Dynamic Hedging Strategies for Option Buyers Using Deep Reinforcement Learning

1st Bhandari, Rucha
Northeastern University
Boston, MA, USA
bhandari.ru@northeastern

2nd Marler, Benjamin
Northeastern University
Boston, MA, USA
marler.b@northeastern

3rd Singh, Harsh
Northeastern University
Bangalore, KA, India
singh.harsh1@northeastern

***Abstract*—While extensive research exists on option hedging strategies from the seller's perspective, there remains a significant gap in developing effective strategies for option buyers. Given a fixed strike price, premium, and expiry date, we explore how buyers should optimally hold the underlying stock over time. We compare Deep Double Q-Network (DDQN) against three Monte Carlo Policy Gradient (MCPG) variants using different risk-based loss functions: entropic, Sharpe ratio, and Markowitz. Our custom trading environment features a 7-dimensional state space including position size, normalized stock price, time to expiry, portfolio value, option Greeks (delta and gamma), and implied volatility with daily data from multiple tickers. DDQN achieves superior mean returns of 352.4% with a 19.9% capture percentage, compared to MCPG variants' 220-264% returns, though at the cost of significantly higher volatility (1159.3% vs. 697-807%). Despite these differences, all models maintain comparable Sharpe ratios (0.30-0.40). Notably, DDQN exhibits weaker downside protection with average losses of -589.2%, compared to MCPG's -261.9% to -344.9%.**

I. Introduction

The financial market often comes with a lot of risk, return and tension. A financial instrument provided by financial institutions is the stock option. The stock option is more a promise that it is a number. A promise that grants the right to buy or sell an asset for a predetermined price at a future date, barring any appreciation or devaluation of the asset. Options offer strategic flexibility, but they introduce new variables to regular portfolio management. We believed that this would make an interesting and complex environment to navigate using artificial intelligence.

A. Motivation

Our team was paired up because of our common interest in creative application of technology in finance. We picked a topic that would expose us to new financial concepts, but also new ML methods we hadn't gotten a chance to implement in classwork or programming assignments, such as Deep Q-Learning and Policy Gradient Descent. One of the most interesting aspects of the project was learning to model a rather complicated environment into a familiar AI agent that we can control. Additionally, most of our initial research took the perspective of the financial institutions who sell these options. There was an asymmetry in this research area, with not many papers covering the point of view of the buyer, which gave us a strong incentive to pursue the project to cater to options trading from the buyer's standpoint.

B. Problem Statement

While the options serve as a risk-reducing instrument, the remaining capital that will be used to maximize profit still rests in the strategic hedging implementation from the buyer's point of view. Our problem statement, thus, can be formulated as the following: "Given a fixed strike price, premium, and expiry date, how should an option buyer dynamically hedge and allocate a fraction of their portfolio to the underlying stock over time in order to maximize their capital and their terminal return. At first, it may seem like any other hedging strategy, but it could be significantly modified and optimized according to the anticipated option payoff.

II. Literature Review

The first paper [1] explores the application of deep reinforcement learning (DRL) to the complex domain of option hedging, with a particular emphasis on American-style options which provide early exercise flexibility. The study evaluates the potential of DRL techniques such as Actor-Critic (A2C), Deep Deterministic Policy Gradient (DDPG), and Proximal Policy Optimization (PPO) to enhance traditional stock option trading strategies. The second paper [2] proposes a standalone dynamic option selling strategy that uses technical indicators, option Greeks, and machine learning for optimization. Their focus on selling options to capture time decay highlights the importance of incorporating risk metrics and the unique challenges associated with the option seller's asymmetric exposure. The third paper [3] offers a comprehensive study, comparing eight DRL algorithms—ranging from Monte Carlo Policy Gradient (MCPG) and PPO to several DQL and DDPG variants—against the classical Black-Scholes delta hedge baseline. This study specifically targets the hedging of American put options using DRL, implementing the DDPG algorithm. It presents one of the first DRL agents specifically for American-style hedging and demonstrates its superiority over Black-Scholes-based methods. The fourth paper [4] introduces the deep hedging framework, which addresses real-world challenges such as transaction costs, liquidity constraints, and risk limits. By training DRL agents to minimize risk measures

like Conditional Value-at-Risk (CVaR) in a model-free environment, the approach shows a robust performance without relying on restrictive market assumptions.

III. Financial Background

A. Options

Options are defined as financial contracts that provide somebody with the right to buy or sell an asset at a certain price after a certain time, despite its devaluation or appreciation within that time period. Working with a "call" option, the option provides us with the ability to purchase Z number of shares at the strike price of K. At exactly T days from the date of purchase, if the stock price, S_T, is below the strike price, the option becomes unprofitable. However, if $S_T > K$, then, the client can exercise their option, purchase Z shares at strike price K, sell them at the higher stock price S_T, and then simply pocket the difference. The client also pays a "premium" as the price of purchasing an option per share. This price, labeled p_0 is the last variable that will define the profit made by an option. The option payoff is defined in equation .

$$V_{option} = max(S_T - K, 0) - p_0 Z \tag{1}$$

Under this, clients also perform hedging in order to maximize their profits, and the option is normally their "safety net". Our model's job is to account for the option's safety net and produce an optimal hedging strategy.

B. Black-Scholes Model

Traditionally, the pricing of an option is modelled by the Black-Scholes Model, developed in 1973, which has proven to be a well-performing model, catering to market dynamicity and volatility. A limitation of this model is that volatility of market and interest rates are constants, and is just taken over a combination of historical data rather than derived, and then set for the model's evaluation period. Therefore, it fails to predict more complex-style options such as ones with early execution incorporated in their offer. The price per premium, p_0, is calculated using equation (2).

$$C = SN(d_1) - Ke^{-RT}N(d_2) \tag{2}$$

$$d_1 = \frac{ln(\frac{S}{K}) + (R + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} \tag{3}$$

$$d_2 = d_1 - \sigma\sqrt{T} \tag{4}$$

In the above formulas, C refers to the call option's price per premium, p_0 that is being computed. S refers to the stock price, $N()$ is the cumulative normal distribution function, K is the strike price, R is the returns, σ was the volatility and T was the time to expiration (in years).

The implied volatility, σ, models how the stock price may fluctuate over time tracking highs and lows. This is a number between zero and one that can be computed in many different ways. For this project, we utilize an exponentially weighted moving average (EWMA) in equation (5). We look back 252 days, as that is typically the number of trading days in a given year. EWMA is defined with σ as follows:

$$EWMA_t = \frac{1}{T}(\Sigma_{t=1}^{T} 63[\log(\frac{S_t}{S_{t-1}})]^2 - 62 * EWMA_{t-1}) \tag{5}$$

$$\sigma = \sqrt{EWMA_t} * \sqrt{252} \tag{6}$$

In (5), 63 is an assumed constant, which is a function of our chosen number of lookback days = 252.

