Dynamic Replication and Hedging Using Reinforcement Learning

Eidelsberg Maxime
Shen Simon

15 May 2025

1 Introduction

Dynamic option hedging constitutes a fundamental problem in finance since the seminal work of Black-Scholes-Merton (BSM). However, traditional assumptions of perfect markets—absence of frictions and continuous trading—prove unrealistic in practice. The studied paper proposes an innovative approach using **reinforcement learning (RL)** to optimize hedging strategies under more realistic conditions incorporating discrete time, transaction costs, and liquidity constraints.

The major contributions of this research revolve around three main axes. First, the authors develop a generic method to train autonomous agents to find the right balance between cost minimization and variance reduction. Second, their approach applies RL to continuous state spaces, thereby circumventing the dreaded curse of dimensionality. Finally, empirical results demonstrate a significant reduction in transaction costs without notable degradation in volatility.

2 Reinforcement Learning (RL)

Reinforcement learning represents a machine learning method where an agent iteratively interacts with an environment to maximize cumulative reward. Unlike traditional supervised approaches, this method does not require pre-labeled data.

From a mathematical perspective, RL is a method for solving multi-period optimal control problems. The agent's policy generally involves explicitly maximizing the action-value function for the current state. Following Sutton and Barto's (2018) notation, the sequence of rewards received after time step t is denoted $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$. The agent's objective is to maximize expected cumulative reward.

The value function, denoted G_t where γ represents the discount factor, estimates the long-term expected gain. This allows the agent to consider potential gains over a longer horizon with a "penalty," due to the discount factor, for moving to a more distant state.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum \gamma^k R_{t+k+1} \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor.

A policy, denoted π, is a way to choose an action a, conditional on the current state s. The policy π, whether stochastic or deterministic, formalizes the decision rules guiding action selection. There are primarily two types of value functions :

— The *action-value function* expresses the value of starting from state s, taking an arbitrary action a, then following policy π thereafter :

$$q_\pi(s, a) := E_\pi[G_t | S_t = s, A_t = a] \tag{2}$$

— The *state-value function* is the action-value function where the first action also comes from policy π :

$$v_\pi(s) = E_\pi[G_t | S_t = s] = q_\pi(s, \pi(s)) \tag{3}$$

An optimal policy is defined as being at least as good as any other policy. All optimal policies share the same optimal state-value and action-value functions :

$$v_*(s) = \sup_\pi v_\pi(s) \tag{4}$$

$$q_*(s, a) = \sup_\pi q_\pi(s, a) \tag{5}$$

The optimal value functions satisfy the Bellman equations :

$$v_*(s) = \max_a \sum_{s', r} p(s', r | s, a)[r + \gamma v_*(s')] \tag{6}$$

$$q_*(s, a) = \sum_{s', r} p(s', r | s, a)[r + \gamma \max_{a'} q_*(s', a')] \tag{7}$$

where $p(s', r|s, a)$denotes the transition probability to state s' with reward r, given state s and action a. The Bellman equations form the mathematical foundation enabling iterative optimization of value functions.

The state-value function $v_*(s)$ has a natural interpretation in derivative pricing theory. In continuous time and frictionless markets, the optimal value function of the dynamic replication strategy equals the arbitrage-free price of the option. Thus, it is natural that RL, which organizes the search for optimal policies through value functions, applies to derivative pricing and hedging.

3 Methodology

3.1 Problem Formulation

The study focuses on the case of a European option to be hedged in a simulated environment. The methodological framework relies on several key elements. The system state s_t integrates three essential variables : the current underlying price S_t, time remaining to maturity τ, and current stock position n_t. Possible actions are limited to buying or selling an integer number of shares to replicate the option.

The reward function, designed to reflect performance and stability objectives, linearly combines instantaneous P&L and its variance according to the equation :

$$R_t = \delta w_t - \frac{K}{2}(\delta w_t)^2$$

where parameter K quantifies the agent's risk aversion.

3.2 Training via Simulation

The RL agent's training process relies on a carefully designed simulated environment to reproduce realistic market conditions. This mechanism generates sufficient data volume to approximate the optimal value functionq_*, while efficiently managing the exploration-exploitation dilemma.

3.2.1 Data Generator

The underlying price simulation follows a geometric Brownian motion (GBM) with specific temporal discretization : five periods per day ($D = 5$), leading to fifty time steps ($T \cdot D = 50$) for a ten-day option ($T = 10$).Fixed parameters include initial price $S_0 = 100$, daily volatility $\sigma = 0.01$, and risk aversion coefficient $K = 0.1$, with the practical constraint of integer number of share trading.

