prompt used to diagnose weaknesses. Right: improvement prompt used to rewrite the strategy.

using a selection method (e.g. roulette wheel, rank, UCB1, uniform selection) to apply mutation to it. The selected strategy is passed to the LLM, which proposes a code modification that is then tested against historical data about the market (backtesting). Based on the performance of the strategy, if it meets the minimum acceptance score (MAS), it is added to the population; otherwise discarded. The process repeats, yielding an open population of candidates while avoiding premature convergence and low-performing individuals. Because the search space of Python code is large, removing the minimum performing strategies (for example, effectively random strategies or fixed strategies that perform worse over big data) helps to focus the search space towards more promising areas. This loop is explained in the following subsections and in the following pseudocode (Algorithm 1).

The LLM is not limited to analyzing the current strategy alone, it can also incorporate the lineage and historical performance of related strategies. Providing this contextual information enables the LLM to reason about how past modifications have influenced performance and to generate more informed improvements.

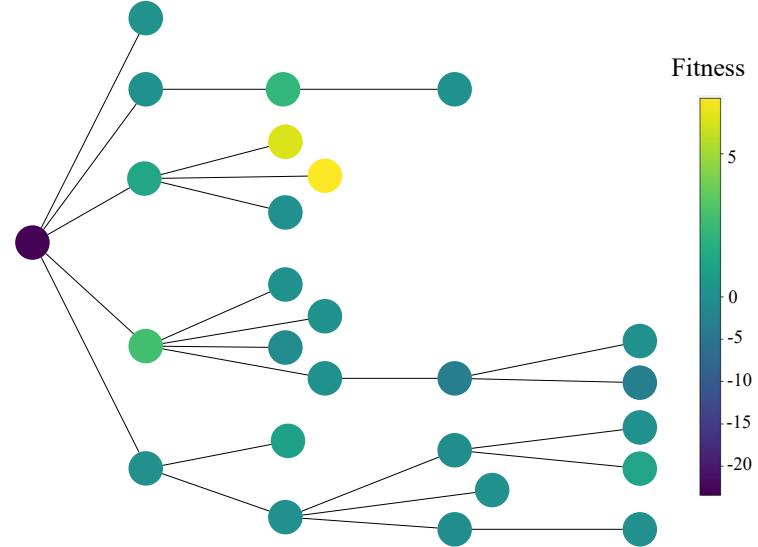


Fig. 2: Population resulting from one sample run of ProFiT. Each node corresponds to a trading strategy variant, with color encoding fitness (annualized return %). The highest fitness achieved in this run is 6.93, and the lowest is -23.41. Edges indicate lineage from parent to child. Only generated strategies exceeding the fixed minimum acceptable score (MAS) are retained, forming an open population of viable candidates.

The overall process comprises three stages: initialization & selection, mutation, and evaluation & addition to the population.

3.1 Initialization & Termination Condition

Each run begins with an initial seed strategy selected from a fixed library (discussed in Section 4). This seed strategy is evaluated, and its performance metrics are recorded and linked to the strategy (see Section 3.4 for details). The framework also initializes an empty population and adds the seed strategy to it. The seed strategy's performance serves as the minimum acceptance score (MAS) for adding new strategies. After initialization, the system proceeds through the following steps for as many generations as desired, or it can operate in an open-ended loop. In the open-ended case, a more effective mechanism for maintaining diversity in the population may be required to prevent the accumulation of highly similar strategies; we leave this consideration for future work.

3.2 Selection: Finding a Parent Strategy

In this step, the system selects a strategy to serve as the parent for generating the next improved strategy. The parent is chosen from the population of eligible

strategies (those with fitness \geq MAS). Since the population only contains strategies that meet the minimum acceptance criteria, various selection methods can be applied, including uniform sampling. An effective selection method should balance exploiting the best-performing strategies with exploring other promising candidates. For example, roulette wheel selection, ϵ -greedy, and UCB-1 are all viable approaches to achieve this balance.

3.3 Mutation: Generating new Strategies

The parent strategy and its recorded performance are provided to the LLM, which is tasked with analyzing the strategy's logic and performance report to generate a list of potential modifications that could improve it. For example, the LLM might suggest adjusting a threshold value or refining risk and position management rules. The suggested modifications, along with the parent strategy, are then fed back to the LLM, instructing it to generate new Python source code that incorporates these changes. If the LLM produces non-functional code, the compiler error and the latest version of the code are returned to it for repair. This repair loop continues for a fixed number of iterations or until the code compiles successfully.