C. Greeks

Our greeks are variables used throughout the computation of the model, primarily featured in our observations and state space. Their incorporation was to help our model stay sensitive to the changes in the market. Our Greeks, delta (δ) and gamma (γ), are computed using equations (3) and (4). Delta represents the sensitivity of the option's pricing with a $1 change in the underlying stock price and is shown in equation (7). Gamma represents the rate of change of Delta and is computed using equation (8).

$$\delta = N(d_1) \tag{7}$$

$$\gamma = \frac{N(d_1)}{S_t\sigma\sqrt{T}} = \frac{\delta}{S_t\sigma\sqrt{T}} \tag{8}$$

IV. Methodology

A. Dataset

The data of the stock price was taken from the "Yahoo! Finance API". The dataset included 25 years of stock data ranging from 2000 - 2024 for major companies in the world such as Apple (AAPL), McDonald's (MCD) and Exxon (EXM). The range of tickers chosen were also from a variety of industries so that any one industry's decline or resurgence did not affect the data in any way. For our purposes, only the opening and closing stock prices at any given day were among the relevant features used from this data.

All models were trained on a random sample of this dataset, and any final models that are reported on, have been trained on a total of 100,000 episodes. As for testing purposes, each model was tested on a total of 10,000 episodes for a randomly sampled ticker. To evaluate model performance fairly on completely unseen data, the ticker was sampled from a company that was disjoint from the set of companies in the training set, but similar profile in terms of industry, size and performance.

B. Trading Environment

1) Environment Features:

i **Risk:** It refers to the fraction of the total money that is being put in by the buyer than they are willing to invest into the portfolio for hedging purposes.

ii **Moneyness:** It is the ratio between the starting stock price and the strike price of the option. It is used as a

qualitative variable that is more descriptive of the state of the relationship, than just the ratio. For example,

- If Moneyness is 1, that means that $K = S_T$, and we are "at-the-money".
- If Moneyness is greater than 1, that means that $K > S_T$, and we are "out-of-the-money."
- If Moneyness is less than 1, that means that $K < S_T$, and we are "in-the-money".

The "Moneyness" also helps us set the initial strike price associated with the option.

$$K = Moneyness * S_0$$

iii **Risk-Free Rate:** It refers to a return expected on any asset at the end of a fiscal year, where no risk is involved or undertaken by the owner. While this value varies from year to year, it is incredibly hard to actually predict it, and hence, we picked a favourable constant, 0.05, for our model predictions.

iv **Action Space:** The action space are integer numbers ranging from 0 to 1, with 0.02 increments, corresponding to the proportion of the money invested into the portfolio. At every time step, the action space reflects the model's choice of how that invested money changes, as part of the maximal or optimal hedging strategy.

v **Expiry Dates:** This refers to the number of days that the purchased option is valid for, at the end of which, the option may be executed, if profitable. In many common options markets, they are sold for a period of 7, 14, 30, 45, 60 and 90 days. We sampled through these expiry dates and used them to train our model so that it may be more sensitive to the amount of time left for hedging and maximizing profit, before the expiry date arrives.

vi **Portfolio Value:** The portfolio value refers to the amount at which the portfolio of the client is valued at. It is a combination of the fruit of their hedging strategy, their stocks owned, their invested money, and their option payoff.

$$V_t = S_t X_t + M_t + V_{option,t} \tag{9}$$

vii **Reward Function:** Our reward function was modeled in equation 10.

$$\textbf{Reward Function, } \mathbf{R} = \ln \frac{V_t}{V_{t-1}} * Factor \tag{10}$$

$$\textbf{where, Factor} = \begin{cases} 1 & S_t \geq K \\ 1 + 10(1 - \frac{S_t}{K}) & S_t < K \end{cases} \tag{11}$$

The reward function is designed to create a reward system based on the jump or fall in the portfolio value at any given time step. It is also multiplied with an urgency factor, which is to say, if our model is "in the money", then the urgency factor is 1, because the model is in the clear and performing well, projected to make gains using the option. However, if the model is "out of the money", then the model is at a place where the option may not be executed, at which point, making money out of an optimal hedging strategy becomes all the more important. Therefore, in this scenario, the reward function will generate an urgency factor between 1 and 11 based on how urgent the scenario is. Or rather, how far below the stock price is compared to the strike price.

2) State Space:

i Position Discrete (0, 1, +0.02). The position at a given state is equivalent to that of the most recent action that was taken, since it is indicative of the number of shares that the client currently holds as a result of hedging, as a proportion of the total shares they may hold.

ii Normalized Stock Price: It is a continuous positive value that refers to the ratio between the stock price at time step t, with respect to the strike price.

iii Time To Expiry: This just refers to the number of days between the current timestep and the expiry time that was chosen at start time, thereby accounting for the number of days the algorithm has left to hedge. It can be represented as $T - t$, and is an integer ranging from 0 to 90.

iv Normalized Portfolio Value: This continuous, positive value refers to the total profit/loss incurred since the start of the episode, as a proportion of the original portfolio value at timestep, $t = 0$. This assists the model in accounting for its own performance to improve.

v Delta - This is the continuous Greek value computed in equation (7) that ranges between 0 and 1.

vi Gamma - This is the continuous, positive Greek value computed in equation (8)

vii Volatility (Sigma) - This is the continuous positive value computed in equation (5)

C. Deep Reinforcement Learning

1) Double Deep Q-Learning Network (DDQN): In 1989, Chris Watkins introduced Q-learning (Watkins & Dayan, 1992). This algorithm estimates the Q-value, value of taking action **a** in state **s** while following a policyπ and receiving a reward. This indicates to the model how effective an action is in a particular state. A higher Q-value indicates greater reward for the agent. The Q-value of a current time step is based on the Q-value of the future time step. While training we update the Q-value in state S_t based on the Q-value in the state S_{t-1}. Using the formula:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \Big]$$

This algorithm performs poorly in some stochastic environments. This is because of the max operator which selects the best next action and evaluates it values. Q-Learning can overestimate the Q-value some certain actions. Over time this leads to overly optimistic estimates and suboptimal policies.

The solution to this was proposed in 2010 (Hasselt, 2010). Where instead of using only one Q-value for each state-action pair, we should use two Q-value estimators Q_A and Q_B. So essentially Q_A picks the best action in the next state and Q_B

evaluates that action and calculates its future rewards. Finally, that is used to update Q_A.