Each simulation episode corresponds to a complete price trajectory until option maturity. During this episode, the agent continuously observes the current system state $s_t = (S_t, \tau, n_t)$, incorporating the underlying price, remaining time, and stock position. The simulator then calculates the instantaneous reward according to the aforementioned formula, including both P&L and transaction costs.

3.2.2 Exploration vs Exploitation

To avoid premature convergence to suboptimal solutions, the algorithm implements a balanced strategy alternating exploration and exploitation. The ϵ-greedy policy, materialized by the following equation, manages this trade-off :

$$\pi_{\epsilon\text{-greedy}}(s) = \begin{cases} \text{Random action} & \text{with probability } \epsilon \\ \arg\max_a \hat{q}(s,a) & \text{with probability } 1-\epsilon \end{cases}$$

Initially, parameter ϵ s set high (typically 1) to favor extensive state space exploration. Over iterations, this value gradually decreases (to about 0.1) to give more weight to exploiting acquired knowledge.

3.2.3 Learning Process

Approximation of the optimal value function q_* is achieved via sophisticated nonlinear regression techniques. The process is organized in three main steps. First, generating batches each containing 750,000 (X_i, Y_i) pairs, where $X_i = (s_i, a_i)$ represents an observed state-action pair and $Y_i = r_{i+1} + \gamma \cdot \hat{q}(s_{i+1}, a_{i+1})$ constitutes the SARSA target.

Next, a regression model is trained on each batch to refine the $\hat{q}$ estimate. This iterative process is repeated over five successive batches ($B = 5$), until performance stabilization indicates algorithm convergence.

3.2.4 Technical Implementation

Practically, all experiments were conducted on standard hardware using a single CPU, with average training time of one hour. This implementation also shows excellent scalability, allowing easy integration of additional constraints or adaptation to time-dependent cost models.dépendants du temps.

4 Results

Performance analysis reveals particularly convincing results for the reinforcement learning approach.