A potential enhancement for future work is to provide the LLM with the historical evolution of the currently selected strategy, allowing it to observe the trajectory of prior modifications. This context could help the LLM avoid redundant changes and behave more like a human designer. However, this direction was not explored in the present work and is left for future research.

3.4 Evaluation & Adding to the Population

Finally, the mutated strategy is evaluated on historical market data using the backtesting engine to generate performance metrics such as annualized return (%), Sharpe ratio, and expectancy (%). We use the annualized return as the fitness measure for evolution; exploring alternative metrics is left for future work.

To be added back to the population, the mutated strategy must achieve a fitness greater than or equal to the minimum acceptable score (MAS). We define the (MAS) as the fitness of the seed strategy. If the mutated strategy fails to meet this threshold, it is discarded to narrow the search space of compilable code. The MAS is initialized from the annualized return of the seed strategy S_0 and remains fixed throughout the evolutionary process. Investigating more adaptive approaches to updating this reference point is left for future work.

4 Experimental Setup

We evaluate our framework on seven assets: E6, ES, KC, SB, A6, NG, and VX. The dataset for each asset is upsampled to a 1-hour timeframe, covering the period from January 2008 to October 2025. Each time series entry contains

Algorithm 1 The ProFiT Evolutionary Loop

```

1: Initialize seed strategy  $S_0$  and compute baseline annualized return  $P_0$ 
2: Set fixed minimum acceptable score (MAS)  $\leftarrow P_0$             $\triangleright$  fitness threshold for
   population inclusion
3: Add the seed strategy with its performance to MAS population ( $A_0 \leftarrow (S_0, P_0)$ )
4: repeat
5:   Selects a strategy and Performance  $((S_t, P_t))$  from the MAS population ( $A_t$ )
6:   Prompt LLMA with  $(S_t, P_t)$  to generate improvement proposal  $\Delta_t$ 
7:   Prompt LLMB with  $(S_t, \Delta_t)$  to synthesize modified strategy  $\hat{S}_t$ 
8:   for up to 10 repair attempts do
9:     Try to compile and backtest  $\hat{S}_t$ 
10:    if failure then
11:      Prompt LLMB with traceback and retry
12:    else
13:      break
14:    Compute the fitness as mean annualized return ( $P_{\hat{S}_t}$ )
15:    if  $P_{\hat{S}_t} \geq$  MAS then
16:      Add the new strategy to the MAS population ( $A_{t+1} \leftarrow (\hat{S}_t, P_{\hat{S}_t})$ )
17:    else
18:      Discard  $\hat{S}_t$ 
19:  until termination criterion met (e.g., 15 iterations)

```

Fold	Train Start	Train End	Val Start	Val End	Test Start	Test End
1	2008-01-02	2010-06-30	2010-06-30	2011-01-11	2011-01-21	2011-07-25
2	2011-07-25	2014-01-20	2014-01-20	2014-08-03	2014-08-13	2015-02-14
3	2015-02-14	2017-08-13	2017-08-13	2018-02-24	2018-03-06	2018-09-07
4	2018-09-07	2021-03-05	2021-03-05	2021-09-16	2021-09-26	2022-03-30
5	2022-03-30	2024-09-25	2024-09-25	2025-04-08	2025-04-18	2025-10-20

Table 1: Walk-forward dataset folds used across all strategy–asset runs.

information for a 1-hour interval, including the asset’s highest and lowest prices, opening and closing values, and the total trading volume during that period.

To ensure robustness, we use cross-validation. Instead of training on the entire dataset, we divide it into five folds (subsets). Each fold consists of a 2.5-year training period, followed by a six-month validation period and a six-month test period. Each intra-fold validation and test pair is separated by a ten-day dormant window to mitigate lookahead effects. The same temporal partitioning is applied across the seven assets. We tested using five seed strategies, chosen because they are commonly used in technical trading. We use the backtest engine ‘Backtesting.py’ benchmark [16] for simulation. These strategies are: Bollinger Mean Reversion, CCI Strategy, EMA Crossover, MACD Strategy, and Williams R Strategy.

Because we use five folds, each seed-strategy–asset pair is evaluated five times to assess stability. The split boundaries are presented in Table 1. The fitness measure used during evolution is the annualized return (%), computed as the average across the five validation folds. Each run starts with an initial trading capital of \$ 10,000, applies a 0.2% transaction cost per trade, and enforces exclusive order execution to prevent overlapping positions. All experiments are conducted using a fixed random seed to ensure reproducibility across folds and runs.