This approach is focused on finding a action a^* that maximizes Q_A in the next state(S'). Then use this action to get the value of $Q_B(S', a^*)$. Finally use it to update $Q_A(S, a*)$.

$$Q_A(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R + \gamma Q_B(S_{t+1}, a) - Q_A(S_t, A_t) \Big]$$

It is mathematically proven that the expected value of Q_B an action a^* is smaller than or equal to the maximum value of $Q_A(S', a^*)$. This means that over a large number of iterations, the expected value of $Q_B(S', a^*)$ is going to be smaller than $Q_A(S', a^*)$. Thus $Q_A(S, a)$ never updates with a maximum value, therefore never overestimates.

In 2015 researchers at DeepMind proposed a paper that mixes Q-learning with neural networks (Hasselt et al., 2015). So instead of using Q-tables, Deep Q-Learning (DQN) uses two neural networks.

Feed forward neural networks are used to predict the best Q-Value. Since the input data is not provided beforehand, the agent stores previous values in local memory called an experience reply, which is then used as input data. Since every Q-Value depends on the policy, the expected outcome also known as the target is continuously changing at every new iteration. This is the main reason we use 2 neural networks. The two networks are called the **Q-network** and the **Target network** . When implementing a Deep Q-learning Network, the Q-network calculates the Q-value for the current state S_t and the Target-network calculates the Q-value for the next state S_{t+1}. The Target-network also picks the best Q value amongst these. The target network is defined by:

$$Y_t^{DQN} \leftarrow R_{t+1} + \gamma maxQ(S_{t+1}, a, \theta_t^-)$$

Since the Target-network both calculates and picks the Q-value for the next state, it leads to the same problem as Q-Learning. The network overestimates the Q-values.

Thus, we used a Double Deep Q-learning network as our model. The Q-network calculates the Q-values of the current state S and the next state S_{t+1} and selects the best action for the next state. The Target-network also calculates the Q-Values for the next state S_{t+1} and provides its evaluation for the action chosen by the Q-network, which is then picked by the Q-network. The target-network in DDQN is defined as:

$$Y_t^{DDQN} \leftarrow R_{t+1} + \gamma Q(S_{t+1}, argmaxQ(S_{t+1}, a; \theta_t), \theta_t^-)$$

Here θ_t represents the parameters of the Q-network at t and θ_t^- represents the parameters of the Target-network at t.

The weights of the second network are replaced with the weights of the target network for the evaluation of the policy. The target network is updated periodically by copying the parameters from the Q-network. This also provides stability, and the network learns better from stable targets.

In our implementation of DDQN for options trading, the states are the 7 observations or our state space, and the actions are the hedge positions. Our goal is to maximize returns from the option hedging strategy.

2) Monte Carlo Policy Gradient (MCPG): Monte Carlo Policy Gradient represents a fundamental departure from value-based methods like Q-learning. While Q-learning updates its value estimates after each action using temporal difference learning, MCPG waits until the end of each episode to observe the terminal reward before updating the policy. This approach is particularly well-suited to option trading, where the true payoff is only realized at expiration.

In our implementation, we train the MCPG agent by running batches of 256 episodes. After collecting the terminal rewards from all episodes in the batch, we update the policy parameters using gradient descent:

$$\theta_{i+1} \leftarrow \theta_i - \alpha \nabla_\theta l(R)$$

where θ represents the policy parameters, $\alpha = 0.003$ is the learning rate, and $l(R)$ is the loss function computed from the terminal rewards R. The learning rate was selected through limited hyperparameter tuning to balance convergence speed with stability.

To investigate how different risk preferences affect trading performance, we implemented three distinct loss functions:

The **entropic risk** loss function focuses solely on downside risk:

$$l(R) = \frac{1}{\lambda} \ln(E[e^{-\lambda R}])$$

This formulation penalizes negative returns exponentially while remaining indifferent to positive outcomes, making it suitable for risk-averse traders primarily concerned with avoiding losses. We set the risk aversion parameter $\lambda = 0.65$ based on limited hyperparameter tuning.

The **Sharpe ratio** loss function balances returns against volatility:

$$l(R) = -\frac{E[R]}{\sigma_R}$$

By maximizing the return-to-risk ratio, this approach seeks strategies that provide consistent performance relative to their variability. Note the negative sign, as we minimize loss rather than maximize the Sharpe ratio.

Finally, the **Markowitz** loss function explicitly trades off expected returns against variance:

$$l(R) = -E[R] + \lambda Var(R)$$

where $\lambda = 0.65$ controls the penalty on variance, using the same risk aversion parameter as the entropic formulation for consistency. This classical mean-variance framework allows for direct control over risk tolerance.

We selected these three formulations to examine how different risk objectives influence the learned trading strategies. Each represents a distinct philosophy: entropic risk for downside protection, Sharpe for efficiency, and Markowitz for explicit risk-return tradeoffs. The hyperparameters were chosen through limited grid search to ensure stable convergence while maintaining reasonable training times.

V. Results

We evaluated the performance of DDQN against three MCPG variants (Entropic, Sharpe, and Markowitz) across 10,000 test episodes. The models were assessed using industry-standard metrics including Sharpe ratio, maximum drawdown, and capture percentage.

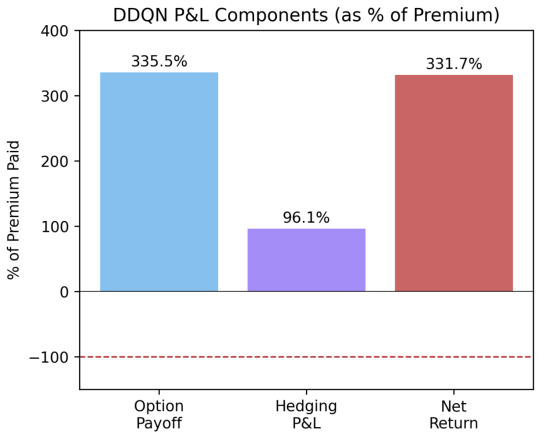


Fig. 1. Decomposition of P&L components for DDQN. The hedging P&L represents the controllable portion through dynamic trading, while option payoff remains fixed across all models as early exercise was not permitted.

Figure 1 illustrates the P&L decomposition, highlighting that while option payoffs remain consistent across all models (as early exercise was disabled), the hedging P&L component represents our primary lever for performance optimization through dynamic trading strategies.

