Deep Hedging for Deep OTM Options: A Multi-Asset Approach in Bates Jump-Diffusion Environments

24167356 [1]

Abstract

Hedging remains an ever-complicating problem in the financial securities industry, especially for deep out-of-the money (OTM) options where traditional models fail. We present a framework for hedging a portfolio of deep OTM European Calls and Puts, considering transactions costs, liquidity constraints, cross-correlations, and jumps. Under a market generated from a Bates stochastic volatility model, we find that a deep Reinforcement Learning (RL) agent significantly outperforms the classic Black–Scholes delta-neutral hedging strategy by 56.89% in VaR, 74.82% in CVaR, 76.79% in max drawdown, 89.41% in P&L, and 81.62% in utility.

1. Introduction

1.1. Motivation

Hedging is the action of purchasing or selling units of a financial instrument to neutralize the risk associated with a held financial asset. Hedging a portfolio of assets remains a pivotal challenge in risk-management due to the simplified assumptions made in traditional models. In idealized efficient and frictionless markets, quantitative finance thrives, developing tractable solutions where traders have commonly hedged the risk associated with derivatives by monitoring the "Greeks."

These measure the sensitivities of the value of the derivative product to changes in a particular parameter—such as implied volatility, interest rate levels, the price of the underlying—while holding the other parameters fixed. The Greeks are given by the closed-form solution of the famous equation developed by Black and Scholes in 1973. One common strategy includes delta-neutral hedging, wherein traders purchase or short a certain amount of the underlying security in accordance with the first derivative of the option value with respect to the underlying price. As theory suggests, if there are no transaction costs or other frictions, it is optimal to rebalance this position continuously. In other words, if the market was ideal, traders could adjust their positions as frequently as they desired and with no costs, thereby completely eliminating the risk associated with holding the derivative.

In real markets, however, trading in any instrument is subject to transaction costs, liquidity constraints, and substantial market shocks. Furthermore, the assumptions underlying classical models break down significantly in extremely volatile markets, such as cryptocurrencies, where jumps are frequent and liquidity can be highly variable. In particular, deep out-of-the-money (OTM) options are challenging to hedge; their Greeks become unreliable due to extreme sensitivity to model assumptions, numerical instability with low option values, and their inability to capture jump risks and volatility skew. In such cases, the conventional delta-hedging approach often fails to capture the true risk profile of hedging these products.

This paper demonstrates how reinforcement learning can be used to derive optimal hedging strategies with derivatives when considering deep OTM options in realistic market conditions; specifically, when taking into account transaction costs, liquidity constraints, jumps, and, most importantly, cross-correlations between assets. In cryptocurrency markets, where tail risks and non-linear effects from significant shocks—such as SEC approved spot ETFs—are pronounced, a deep hedging framework is particularly valuable. Utilizing this structure helps answer the question: *How can we effectively hedge the risks of a portfolio of deep OTM options in volatile markets, where traditional models often struggle?*

1.2. Background

Seminal deep hedging research was conducted by Hans Buehler, the former Global Head of Equities Analytics, Automation and Optimization (AAA) at J.P. Morgan. In 2018, Buehler, alongside several researchers from the University of St Gallen and the University of Zurich, analyzed how a reinforcement learning agent placed in a market simulated by the Heston model outperformed "complete" traditional models (Buehler et al., 2018). This coterie built upon their study in 2022, transitioning to an actor-critic-type reinforcement learning model, citing the use of a continu-

[*]UCL: 24167356

ous state Markov Decision Process (MDP) (Buehler et al., 2022). Most significantly, this newly trained model was able to provide an optimal hedge for arbitrary initial portfolios.

Subsequent research aimed to address the anomalies that Buehler et al. had not. Neagu et al. (2024), for example, extended the existing RL framework to account for market impacts from finite liquidity, and Mikkilä and Kanniainen (2023) adapted the original model by gathering inordinate historical intra-day data to run the deep RL agent—instead of relying on synthetic data. While Mikkilä and Kanniainen demonstrated superior performance using historical data, they solely focused on hedging with a single instrument, suggesting the complexities of hedging a portfolio of assets, as addressed by Buehler et al., requires even more synthetic data.