EXHIBIT 1

Out-of-Sample Simulation of a Trained RL Agent

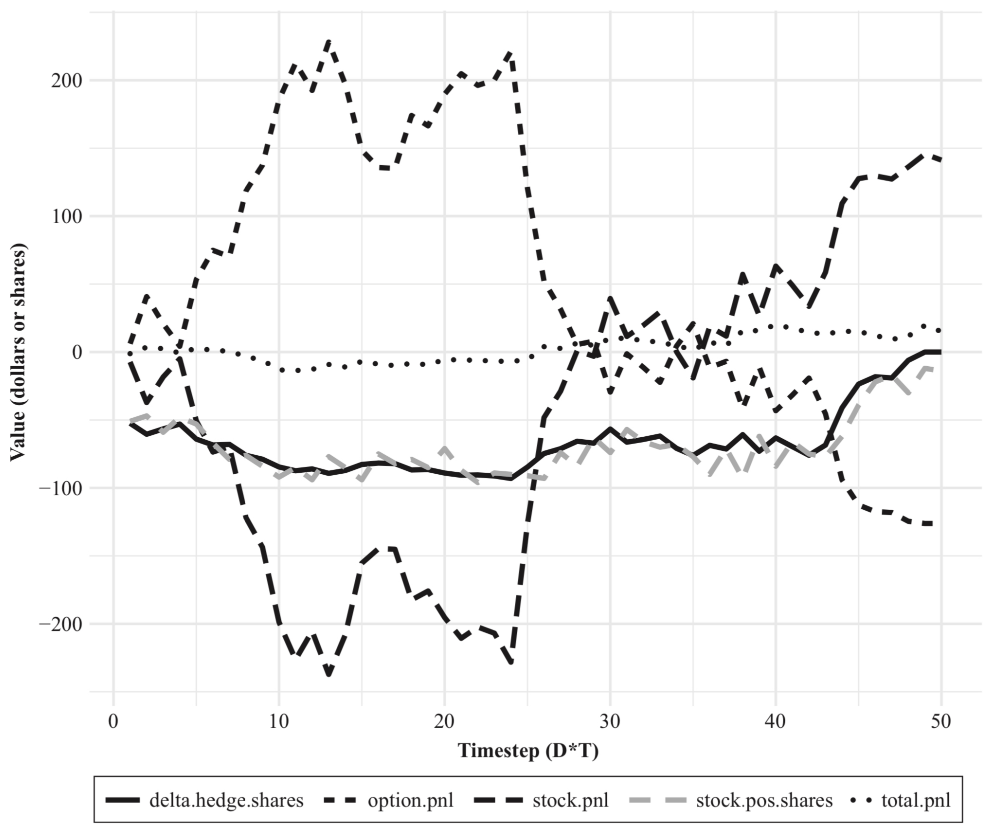

FIGURE 1 – RL agent performance in a transaction cost-free environment

Figure 1 illustrates how the agent maintains effective hedging. Total P&L remains remarkably close to zero despite temporal discretization and integer lot trading constraints. The stock position closely follows theoretical delta evolution, demonstrating the agent correctly learned dynamic hedging principles.

Comparison with traditional methods reveals decisive advantages :

EXHIBIT 4

Kernel Density Estimates for Total Cost (left panel) and Volatility of Total P&L (right panel) from N = 10,000 Out-of-Sample Simulations

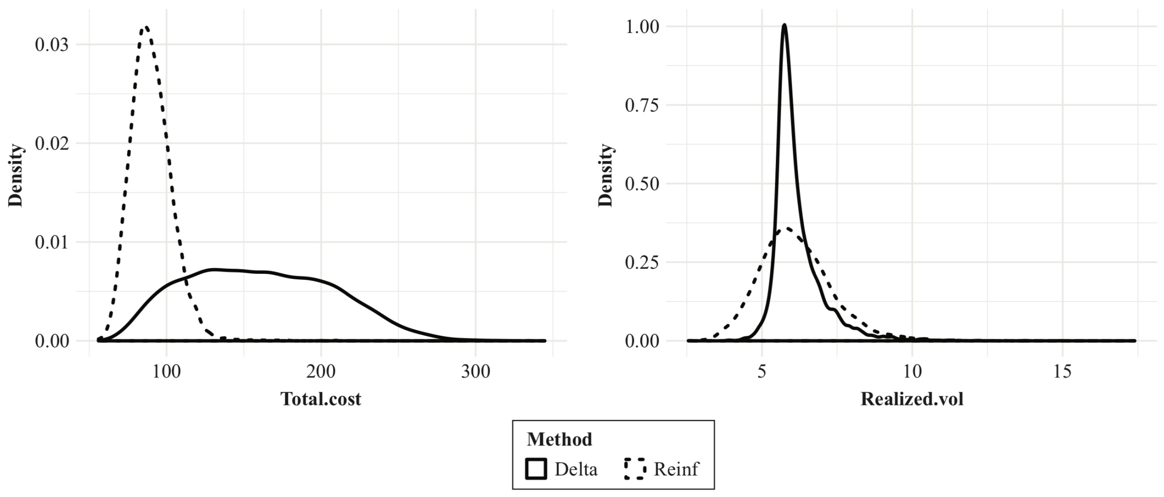

FIGURE 2 – Comparison of cost and volatility distributions

As Figure 2, shows, the RL approach reduces costs without increasing volatility. This substantial saving comes with a distribution more concentrated around the median.