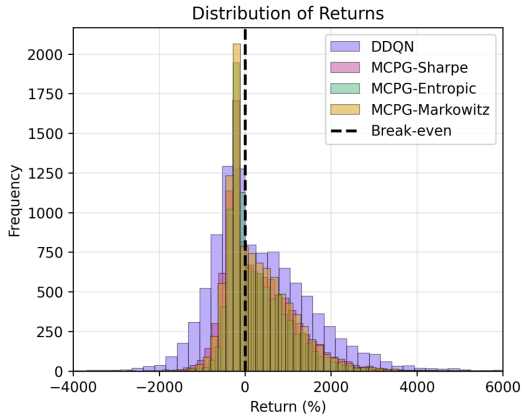


Fig. 2. Distribution of returns across 10,000 test episodes.

The return distributions (Figure 2) reveal distinct behavioral patterns: MCPG models exhibit relatively consistent Gaussian curves, while DDQN demonstrates significantly higher standard deviation (1159.3% vs. 697-807% for MCPG variants). This increased variance is expected as DDQN's Q-learning framework does not explicitly account for variance in its loss calculations.

Table I presents comprehensive performance metrics. Despite DDQN's superior mean return (352.4%), Sharpe ratios remain comparable across all models (0.30-0.40) due to DDQN's proportionally higher volatility. The median returns scale consistently with mean returns, suggesting stable performance characteristics.

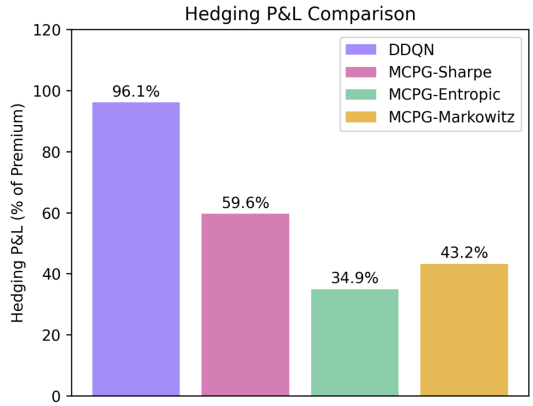


Fig. 3. Hedging P&L comparison across models.

Notably, DDQN's capture percentage of 19.9% (representing the fraction of maximum theoretical returns achieved) exceeds all MCPG variants. The hedging effectiveness analysis (Figure 3) reveals that DDQN's average win percentage (1160.6%) significantly outperforms MCPG models, with the performance gap in wins exceeding the gap in overall returns. This indicates particularly effective hedging when options expire in-the-money.

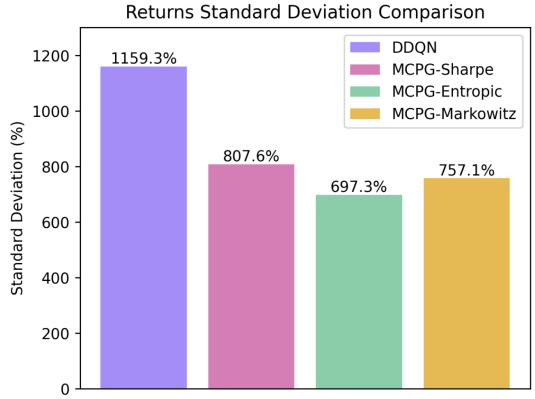


Fig. 4. Standard deviation comparison across models.

However, DDQN's average loss of -589.2% reveals significantly larger downside exposure compared to MCPG variants (-261.9% to -344.9%), indicating weaker downside protection. This is a critical concern for hedging strategies. This asymmetric performance profile suggests DDQN adopts more aggressive positioning that amplifies both gains and losses, prioritizing upside capture over risk mitigation.

The episodic return analysis (Figure 5) confirms that while all models exhibit sporadic return patterns characteristic of option hedging, DDQN maintains consistent outperformance throughout the test period. This dominance, combined with the model's ability to minimize losses while maximizing gains, is

Metric	DDQN	MCPG-Entropic	MCPG-Sharpe	MCPG-Markowitz
Mean Return (%)	**352.4**	278.2	294.7	284.8
Median Return (%)	**126.6**	57.0	68.5	66.7
Std Deviation (%)	1159.3	697.3	807.6	757.1
Sharpe Ratio	0.30	**0.40**	0.36	0.38
Win Rate (%)	**53.8**	52.5	52.9	52.9
Avg Win (%)	1160.6	766.5	864.7	816.9
Avg Loss (%)	**-589.2**	-261.9	-344.9	-312.6
Capture Percent (%)	**19.9**	15.7	16.7	16.3

TABLE I
PERFORMANCE COMPARISON ACROSS MODELS.

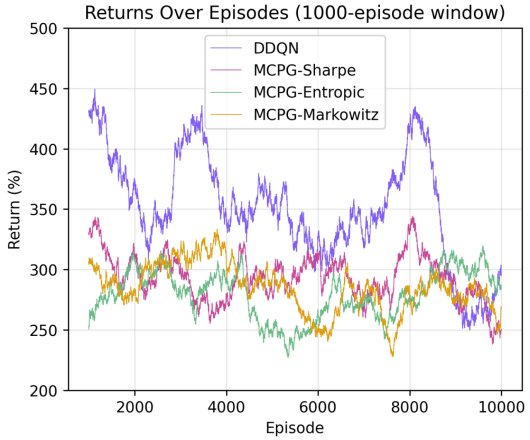


Fig. 5. Rolling average returns over 10,000 episodes.

important, but does not necessarily make it the superior model. As emphasized earlier, the higher variance and lower losses make it less viable for the main purpose of hedging.

VI. CONCLUSION

A. Limitations

MCPG is an algorithm that often yields high variance values, which contributed to our less than ideal Sharpe Ratio values, which we hope to adjust by exploring techniques like the Generalized Advantage Estimation. We also assumed global risk aversion factor. Ideally, the risk-aversiveness could be a function of the market's current state or volatility. Given that we have a neural network that does the heavy-lifting in the machine learning model, we want to explore the SHAP explainability of our model. We think that coming up with a good model may not be enough. Being able to explain or determine which features of our model end up driving our hedging decisions and why our model makes certain good decisions and avoids others, will give our model methodology extra credibility.

Currently, our action space is discretized, which may constrain the model. We'd like to explore continuous action spaces and try other policy gradient descent algorithms that work better on continuous data. Another limitation would be the high training period and trying to find better physical resources to speed up our model training, since our training times and computational efficieny was not comparable to other related research and it was very much a physical limitation.