Cao et al. go on to contrast Buehler's simulation of a market in a Heston environment with one constructed by a Geometric Brownian Motion. Additionally, they extend the basic RL standard by incorporating Q-functions, in order to consider both the expected value of the cost and the square root of the cost for various state/action combinations. It is worth noting, as in Cao et al. (2019), traditional delta hedging is widely used as a benchmark in the literature, even though it typically involves trading the underlying asset. Therefore, we will incorporate it as our baseline.

The inspiration for deep hedging is often tied to option pricing, wherein deep learning solves issues related to the dimensionality of traditional methods, like the Finite Difference Method and Finite Elements. Carbonneau and Godin (2020), for example, described a methodology based on equal risk pricing, which equates the residual risk exposure of long and short positions in a derivative after optimal hedging. This framework inherently ties pricing and hedging together, as the price of the derivative depends on the cost and effectiveness of hedging it. This extension has garnered widespread attention, as demonstrated in studies by Noguer i Alonso and Haida (2024) and Bolfake et al. (2023).

Beyond the constraints of traditional financial models, one might wonder why simpler statistical measures are ineffective, coined as "Statistical Hedging" by Buehler. While useful for straightforward conditions, statistical hedging, such as linear or simple regression models, struggle when used on portfolios with multiple interacting risk factors. Such problems require models that can learn intricate patterns, leading to our use of Machine Learning. Furthermore, Reinforcement Learning is particularly well-suited as hedging is a sequential decision-making problem, with each trading action affecting future states and opportunities. RL excels by learning optimal policies over time rather than making single-point predictions.

2. Methodology

In dynamic hedging, the problem is inherently sequential and stochastic. The agent must decide, at each time step, how many units of each hedging instrument to trade based on continuously changing market states. Reinforcement learning (RL) provides a framework where the agent learns directly from interacting with the market simulator. It captures complex dependencies and optimizes the overall risk-adjusted profit-and-loss (P&L) even in the presence of frictions like transaction costs and liquidity constraints.

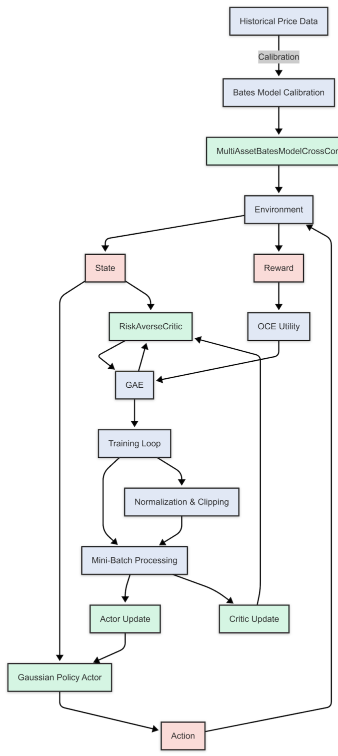

Figure 1. Deep Hedging Reinforcement Learning Flow Chart.

Initial experiments led us to explore policy-based reinforcement learning approaches. While our goal of directly optimizing a trading strategy naturally suggested policy gradient methods:

$$\mathcal{L} = \log \pi(a|s) \tag{1}$$

We discovered that basic policy gradient methods (like REINFORCE)—as suggested in prior literature—performed poorly due to high variance in gradient estimates when learning from complete hedging episodes. To address this limitation, we implemented an actor-critic architecture with Generalized Advantage Estimation (GAE), which introduces a second network—the critic—to estimate the value function V(s) (Schulman et al., 2016).