Results are aggregated along two dimensions: (i) the seed strategy and (ii) the liquid futures asset. This enables us to evaluate which strategies perform best across different assets. In future work, it would be interesting to evolve strategies that generalize across multiple assets or to explore cross-parent evolution between markets.

For each (strategy, asset) pair, we compare the best-performing strategy from the population after 15 generations to two baselines that are not part of the seed strategies. The first baseline is a *random strategy*, denoted R_0 , defined as follows: when flat, the agent chooses uniformly among {enter long, enter short, do nothing}; when in a position (acquired a asset), the agent exits with probability 0.5. The second baseline is a passive *Buy-and-Hold* strategy, denoted $B\&H$, which just uses all the capital to buy an asset and hold it over time, hoping that it will make money and the asset will increase. This provides a non-active reference. The best evolved strategy is evaluated based on average annualized return (%), expectancy (%), and Sharpe ratio across the test folds.

For each test fold (i), the improvement (Δ_i) is calculated as the difference between the current strategy (C) and the starting strategy (S) on a given performance metric (P):

$$\Delta_i = P_{C,i} - P_{S,i} \quad \forall 1 \leq i \leq 5. \quad (1)$$

A paired, one-sided Wilcoxon signed-rank test assesses whether $\text{median}(\Delta_i) > 0$, thereby testing whether the improved strategy consistently outperforms its respective baseline across test folds. We report resulting p -values for each experiment, with statistical significance defined as $p < 0.05$.



Fig. 3: First and best strategies along an evolutionary path.

5 Results

We ran 35 experiments ($7 \text{ assets} \times 5 \text{ seed strategies}$). Fitness growth tapers off after approximately 15 iterations, suggesting a local optimum. We attribute this behavior to the use of historical data rather than real-time market conditions, a point we discuss further later in the paper.

Table 2 reports aggregate results across all seven futures assets and five seed strategies. On average, ProFiT yields positive improvements in more than 75% of all experiments, confirming that its evolutionary process produces meaningful, generalizable performance gains rather than overfitting to specific assets or folds. The strongest relative gains occur in annualized return, where ProFiT achieves a mean increase of $+44.21\% \pm 4.31$ over the seed strategy, $+3.41\% \pm 1.27$ over Buy-and-Hold, and a remarkable $+86.47\% \pm 1.21$ over the random baseline. In terms of risk-adjusted performance, ProFiT improves the Sharpe ratio by $+0.57 \pm 0.04$ versus B0, $+2.69 \pm 0.97$ versus B&H, and $+0.62 \pm 0.04$ versus the random baseline. Expectancy shows similar gains, underscoring enhanced trade quality and consistency.

Importantly, ProFiT outperforms the random baseline in 100% of all evaluated cases, demonstrating that its improvements are not the byproduct of stochastic variation but stem from directed, LLM-guided evolutionary search. More than 60% of all comparisons across metrics achieve statistical significance at $p < 0.05$ under a one-sided Wilcoxon signed-rank test, further validating the robustness of the observed results. These findings indicate that ProFiT generalizes effectively across diverse assets and strategy archetypes. ProFiT systematically discovers and preserves modifications that enhance performance and robustness, significantly outperforming both the B&H and random baselines.

5.1 Cross-Assets and Cross-Strategy Analysis

Figure 4 illustrates cross-sectional improvements produced by ProFiT relative to the initial seed strategy baselines (B0) across the seven futures assets. Each cell reports the mean test-fold Δ for the corresponding (strategy, asset) pair, aggregated over five independent runs. The three heatmaps correspond to the three fitness metrics tracked: annualized return (%), expectancy (%), and Sharpe ratio.

Return improvements. As shown in the top panel of Figure 4, ProFiT produces large positive improvements in annualized return across nearly all assets. The strongest gains occur for *williams_r_strategy*, which achieves mean increases between +63% and +90% across NG, SB, and VX. Similarly, *cci_strategy* and *macd_strategy* consistently improve returns by +40–+60% on most assets. Only *ema_crossover* and *macd_strategy* on KC show degraded performance after ProFiT.