Exhibit 3

Out-of-Sample Simulation of Our Trained RL Agent

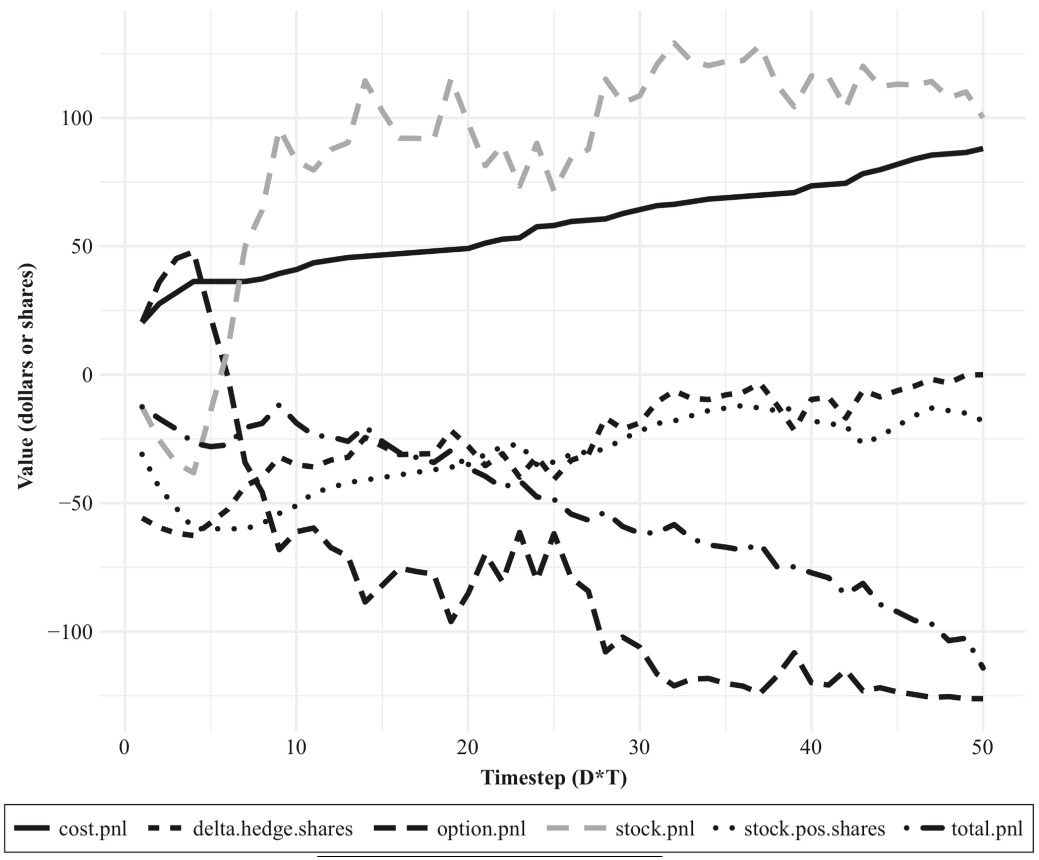

Figure 3 – RL strategy adaptation to transaction costs

Figure 3 demonstrates agent adaptability. Adjustments become less frequent but more strategic, limiting cost accumulation while maintaining effective hedging.

Exhibit 5

Kernel Density Estimates of the t-Statistic of Total P&L for Each of Our Out-of-Sample Simulation Runs and for Both Policies Represented Previously (delta and reinf)

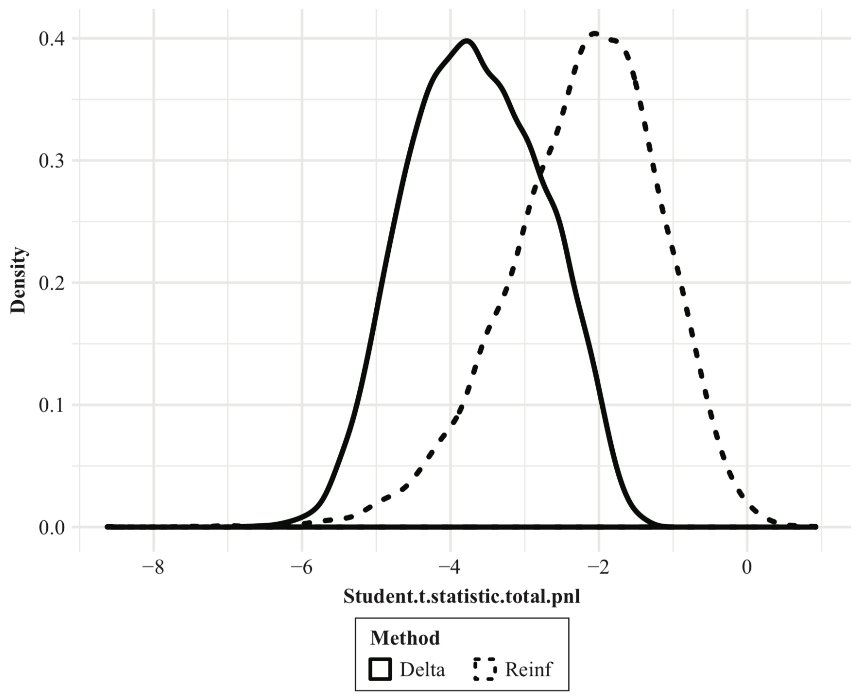

Figure 4 – Statistical robustness of RL strategy

Statistical robustness (Figure 4) translates to P&L t-statistics distribution tightly centered around zero, with fewer extreme values than the delta method.

