REINFORCEMENT LEARNING AND INVERSE REINFORCEMENT LEARNING: A PRACTITIONER'S GUIDE FOR INVESTMENT MANAGEMENT

Igor Halperin, PhD
Group Data Science Leader, GenAI Asset Management Technology, Fidelity Investments

Petter N. Kolm, PhD
Clinical Professor, Director of the MS in Mathematics in Finance Program, Courant Institute of Mathematical Sciences, New York University

Gordon Ritter, PhD
Adjunct Professor, Courant Institute of Mathematical Sciences and Tandon School of Engineering, New York University
Partner, Ritter Alpha, LP

Introduction

Traditional quantitative finance relies heavily on predictive models: forecasting returns, estimating volatilities, predicting default probabilities. These supervised learning approaches excel at pattern recognition but fall short when the goal is to determine *what actions to take* in dynamic, uncertain environments where decisions have lasting consequences.

Consider a portfolio manager deciding how to rebalance a multiasset portfolio. Traditional approaches might forecast expected returns and use mean–variance optimization. This static approach ignores several crucial factors, however:

- Transaction costs and market impact that depend on order size and timing
- The dynamic nature of markets where today's trades affect tomorrow's opportunities
- The need to adapt strategies as market conditions evolve
- Risk considerations that go beyond simple variance measures

Reinforcement learning (RL) is a branch of machine learning capable of addressing these issues. RL is concerned with how intelligent agents should take actions in an environment to maximize some notion of cumulative reward, as we will explain in more detail later. Unlike supervised learning, where an agent is given explicit "correct answers," an RL agent learns through experience—by trying out actions and observing the consequences.

Inverse reinforcement learning addresses the complementary problem: Given observed behavior from an expert (successful trader, market participant, or even the market itself), what underlying objectives or preferences explain that behavior? This approach is particularly

valuable in finance, where true utility functions are rarely known explicitly but observed behavior is abundant.

Why RL and IRL Matter for Finance

The relevance of RL to finance stems from several key characteristics:

Sequential decision making: Most financial problems involve sequences of interdependent decisions. A trade execution strategy unfolds over time, with each trade affecting market conditions for subsequent trades. Portfolio rebalancing decisions today influence the risk–return profile tomorrow.

Uncertainty and adaptation: Financial markets are inherently uncertain and nonstationary. Successful strategies must adapt to changing conditions. RL agents can learn to recognize different market regimes and adjust their behavior accordingly.

Delayed and complex rewards: The consequences of financial decisions often manifest with significant delays and through complex causal chains. A hedging decision today may prove its worth only during a market crisis months later.

Market impact and feedback loops: In many financial contexts, an agent's actions influence the environment itself. Large trades move prices, which affects subsequent trading opportunities. This dynamic creates feedback loops that traditional static optimization cannot capture.

A Practitioner's Guide to RL Fundamentals

The RL Framework: Key Components

Understanding RL begins with its core components (Sutton and Barto 2018).

Agent

The agent is a decision maker that represents a market participant, such as a risk taker, liquidity provider, market maker, or institutional or retail trader. The agent operates (acts) over a time interval $[0, T]$, where T is the planning/acting time horizon. Most RL algorithms are formulated for discrete time steps. With a slight abuse of notation, which is quite common in the RL literature, we use the symbol t to denote both the discrete-value time and the index on the time grid, so that we can write $t = 0,1,\ldots,T$. Depending on the specific setting, T can vary between milliseconds and months or even years.

Environment

The environment is the market structure or venue where the agent operates and takes actions. In finance, this includes the following:

- Electronic limit order books (exchanges, such as NYSE and Nasdaq)
- Over-the-counter (OTC) markets for bonds and derivatives
- Alternative trading systems (dark pools, crossing networks)
- Auction mechanisms (opening/closing auctions)

- Bilateral negotiation markets (institutional block trading)
- Market-maker networks and dealer markets

State (S)

The state is a snapshot of all relevant information available to the agent at a given time. Examples include the following:

- Current portfolio positions and cash holdings
- Market data: prices, volumes, spreads, volatility measures
- Order book depth and imbalance (if observable)
- Recent price movements and technical indicators
- Time of day, calendar effects, and time to market close
- Macroeconomic indicators and news sentiment
- Risk metrics: value at risk (VaR), exposure concentrations, correlation estimates

Action (A)

Action consists of the choices available to the agent:

- Trade quantities and directions (buy/sell/hold)
- Order types and timing
- Portfolio allocation adjustments
- Risk management decisions (hedging, position sizing)

Reward (R)

Reward is the feedback signal that guides learning. The reward in RL is typically assumed to be received at each time step $t \in [0, T]$ over the course of action of the agent, where T is a time horizon. The reward at time step t is usually defined to be a function of action A_t taken at this step, as well as the current-step and next-step values of the state variable—respectively, S_t and S_{t+1}, so it is usually written as $R(S_t, A_t, S_{t+1})$. In finance, rewards typically reflect the following:

- Profit and loss (P&L)
- Risk-adjusted returns (Sharpe ratio, information ratio)
- Transaction costs and market impact
- Risk penalties (VaR, conditional value at risk, drawdown measures)

Policy (π)

The policy is the agent's strategy—a mapping from states to actions. This is what the RL algorithm learns and optimizes. Depending on the type of the specific RL algorithm, the policy can be deterministic, when the same current state always produces the same action, or it can be stochastic. In the latter case, instead of being a function, the policy is given by a probability

distribution, such that for a given state, only the probabilities of different actions, rather than actions themselves, are fixed.

By definition, the optimal policy is a policy that maximizes the total expected reward from taking actions.[1] The latter quantity is often written as follows:

$$E^{\pi}\left[\sum_{t=0}^{T} e^{-\gamma t} R(S_t, A_t, S_{t+1})\right].$$

Here, $E^{\pi}[\cdot]$ stands for the expected value assuming that policy π will be used to pick all future actions A_t. Furthermore, $\gamma \in [0,1]$ is a discount factor that specifies how much immediate rewards are more preferred over rewards received in the future. The meaning of the discount factor, γ, in RL is quite similar to the meaning of the discount factor in finance. The expected total reward defined in Equation 1 is often referred to as the *return* in the RL literature.

Example: Optimal Trade Execution

To help contextualize the foregoing information, consider an optimal execution problem in which you need to sell 100,000 shares of a stock over the next hour:

State: Remaining shares to sell (100,000 $\rightarrow$ 0), current stock price, bid–ask spread, order book depth, time remaining (60 minutes $\rightarrow$ 0), recent volatility

Actions: Number of shares to sell in the current time period (0 to remaining quantity)

Rewards: Execution price received minus a penalty for price impact and inventory risk—for example,

$$R_t = \text{Price received} - \lambda_1 \times \text{Market impact} - \lambda_2 \times \text{Inventory risk}.$$

Policy: A function that determines how many shares to sell given the current state

The RL agent learns through trial and error, experimenting with different execution rates under various market conditions, gradually improving its strategy based on the rewards received.

Mathematical Foundations: MDPs and Beyond

Markov Decision Processes

The mathematical foundation for many RL problems is the *Markov decision process* (MDP), defined by the tuple (S,A,P,R,γ):

- S = Set of possible states. States can be continuous or discrete.
- A = Set of possible actions. Actions can be continuous or discrete.
- $P(S_{t+1} \mid S_t, A_t)$ = State transition probabilities.
- $R(S_t, A_t, S_{t+1})$ = Reward function.
- $\gamma \in [0,1]$ = Discount factor for future rewards.

[1]This definition applies to the most commonly used version of RL, called the *risk-neutral RL*. Other modeling choices will be described later.

The key assumption is the *Markov property*: The future depends on only the current state, not the history of how we arrived there. This assumption allows one to define the state dynamics in terms of single-step transition probabilities $P(S_{t+1} \mid S_t, A_t)$ that reference only the current-step and next-step values of the state variable. Although the Markovian assumption may appear somewhat restrictive for financial applications, it can often be satisfied by including sufficient information in the state representation.

Partially Observable Environments

In practice, financial environments are often partially observable. Traders do not have access to all relevant information—such as other participants' intentions, unrevealed news, central bank decisions, and so on. This situation leads to *partially observable MDPs* (POMDPs), where agents receive observations O_t that provide only partial information about the true state, S_t.

Financial POMDP Example: Trading with Hidden Liquidity

Consider a trader executing a large order in a market with hidden liquidity (e.g., dark pools or iceberg orders):

- True state S_t: Total available liquidity at each price level (including hidden orders)
- Observation O_t: Only visible order book depth
- Belief state b_t: Probability distribution over possible hidden liquidity levels
- Action A_t: Order size and aggressiveness

The agent must maintain a belief state $b_t = P(S_t \mid O_{1:t}, A_{1:t-1})$ and update it using Bayesian inference:

$$b_{t+1}(s') = \frac{P(\quad_{t+1} \mid s')}{P(\quad_{t+1} \mid b_t)} \sum_s P(s' \mid A \quad_t) b_t(s).$$

POMDPs are significantly more complex to solve, often requiring agents to maintain belief states (probability distributions over possible true states). However, they provide a more realistic model of financial decision making under uncertainty.

Bellman Equations: The Optimization Principle

Historically, RL grew out of dynamic programming, developed by Richard Bellman in the 1960s. Dynamic programming offers a systematic approach to the problem of sequential control that essentially relies on additivity of the RL return defined in Equation 1. With dynamic programming, the total return is viewed as a function of either the current state or a combination of the current state and the first action taken. In the former case, the total return (Equation 1) is referred to as the value function (or *V*-function), while in the latter case, it is referred to as the value-action function (or *Q*-function). The policy optimization amounts to finding an optimal policy π_* that maximizes the *V*-function or the *Q*-function. The optimal *V*- and *Q*-functions are denoted, respectively, as $V^*(S_t)$ and $Q^*(S_t, A_t)$. These functions satisfy recursive relations known as Bellman optimality equations:

$$V^*(\quad) = \max_a \left[\mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] + \gamma \sum_{s'} P(s' \mid s, a) V^*(\quad') \right].$$

$$Q^*(s,a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q^*(s',a').$$

These equations state that the optimal value of a state (or state-action pair) equals the expected immediate reward plus the discounted expected value of the best possible next state.

The classical dynamic programming typically focuses on solution of the first Bellman optimality equation for the *V*-function for low-dimensional and discrete sets of states and action, while assuming that the dynamics of the transitions between states are known. Such constraints on the dimensionality of a state-action space and the need for an explicit model of the environment presented severe limitations for the use of dynamic programming for many real-life applications where the dimensionality of the state-action space is high and where the explicit model of the world is typically not available or hard to estimate.

Respectively, one approach of RL extends the dynamic programming formulation of sequential decision-making problems to models with a high-dimensional continuous or discrete state-action space, without assuming that dynamics of the world are known. Instead, this approach, known as the *value-based RL*, relies on samples from data obtained from interactions of an agent with its environment, giving rise to sample-based solutions of Bellman optimality equations.