Rather than imposing a terminal maturity, the formulation of our problem is more open-ended, thereby infinitehorizon in spirit. As decisions are made iteratively without a hard terminal boundary, this is representative of a discrete-time MDP. We employ a Bellman Equation to guide the critics target, while having an Optimized Certainty Equivalent (OCE) define our risk-adjusted return. Additionally, our model follows the flow chart in figure 1, which can be useful for reference throughout this methodology section.

2.1. Environment and Transitions

In this multi-asset deep hedging problem, the agent aims to dynamically hedge a portfolio of cryptocurrency assets, including BTC, ETH, and LTC, by trading in 12 hedging instruments—2 European Calls and 2 Puts for each asset.

The agent operates within a discrete-time MDP, making decisions at fixed intervals over 60 time steps per episode, with each step representing a day. The MDP framework defines the problem structure and is formally represented as:

$$\text{MDP} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma) \tag{2}$$

where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition probability function, $\mathcal{R}$ is the reward function, and γ is the discount factor. At each time step $t = 0, 1, \ldots, 59$, the agent observes the state $s_t = (z_t, m_t)$—where z_t represents the portfolio's mark-to-market value, computed using the Black-Scholes formula for each instrument, and m_t includes market observables—takes an action a_t, receives a reward r_t, and transitions to the next state $s_{t+1} = (z_{t+1}, m_{t+1})$, following the Markov property. Each component of the MDP will be discussed in detail in the subsequent sections.

2.2. State

The state $s_t = (z_t, m_t)$ comprises the portfolio value $z_t \in \mathbb{R}$ and the market state $m_t \in \mathbb{R}^{d_m}$, with $d_m \geq 6$, including spot prices and volatilities for three assets (BTC, ETH, LTC), the risk-free rate and the time to maturity. The market state m_t evolves stochastically, driven by the Bates model, which captures the dynamics of cryptocurrency markets. These dynamics introduce uncertainty through stochastic volatility and jumps, which the agent must mitigate through its trading decisions.

The Bates model combines the Heston stochastic volatility model with Merton's jump-diffusion model. The spot price S_t of each asset follows the stochastic differential equation (SDE):

$$\frac{dS_t}{S_t} = \mu\, dt + \sqrt{v_t}\, dW_t^S + (e^{J_t} - 1)\, dN_t \tag{3}$$

$$dv_t = \kappa(\theta - v_t)\, dt + \sigma\sqrt{v_t}\, dW_t^V \tag{4}$$

where μ is the drift, v_t is the variance process, W_t^S and W_t^V are Brownian motions with correlation ρ, N_t is a Poisson process with jump intensity λ_j, and $J_t \sim \text{Lognormal}(\mu_j, \sigma_j^2)$ is the lognormal jump size. The variance process follows a mean-reverting square-root diffusion with mean reversion speed κ, long-run variance θ, and volatility-of-volatility σ.

The Bates model does not have a closed-form probability density function (PDF) for log-returns $r_t = \ln(S_{t+1}/S_t)$; therefore, in order to get accurate measurements for its parameters we utilized its characteristic function $\phi(u) = \mathbb{E}[\exp(iu\ln(S_{t+\Delta t}/S_t))]$ which can be derived in closed form. The characteristic function for the Bates model is:

$$\phi_{\text{Bates}}(u) = \phi_{\text{Heston}}(u) \cdot \exp\left(\lambda_j \Delta t\left(\exp\left(iu\mu_j - \frac{1}{2}\sigma_j^2 u^2\right) - 1\right)\right) \tag{5}$$

where $\phi_{\text{Heston}}(u)$ is the Heston characteristic function. To calibrate the model, we compute the PDF of log-returns using the Fast Fourier Transform (FFT). First, we evaluate $\phi_{\text{Bates}}(u)$ on a frequency grid v, utilizing the Nyquist relation, where $v_j = j \cdot \Delta v$, $\Delta v = 2\pi/(N\Delta t)$, $N = 64$, and $\Delta t = 1/252$ (daily steps). The corresponding spatial grid for log-returns is $k_j = -\pi/\Delta t + j \cdot \Delta k$, with $\Delta k = 2\pi/(N\Delta v)$. The PDF is then obtained by FFT:

$$p(k) \approx \text{Re}\left(\text{FFT}(\phi_{\text{Bates}}(v))\right) \tag{6}$$

and interpolated to estimate the density at specific log-return values r_t. To find the maximum likelihood estimation (MLE), we equivalently minimize the negative log-likelihood (NLL):

$$\text{NLL} = -\sum_t \ln p(r_t) \tag{7}$$

using the L-BFGS-B optimization algorithm. This process leverages historical price data for each asset to estimate the model parameters $(\kappa, \theta, \sigma, \rho, \lambda_j, \mu_j, \sigma_j, r, \mu)$ and the initial variance v_0, ensuring the model fits the observed log-return distributions.

Following single-asset calibration, we construct a $2N \times 2N$ correlation matrix (where $N = 3$ for three assets) to capture cross-asset dependencies. This matrix incorporates correlations between asset returns (W_i^S and W_j^S), variance processes (W_i^V and W_j^V), and leverage effects (between W_i^V and W_i^S for the same asset), derived from historical price data using a rolling window approach. The matrix is adjusted to be symmetric and positive semi-definite, ensuring valid stochastic simulations.

These calibrated Bates parameters are then used to simulate future market states m_t, generating realistic trajectories of spot prices and volatilities that reflect the stochastic volatility and jump behavior observed in historical cryptocurrency data.

2.3. Actor-Critic Model

As previously mentioned, our approach employs an **on-policy** reinforcement learning method, specifically an actor-critic framework with Generalized Advantage Estimation (GAE). On-policy methods learn directly from the current policy's experience, ensuring that the collected data reflects the policy being optimized.

This contrasts with off-policy methods, which can leverage past experiences collected under different policies, potentially improving sample efficiency but introducing complexity in our risk-averse setting with OCE. We chose an on-policy method to maintain stability in policy updates, as the OCE utility introduces non-linear transformations of rewards, making off-policy corrections challenging.

To handle our continuous action space, the agent employs a multivariate Gaussian policy, implemented as `GaussianPolicyActor`. The action space is continuous as the agent can trade (buy or sell) a real number of units of each instrument. The actor uses an LSTM-based architecture to capture temporal dependencies in the sequential market states. The LSTM cell processes the state $s_t = (z_t, m_t)$ at each time step, maintaining a hidden state h_t and cell state c_t.

These hidden states condition the policy on the history of states, not just the current state, enabling the agent to adapt to evolving market dynamics. The LSTM output is passed through two linear heads: one for the mean ($\mu \in \mathbb{R}^{12}$) and one for the log standard deviation ($\log \sigma \in \mathbb{R}^{12}$) of the action distribution. The action a_t is then sampled from a multivariate Gaussian $\mathcal{N}(\mu, \Sigma)$, where $\Sigma = \text{diag}(\exp(\log \sigma))$ is a diagonal covariance matrix. This suggests the uncertainties are independent per instrument, though the LSTM's hidden state indirectly captures correlations across instruments.

Correspondingly, our model uses a "critic", which is a network that approximates the value function $V^\pi(s)$. In a risk-neutral setting, the Bellman equation gives us:

$$V(s) = \sup_a \mathbb{E}\Big[r + \gamma V(s')\Big] \tag{8}$$

where (r) is the immediate reward after taking action (a) in state (s), (γ) is the discount factor, and (s') is the next state. This recursive equation expresses the value of a state as the expected sum of immediate reward and discounted future value.