E X H I B I T 2

Out-of-Sample Simulation of a Baseline RL Agent That Uses Policy Delta or π_{DH}, Defined in Equation 12

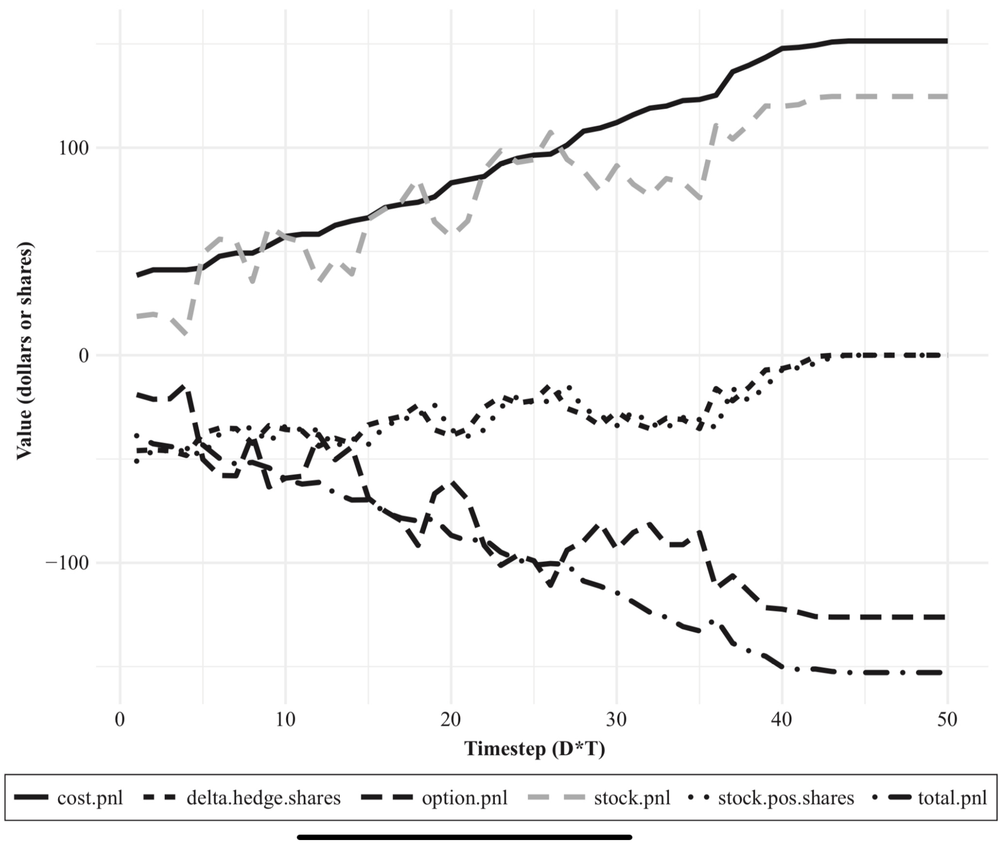

FIGURE 5 – Reference delta method performance

In contrast, Figure 5 shows traditional approach limitations, with excessive costs without variance reduction benefit.

The statistical robustness of the method clearly emerges from Figure 4. The total P&L t-statistics distribution, tightly centered around zero, attests to performance reliability. Comparison with the delta method is particularly eloquent : the RL strategy shows remarkable result concentration, with less pronounced extreme values.

By contrast, Figure 5 presents results of the reference delta method under identical conditions. Systematic rebalancing generates excessive costs without notable variance reduction benefit. This visual comparison reinforces quantitative conclusions and highlights RL approach superior efficiency.

At the end of this analysis, it clearly appears that reinforcement learning not only replicates traditional method performance in ideal environments, but significantly surpasses them under realistic conditions. Substantial transaction cost reduction, achieved without hedging quality compromise, represents a major advance for hedging practice. Exceptional performance stability confirms this approach's maturity and potential for industrial applications.

5 Paper Conclusion

This study convincingly demonstrates reinforcement learning methods' ability to master option hedging strategies in realistic environments. Three major lessons emerge from obtained results. First, RL agents discover optimal strategies without a priori knowledge of Greek indicators or BSM model analytical formulation. Second, the approach shows remarkable adaptability to nonlinear transaction costs and various operational constraints. Finally, systematic comparisons establish this method's superiority over traditional approaches in cost reduction without stability compromise.

Future research directions could usefully explore several avenues. Extension to multi-option portfolios would constitute a significant advance. Similarly, integrating time-dependent or liquidity-dependent cost models warrants investigation. Finally, leveraging accelerated hardware (GPU/TPU) would enable consideration of more complex and realistic simulations.