Value-Based RL vs. Policy-Gradient vs. Actor-Critic Methods

The value-based RL that uses value or action-value functions together with Bellman equations is not the only available approach to RL. Other approaches exist that do not rely on Bellman equations but, rather, directly optimize a parameterized policy to maximize the total return defined in Equation 1. These methods are collectively known as *policy-gradient methods*. The most basic policy-gradient method, REINFORCE, is very simple to implement but has the drawback of producing high variance for total returns (Sutton and Barto 2018). Finally, with actor-critic RL methods, two different parameterized functions are used to represent the value function and the policy function. Actor-critic methods produce lower variance of total returns than pure policy-gradient methods.

RL Algorithm Landscape

Classical Algorithms

Perhaps the most famous RL algorithm is the value-based RL algorithm called *Q-learning*. It learns the optimal action-value function, $Q^*(s,a)$, through temporal difference updates:

$$Q(\ ,a) \leftarrow Q(\ ,a) + \alpha\left[R_{t+1} + \gamma \max_{a'} Q(\ _{t+1}, a') - Q(\ ,a)\right].$$

The Q-learning algorithm can be interpreted as a sample-based solution of the Bellman optimality equation for the Q-function. For discrete state-action systems, values of the Q-functions for different combinations of the state and action can be stored in a table in which rows and columns correspond to different values of the state and action, while the Q-update equation shown here is used to update the values in the table upon observing a new state and rewards from taking a certain action in the current state. This process is referred to as *tabular Q-learning*. Q-learning is considered "off-policy," meaning it can learn the optimal policy while

following a different (exploratory) policy. A different form of tabular learning is presented by the SARSA algorithm, an "on-policy" alternative that updates according to the action actually taken:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s,a)].$$

From Tabular to Deep Learning: Function Approximation and Deep RL

Financial state spaces are typically enormous or continuous (asset prices, portfolio weights, market indicators). *Function approximation* addresses this situation by learning parameterized functions that approximate value functions (for value-based RL methods), policies (for policy-gradient methods), or both (for actor-critic methods). Although many machine learning approaches (e.g., trees) are able to produce universal function approximation methods, today, various versions of artificial neural networks serve as the most popular function approximation approach.

Among value-based deep RL methods, *deep Q-networks* (DQNs) are most widely known. DQNs combine Q-learning with deep neural networks, enabling RL to handle high-dimensional state spaces. In addition to using deep neural networks for value function approximation, DQNs introduce other innovations that improve their training. In particular, they use experience replay, which amounts to storing and randomly sampling past experiences for training. The other innovations are target networks: separate networks for stability during training.

With policy-gradient methods, function approximations are used to approximate policies (for pure policy-based approaches, such as REINFORCE) or both value and policy functions (for actor-critic methods). Function approximations for both functions are also used in proximal policy optimization (PPO) algorithms—a policy-gradient method that uses a clipped objective function to ensure that policy updates stay within a certain "trust region," which makes training more stable.

Model-Based vs. Model-Free RL

One of the main paradigms of RL is its reliance on learning directly from experience, without modeling environment dynamics. This is commonly referred to as *model-free RL*. Examples of model-free RL methods include Q-learning and policy gradients, outlined previously. Model-free RL approaches have both pros and cons:

- *Pros:* Simple to implement, effective when dynamics are unknown/complex
- *Cons:* Sample inefficient, requires many interactions

An alternative to model-free RL is *model-based RL*. With this approach, one first learns a model of environment dynamics and then uses this model for planning. In a sense, model-based RL takes us one step back to the setting of dynamic programming, which likewise assumes that the dynamics of the environment are known. Model-based RL methods have their own pros and cons:

- *Pros:* More sample efficient, enables planning and scenario analysis
- *Cons:* Possibility of model errors compounding, more complex to implement

For financial applications, the choice between model-free and model-based RL depends on the specific problem. High-frequency trading with complex, fast-changing dynamics might favor model-free approaches, whereas strategic asset allocation could benefit from model-based methods.

Online vs. Offline RL

Another important distinction between different RL algorithms concerns how they are trained. In online RL methods, an agent learns by taking actions and observing rewards received from its environment. Such settings are very common in applications of RL in robotics. The important question faced by the agent is how to optimally combine taking actions that have previously shown good reward versus trying new actions whose rewards will be observed only after trying them. This problem is known as the exploration–exploitation dilemma of RL (Sutton and Barto 2018).

A different setting is provided by offline RL. In this formulation, the agent learns the optimal policy without a direct interaction with its environment but, rather, using historical data collected from previous interactions of another (or the same) agent with the same environment. In this setting, the agent can no longer rely on trial-and-error methods to find optimal actions: The exploration–exploitation dilemma was already addressed (potentially not optimally!) by the previous agent. As a result of its inability to explore different actions in interaction with its environment, offline RL is generally harder than online RL. However, the setting of offline RL closely matches the setting of classical financial models that are typically fit to fixed datasets consisting some historical data.

Risk-Aware RL: Beyond Expected Returns

The total return given by the sum of all future reward is clearly a random quantity as seen at the start of agent's action (at time 0) because it depends on future states and actions that are not yet known at time 0. This situation is quite similar to how the future return of an investment portfolio is a random quantity at the initial time of portfolio initiation. However, the standard RL optimizes only the expected cumulative reward (i.e., the mean of this distribution) and does not try to control its higher moments, such as variance, kurtosis, or tail risk measures. Again similar to classical financial settings, this condition is often not sufficient for the practical purposes of using RL for financial applications, where we want to control not only the mean future total reward but also risk, or variability around this mean, as expressed, for example, by the variance of the total reward. Because standard RL does not address risk/uncertainty around the mean expected total reward, it is sometimes referred to as *risk-neutral RL*.

Because financial decision making is inherently risk sensitive, risk-neutral RL is often inadequate for this task, especially if pure P&L (or return) is taken as the reward for RL. Several extensions address this limitation.

Risk-Sensitive RL

Mean–variance RL incorporates both expected return and variance into the reward:

$$\text{Reward} = \mathbb{E}[\text{Return}] - \lambda \times \text{var}[\text{Return}].$$

In this approach, one uses a risk-adjusted return as a reward function, very much in the spirit of the classical Markowitz portfolio theory. The attractive property of this approach is that it still can use the risk-neutral RL formulation, where the variance penalty is simply added to the definition of the reward such that we still optimize its expected (mean) value. The other attractive feature of this specification is that the reward defined by this relation is a quadratic function of actions, which tremendously simplifies the RL algorithm and in fact enables its implementation without using any neural networks altogether (Dixon, Halperin, and Bilokon 2020). The known drawback of using variance as a risk measure is that it penalizes both negative and positive returns, while ideally we may want to penalize only negative returns.

By using *conditional value at risk* (CVaR) as the risk penalty component of the reward function, RL methods with such rewards directly optimize tail risk measures, which is crucial for downside protection. Unlike mean–variance RL, which can proceed without using neural nets, RL methods that use CVAR typically need to use deep neural networks as function approximation tools.

Risk-constrained RL methods are similar to CVAR-based approaches. They maximize expected returns subject to risk constraints, such as maximum drawdown or VaR limits.

Distributional RL

Instead of learning expected values, *distributional RL* learns the full probability distribution of cumulative future rewards. Having access to the full distribution of the total reward allows one to compute any risk measure (e.g., VaR, CVaR, skewness) for the ultimate decision making. Such algorithms as quantile regression DQN (QR-DQN) and implicit quantile networks (IQNs) represent the return distribution using quantiles, making them particularly suitable for financial applications focused on tail risks. Distributional RL can also be constructed with the continuous time formulation. With this approach, policy optimization amounts to a numerical solution of certain partial differential equations (Halperin 2024).

Use Cases for RL Applications in Finance

Here, we provide a brief outlook for various use cases for RL in financial applications. Without attempting a detailed presentation, we focus here on the few key elements, including specifications of state, action, and reward, as well as outlining implementation consideration.

Optimal Trade Execution

Problem Setting

A fundamental problem for any large institutional investor is how to execute a large order with minimal price disruption. Standard benchmarks such as time-weighted average price or volume-weighted average price provide simple, static schedules but fail to adapt to changing market conditions during the execution horizon. An RL agent, in contrast, can learn a dynamic policy that breaks a large "parent" order into a sequence of smaller "child" orders, adapting the pace of trading to minimize costs. The core challenge is to balance the trade-off between the *market impact cost* of trading aggressively and the *timing risk* of trading slowly in a volatile market.

RL Formulation

The problem is naturally framed as a finite-horizon MDP. The *state space* typically includes the remaining quantity to be traded, the time left in the execution window, current asset price, and potentially microstructure features, such as order book imbalance or recent volatility. The *action* is the size of the child order to submit in the current time slice. The *reward function* is crucial and must encapsulate the trade-off: It is often formulated as the negative of implementation shortfall or, more explicitly, the execution price achieved, penalized by terms representing price slippage resulting from market impact and the risk exposure of the remaining inventory.

Implementation Considerations

A significant challenge is developing a realistic market impact model, which can be either estimated offline from historical data or learned online as part of the RL agent's interaction. Furthermore, a robust execution agent must be able to generalize across different market volatility regimes and asset-specific behaviors. For training and validation, a high-fidelity market simulator that can accurately model price dynamics and market impact is indispensable because training in a live market is impractical and costly. Finally, the learned policy must operate within the constraints of regulatory requirements, such as "best execution" mandates.

Dynamic Portfolio Optimization

Problem Setting

Classical single-period portfolio optimization, such as Markowitz's mean–variance framework, is static and highly sensitive to estimation errors in its inputs (expected returns and covariances). Its static nature prevents optimal portfolio allocation decisions in terms of minimization of transaction costs or optimal use of multihorizon predictive signals. Dynamic portfolio optimization extends the static mean–variance framework to a multiperiod setting, where an RL agent learns a rebalancing policy over a long horizon. This approach aims to maximize cumulative risk-adjusted returns while accounting for real-world frictions, such as transaction costs and market impact.

RL Formulation

The *state space* for a dynamic portfolio agent includes the current portfolio weights, recent asset returns, market regime indicators (e.g., from a hidden Markov model), and predictive macroeconomic factors. The *action space* is the set of target portfolio weights for the next period. Because weights are continuous variables, this problem is ill suited for tabular RL methods and requires function approximation. The *reward* is typically a risk-adjusted return metric, such as the period's Sharpe ratio or a utility function of the portfolio's return, net of transaction and holding costs.

Implementation Considerations

The continuous and high-dimensional nature of the state and action spaces makes this application of RL challenging in practice. Advanced techniques are often used in this setting. In particular, with hierarchical RL approaches, the problem can be decomposed into a high-level strategic allocation agent that sets broad targets over long horizons (e.g., quarterly) and a low-level tactical agent that makes finer adjustments (e.g., monthly) to achieve those targets. Alternatively, multi-objective RL can be considered to navigate the complex trade-offs between competing

goals, such as maximizing returns, minimizing volatility, and reducing portfolio turnover to limit transaction costs. Lastly, a computationally efficient and noise-robust approach particularly helpful for financial portfolio optimization tasks is provided by G-learning, a probabilistic extension of Q-learning. This method was applied to goal-based wealth management by Dixon and Halperin (2020).