B. Future Work

This project has really intrigued us about the world of finance since we got to model this dynamic financial environment. For this project, our options were free to execute at the end of T days, time period. These stock options are a lot more common in Europe, while America also offers options that may be executed "early" at any point in time between timestep $0 < t < T$, which changes the problem statement significantly. Given that our model can make predictions and analysis about a given option, its scope can be widened to accommodate the input of many option choices, where it can then recommend the best choice based on the expected hedging and outcome. We can also incorporate the input of a risk aversion factor making it user-centric and the user may decide how averse to risk they are in developing their portfolio. For model improvements, we'd like to explore with random restarts, since local peaks were a common problem with MCPG models and their loss minimization.

Link To GitHub Repository:
https://github.com/bamarler/Deep-RL-for-Option-Hedging

REFERENCES

[1] Garza, J. (2023). *Reinforcement Learning for Stock Option Trading*. https://doi.org/10.63227/566.706.52

[2] Joshi, A., Venkateswaran, B., & Bhattacharyya, R. (2024, March 20). *Options Selling Using Machine Learning*. Social Science Research Network. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4766370

[3] Neagu, A., Godin, F., & Kosseim, L. (2025). *Deep Reinforcement Learning Algorithms for Option Hedging*. ArXiv.org. https://arxiv.org/abs/2504.05521

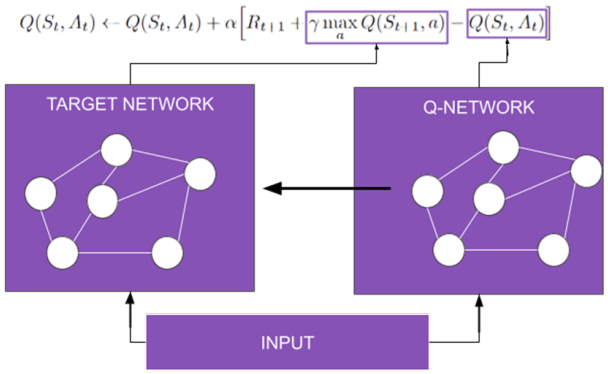
[4] Buehler, H., Gonon, L., Teichmann, J., & Wood, B. (n.d.). *Deep Hedging*. ArXiv.org. https://arxiv.org/pdf/1802.03042

[5] Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3–4), 279–292. https://doi.org/10.1007/bf00992698

[6] Hasselt, H. (2010). Double Q-learning. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc. https://papers.nips.cc/paper_files/paper/2010/hash/091d584fced301b442654dd8c23b3fc9-Abstract.html

[7] Van Hasselt, H., Guez, A., & Silver, D. (2015). *Deep Reinforcement Learning with Double Q-learning*. ArXiv.org. https://arxiv.org/abs/1509.06461

Appendix A. DDQN

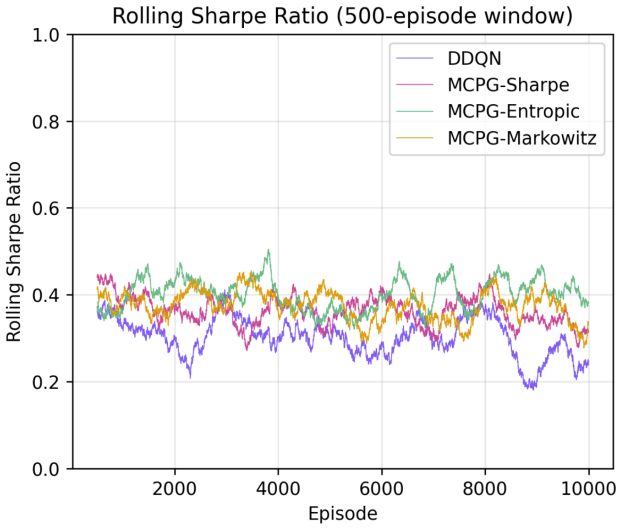


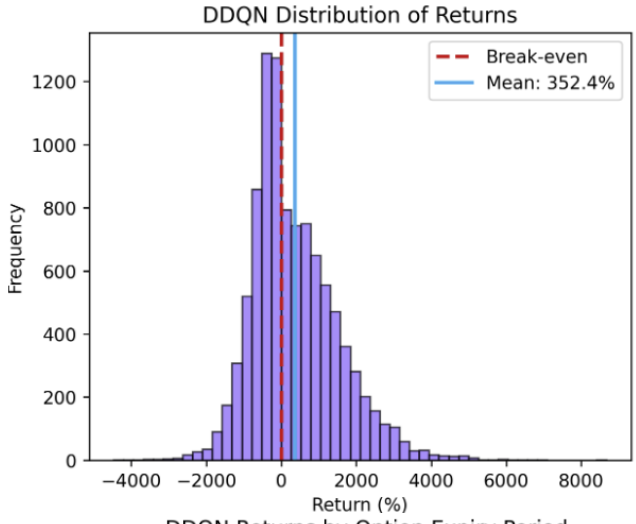

```python
class DDQNNetwork(nn.Module):
    def __init__(self, num_actions=51):
        super().__init__()

        input_dim = 7

        self.network = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.LayerNorm(128),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(128, 128),
            nn.LayerNorm(128),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, num_actions)
        )
```