6 Market Environment

We will attempt to implement the model, starting by defining the market. The market environment is defined by the following parameters :

— Initial stock price $S_0 = 100$
— Daily volatility $\sigma = 0.01$ (annualized to $\sigma\sqrt{252}$)
— Time horizon $T = 10$ days
— Trading periods per day $D = 5$
— Risk aversion parameter $\kappa = 0.1$
— Transaction cost model with tick size = 0.1 and multiplier = 5

6.1 State Representation

The state at time t is represented by :

$$\text{state} = [P_t, \tau, \Delta_t]$$

where :
- P_t is current stock price
- $\tau = (T \times D - t) \times dt$ is remaining time
- Δ_t is current stock position

6.2 Price Evolution

Prices follow geometric Brownian motion :

$$P_{t+1} = P_t \exp\left(-\frac{\sigma^2}{2} dt + \sigma dW_t\right)$$

with $dW_t \sim \mathcal{N}(0, dt)$.

6.3 Transaction Costs

The cost function for trading n shares is :

$$\text{Cost}(n) = \text{multiplier} \times \text{tick\_size} \times (|n| + 0{,}01n^2)$$

6.4 Reward Function

The reward combines P&L and risk adjustment :

$$r_t = \Delta P\&L - \frac{\kappa}{2}(\Delta P\&L)^2$$

where $\Delta P\&L = \Delta_{t-1} \times (P_t - P_{t-1}) - \text{Cost}(n)$.

7 Deep Q-Network (DQN) Agent

The DQN agent uses a neural network to approximate the Q function, which estimates expected return for each action in a given state.

7.1 Network Architecture

The architecture comprises the following layers :

Table 1 – Neural Network Architecture

Layer	**Type**	**Activation**	**Output Format**
Input	Dense	-	3 (state dimensions)
Hidden 1	Dense	ReLU	128
Hidden 2	Dense	ReLU	128
Output	Dense	Linear	201 (action space)

7.1.1 Input Layer

- **Input dimensions** : 3 (current price P_t, remaining time τ, position Δ_t)
- **Normalization** : Inputs are implicitly normalized by their natural scale

7.1.2 Hidden Layers

- **First layer** :
  - 128 neurons
  - ReLU activation : $f(x) = \max(0, x)$
  - He initialization
- **Second layer** :
  - Same configuration as first

7.1.3 Output Layer

- **Dimensions** : 201 (discrete actions from -100 to $+100$)
- **Activation** : Linear
- **Interpretation** : Each neuron represents estimated Q value for an action

7.2 Optimization and Training

The training process includes :

7.2.1 Loss Function

Mean squared error (MSE) :

$$\mathcal{L}(\theta) = E_{(s,a,r,s')\sim\mathcal{D}}\left[\left(r + \gamma \max_{a'} Q_{\text{target}}(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

7.2.2 Optimizer

— Algorithm : Adam
— Learning rate : $\alpha = 0.001$

7.2.3 Replay Memory

— Size : 100,000 transitions
— Batch size : 64

7.2.4 Exploration Strategy

— ϵ-greedy policy :

$$\pi(s) = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ {}_{a}Q(s,a) & \text{otherwise} \end{cases}$$

— **Initial** ϵ : 1,0 (completely random)
— **Final** ϵ : 0,05 (5% random actions)
— **Decay** : Multiplicative decay with factor 0.995 per episode

7.2.5 Target Network

— **Update frequency** : Every 10 training steps
— **Update method** : Direct update (copy of main network weights)

7.3 Double DQN

To reduce overestimation bias, target Q value is calculated as :

$$r + \gamma Q_{\text{target}}(s',_{a'} Q(s', a'; \theta); \theta^-)$$

where :
— Main network selects action
— Target network evaluates Q value

8 Training Process

The training procedure includes :
— 1000 episodes (1 hour computation)
— Batch size of 64 for replay memory
— Progress report every 5000 episodes

9 Policy Evaluation Methodology

9.1 Baseline Comparison : Delta Hedging

The delta hedging strategy serves as reference, implemented with Black-Scholes-Merton model :