However, in our risk-averse, OCE setting (like the Buehler et al. (2022) approach), the Bellman operator uses a monetary utility U, becoming:

$$V^*(s) = \sup_a U\Big[\beta V^*(s') + r\Big] \tag{9}$$

The risk-averse critic, implemented as `RiskAverseCritic`, is thus a neural network—guided by a mean-squared erro (MSE) loss function—that (1) inputs the current state s and (2) outputs an approximation of $V^*(s)$ under this risk measure. It is a feedforward network with three layers (input $\rightarrow$ hidden $\rightarrow$ hidden $\rightarrow$ output), using ReLU activations to map the flattened state to a scalar $V^*(s)$. Accordingly, instead of approximating $\mathbb{E}$, it approximates the $\sup_a U[\cdot]$ form, which we incorporate parametrically by applying the OCE Utility U to the returns during training.

Specifically, the `oce_utility` function transforms rewards using an exponential utility $U(x) = -\exp(-\lambda x)$ embedding risk aversion into the value function. We, additionally, test the use of a power function $U(x) = \frac{|x|^{1-\gamma}}{1-\gamma} \cdot \text{sign}(x)$.

2.4. Actions and Policy

In our multi-asset hedging environment, the agent's actions represent the number of units to trade for each hedging instrument. Specifically, the action vector $a_t \in \mathbb{R}^{12}$ corresponds to 12 instruments (call and put options), where each coordinate $a_t[i]$ denotes the number of units to trade for instrument i. Positive values indicate buying, while negative values indicate selling.

The action space is continuous, as the agent can trade any real number of units, subject to a liquidity constraint: the absolute position for each instrument must not exceed a `liquidity_limit` of 100.0 (i.e. the max number of units traded), with a `big_penalty` applied for violations, as implemented in `MultiAssetHedgingEnv`.

A Gaussian policy is chosen because it handles continuous action spaces and balances exploration and exploitation. Early in training, the standard deviation ($\sigma = \exp(\log \sigma)$) is larger, encouraging exploration by sampling actions that deviate more from the mean, useful for our stochastic market dynamics.

Additionally, the Gaussian policy enables gradient-based optimization through reparameterization. Instead of directly sampling $a \sim \mathcal{N}(\mu, \sigma)$, we sample a noise vector $\epsilon \sim \mathcal{N}(0, I)$ and compute the action as $a = \mu + \sigma \odot \epsilon$. This reparameterization makes the sampling process differentiable with respect to μ and σ—unlike direct sampling from a Gaussian policy, which breaks gradient flow— allowing gradients to propagate backward during optimization. Consequently, we can sample actions differentiably during training, as in `train_gae.py` when computing the actor loss: $-\text{advantage} \times \log \text{prob}(a|s)$.

To summarize, our training process uses an on-policy rollout, meaning the agent interacts with the environment using its current policy to collect transition sequences. Each episode consists of 60 time steps (days), during which the agent generates a trajectory while updating the LSTM's hidden states to maintain sequential dependencies in the trading process.

These new trajectories are used to compute advantages through Generalized Advantage Estimation (GAE), which are then used to update the policy (5 updates per episode, leading to 1000 update steps) and critic networks in `train_gae.py`. By trajectories, we are referring to the sequences of states, actions, rewards, and next states that the agent collects while interacting with the environment during an episode, represented as $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T, s_{T+1})$.

2.5. Bellman-Equations and Risk Measures

With each action $a_t \in \mathbb{R}^{12}$ the environment transitions to (z_{t+1}, m_{t+1}) through a known rule:

$$\begin{aligned} z_{t+1} &= \text{MtM}\big((z_t, a_t), m_{t+1}\big) \qquad (10) \\ m_{t+1} &\sim P\big(\cdot \mid m_t\big) \end{aligned}$$

where $\text{MtM}(\cdot)$ is the mark-to-market function that updates the portfolio value based on the new positions after applying a_t, and $P(\cdot \mid m_t)$ is the Bates model's distribution for the next-step market state.