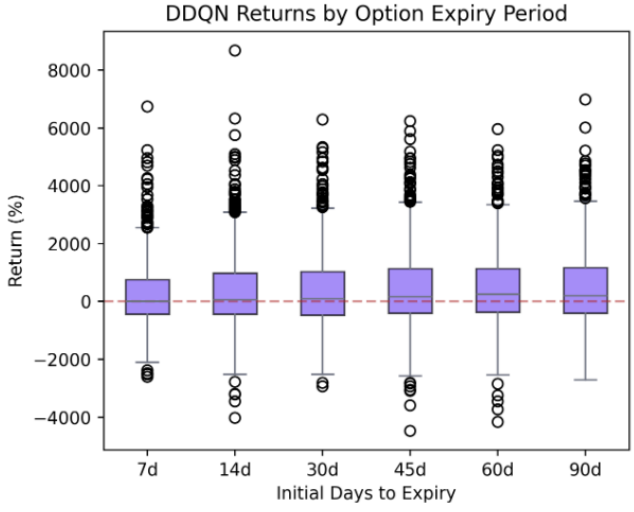
Appendix B. MCPG

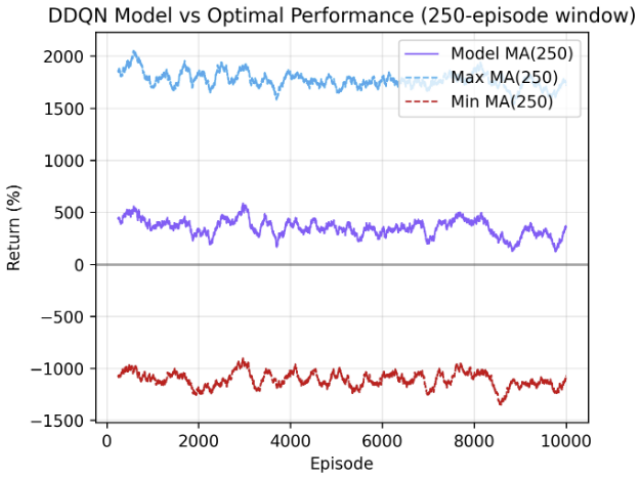
```python
class MCPGPositionNetwork(nn.Module):
    def __init__(self, num_actions=51):
        super().__init__()

        input_dim = 7

        self.stack = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.LayerNorm(128),
            nn.LeakyReLU(),
            nn.Dropout(0.2),
            nn.Linear(128, 128),
            nn.LayerNorm(128),
            nn.GELU(),
            nn.Dropout(0.2),
            nn.Linear(128, 64),
            nn.Tanh(),
            nn.Linear(64, 64),
            nn.ReLU(),
            nn.Linear(64, num_actions)
        )
```