Option Pricing and Hedging

Problem Setting

The classical Black–Scholes–Merton model (Black and Scholes 1973; Merton 1974) provides a cornerstone for derivative pricing but relies on idealized assumptions, such as continuous and costless hedging, that do not hold in practice. When hedging is discrete and involves transaction costs, the problem of pricing and hedging a derivative becomes a sequential decision-making problem under uncertainty. The goal is to design a dynamic hedging strategy that minimizes a risk-adjusted measure of the total hedging cost. Several RL-inspired frameworks have been developed to address this problem, moving beyond the classical risk-neutral paradigm.

RL Formulation and Approaches

Two main classes of methods have emerged, which can be broadly understood as value-based and policy-based approaches to the hedging problem.

Value-Based RL

The QLBS Model proposed by Halperin (2020) directly applies the principles of value-based RL. The problem is framed as an MDP where the agent (an option seller) seeks to learn an optimal hedging policy. The *state* is defined by the underlying asset price and time to expiration. The *action* is the hedge adjustment (the amount of the underlying to hold). The *reward function* is defined as the negative of a risk-adjusted hedging cost. For tractability, this cost takes a quadratic form (mean–variance utility), including a penalty for the variance of the hedge portfolio's value, which directly incorporates the hedger's risk aversion. The agent learns an optimal action-value function (Q-function), from which both the optimal hedge policy and the corresponding option price (the negative of the Q-value) are derived simultaneously.

Deep value-based RL for option hedging is a more end-to-end approach that has been explored. Du, Jin, Kolm, Ritter, Wang, and Zhang (2020) applied state-of-the-art DRL algorithms, such as proximal policy optimization (PPO), to learn hedging strategies directly from market simulations. A key advantage of their framework is its ability to handle practical market frictions, such as discrete trading times and nonlinear transaction costs. Furthermore, their approach is highly efficient because a single trained DRL agent can learn to hedge a whole range of option strikes simultaneously, eliminating the need for retraining on a per-strike basis. The authors demonstrated that the DRL agent can learn strategies that match or outperform traditional delta hedging in terms of profit and loss.

Direct Policy Optimization

An alternative framework is *deep hedging*, pioneered by Buehler, Gonon, Teichmann, and Wood (2019). This approach can be seen as a form of direct policy optimization. Instead of solving

the recursive Bellman equation, it frames the entire hedging problem as a single end-to-end optimization. A neural network is used to directly represent the hedging policy, taking the current market state as input and outputting the hedge position. The training process involves the following:

- Simulating a large number of market scenarios (paths) for the underlying asset(s)
- For each path, applying the hedging strategy defined by the current neural network
- Calculating the final P&L distribution of the fully hedged portfolio across all paths
- Using backpropagation to update the neural network's weights to minimize a chosen risk measure of this final P&L distribution (e.g., CVaR, mean–variance utility, or expected shortfall)

Implementation Considerations

All RL approaches directly address the limitations of the classical model by embedding real-world frictions, such as *discrete hedging* and *transaction costs*. However, they differ in their philosophy and implementation. The QLBS approach aligns closely with traditional RL theory (MDPs, Bellman equations, Q-functions) and can be computationally efficient when its quadratic reward assumption leads to semi-analytical solutions, as explored in fitted Q-iteration (Ernst, Geurts, and Wehenkel 2005). Also, because Q-learning is a model-independent method, the QLBS approach amounts to a model-independent and data-driven option hedging and pricing method. Deep hedging, while conceptually related, is implemented as a global optimization that is very flexible; changing the risk objective simply means changing the final loss function. It is inherently a model-based approach because it learns the optimal policy for a given simulation model of the market (e.g., Heston, SABR). The power of neural networks also allows it to handle very complex and non-Markovian state representations.

End-to-End Deep RL for Asset Allocation

Problem Setting

Traditional quantitative asset allocation involves a two-step process: First, predict expected returns and covariances, and second, use these predictions in an optimization framework (e.g., mean–variance). This process is fragile because performance is highly sensitive to errors in the initial prediction step. A more robust approach would be to learn the mapping from market data to portfolio weights in an end-to-end fashion.

RL Formulation and Approaches

Noguer i Alonso and Srivastava (2020) proposed a model-free, end-to-end deep reinforcement learning approach that bypasses the explicit forecasting step. The RL agent learns a direct mapping from raw market data to optimal portfolio weights.

- *State:* The state is represented as a tensor of recent price history (e.g., 50 days of high, low, and close prices for a universe of stocks).
- *Agent:* The agent is a deep neural network (the authors tested various architectures, such as CNNs, RNNs, and LSTMs) that learns the allocation policy.

- *Action:* The action is an output vector of portfolio weights for the next period.
- *Reward:* The reward is a simple and direct financial objective, such as the portfolio's average logarithmic return, adjusted for transaction costs. To encourage stable portfolios, the architecture incorporates a "portfolio vector memory," which considers past weights when determining new ones and implicitly penalizes high turnover.

Implementation Considerations

This end-to-end framework learns to predict and optimize simultaneously. Noguer i Alonso and Srivastava (2020) showed it can construct portfolios that outperform traditional methods, such as mean–variance optimization and risk parity, even when only given raw price series as input. The use of a portfolio memory is a key architectural choice to control for turnover, which is a critical practical consideration.

Algorithmic Trading

Problem Setting

Algorithmic trading seeks to automate trading decisions to capitalize on market opportunities more efficiently than human traders can. The core challenge is to develop a strategy that can adapt to changing market conditions and dynamically balance the trade-off between expected returns and various forms of risk.

RL Formulation and Approaches

The problem is naturally framed as an MDP where an agent learns an optimal trading policy.

- *State space:* This includes such features as historical price data, technical indicators (e.g., moving averages), market volatility, and the agent's current portfolio state (cash and asset holdings).
- *Action space:* This consists of such actions as buying, selling, and holding assets. The action can be discrete (e.g., buy one unit) or continuous (e.g., allocate 15% of capital).
- *Reward function:* The reward function is typically defined to reflect a risk-adjusted return measure. A simple profit-and-loss reward can be augmented with risk measures, such as a penalty for high portfolio variance, large drawdowns, or a direct optimization of the Sharpe ratio.

A comprehensive review by Pricope (2021) considered the application of deep RL to this problem, noting that many studies show statistically significant outperformance over simpler baselines in simulated environments.

Implementation Considerations

A key challenge highlighted in the literature is the gap between simulated performance and real-world applicability. Many studies are proofs of concept conducted in environments that do not fully capture real-time market frictions, latency, and data imperfections. Successful implementation requires careful consideration of transaction costs, market impact, and robust out-of-sample validation.

Market Making

Problem Setting

Market making is a high-frequency strategy in which a trader provides liquidity by simultaneously placing bid and ask limit orders, aiming to profit from the spread. The main challenge is to set optimal quotes that balance maximizing spread capture against managing two key risks: inventory risk (holding a large, undiversified position) and adverse selection risk (trading against an informed counterparty who knows where the price is headed).

RL Formulation and Approaches

The market making problem is well suited to an MDP formulation where the agent learns an optimal quoting strategy.

- *State space:* This encompasses the agent's current inventory, features of the limit order book (e.g., bid–ask spread, volume imbalance), and market indicators, such as volatility.
- *Action space:* This includes setting the bid and ask quotes relative to the market midpoint and deciding the size of the orders to be placed.
- *Reward function:* Carefully designed, this function rewards captured spreads while penalizing inventory risk and losses from adverse selection.

Spooner, Fearnley, Savani, and Koukorinis (2018) presented a foundational example in which a deep RL agent learns to perform market making. Their agent learns a value function to optimize quotes, demonstrating that an RL approach can outperform traditional stochastic control-based strategies in simulated environments.

Implementation Considerations

The primary challenge in applying RL to live market making is latency. High-frequency environments require decisions on microsecond timescales, which can be difficult for complex neural network models to meet. Furthermore, creating a high-fidelity market simulator that accurately captures order flow dynamics and adverse selection is a significant undertaking yet is crucial for training a robust agent.

Inverse Reinforcement Learning: Inferring Hidden Objectives

Reinforcement learning learns optimal policies given known objectives, as codified by rewards. The reward in RL is observable, as a result of online interaction of an agent with its environment or for offline RL, as a part of historical data. In many real-world problems, however, we observe only agents' actions—not their reward. Arguably, in real life, such scenarios are encountered more often relative to scenarios where rewards are observed. The ultimate objective of learning in this setting is still the same as in the RL scenario—that is, to learn the optimal policy by observing the agent's behavior. In contrast to the standard RL scenario, however, we now observe only the states of the environment and actions taken by the agent and do not observe rewards received by the agent.

Clearly, without additional assumptions, such problems do not have a solution. For example, if an agent's actions are purely random, not much (if anything) can be learned from such observations. If we assume that the agent's actions were supposed to achieve some objectives, however, then we can address the inverse problem: Given observed behavior, what objectives explain that behavior?

If the objective is to maximize the total expected reward, this inverse problem formulation gives rise to *inverse reinforcement learning* (IRL). IRL uses the observed behavior of an agent and infers the reward function that is assumed to be optimized by the agent. Note that in most IRL applications, such inference of the reward function is *not* the end goal. The end goal is rather the same as in conventional (direct) RL, which is to find the optimal policy that maximizes the total expected reward. Unlike the task of RL, however, where rewards are a part of the inputs, in IRL we first have to infer them from the behavior, and then we use them to find the optimal policy.

The Challenge of IRL: An Ill-Posed Problem

As with many inverse problems, IRL is an *ill-posed problem*: For any given set of observed behaviors, an infinite number of reward functions could explain that behavior. For instance, a policy of doing nothing is optimal for a reward of zero but also for any reward function that depends only on the state, not the action. This is the issue of *reward shaping invariance*: an optimal policy is unchanged if we transform the reward function by adding a potential-based term (Ng, Harada, and Russell 1999).

In order to have a solution, one needs to make additional assumptions about the agent in order to select the "best" or most plausible reward function from the many possibilities. One such assumption can be that the agent's behavior is optimal or close to optimal. Although such assumptions may be reasonable in some applications (e.g., robotics), it is hardly appropriate in finance, where optimality does not even exist in absolute terms and can be defined only relative to some benchmark.

IRL vs. Imitation Learning

The other question that can be asked in relation to the declared objective of IRL is, Why do we need to first infer the reward, as long as we assume that the observed behavior is already optimal? Can we simply build a supervised learning–type model that would simply mimic the observed behavior of an agent in different states of the world, without even asking a question about the agent's reward function? It turns out that such strategies are indeed feasible in certain circumstances, and the subfield of machine learning that studies these methods is known as imitation learning (IL). Although IL and IRL have the shared final goal of finding optimal policy from the observed behavior, they differ in the intermediate steps. IRL infers the reward function as the intermediate step, while IL proceeds without it. In general, IRL methods often work better than IL methods, and they offer more flexible and portable solutions because a reward function offers a succinct description of agents' goals that is portable across different environments (Dixon et al. 2020). Therefore, we will mostly focus on IRL methods for financial applications.

Key IRL Approaches

To deal with its ill-posed nature, IRL methods impose additional principles to find a unique reward function.

Maximum Entropy IRL

The *maximum entropy (MaxEnt) IRL* principle is a popular approach to regularize the problem (Ziebart, Maas, Bagnell, and Dey 2008).