Risk-adjusted performance. The middle panel shows that ProFiT also enhances risk-adjusted performance, in the form of the Sharpe ratio. Gains are broadly

	Baseline	Improvement [%]	$p < 0.05$ [%]	Δ Mean (\pm SE)
Expectancy				
Random	100.0	64.3	0.94 \pm 0.14	
B0	97.1	62.9	0.75 \pm 0.13	
B&H	77.1	60.0	2.59 \pm 1.03	
Return				
Random	100.0	83.3	86.47 \pm 1.21	
B0	94.3	94.3	44.21 \pm 4.31	
B&H	77.1	54.3	3.41 \pm 1.27	
Sharpe Ratio				
Random	100.0	59.5	0.62 \pm 0.04	
B0	100.0	71.4	0.57 \pm 0.04	
B&H	74.3	62.9	2.69 \pm 0.97	

Table 2: Mean ProFiT performance relative to baselines across all (asset, strategy) pairs. The evolved strategies consistently outperform a random trading agent (Random), a Buy-and-Hold strategy (B&H), and the initial seed strategies used for each run (B0). Improvement (%) is the fraction of pairs with positive mean performance gains across all runs. $p < 0.05$ (%) is the fraction significant under a one-sided Wilcoxon signed-rank test computed across runs for each pair. Δ Mean (\pm SE) reports the average improvement and its standard error across pairs within each (Performance Metric, Baseline) bucket.

distributed, with typical improvements of +0.5–+1.0 across strategies. Notably, *cci_strategy* achieves the highest Sharpe improvement (+1.07) on E6, while *bollinger_mean_reversion* on ES and *macd_strategy* on SB each reach values of +0.91. These consistent, positive Sharpe improvements indicate structurally better reward–risk profiles.

Expectancy and trade quality. The bottom panel highlights that expectancy (average net profit per trade) follows the same trends as both annualized return and Sharpe ratio. The *macd_strategy* on SB shows the highest single-cell improvement (+3.73%), and *williams_r_strategy* exhibits broad-based gains across KC, NG, and VX. More conservative strategies such as *bollinger_mean_reversion* and *cci_strategy* also show uniform positive improvements across most assets.

Overall pattern. Across all three metrics, improvements are systematic. Most cells exhibit positive deltas, indicating that ProFiT successfully generalizes across both trending (e.g., *cci_strategy*) and mean-reverting (e.g., *bollinger_mean_reversion*) regimes. The alignment of high- Δ values across metrics suggests that ProFit’s

modifications tend to enhance both profitability and consistency simultaneously. Heatmaps for comparisons to Buy-and-Hold appear in the appendix.

5.2 Statistical Significance and Robustness

To ensure that ProFiT-observed performance improvements are not due to chance, we evaluate statistical significance using a (ProFiT, baseline) paired, one-sided Wilcoxon signed-rank test on the mean test-fold performance across five runs for each (ProFiT, strategy) pair. The null hypothesis H_0 assumes $\text{median}(\Delta_i) = 0$, while the alternative H_1 tests whether $\text{median}(\Delta_i) > 0$. This nonparametric test was selected due to the small number of independent runs.

Across all metrics and comparisons, more than 60% of the (ProFiT, strategy) pairs achieve statistical significance, confirming that the majority of observed improvements reflect systematic effects rather than stochasticity. In particular, 71.4% of Sharpe ratio improvements and 62.9% of expectancy improvements versus the baseline (B0) are significant, underscoring the reliability of ProFiT’s evolutionary updates. This tendency is reinforced by the narrow standard errors reported in Table 2, which demonstrate consistency across runs. Variability is largely confined to a few high-volatility assets, namely NG, VX, and ES. The convergence of both return and risk-adjusted metrics toward positive medians across runs suggests that ProFiT’s search dynamics are stable.

6 Discussion

Beyond demonstrating empirical viability, the results establish ProFiT as a viable mechanism for algorithmic trading systems, bridging symbolic reasoning with empirical feedback. A key area for future exploration lies in prompt-level evolution and meta-optimization. While our experiments featured fixed prompt templates for consistency, early observations suggested that the prompt layer could benefit from an improvement loop. This enhancement could potentially amplify discovery efficiency and diversity, which could lead to emergent improvements in search depth and generalization across market regimes, assets, and initial seeds. One obvious insight from our work is that autonomous systems can iteratively refine real-world trading policies without manual tuning. More broadly, when large language models are coupled with grounded, test-based feedback, they can serve not only as reasoning tools but as active facilitators in the process of scientific and algorithmic discovery.

One limitation of the work in its current form is the test against fixed historical data without taking in consideration how the system will modify itself over time if the market changes too much. We are currently rectifying this.