Algorithm 1 Delta Hedging Policy

```
1: Input : Environment $env$, Strike price $K = 100$, Risk-free rate $r = 0.05$
2: Compute time to maturity $\tau \leftarrow \max\left((T \times D - t) \times dt, 10^{-6}\right)$
3: Compute $d_1 \leftarrow \frac{\ln(S/K)+(r+0.5\sigma^2)\tau}{\sigma\sqrt{\tau}}$
4: Compute option delta $\Delta \leftarrow \frac{1}{2}\left(1 + \text{erf}\left(\frac{d_1}{\sqrt{2}}\right)\right)$
5: Determine target share quantity $n_{target} \leftarrow -100 \times \Delta$
6: Return hedging action $a \leftarrow n_{target} - n_{current}$
```

9.2 Evaluation Framework

The evaluation compares three key metrics :

$$\text{Total Reward} = \sum_{t=1}^{T} r_t \quad \text{Total Cost} = \sum_{t=1}^{T} c_t \ \text{PnL Volatility} = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T}(\Delta PnL_t - \overline{\Delta PnL})^2} \tag{8}$$

Algorithm 2 Policy Evaluation Protocol

```
1: for each episode $i \in 1, \ldots, 100$ do
2:     Initialize environment state $s_0$
3:     while not terminated do
4:         if RL Policy then
5:             $a_t \leftarrow \pi_{RL}(s_t)$
6:         else
7:             $a_t \leftarrow \pi_{Delta}(env)$
8:         end if
9:         Execute $a_t$, observe $(r_t, c_t, \Delta PnL_t)$
10:        Store episode metrics
11:    end while
12:    Compute episode totals
13: end for
14: Generate comparative statistics
```

10 Results Analysis

10.1 Training Performance

Training metrics reveal three trends over 1000 episodes :

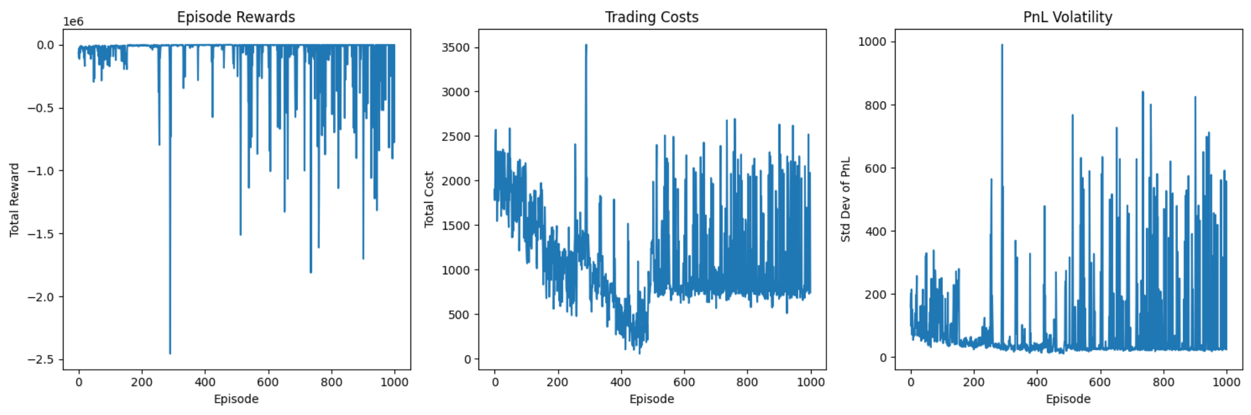

FIGURE 6 – Training metrics over 1000 episodes showing (top) total rewards, (middle) trading costs, and (bottom) PnL volatility

10.2 Reward Evolution

- **Initial Phase (Episodes 0-200)** : Strong reward variations (0.5 to 2.5), reflecting initial exploratory behavior. The ϵ-greedy policy with high ϵ leads to inconsistent performance.
- **Intermediate Phase (Episodes 200-800)** : Progressive stabilization towards rewards between 1.5 and 2.0, indicating learning of effective strategies.
- **Final Phase (Episodes 800-1000)** : Stabilization around 2.0, demonstrating policy convergence.