- *Principle:* Among all reward functions that explain the expert's behavior, choose the one that makes the expert's policy as random as possible (i.e., maximizes its entropy). The intuition is to match the observed behavior while being maximally non-committal about behavior that hasn't been observed.
- *Probabilistic policy:* This principle naturally leads to a stochastic policy in which the probability of taking an action is exponentially proportional to its value. This is often called a Boltzmann or softmax policy.
- *Implementation:* This approach turns IRL into a maximum likelihood problem on the observed expert trajectories. The main computational challenge is often calculating the normalization constant (partition function) for this policy distribution, which requires summing or integrating over all possible actions at each step.

Bayesian and Gaussian Process IRL

Another way to handle the ill-posed nature of IRL is through a Bayesian lens. Instead of seeking a single best-fit reward function, Bayesian IRL aims to find a posterior distribution over all plausible reward functions, given the observed expert data.

Gaussian process IRL (GPIRL) is a specific and powerful nonparametric implementation of this idea (Levine, Popović, and Koltun 2011).

- *Principle:* A Gaussian process (GP) is placed as a prior over the unknown reward function. A GP can be thought of as a "distribution over functions." It provides a flexible way to represent the belief that the reward function is likely to be smooth, without having to specify its exact functional form (e.g., linear or quadratic).
- *Learning process:* Starting with this GP prior, the algorithm observes the expert's state-action trajectories. It then uses Bayesian inference to update the prior, resulting in a posterior distribution over reward functions that are consistent with the observed behavior.
- *Benefits:* The main advantage is flexibility. GPIRL can capture complex, nonlinear reward functions without manual feature engineering. This ability makes it particularly suitable for financial applications for which the relationship between market states and a trader's implicit rewards can be highly nuanced. This method was notably used in the financial applications discussed later.

Adversarial Imitation Learning

A different and powerful class of methods frames imitation learning as a two-player game.

- *Generative adversarial imitation learning (GAIL):* Ho and Ermon (2016) proposed GAIL, which does not explicitly recover a reward function. It trains a *generator* (the agent's policy) to produce state-action trajectories that are indistinguishable from an expert's trajectories, as judged by a *discriminator*. The discriminator is trained simultaneously to tell the difference between the agent's and the expert's behavior. GAIL is pure imitation learning.

- *Adversarial inverse reinforcement learning (AIRL):* Finn, Christiano, Abbeel, and Levine (2016) extended this framework by structuring the discriminator in a specific way. In AIRL, the discriminator's output can be decomposed to recover not only a policy but also a reward function. This approach elegantly connects adversarial learning back to the original goal of IRL.

T-REX: Learning from Ranked Demonstrations

Standard IRL often assumes expert demonstrations are (nearly) optimal. This can be a strong and often incorrect assumption. *Trajectory-ranked reward extrapolation (T-REX)* offers a powerful alternative by learning from demonstrations of varying quality (Brown, Goo, Nagarajan, and Niekum 2019).

- *Principle:* Instead of a set of optimal demonstrations, T-REX uses a set of trajectories that have been *ranked* by preference (e.g., "trajectory A is better than B"). It does not require knowing the absolute quality, only the relative ranking.
- *Learning intent:* The objective is to learn a reward function such that the total reward assigned to each trajectory is consistent with the given ranking. By learning what makes one trajectory better than another, T-REX can infer the underlying *intent* of the demonstrator.
- *Surpassing the teacher:* Because it learns an underlying reward function rather than simply mimicking actions, the learned reward can be used with a standard RL algorithm to find a policy that is even better than the best demonstration provided. This is a crucial step toward building agents that can learn from and improve on human behavior.

Inverse Reinforcement Learning in Action: Financial Use Cases

IRL opens up new avenues for analyzing financial behavior and markets. In this section, we present a short and nonexhaustive overview of applications of IRL in the financial domain.

Algorithmic Trading Strategy Identification

- *Problem:* High-frequency trading (HFT) firms use a diverse set of strategies. Regulators and market operators are interested in identifying and clustering these strategies from observable order data to monitor market health and detect manipulative behavior. Standard clustering based on statistical features of trading activity (e.g., order-to-trade ratio) can be crude and may not capture the underlying objectives.
- *IRL approach:* Yang, Qiao, Beling, Scherer, and Kirilenko (2015) pioneered an approach using Bayesian IRL (specifically, Gaussian process IRL). Their approach treats HFT strategies as "experts" and uses their observed order placements (actions) in the limit order book (state) to infer the reward function each strategy is optimizing. By clustering strategies based on the parameters of their learned reward functions, they achieved more meaningful and interpretable groupings of behavior than by using simple statistical features. The reward function captures the agent's implicit trade-offs between, for instance, aggressive execution and inventory risk.

Sentiment-Based Trading Strategies

- *Problem:* Build a trading system that systematically exploits the relationship between investor sentiment and market dynamics.
- *IRL approach:* Yang, Yu, and Almahdi (2018) framed the problem using GPIRL. Their approach treats aggregate news sentiment as the "action" of a single, collective market agent. The market state is defined by recent price dynamics, and GPIRL is used to infer the reward function that this collective agent is maximizing. This learned reward function, which implicitly captures how the "market" values taking bullish or bearish actions given certain conditions, can then be used by a direct RL agent to make its own trading decisions.

Inferring Customer Preferences in Consumer Finance

- *Problem:* Businesses offering subscription or recurring utility services (e.g., mobile data plans, cloud computing, energy) need to understand customer behavior to design better products and pricing plans. Customer decisions, such as daily consumption, are sequential and depend on the current state (e.g., remaining quota, days left in the billing cycle).
- *IRL approach:* This problem can be framed as inferring a customer's latent utility (reward) function from their observed consumption patterns. A MaxEnt IRL algorithm for this problem was proposed by Dixon et al. (2020). It uses a parametric reward function that captures the trade-offs a customer makes, including the utility of consumption, a penalty for exceeding a quota (and paying an overage price), and a potential reward for forgoing consumption. By observing a customer's consumption history, the MaxEnt IRL algorithm finds the utility parameters that make the observed behavior most probable.

Once this customer-specific utility function is learned, the firm can perform powerful counterfactual simulations. For example, it can predict how that customer's consumption would change if the monthly price were lowered or the data quota were increased. This ability provides a principled, data-driven method for product design and targeted marketing that goes far beyond simple statistical analysis.

Goal-Based Wealth Management and Robo-Advising

- *Problem:* A core challenge in wealth management is optimizing a client's portfolio over a long horizon to meet a specific goal, such as funding retirement. This problem differs from standard portfolio optimization because it involves periodic cash flows (contributions during the accumulation phase, withdrawals during decumulation) in addition to asset rebalancing. The objective is often to reach a target wealth level, a more intuitive goal for retail investors than maximizing a Sharpe ratio or tracking a mean–variance-efficient frontier.
- *IRL–RL approach:* Dixon and Halperin (2020) proposed a two-part framework to tackle this problem.
    1. *G-learner (the RL agent):* First, they define a direct RL agent, the *G-learner*, which solves the goal-based wealth management problem. The G-learner uses a specific quadratic reward function that penalizes underperformance relative to a target wealth path and accounts for transaction costs. By defining actions as absolute dollar changes in asset positions, it handles cash flows naturally and scales to high-dimensional portfolios.

This G-learner serves as a powerful, computationally tractable solver for the direct RL problem.

2. *GIRL (the IRL method):* The second part, *GIRL (G-learning IRL)*, addresses the inverse problem. Many investors cannot explicitly define their reward function parameters (e.g., their exact risk aversion or how they weigh tracking a benchmark versus growth). GIRL takes the observed trading history of an investor (or a human portfolio manager) and infers the most likely reward function parameters that the investor was implicitly optimizing. It assumes the investor behaves like a G-learner and uses maximum likelihood to find the reward parameters that best explain the observed actions.

- *Application to robo-advising:* The combination of these two algorithms creates a powerful tool for robo-advising. GIRL can be used to learn the implicit reward functions (i.e., the investment "styles" and risk preferences) of successful human portfolio managers. This learned "best-in-class" reward function can then be given to the G-learner, which computes a new, enhanced optimal policy. This process creates a system that can learn from human expertise, formalize it, and then use AI to find an even better strategy, providing superior, data-driven recommendations.

Learning Optimal Asset Allocation from Collective Behavior of Fund Managers

- *Problem:* How can we learn from the collective behavior of a group of active fund managers to provide improved asset allocation recommendations? Although individual managers are experts, their decisions can contain noise or suboptimal biases. A method is needed to distill their collective wisdom while filtering out individual errors.
- *IRL–RL approach:* Halperin, Liu, and Zhang (2022) proposed a practical two-step framework that combines IRL and RL to learn from and improve on the investment practices of a group of fund managers.

  Step 1: Infer collective intent (IRL): The framework first takes the historical trading data from a group of fund managers with similar investment mandates (e.g., large-cap growth funds). The historical performance of these funds is used to rank their trajectories. Using the T-REX algorithm, the system learns a single, shared reward function whose parameters are optimized to be consistent with these performance rankings. This step infers the *collective intent* of the group—what objectives, on average, lead to better performance within this peer group.

  Step 2: Optimize policy (RL): The collective reward function learned in the IRL step is then passed to a direct RL agent (the G-learner). This agent solves for the optimal asset allocation policy that maximizes this reward function. Because the reward function is based on the distilled wisdom of the entire group, the resulting policy is often superior to the strategies of the individual managers it learned from.
- *Application as an assistant:* This framework is designed not to replace portfolio managers but to *assist* them. The output of the RL step is a set of recommended asset allocation changes (e.g., reweighting portfolio exposure across industry sectors). Managers can use these recommendations as a data-driven input to refine their own decisions, leveraging the collective intelligence of their peers to improve performance. This demonstrates a practical

human–machine interaction loop, where IRL learns from human experts and RL provides optimized suggestions to them.

High-Frequency Market Making with Imitation Learning

- *Problem:* Traditional HFT models for market making, such as the Avellaneda–Stoikov model, are often calibrated on historical data and make strict assumptions about market dynamics (e.g., stable order flow, specific price processes). These models struggle to adapt when real-world market conditions diverge from these assumptions. Standard RL approaches, on the other hand, can be sample-inefficient and often optimize myopically for single-step actions, which can lead to compounding errors and poor inventory management in HFT.
- *Imitation learning approach (FlowHFT):* To address these challenges, Li, Chen, and Yang (2025) proposed FlowHFT, a novel framework based on imitation learning. Instead of assuming a single expert model is best for all conditions, FlowHFT learns from a diverse set of expert demonstrations. It simulates various market scenarios (e.g., high/low volatility, trending/mean-reverting) and identifies the best-performing traditional model (e.g., AS, GLFT) for each specific scenario.
- *Flow-matching policy:* The core of the framework is a flow-matching policy. This is a sophisticated generative model that learns to map a market state to a sequence of optimal trading actions. It does so by learning a "flow" that transforms a simple noise distribution into the complex distribution of expert actions observed across all market scenarios. This process allows a single, adaptive model to integrate the knowledge of many specialized experts. Crucially, it learns to generate entire action *sequences* over a planning horizon, which inherently considers the near-term consequences of actions and helps mitigate the compounding errors seen in single-step RL.
- *Application:* The trained FlowHFT model can adaptively generate trading decisions suitable for the prevailing market state, effectively leveraging the best strategy from its library of learned experts. Li et al. (2025) showed that their single framework can consistently outperform the best individual expert model in each tested market condition, demonstrating a powerful application of imitation learning to build robust, adaptive HFT agents.