7 Conclusion

Across seven futures assets and five distinct strategy baselines, ProFiT consistently improved both profitability and risk-adjusted performance relative to

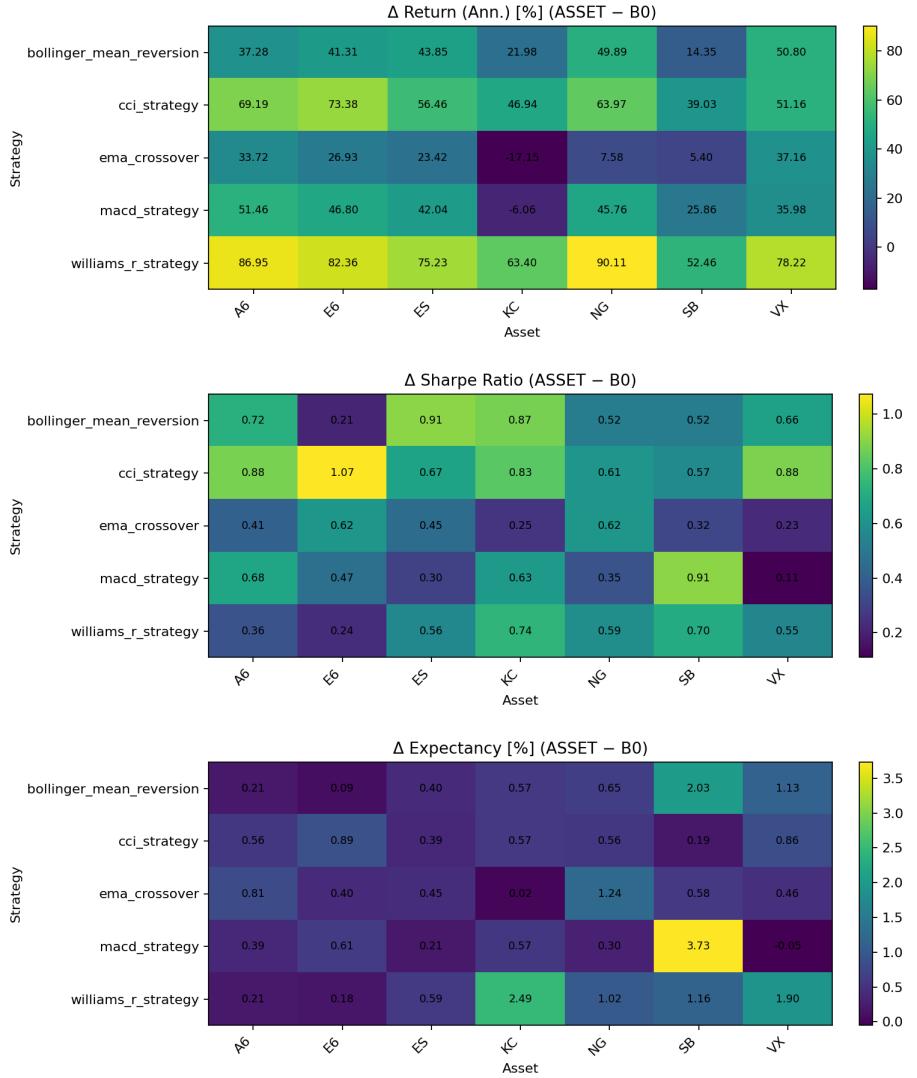


Fig. 4: ProFiT and cross-strategy mean improvements of ProFiT relative to the baseline (B0). Each cell represents the mean Δ over test folds for the given (strategy, asset) pair. Top: Ann. Return [%]; Mid: Sharpe; Bottom: Expectancy.

Buy-and-Hold and the initial strategies. Averaged across runs and temporal folds, ProFiT achieved mean improvements of +44.2% in annualized return and +0.57 in Sharpe ratio versus the baseline, with over 70% of all (asset, strategy) pairs reaching statistical significance ($p < 0.05$). These findings provide strong evi-

dence that self-improving, code-level search guided by empirical validation can produce reliable and interpretable improvements in quantitative trading systems.