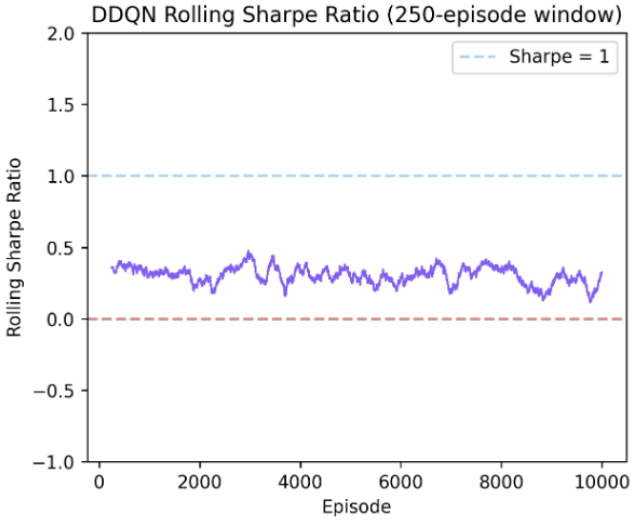
APPENDIX C. EXTENDED RESULTS

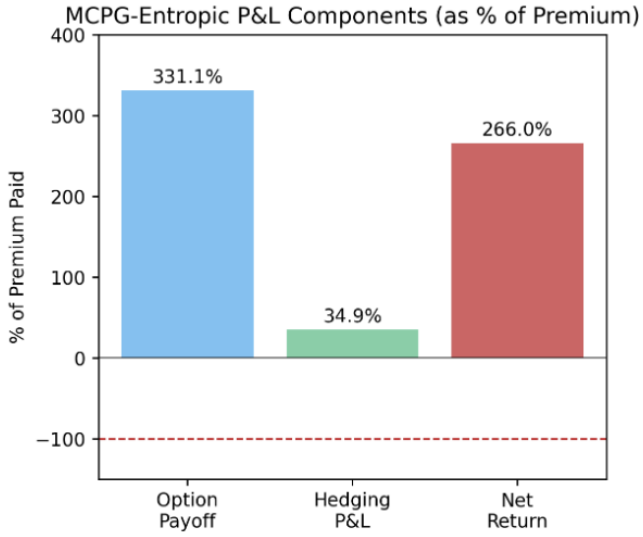


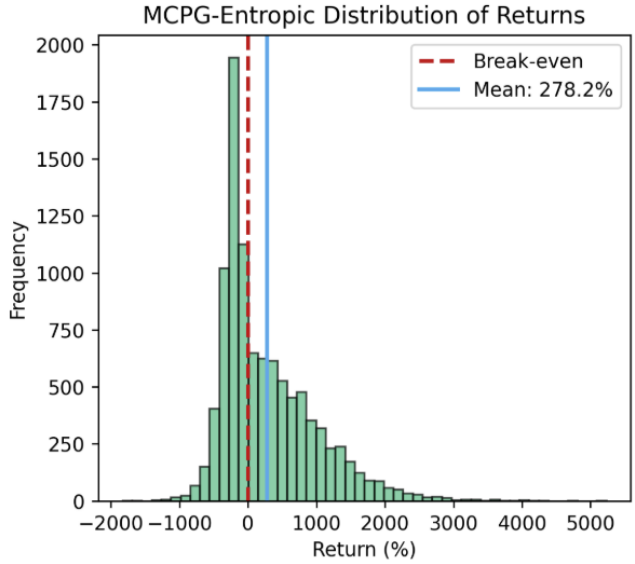


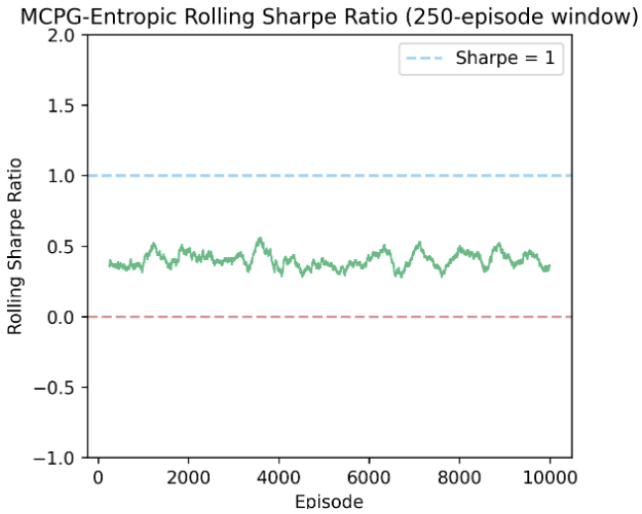


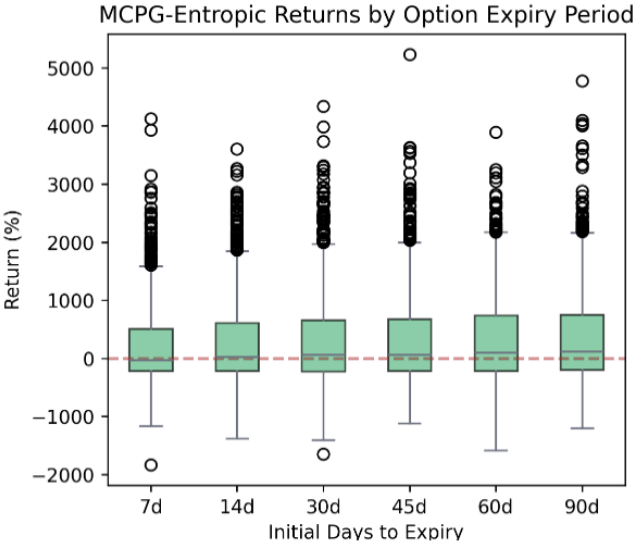


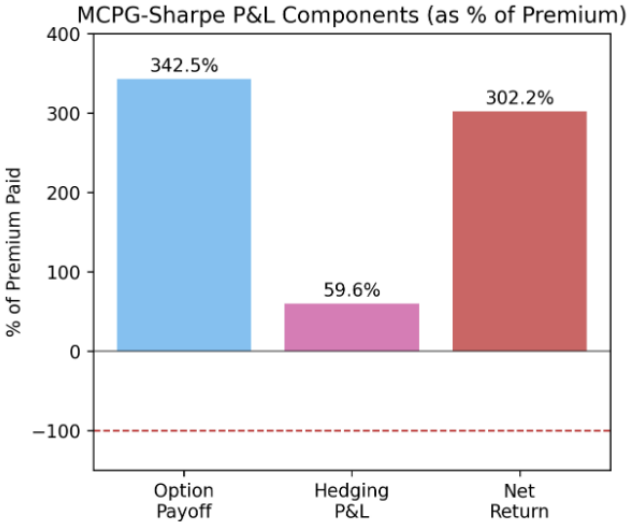


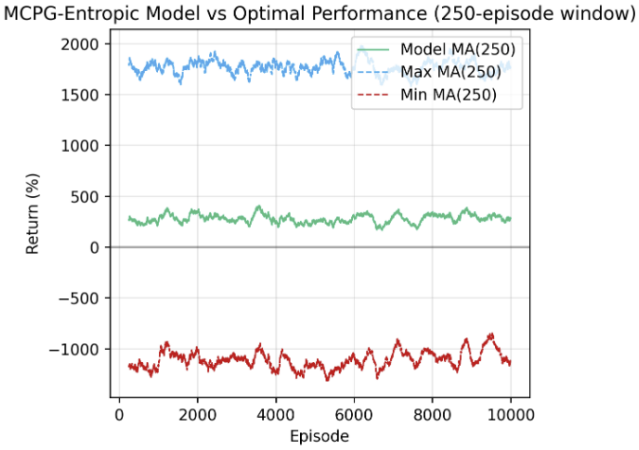


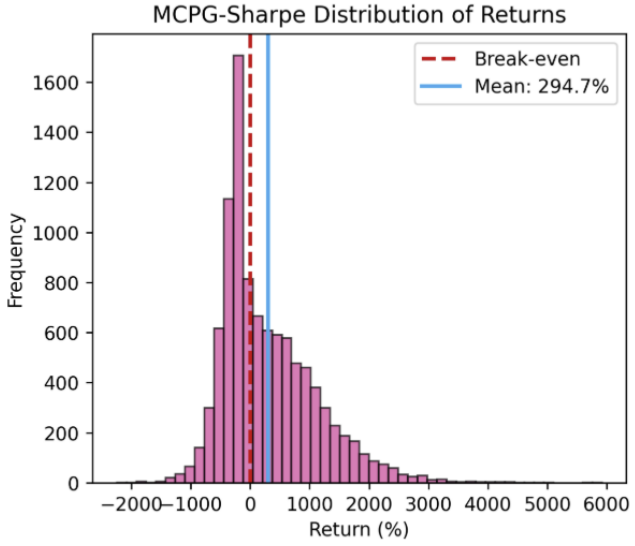

MCPG-Entropic Distribution of Returns
Break-even
Mean: 278.2%
Frequency
Return (%)

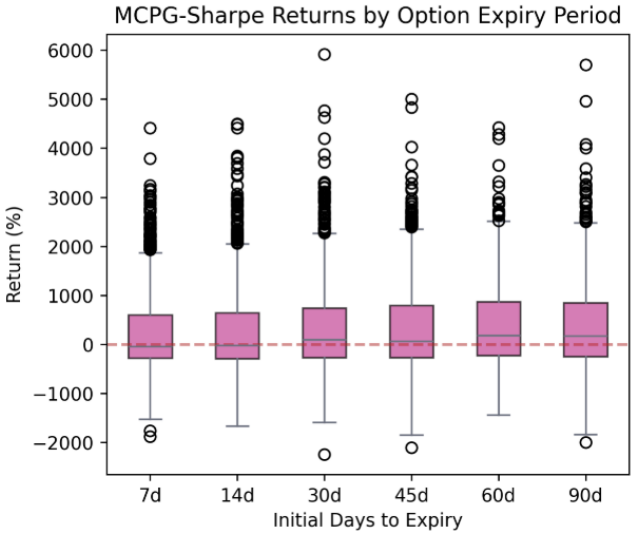

MCPG-Entropic Rolling Sharpe Ratio (250-episode window)
Sharpe = 1
Rolling Sharpe Ratio
Episode

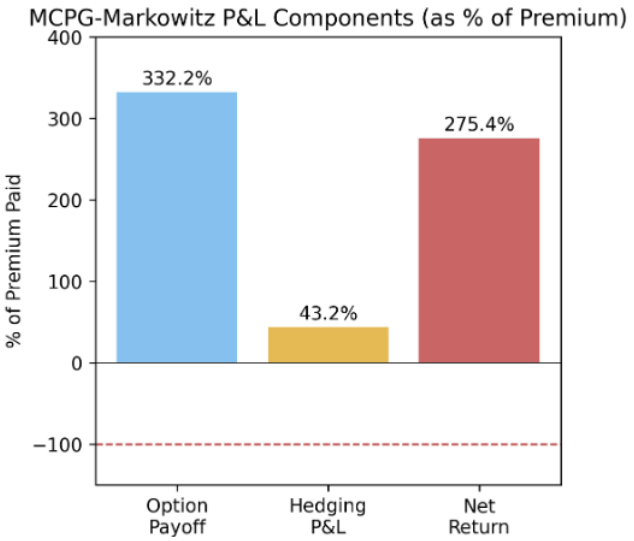

MCPG-Entropic Returns by Option Expiry Period
Return (%)
7d
14d
30d
45d
60d
90d
Initial Days to Expiry