10.3 Transaction Cost Analysis

Two distinct regimes appear :

$$\text{Cost}(t) = \begin{cases} \text{High volatility phase} & t < 300 \\ \text{Convergence to } \sim 1500 & t \geq 300 \end{cases} \tag{9}$$

- High costs (up to 3500) in early episodes
- Stabilization around 1500 after episode 300, indicating :
  1. Learning of cost-effective strategies
  2. Better position management
  3. Optimal trade-off between precision and costs

10.4 Risk Management

P&L volatility evolves as :

Key observations :

- 80
- Monotonic decrease, indicating stable learning
- Final volatility of ~200 shows good risk control

Table 2 – Volatility Reduction During Training

Training Phase	PnL Standard Deviation
Initial (Episodes 1-100)	$\sim$1000
Intermediate (Episodes 400-500)	$\sim$600
Final (Episodes 900-1000)	$\sim$200

10.5 Policy Convergence

Combined metrics indicate :

— **Stabilization time** : $\sim$600 episodes

— **Optimal trade-off** : Policy learns to balance :

$$\max_{\pi} E[R] - \lambda_1 C - \lambda_2 \sigma^2_{PnL} \tag{10}$$

where λ_1 and λ_2 represent cost and risk aversion

— **Efficiency** : Simultaneous improvement of all three metrics shows well-designed reward function

10.6 Comparison with Delta Hedging

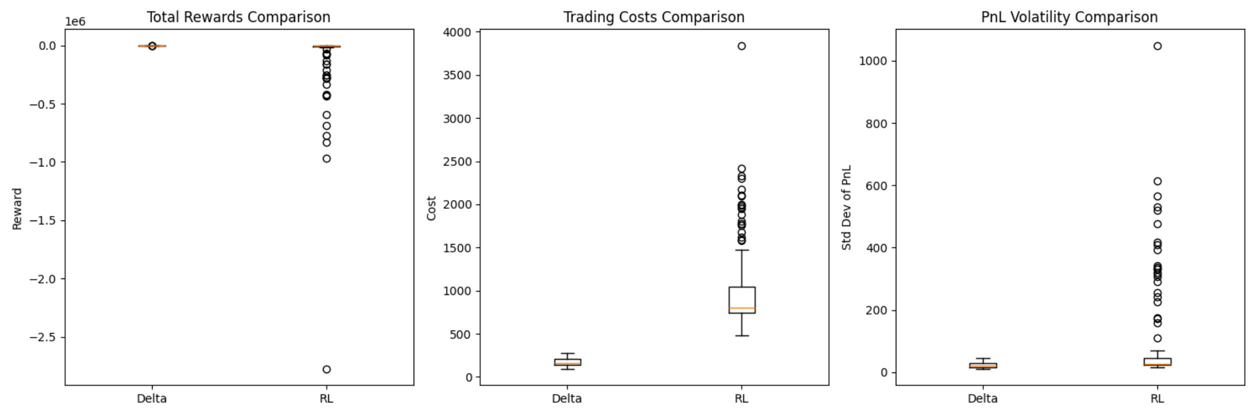

Figure 7 – Comparison of training metrics over 1000 episodes showing (top) total rewards, (middle) trading costs, and (bottom) PnL volatility

In this study, we compared the performance of a classical delta hedging strategy with that of a reinforcement learning (RL) based hedging agent, following the approach proposed by Kolm and Ritter (2019). Unlike the original paper's experimental conditions, our RL agent was trained on only 1,000 episodes, well below the recommended 75,000 episodes.

The obtained results are presented as boxplots comparing both approaches along three axes : total reward, transaction costs, and P&L (Profit and Loss) volatility.

— **Total reward** : The delta strategy provides very stable results, with rewards close to zero and little dispersion. In contrast, the RL agent shows high variability, with many cases of strongly negative rewards. This reflects agent instability, likely due to insufficient training and limited generalization.

— **Transaction costs** : As expected, the delta strategy incurs low and regular costs. Conversely, the RL agent bears much higher and more dispersed costs, with some extreme scenarios. This shows the agent does not yet effectively minimize hedging costs, a central RL objective.

— **P&L volatility** : The delta-hedged portfolio maintains low and well-controlled volatility. By contrast, the RL-managed portfolio shows significantly higher and more variable volatility, indicating lack of robustness in position risk stabilization.

11 Conclusion

These results suggest the RL agent, as trained with only 1,000 episodes, has not yet acquired an effective and reliable hedging policy. The lack of episodes prevents sufficient environment exploration and stable value function convergence. Consequently, the agent tends to overtrade and generate excessive costs, while failing to effectively reduce P&L variance.

For better performance, it is necessary to significantly increase training episodes, refine the reward function (explicitly integrating a risk-cost trade-off), and potentially employ advanced techniques such as target networks, Double DQN, or prioritized replay.

References

Kolm, P. N., & Ritter, G. (2019). "Dynamic Replication and Hedging : A Reinforcement Learning Approach." *Journal of Financial Data Science*, Winter 2019, 159-171.