References

1. Banzhaf, W.: Genetic Programming: An Introduction on the Automatic Evolution of computer programs and its Applications. Morgan Kaufmann Publishers (1998)
2. Benbassat, A., Sipper, M.: EvoMCTS: A scalable approach for general game learning. *Transactions on Computational Intelligence and AI in Games* **6**(4), 382–394 (2014)
3. Brabazon, A., Kampouridis, M., O’Neill, M.: Applications of genetic programming to finance and economics: past, present, future. *Genetic Programming and Evolvable Machines* **21**(1), 33–53 (2020)
4. Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H.P., Kaplan, J., Edwards, H., Yuri, B., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F.P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W.H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A.N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., Zaremba, W.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)
5. Clune, J.: AI-GAs: AI-generating algorithms, an alternate paradigm for producing general artificial intelligence. arXiv preprint arXiv:1905.10985 (2019)
6. Contreras, I., Hidalgo, J.I., Nunez-Letamendia, L.: A hybrid automated trading system based on multi-objective grammatical evolution. *Journal of Intelligent & Fuzzy Systems* **32**(3), 2461–2475 (2017)
7. Deng, Y., Bao, F., Kong, Y., Ren, Z., Dai, Q.: Deep direct reinforcement learning for financial signal representation and trading. *Transactions on neural networks and learning systems* **28**(3), 653–664 (2016)
8. Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K.O., Clune, J.: First return, then explore. *Nature* **590**(7847), 580–586 (2021)
9. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. *Statistics and computing* **4**(2), 87–112 (1994)
10. Langdon, W.B., Poli, R.: Foundations of genetic programming. Springer Science & Business Media (2013)
11. Lehman, J., Gordon, J., Jain, S., Ndousse, K., Yeh, C., Stanley, K.O.: Evolution through large models. In: *Handbook of evolutionary machine learning*, pp. 331–366. Springer (2023)
12. Lehman, J., Miikkulainen, R.: Neuroevolution. *Scholarpedia* **8**(6), 30977 (2013)
13. Lehman, J., Stanley, K.O.: Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation* **19**(2), 189–223 (2011)
14. Lehman, J., Stanley, K.O.: Evolving a diversity of virtual creatures through novelty search and local competition. In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. pp. 211–218 (2011)
15. Li, J., Taiwo, S.: Enhancing financial decision making using multi-objective financial genetic programming. In: *International Conference on Evolutionary Computation*. pp. 2171–2178. IEEE (2006)

16. Lüster, Z.: Backtesting.py: A python framework for strategy backtesting. <https://kernc.github.io/backtesting.py/> (2025), accessed: August 26, 2025
17. Menoita, R., Silva, S.: Evolving financial trading strategies with vectorial genetic programming (2025), <https://arxiv.org/abs/2504.05418>
18. Moody, J., Wu, L., Liao, Y., Saffell, M.: Performance functions and reinforcement learning for trading systems and portfolios. *Journal of forecasting* **17**(5-6), 441–470 (1998)
19. Mouret, J.B., Clune, J.: Illuminating search spaces by mapping elites. arXiv preprint arXiv:1504.04909 (2015)
20. Perez, D., Samothrakis, S., Lucas, S.: Online and offline learning in multi-objective monte carlo tree search. In: Conference on computational intelligence in games (CIG). pp. 1–8. IEEE (2013)
21. Poli, R., Langdon, W.B., McPhee, N.F., Koza, J.R.: *A Field Guide to Genetic Programming*. Lulu (2008)
22. Pugh, J.K., Soros, L.B., Stanley, K.O.: Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI* **3**, 40 (2016)
23. Schmidhuber, J.: Gödel machines: self-referential universal problem solvers making provably optimal self-improvements. arXiv preprint cs/0309048 (2003)
24. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. *Evolutionary computation* **10**(2), 99–127 (2002)
25. Trifonov, V., Kononov, I., Sherki, D., Svidchenko, O., Shpilman, A., Muravleva, E.: AI-powered platform for scientific discovery. In: AI4X International Conference (2025)
26. Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., Anandkumar, A.: Voyager: An open-ended embodied agent with large language models. arXiv preprint arXiv:2305.16291 (2023)
27. Wang, R., Lehman, J., Clune, J., Stanley, K.O.: POET: open-ended coevolution of environments and their optimized solutions. In: The genetic and evolutionary computation conference. pp. 142–151 (2019)
28. Yao, X.: Evolving artificial neural networks. *Proceedings of the IEEE* **87**(9), 1423–1447 (1999)
29. Zhang, J., Hu, S., Lu, C., Lange, R., Clune, J.: Darwin Gödel Machine: Open-ended evolution of self-improving agents. arXiv preprint arXiv:2505.22954 (2025)