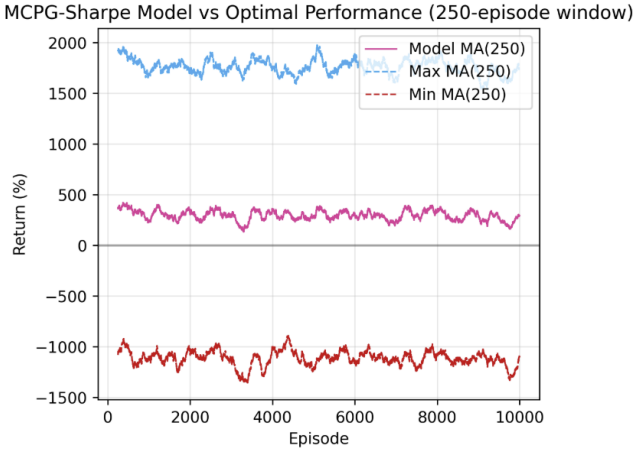

MCPG-Sharpe P&L Components (as % of Premium)
342.5%
59.6%
302.2%
% of Premium Paid
Option Payoff
Hedging P&L
Net Return

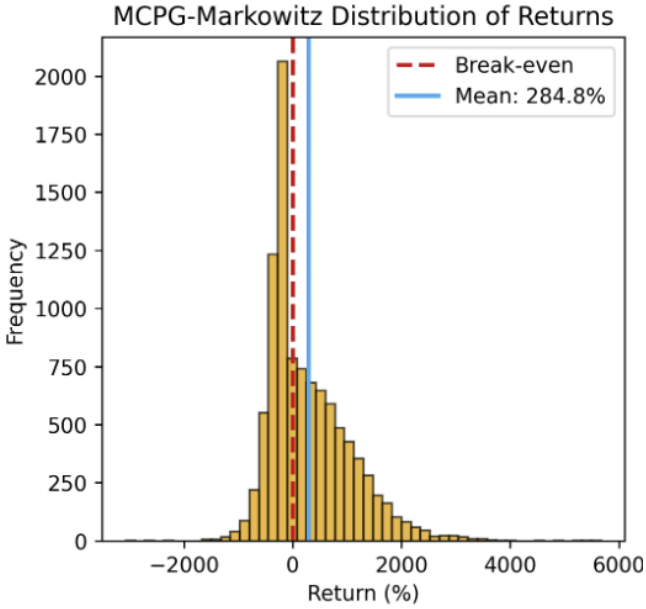

MCPG-Entropic Model vs Optimal Performance (250-episode window)
Model MA(250)
Max MA(250)
Min MA(250)
Return (%)
Episode

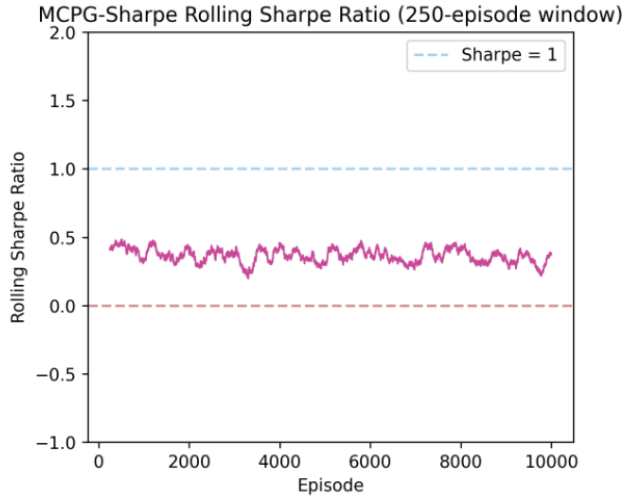

MCPG-Sharpe Distribution of Returns
Break-even
Mean: 294.7%
Frequency
Return (%)

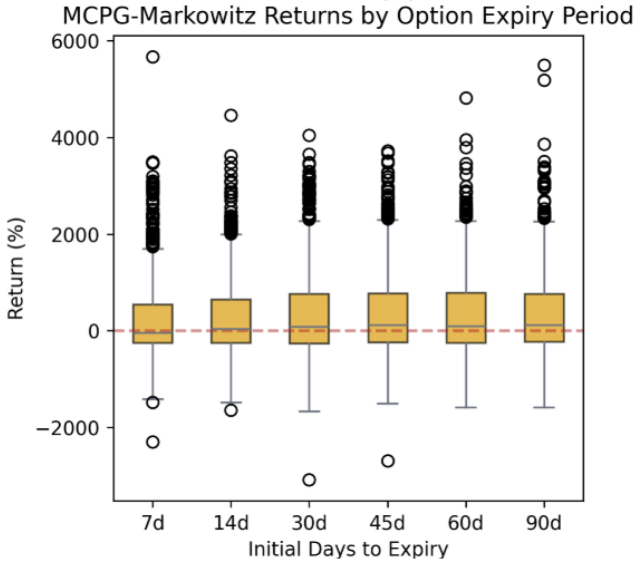

MCPG-Sharpe Returns by Option Expiry Period
Return (%)
6000
5000
4000
3000
2000
1000
0
−1000
−2000
7d
14d
30d
45d
60d
90d
Initial Days to Expiry


MCPG-Markowitz P&L Components (as % of Premium)
% of Premium Paid
400
300
200
100
0
−100
332.2%
43.2%
275.4%
Option Payoff
Hedging P&L
Net Return

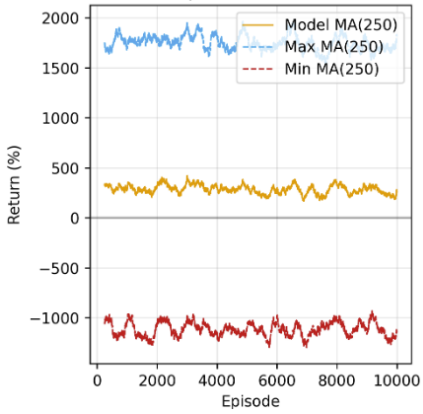

MCPG-Sharpe Model vs Optimal Performance (250-episode window)
Model MA(250)
Max MA(250)
Min MA(250)
Return (%)
2000
1500
1000
500
0
−500
−1000
−1500
0
2000
4000
6000
8000
10000
Episode


MCPG-Markowitz Distribution of Returns
Break-even
Mean: 284.8%
Frequency
2000
1750
1500
1250
1000
750
500
250
0
−2000
0
2000
4000
6000
Return (%)


MCPG-Sharpe Rolling Sharpe Ratio (250-episode window)
Sharpe = 1
Rolling Sharpe Ratio
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0
2000
4000
6000
8000
10000
Episode


MCPG-Markowitz Returns by Option Expiry Period
Return (%)
6000
4000
2000
0
−2000
7d
14d
30d
45d
60d
90d
Initial Days to Expiry

MCPG-Markowitz Model vs Optimal Performance (250-episode window)



MCPG-Markowitz Rolling Sharpe Ratio (250-episode window)