The MtM function computes the value of each instrument using the Black-Scholes formula, assuming a dividend yield $q = 0$, which is reasonable for cryptocurrency options as they typically do not pay dividends. Specifically, for a European call option with strike K, time to maturity $T - t$, spot price S_t (from m_t), volatility $\sqrt{v_t}$ (from m_t), and risk-free rate r, the price is:

$$C(S_t, K, T-t, v_t, r) = S_t \Phi(d_1) - K e^{-r(T-t)} \Phi(d_2) \qquad (11)$$

where Φ is the standard normal CDF, and d_1 and d_2 follow standard definitions. The portfolio value z_t is the sum of the values of all positions, weighted by their quantities.

The reward $R\big(a_t;\, s_t,\, s_{t+1}\big)$ is the one-step profit-and-loss (P&L) from holding z_t and adding trades a_t, minus transaction costs and penalties:

$$\begin{aligned} &R(a_t;\, z_t, m_t,\, z_{t+1}, m_{t+1}) = \\ &\Delta\text{MtM}\big(z_t, a_t, m_t,\, m_{t+1}\big) - \text{Cost}\big(a_t;\, m_t\big) - \text{Penalty}\big(a_t\big) \qquad (12) \end{aligned}$$

where $\Delta\text{MtM} = z_{t+1} - z_t$, Cost is the transaction cost, and Penalty is a large penalty for exceeding the liquidity limit (`liquidity_limit=100.0`).

With a discount factor $\beta = 0.99$ and a concave, monotone utility operator U, the **Bellman equation** for the risk-averse value function $V^*(z, m)$ is below. This takes the general form we gave before in the actor-critic model section, adapting it to our specific state and reward conditions:

$$\begin{aligned} V^*(z, m) = \sup_{a \in A(z,m)} \; & U\Big[\beta V^*\big(z_{t+1},\, m_{t+1}\big) \\ & + R\big(a;\, z, m, z_{t+1}, m_{t+1}\big)\Big] \end{aligned} \qquad (13)$$

This yields a continuous-state, continuous-action MDP. Our goal is to approximate the optimal policy π^* and value V^* using an actor-critic algorithm in PyTorch, with the actor (`GaussianPolicyActor`) and critic (`RiskAverseCritic`).

To train the critic, we use Generalized Advantage Estimation (GAE), which builds on Temporal-Difference (TD) learning. The Critic (`RiskAverseCritic`) estimates the risk-adjusted value function $V^*(s)$, providing $V(s_t)$ and $V(s_{t+1})$ to compute TD errors:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \qquad (14)$$

GAE then calculates the advantage:

$$\hat{A}_t = \delta_t + \gamma \lambda \hat{A}_{t+1} \qquad (15)$$

with $\gamma = 0.99$, $\lambda = 0.95$, and r_t as the OCE-transformed reward (through `oce_utility`, e.g., exponential utility).

This advantage informs the Actor (`GaussianPolicyActor`) by indicating how much better an action is compared to the expected value, guiding policy updates via the loss $\mathcal{L}_{\text{actor}} = -\hat{A}_t \log \pi(a_t|s_t)$.

Simultaneously, the Critic uses $\hat{A}_t$ to compute target returns, $\text{return}_t = V(s_t) + \hat{A}_t$, minimizing its MSE loss to refine $V^*(s)$. GAE thus acts as a bridge, leveraging the Critic's values to produce stable advantages for the Actor while supporting the Critic's learning, balancing variance and bias by smoothing TD errors over multiple steps.

To derive the policy gradient for the actor, we use the Policy Gradient Theorem, optimizing the policy $\pi_\theta(a_t|s_t)$ to maximize the expected cumulative discounted reward:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T-1} \gamma^t r_t\right]$$

where $r_t \sim \mathcal{R}(s_t, a_t, s_{t+1})$, $\gamma = 0.99$, $T = 60$ is the episode length, and $\tau = (s_0, a_0, r_0, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$ are trajectories sampled from the policy π_θ.