Computational Requirements and Infrastructure: Hardware and Software Ecosystem

Implementing RL for financial applications requires careful consideration of computational infrastructure:

Hardware requirements

- *Development phase:* GPU-enabled workstations (NVIDIA RTX 3090 or better) for deep RL algorithms
- *Production phase:* Low-latency inference servers for real-time decision making
- *Memory requirements:* 32GB+ RAM for experience replay buffers in high-frequency applications

Software stack

- *RL frameworks:*
  - Stable Baselines3: Production-ready implementations of standard algorithms
  - RLlib: Distributed training for large-scale applications
  - TF-Agents/PyTorch RL: For custom algorithm development
- *Market simulators:*
  - ABIDES: Agent-based market simulator for microstructure research
  - FinRL: Integrated environment for financial RL applications
  - Custom simulators using historical tick data

Challenges and Frontiers

Although RL and IRL hold immense promise for finance, practitioners must navigate a set of significant challenges before these methods can be widely and reliably deployed.

Key Challenges

The following are some of the main challenges for RL and IRL:

- *Sample efficiency and data requirements:* RL agents, particularly model-free ones, often learn through extensive trial and error. Financial data, although vast, can be noisy, and the number of truly independent historical scenarios is limited. This situation makes it difficult to train agents that are robust to rare but critical market events, such as financial crises.
- *Nonstationarity of financial markets:* The core assumption of a stationary MDP is often violated in finance. Market dynamics, volatility regimes, and correlation structures evolve over time. A policy learned on historical data may become suboptimal or even detrimental when market conditions change. This dynamic necessitates continuous learning or adaptive models that can detect and adjust to regime shifts.
- *Fidelity of simulation environments:* Training and validating RL agents, especially for high-stakes applications, require a realistic market simulator. For example, for applications for trading in the limit order book, building a simulator that accurately captures market micro-structure, order flow dynamics, latency, and the feedback loop of market impact is an extremely challenging problem in itself. An agent that performs well in a flawed simulation may fail spectacularly in a live market.
- *Reward specification and risk sensitivity:* For direct RL, defining a reward function that perfectly aligns with a long-term financial objective is nontrivial. A myopic reward (e.g., single-period profit) can lead to undesirable behavior, such as excessive risk taking. As discussed, standard RL optimizes for expected returns (risk neutrality), whereas financial applications almost always demand explicit management of risk (e.g., variance, tail risk, drawdown). Risk-sensitive and distributional RL can therefore present a particular interest for financial applications.
- *Interpretability and trust:* Many modern RL agents, especially those using deep neural networks, function as "black boxes." This lack of transparency is a major hurdle for adoption in a highly regulated industry where portfolio managers and risk officers need to understand and justify investment decisions.

The Research Frontiers

Addressing these challenges is the focus of ongoing research. Several exciting frontiers are emerging that are particularly relevant for finance:

- *Model-based RL and learned simulators:* To improve sample efficiency, researchers are developing agents that simultaneously learn a policy and a model of the environment. A learned world model can be used to generate simulated experiences, allowing the agent to "plan" and learn much faster than by relying solely on real market data.
- *Explainable AI (XAI) for RL/IRL:* A critical area of research is the development of methods to make the decisions of RL/IRL agents more transparent. Such techniques as sensitivity analysis and feature attribution can help practitioners understand which market signals are driving an agent's actions, fostering greater trust and facilitating model risk management.
- *Multi-agent RL (MARL):* Financial markets are inherently multi-agent systems. MARL moves beyond the single-agent paradigm to model the strategic interactions between multiple learning agents (e.g., competing high-frequency traders, interacting institutional investors, networks of broker/dealers in OTC markets). Similarly, multi-agent IRL aims to deconvolve the observed market dynamics into the behaviors and objectives of distinct classes of agents, which is a significant step beyond "single representative agent" models.
- *RL with large language models (LLMs):* The intersection of RL and LLMs is a rapidly developing area with significant implications for finance. LLMs can process and synthesize vast amounts of unstructured text data, such as news, filings, and social media, creating a richer, more nuanced state representation for an RL agent. Beyond simply enhancing the state, the methods used to align LLMs with human intent are directly applicable to financial decision making.
  - *Reinforcement learning from human feedback (RLHF):* This is the established technique used to fine-tune models such as ChatGPT. It involves a multistage process: First, a reward model is trained on human preference data (e.g., a human ranks several model-generated responses). Then, this reward model is used to fine-tune the LLM's policy using an RL algorithm (such as PPO). In a financial context, this approach could be adapted to align a trading or allocation agent with the complex, hard-to-specify intuition of an expert portfolio manager. A manager could provide qualitative feedback by ranking several trade proposals generated by the agent, allowing the system to learn the manager's implicit risk preferences and market views without the manager having to articulate an explicit utility function.
  - *Direct policy optimization:* Although powerful, RLHF is a complex and potentially unstable multistage process. Rafailov, Sharma, Mitchell, Ermon, Manning, and Finn (2023) introduced a more elegant and direct method called direct preference optimization (DPO), which bypasses the need for an explicit reward model. It uses a clever mathematical re-parameterization to show that the reward-modeling-plus-RL objective can be optimized directly on the preference data with a single, stable loss function. This approach significantly simplifies the training process. For finance, DPO offers a more robust and efficient way to fine-tune an RL agent's policy based on direct human preference data.
  - *Group preference optimization:* A limitation of both RLHF and DPO is their focus on a single preference provider. In many financial settings, decisions must satisfy multiple stakeholders with potentially conflicting objectives (e.g., a pension fund manager

balancing the needs of different beneficiary groups or a team of traders with diverse market views). Zeng, Zhang, Yang, and Chen (2024) formalized this problem as group preference optimization (GPO), a multi-agent extension of DPO. GPO learns a single policy that represents a social welfare optimum—a Condorcet winner—that is most preferred by the group as a whole. This approach provides a principled framework for building RL agents that can learn from and make decisions for a group of diverse financial experts or stakeholders.

Case Study: Trading and Option Hedging

In this section, we apply RL to building trading and option hedging strategies. We begin by exploring how RL can be used in trading, focusing on the ability of AI to discover optimal strategies despite market frictions, the definition of optimality in finance, and ways to encode risk preferences within the RL framework.

Next, we examine how various reward functions and state representations arise naturally in trading problems. We discuss how to formulate RL problems to reflect real financial objectives, such as maximizing expected utility, managing risk, or hedging derivatives.

We then introduce simple illustrative models—mean-reversion trading and option hedging—that demonstrate the mechanics of RL in finance and reveal key practical challenges. These foundational examples set the stage for more-advanced RL methods, including policy-gradient and deep RL approaches. As a practical complement to the discussion, we provide an open-source code example implementing the mean-reversion trading model, allowing readers to experiment with RL in a real trading context.

By the end of the case study, you will have a practical understanding of how to frame trading problems within the RL paradigm, design reward and state structures aligned with investment objectives, and interpret the results and limitations of RL-based trading strategies. This foundation prepares you for further study of deep RL, risk-sensitive methods, and real-world deployment in quantitative finance.

Defining Optimality and the Role of the Reward Function in Trading

We begin this section by discussing some worthwhile questions that motivated the first investigations into the application of RL to optimal trading strategies.

Question 1. *Can an artificial intelligence autonomously identify an optimal dynamic trading strategy, accounting for transaction costs, without prior knowledge of the strategy's structure?*

AlphaGo Zero (Silver, Schrittwieser, Simonyan, Antonoglou, Huang, Guez, Hubert, et al. 2017) is a historically important RL system that learned to play with "zero" human guidance, given only the rules of the game and the chance to play against a simulator. Question 1 asks, What are the various financial analogues of AlphaGo Zero—where an RL agent learns trading or investment strategies from first principles, with minimal human guidance?

Question 2. *In the context of Question 1, how should we define an optimal strategy? Is optimality inherently subjective, or can we rigorously quantify the strategy that a rational decision maker would employ?*

In finance, we define a strategy as optimal if it maximizes the expected utility of terminal wealth. This notion is grounded in the seminal work of von Neumann and Morgenstern (1944), who showed that if a decision maker (a) faces uncertain (probabilistic) outcomes and (b) has preferences satisfying four axioms of rational behavior, then the decision maker will behave as if maximizing the expected value of a utility function u, defined over possible outcomes. For further details, see Benveniste, Kolm, and Ritter (2024). When applied to trading or investment management, the relevant outcome is typically terminal wealth w_T, so the rational agent's objective becomes

$$E[u(w_T)]. \tag{1}$$

This principle underpins modern portfolio theory (Markowitz 1952; Merton 1969; Merton 1971) and provides the foundation for quantitative models of optimal investment and trading. We express terminal wealth as the sum of initial wealth w_0 and the cumulative increments in wealth over time:

$$w_T = w_0 + \sum_{t=1}^{T} \delta w \ , \tag{2}$$

where

$$\delta w_t := w_t - w_{t-1} \tag{3}$$

denotes the wealth increment at time t and T is the final time.

Although it might be tempting to maximize $E[w_T]$ directly, doing so ignores risk and can lead to paradoxes or undesirable outcomes. In contrast, maximizing expected utility $E[u(w_T)]$ incorporates risk preferences appropriately. Chamberlain (1983) and Benveniste et al. (2024) showed that the equivalence between maximizing expected utility and using mean–variance optimization is much broader than most practitioners realize. The classic mean–variance approach remains theoretically justified across a wide range of realistic return distributions—not just under the normality assumption. Specifically, this result applies whenever the distribution of terminal wealth is mean–variance equivalent (MVE), a broad class that includes all elliptical distributions, such as the normal and multivariate t-distributions, as well as a family of asymmetric distributions with well-defined first and second moments. In such cases, a rational agent's preferences over risky outcomes can be fully characterized by the mean and variance of terminal wealth, and the optimal strategy reduces to maximizing expected return penalized by risk, as in the classical Markowitz mean–variance framework (Markowitz 1952). We clarify those assumptions next.

Assumption 1 (Discreteness). *Trading occurs at discrete times* ($t = 1,\ldots,T$)*, and final wealth is given by Equation* 2.

Assumption 2 (Portfolios). *There exists a set of portfolios* ($h_0,\ldots,h_{T-1}$) *known at* $t = 0$ *such that*

$$\delta w_t = h_{t-1} \cdot r_t, \tag{4}$$

where h_t *is the dollar holdings vector at time* t *and* r_t *is the random vector of asset returns over* $[t-1,t]$.

Assumption 3 (Independence). *If* $t \neq s$*, then* r_t *and* r_s *are independent.*

Assumption 4 (Mean–Variance Equivalence). *For each t, the distribution of r_t is mean–variance equivalent.*

Assumption 5 (Utility). *The utility function is increasing and concave, and it has continuous derivatives up to second order.*

The assumption that the distribution is MVE is perhaps the most interesting one. Although the mean–variance optimization framework dates back to Markowitz's pioneering work in the early 1950s, the precise conditions under which mean–variance optimization is truly equivalent to expected utility maximization were only recently clarified by Benveniste et al. (2024). This distinction is central in our context, because a von Neumann–Morgenstern rational investor aims to maximize $E[u(w_T)]$, whereas the reward signal in Equation 11 takes a mean–variance form. Under the MVE condition, these objectives are, in fact, equivalent. Specifically, there exists some constant

$$\kappa > 0, \tag{5}$$

that depends on initial wealth w_0 and the investor's utility function, such that maximizing

$$E[u(w_T)] \tag{6}$$

is equivalent to maximizing

$$E[w_T] - \frac{1}{2}\kappa \mathbb{V}[w_T]. \tag{7}$$

In the following, we focus on

$$\text{maximize}\left\{\mathbb{E}[w_T] - \frac{\kappa}{2}\mathbb{V}[w_T]\right\}. \tag{8}$$

The first example of a reward signal appropriate for mean–variance utility in the context of RL was provided by Ritter (2017), which we now describe (see also Ritter 2018). Suppose we could invent some definition of reward (R_t) such that

$$E[w_T] - \frac{\kappa}{2}V[w_T] \approx \sum_{t=1}^{T} R\ . \tag{9}$$

Then, the optimization problem (Equation 8) looks like the kind of "cumulative reward over time" problem that is typical in RL. RL searches for policies that maximize

$$E[G_t] = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots], \tag{10}$$

which by Equation 9 would then maximize expected utility as long as $\gamma \approx 1$.

We consider the reward function proposed by Ritter (2017):

$$w_t := \delta_t - \frac{\kappa}{2}\left(\delta w_t - \hat{\mu}\right)^2, \tag{11}$$

where $\hat{\mu}$ is an estimate of a parameter representing the mean wealth increment over one period, ($\mu := E[\delta w_t]$). Then, averaging over T periods yields

$$\frac{1}{T}\sum_{t=1}^{T} w_t = \underbrace{\frac{1}{T}\sum_{t=1}^{T}\delta_t}_{\to \mathbb{E}[\delta w_t]} - \frac{\kappa}{2}\underbrace{\frac{1}{T}\sum_{t=1}^{T}\left(\delta w_t - \hat{\mu}\right)^2}_{\to \mathbb{V}[\delta w_t]}, \tag{12}$$

where for large T, the first term approaches the sample mean and the second approaches the sample variance of the wealth increments. Thus, with the reward function (Equation 11), maximizing cumulative reward implies maximizing the mean–variance form of expected utility.

State Variables for Trading Problems

The state variable (s_t) is a data structure that, simply put, should contain everything the agent needs to make an informed trading decision and nothing else. Variables that are natural candidates for inclusion in the state are

- the current position or holding in the asset,
- the values of any signals that are believed to be predictive,
- the current state of the market, including current price and any relevant microstructure/limit order book details, and
- if the portfolio includes contracts with expirations, such as futures or options, the time remaining until expiry.

In trading problems, the most natural choice for an action is the number of shares to trade, δn_t. This choice identifies the action space ($\mathcal{A} \subset Z$). When market microstructure effects are significant, the action space may need to be expanded. For example, the agent could decide which execution algorithm to use, choose between crossing the spread or submitting a passive order, or set the target participation rate. If one of the assets is an option, the agent may also have the ability to take additional actions, such as early exercise.

Trading Examples

In this section, we present two simple examples—mean-reversion trading and option hedging—that highlight key ideas and challenges in applying RL in trading.

Mean Reversion

We assume there exists a tradable asset with a strictly positive price process $p_t > 0$. This "asset" could be a portfolio of other assets, such as an exchange-traded fund or a hedged relative value trade. Further suppose that there exists an "equilibrium price" p_e such that $x_t = \log(p_t/p_e)$ has dynamics

$$dx_t = -\lambda x_t + \sigma\xi_t, \tag{13}$$

where $\xi_t \sim N(0,1)$ and ξ_t, ξ_s are independent when $t \neq s$. This means that p_t tends to revert to its long-run equilibrium level, p_e, with mean-reversion rate λ.

These assumptions describe a scenario that is close to an arbitrage: Positions established far from equilibrium have a very small probability of loss and highly asymmetric gain/loss profiles. Initially, we do not allow the agent to know anything about the dynamics. That is, the agent does not know λ, σ, or even that some dynamics of Equation 13 are valid. The agent also does not know there are trading costs. We impose a spread cost of one tick per trade. If the bid–offer

spread were equal to two ticks, then this proportional cost would correspond to the slippage incurred by an aggressive fill that crosses the spread to execute. Hence,

$$\text{SpreadCost}(\delta n) = \text{TickSize} \cdot |\delta n|. \tag{14}$$

Additionally, we assume there is a temporary price impact with a linear functional form: Each round lot traded is assumed to move the price one tick, hence leading to a dollar cost

$$\delta n_t \times \text{TickSize/LotSize} \tag{15}$$

per share traded, for a total dollar cost for all shares

$$\text{ImpactCost}(\delta n) = (\delta n)^2 \times \text{TickSize/LotSize}. \tag{16}$$

Together, the total trading cost is given by

$$\text{Cost}(\delta n) = \text{Multiplier} \times [\text{SpreadCost}(\delta n) + \text{ImpactCost}(\delta n)]. \tag{17}$$

This functional form matches the *t*-cost assumptions of Almgren and Chriss (1999). In fact, the Almgren–Chriss model of optimal execution can be learned by an RL agent, similar to the Ornstein–Uhlenbeck trading model discussed here.

The state variable at time *t*,

$$s_t = (p_t, n_{t-1}), \tag{18}$$

consists of the current asset price, p_t, and the agent's position in shares, n_{t-1}, at the start of the period.

As a proof of concept and in the spirit of exhausting the simplest method first, Ritter (2017) trained a tabular Q-learner with $n_{train} = 10^7$ training steps and then evaluated the system on 5,000 new samples of the stochastic process; see **Exhibit 1**.

These results look encouraging. We simulated a simple dynamic wherein we know there is an arbitrage, analogous to a game where it is actually possible to win. The machine then learns to play this game and develops a profitable strategy. How well did the agent really learn? To find the answer, we examine a cross-section of its learned action-value function $\hat{q}$, a diagnostic that reveals not only whether the agent profits but also how it chooses actions across different states. This deeper analysis helps us move beyond performance to evaluate the quality and reliability of the learned policy (see **Exhibit 2**).

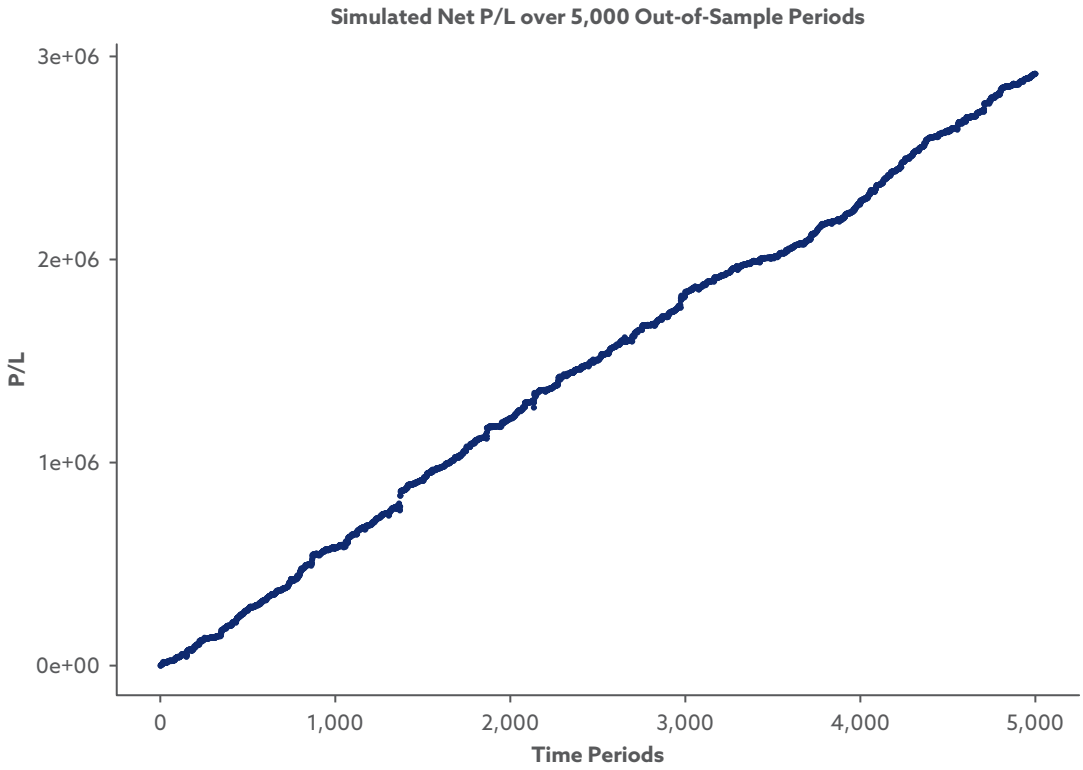
The main weakness of the tabular method is that it estimates each element

$$\hat{q}(s,a) \tag{19}$$

in isolation, without any smoothing across neighboring states or any inherent continuity. In this example, the optimal action choice exhibits a natural monotonicity, which we now describe intuitively.

If the current holding is $h = 0$ and for some price $p < p_e$ the optimal action is to buy 100 shares, then for any lower price $p' < p$, the optimal action should be to buy at least 100 shares. As Exhibit 2 illustrates, however, for large prices, the tabular value function often oscillates between different decisions, violating this monotonicity. This behavior reflects estimation

Exhibit 1. Tabular Q-Learner with 10^7 Training Steps, Evaluated on 5,000 New Samples



Notes: This exhibit uses simulated data. P/L stands for profit/loss.

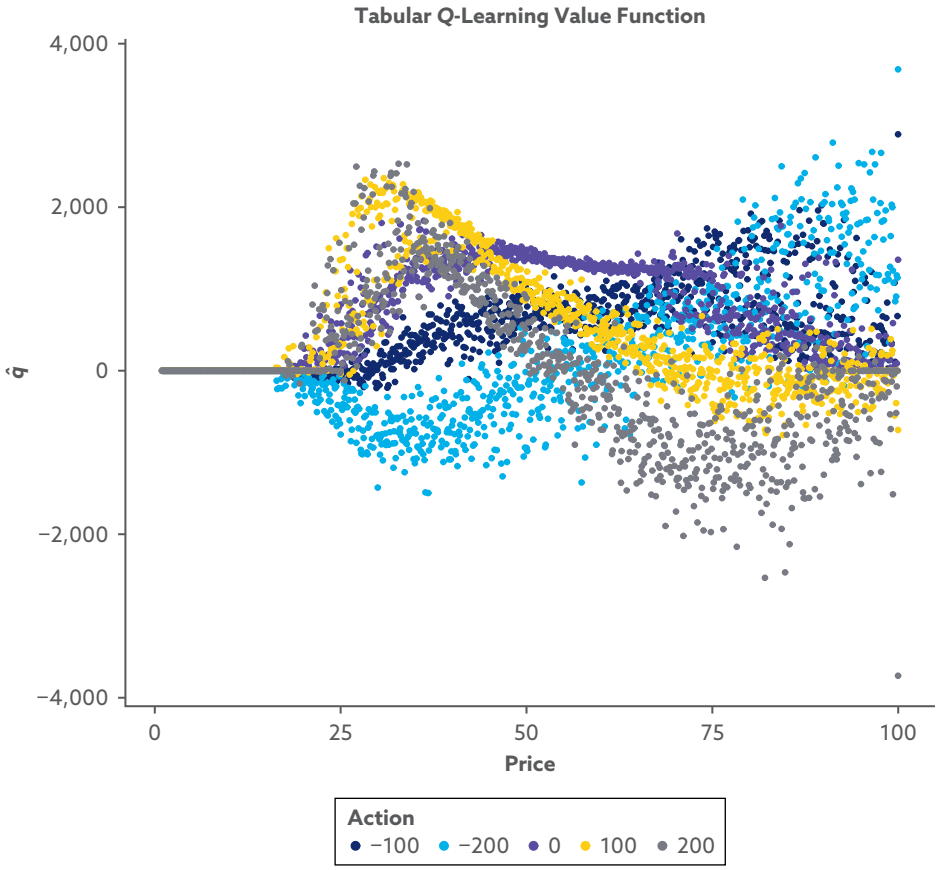
error and incomplete convergence, even after millions of iterations. The tabular value function also tends to collapse to a trivial form in the far-left tail because those states are rarely visited during training.

All these issues stem from the same fundamental limitation: the use of a finite, tabular state space. Methods that rely on discretizing the state space inevitably break down as dimensionality increases. For example, if the state vector included 10 variables, each taking 100 possible values, the number of parameters to estimate would reach into the billions. The solution to this curse of dimensionality is to move beyond tabular methods and adopt continuous state spaces with function approximation in RL. In the next section, we illustrate this approach with a concrete example.

Option Hedging with Transaction Costs

We now consider another problem of interest to traders: hedging an option position. For clarity, we focus on the simplest possible case—a European call option with strike K and expiry T on a non-dividend-paying stock. We set the strike and maturity as exogenous constants and, for

Exhibit 2. Value Function $p \to \hat{q}\,[(0, p), a]$, Where $\hat{q}$ Is Estimated by the Tabular Method



simplicity, assume a risk-free rate of zero. We train the agent to hedge this specific option with its given strike and maturity, rather than teaching it to hedge options with arbitrary parameters.

To hedge a European option, the state must at least contain the current price, S_t, of the underlying; the time remaining to expiry,

$$\tau := T - t > 0; \tag{20}$$

and our current position of n shares in the underlier. The state is thus naturally an element of

$$\mathcal{S} := \mathbb{R}_+^2 \times \mathbb{Z} = \{(S, \tau, n) \mid \quad > 0, \tau > 0, n \in \mathbb{Z}\}. \tag{21}$$

We emphasize that the state does not need to contain the option Greeks, because these quantities are nonlinear functions of the state variables already available to the agent. We expect agents to learn such nonlinear functions on their own as needed. This approach has the advantage of not requiring any model-based calculations. First, we consider a "frictionless" world (i.e., without trading costs) and ask whether it is possible for a machine to learn what we teach

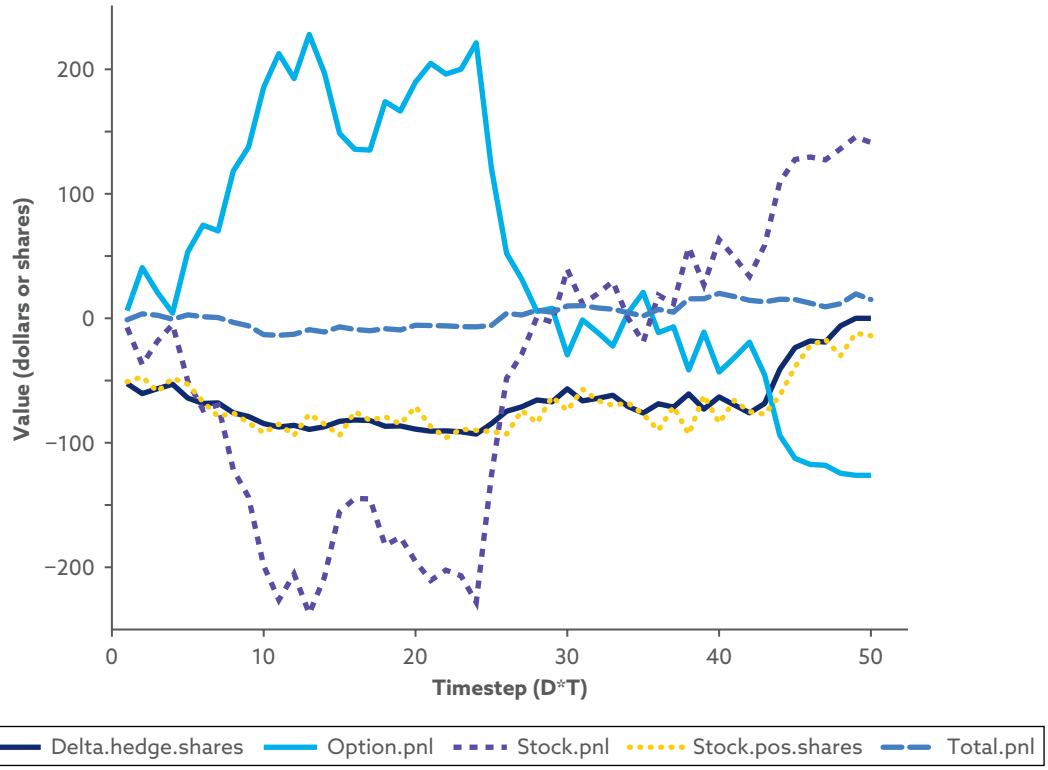
students in their first semester of business school: the dynamic replicating portfolio strategy. Unlike our students, the machine can learn only by observing and interacting with the environment, without any explicit instruction.

The RL agent is initially at a disadvantage. Recall that it does not know any of the following pertinent pieces of information:

- The strike price, K
- The fact that the stock price process is a geometric Brownian motion
- The volatility of the price process
- The Black–Scholes–Merton formula
- The payoff function, $(S - K)_+$, at maturity
- Any of the Greeks

It must infer the relevant information from the state variables, insofar as it affects the value function, by interacting with the environment. We present the results in **Exhibit 3**. Notably, the cumulative stock and option P&L largely offset each other, resulting in a total P&L with relatively

Exhibit 3. Out-of-Sample Simulation of a Trained Agent



Note: We depict cumulative stock, option, and total P&L; the RL agent's position in shares (Stock.pos.shares); and $-100 \cdot \Delta$ (Delta.hedge.shares).

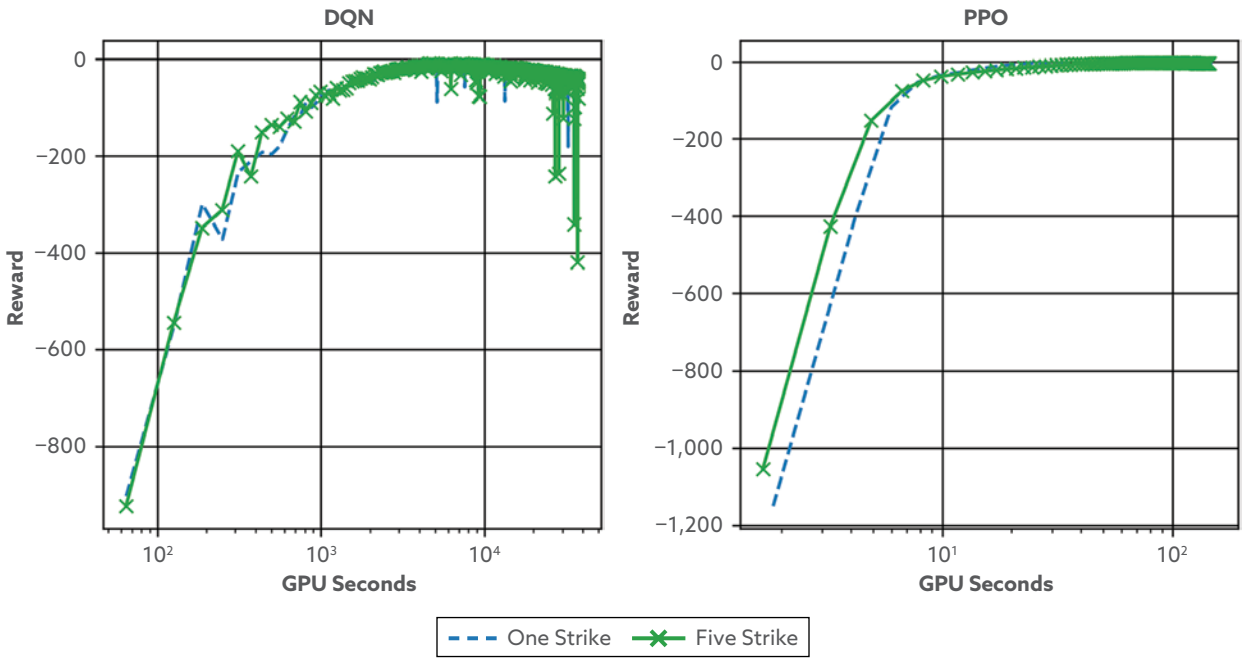
low variance. We also see that the RL agent's position closely tracks the delta position, despite having no direct access to it. For a more detailed discussion, see Kolm and Ritter (2019).

Policy Gradient Methods

When working in continuous state spaces, policy gradient methods frequently outperform value-based approaches. Du, Jin, Kolm, Ritter, Wang, and Zhang (2020) were the first to apply proximal policy optimization (PPO) to the problem of option hedging with transaction costs. The authors developed models that replicate options across a wide range of strikes while incorporating discrete trading, round lots, and nonlinear transaction costs. These models use deep RL techniques—including deep Q-learning and PPO—and are built to interface easily with any option pricing and simulation library, enabling users to train them on arbitrary option portfolios without further modification. Our empirical studies demonstrate that deep RL models can match or surpass traditional delta hedging, with PPO delivering superior results in terms of profit and loss, training speed, and data efficiency; see **Exhibit 4**.

Note that policy gradient methods, including PPO and asynchronous actor-critic (A2C), can be applied to the mean-reversion trading model discussed earlier and, in practice, yield significantly better results than the simple tabular Q-learning approach used by Ritter (2017).

Exhibit 4. Average Reward vs. GPU Seconds for the DQN and PPO Agents in the One and Five Strike Scenarios



Computational Implementations

The state of the art for computational AI is evolving at an extraordinary pace. Nevertheless, we believe that providing readers with a short, self-contained program that implements the mean-reversion example we showed will contribute to practical understanding and further progress in this area.

The code for this guide was developed with the goal of prioritizing simplicity, brevity, and reproducibility, using only open-source frameworks. Computational efficiency was intentionally not the aim, in the spirit of Knuth's well-known advice that premature optimization is the root of all (programming) evil.

For our implementation, we chose Stable Baselines 3 (SB3), a set of reliable, well-tested, and standardized RL algorithms built on PyTorch. SB3 offers a clean, modular codebase and a consistent API for training, evaluating, and deploying RL agents. It is compatible with gymnasium environments. These features make SB3 an ideal computational engine for our examples.

For clarity, our code sample consists of just two short files: reversion.py defines the custom environment for the mean-reversion trading problem described previously, and main.py is a main script that orchestrates the training and performance evaluation. The primary outputs of the main script are a set of image files that summarize the learning curve, the behavior of the trained agent, and the key performance metrics, such as P/L and the Sharpe ratio. The trained model is also saved as a file that can be reloaded for later use. Setting the Boolean variable discrete to true switches the model to a discrete action space; by default, the model operates with a continuous action space.

In summary, machines can identify arbitrage opportunities in data when such opportunities exist and can learn to optimize over long horizons in the presence of transaction costs. Despite these advances, the field of RL in finance remains in its early stages, and we are still far from a fully autonomous "Skynet" for trading. In practice, working in continuous state spaces is most effective because most optimal value functions in finance are continuous—and often monotonic—functions of the state, with properties grounded in economic intuition. Many optimal value functions are smooth or piecewise smooth, as seen in such classic problems as the linear-quadratic regulator. These same RL methods can be applied to derivative hedging in illiquid markets, where trading costs play a crucial role. RL agents can learn to price and hedge derivatives in environments where perfect replication is either impossible or prohibitively expensive.

Conclusion and Outlook

Reinforcement learning and inverse reinforcement learning mark a paradigm shift in quantitative finance, transitioning from static predictive models to dynamic, adaptive decision-making systems. RL excels in optimizing sequential decisions under uncertainty, making it ideal for such tasks as asset allocation, trade execution, and risk management. IRL, although less mature, enables the inference of preferences or objectives from observed behaviors, offering novel insights into individual trader strategies or market dynamics.

For practitioners aiming to adopt these technologies, a disciplined and strategic approach is essential. The path to real-world deployment is complex, but the benefits—improved performance and deeper market understanding—are significant. We provide the following key recommendations:

- *Target well-defined problems:* Choose applications with clearly defined states, actions, and rewards to ensure effective RL implementation. Identify problems aligned with your specific objectives, where the decision-making process can be modeled and optimized systematically.
- *Integrate risk management early:* Incorporate risk considerations directly into the reward function, using risk-averse or distributional RL frameworks. Doing so ensures alignment with the desired risk–return profile, avoiding the pitfalls of retrofitting risk controls.
- *Prioritize robust simulation:* Success hinges on high-quality data and realistic simulation environments. Rigorous backtesting, out-of-sample validation, and stress testing across diverse market scenarios are critical before deployment.
- *Leverage hybrid intelligence:* Combine RL and IRL with human expertise for optimal results. Use IRL to codify successful human strategies and RL to refine them, creating systems that augment human decision making while addressing ethical considerations.
- *Address practical constraints:* Design frameworks to meet computational, latency, and regulatory requirements. Interpretability is essential in regulated environments and should be a core design principle.

The future of quantitative finance lies in adaptive, real-time learning systems. RL and IRL provide the theoretical and practical tools to realize this vision. Despite challenges, their ability to enhance performance and uncover new market insights makes them critical for practitioners to master. As computational power grows and algorithms advance, RL and IRL are poised to become indispensable components of the quantitative finance toolkit, driving widespread adoption across the industry.

References

Almgren, Robert, and Neil Chriss. 1999. "Value under Liquidation." *Risk* (December): 61–63.

Benveniste, Jerome, Petter N. Kolm, and Gordon Ritter. 2024. "Untangling Universality and Dispelling Myths in Mean–Variance Optimization." *Journal of Portfolio Management* 50 (8): 90–116. doi:10.3905/jpm.2024.50.8.090.

Black, Fischer, and Myron Scholes. 1973. "The Pricing of Options and Corporate Liabilities." *Journal of Political Economy* 81 (3): 637–54. doi:10.1086/260062.

Brown, Daniel S., Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. 2019. "Extrapolating beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations." In *International Conference on Machine Learning*, 783–792. doi:10.48550/arXiv.1904.06387.

Buehler, Hans, Lukas Gonon, Josef Teichmann, and Ben Wood. 2019. "Deep Hedging." *Quantitative Finance* 19 (8): 1271–91. doi:10.1080/14697688.2019.1571683.

Chamberlain, Gary. 1983. "A Characterization of the Distributions That Imply Mean–Variance Utility Functions." *Journal of Economic Theory* 29 (1): 185–201. doi:10.1016/0022-0531(83)90129-1.

Dixon, Matthew Francis, and Igor Halperin. 2020. "G-Learner and GIRL: Goal Based Wealth Management with Reinforcement Learning." Working paper (25 February). doi:10.2139/ssrn.3543852.

Dixon, Matthew F., Igor Halperin, and Paul Bilokon. 2020. *Machine Learning in Finance: From Theory to Practice*. Cham, Switzerland: Springer. doi:10.1007/978-3-030-41068-1.

Du, Jiayi, Muyang Jin, Petter N. Kolm, Gordon Ritter, Yixuan Wang, and Bofei Zhang. 2020. "Deep Reinforcement Learning for Option Replication and Hedging." *Journal of Financial Data Science* 2 (4): 44–57. doi:10.3905/jfds.2020.1.045.

Ernst, Damien, Pierre Geurts, and Louis Wehenkel. 2005. "Tree-Based Batch Mode Reinforcement Learning." *Journal of Machine Learning Research* 6 (18): 503–56.

Finn, Chelsea, Paul Christiano, Pieter Abbeel, and Sergey Levine. 2016. "A Connection Between Generative Adversarial Networks, Inverse Reinforcement Learning, and Energy-Based Models." Working paper (25 November). doi:10.48550/arXiv.1611.03852.

Halperin, Igor. 2020. "QLBS: Q-Learner in the Black-Scholes(-Merton) Worlds." *Journal of Derivatives* 28 (1): 99–122. doi:10.3905/jod.2020.1.108.

Halperin, Igor. 2024. "Distributional Offline Continuous-Time Reinforcement Learning with Neural Physics-Informed PDEs (SciPhy RL for DOCTR-L)." *Neural Computing & Applications* 36: 4643–59. doi:10.1007/s00521-023-09300-7.

Halperin, Igor, Jiayo Liu, and Xiao Zhang. 2022. "Asset Allocation with Inverse Reinforcement Learning." *Risk.net* (30 November). www.risk.net/cutting-edge/7955333/asset-allocation-with-inverse-reinforcement-learning.

Ho, Jonathan, and Stefano Ermon. 2016. "Generative Adversarial Imitation Learning." In *Advances in Neural Information Processing Systems 29*. doi:10.48550/arXiv.1606.03476.

Kolm, Petter N., and Gordon Ritter. 2019. "Dynamic Replication and Hedging: A Reinforcement Learning Approach." *Journal of Financial Data Science* 1 (1): 159–71. doi:10.3905/jfds.2019.1.1.159.

Levine, Sergey, Zoran Popović, and Vladlen Koltun. 2011. "Nonlinear Inverse Reinforcement Learning with Gaussian Processes." In *NIPS'11: Proceedings of the 25th International Conference on Neural Information Processing Systems*, 19–27. https://dl.acm.org/doi/abs/10.5555/2986459.2986462.

Li, Yang, Zhi Chen, and Steve Yang. 2025. "FlowHFT: Flow Policy Induced Optimal High-Frequency Trading under Diverse Market Conditions." Working paper (22 May). doi:10.48550/arXiv.2505.05784.

Markowitz, Harry. 1952. "Portfolio Selection." *Journal of Finance* 7 (1): 77–91.

Merton, Robert C. 1969. "Lifetime Portfolio Selection under Uncertainty: The Continuous-Time Case." *Review of Economics and Statistics* 51 (3): 247–57. doi:10.2307/1926560.

Merton, Robert C. 1971. "Optimum Consumption and Portfolio Rules in a Continuous-Time Model." *Journal of Economic Theory* 3 (4): 373–413. doi:10.1016/0022-0531(71)90038-X.

Merton, Robert C. 1974. "On the Pricing of Corporate Debt: The Risk Structure of Interest Rates." *Journal of Finance* 29 (2): 449–70. doi:10.2307/2978814.

Ng, Andrew Y., Daishi Harada, and Stuart J. Russell. 1999. "Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping." In *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, 278–87.

Noguer i Alonso, Miquel, and Sonam Srivastava. 2020. "Deep Reinforcement Learning for Asset Allocation in US Equities." Working paper (9 October). doi:10.2139/ssrn.3711487.

Pricope, Tidor-Vlad. 2021. "Deep Reinforcement Learning in Quantitative Algorithmic Trading: A Review." Working paper (31 May). doi:10.48550/arXiv.2106.00123.

Rafailov, Rafael, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. "Direct Preference Optimization: Your Language Model Is Secretly a Reward Model." Working paper (13 December). doi:10.48550/arXiv.2305.18290.

Ritter, Gordon. 2017. "Machine Learning for Trading." Working paper (8 August). doi:10.2139/ssrn.3015609.

Ritter, Gordon. 2018. "Reinforcement Learning in Finance." In *Big Data and Machine Learning in Quantitative Investment*, edited by Tony Guida, 225–250. Hoboken, NJ: John Wiley & Sons.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, et al. 2017. "Mastering the Game of Go without Human Knowledge." *Nature* 550 (7676): 354–59. doi:10.1038/nature24270.

Spooner, Thomas, John Fearnley, Rahul Savani, and Andreas Koukorinis. 2018. "Market Making via Reinforcement Learning." Working paper (11 April). doi:10.48550/arXiv.1804.04216.

Sutton, Richard S., and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press.

von Neumann, John, and Oskar Morgenstern. 1944. *Theory of Games and Economic Behavior*. Princeton, NJ: Princeton University Press.

Yang, Steve Y., Qifeng Qiao, Peter A. Beling, William T. Scherer, and Andrei A. Kirilenko. 2015. "Gaussian Process-Based Algorithmic Trading Strategy Identification." *Quantitative Finance* 15 (10): 1683–703. doi:10.1080/14697688.2015.1011684.

Yang, Steve Y., Yangyang Yu, and Saud Almahdi. 2018. "An Investor Sentiment Reward-Based Trading System Using Gaussian Inverse Reinforcement Learning Algorithm." *Expert Systems with Applications* 114 (30 December): 388–401. doi:10.1016/j.eswa.2018.07.056.

Zeng, J.-C., Z. Zhang, Y. Yang, and J. Chen. 2024. "Group Preference Optimization: A Multi-Agent Formalization of RLHF for Scenarios with Multiple Stakeholders."

Ziebart, Brian D., Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. 2008. "Maximum Entropy Inverse Reinforcement Learning." In *Twenty-Third AAAI Conference on Artificial Intelligence*, 1433–38.