The Policy Gradient Theorem provides a way to compute the gradient of $J(\theta)$ with respect to the policy parameters

θ, allowing us to update π_θ in the direction that increases the expected reward. Specifically, the theorem states that the gradient is proportional to the expected value of the score function $\nabla_\theta \log \pi_\theta(a_t|s_t)$, weighted by the action-value function, yielding:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) Q^\pi(s_t, a_t)\right]$$

where the state-action value function $Q^\pi(s_t, a_t)$ represents the expected cumulative discounted reward starting from state s_t, taking action a_t, and following π_θ thereafter:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{k=t}^{T-1} \gamma^{k-t} r_k \mid s_t, a_t\right]$$

Consequently, the gradient $\nabla_\theta J(\theta)$ increases the probability of actions a_t that lead to higher expected returns, as measured by $Q^\pi(s_t, a_t)$. Estimating Q^π directly is difficult due to future reward dependence. Therefore, we define the advantage function to stabilize learning:

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

where the state-value function $V^\pi(s_t)$ is the expected cumulative discounted reward starting from state s_t:

$$V^\pi(s_t) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{k=t}^{T-1} \gamma^{k-t} r_k \mid s_t\right]$$

Using the advantage function, the policy gradient becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A^\pi(s_t, a_t)\right]$$

This shifts the problem to estimating $A^\pi(s_t, a_t)$ efficiently. A common method uses the TD (Temporal Difference) error:

$$\delta_t = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$$

which shows how much better or worse an action was relative to the expected value of the state. Single-step TD has high variance, so we use a multi-step approach to smooth the estimation, leading to Generalized Advantage Estimation (GAE). GAE computes the advantage as:

$$\hat{A}_t^{\text{GAE}}(\lambda) = \sum_{l=0}^{T-1-t} (\gamma\lambda)^l \delta_{t+l}$$

where $\lambda \in [0, 1]$ trades off bias and variance ($\lambda = 1$: Monte Carlo, low bias, high variance; $\lambda = 0$: one-step TD, high bias, low variance). Expanding the sum explicitly:

$$\hat{A}_t^{\text{GAE}}(\lambda) = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots + (\gamma\lambda)^{T-1-t}\delta_{T-1}$$

and substituting $\delta_t = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$:

$$\begin{aligned}\hat{A}_t^{\text{GAE}}(\lambda) = (r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)) + (\gamma\lambda)(r_{t+1} \\ + \gamma V^\pi(s_{t+2}) - V^\pi(s_{t+1})) + \cdots + (\gamma\lambda)^{T-1-t}(r_{T-1} + \\ \gamma V^\pi(s_T) - V^\pi(s_{T-1}))\end{aligned}$$

smoothing the advantage over multiple steps to reduce variance while capturing long-term effects. The final policy gradient with GAE is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t^{\text{GAE}}(\lambda)\right] \quad (16)$$

This gradient guides policy optimization by weighting policy adjustments with GAE advantages. In training, we approximate this expectation using sampled trajectories, implementing the gradient through the actor's loss $\mathcal{L}_{\text{actor}} = -\hat{A} \log \pi(a|s)$, where $\hat{A} = \hat{A}_t^{\text{GAE}}$, as detailed in Section 2.6.

2.6. Training

The OCE framework is implemented in `train_gae.py` via the `oce_utility` function, which applies a risk-averse utility to transform rewards during GAE computation and the critic's MSE loss calculation.

Notably, OCE is not included in the environment's raw rewards, but it indirectly shapes the agent's strategy by adjusting the learning objective to prioritize risk-adjusted returns. This separation ensures that the environment provides an unbiased performance signal, while the OCE utility aligns the policy with practical financial hedging strategies that account for risk.

The actor optimizes the policy gradient loss weighted by the advantage function:

$$\mathcal{L}_{\text{actor}} = -\hat{A} \log \pi(a|s). \quad (17)$$

To leverage the collected trajectory data efficiently, training employs mini-batches, which are integral to the learning process. After each episode, the agent gathers a full trajectory of transitions (s_t, a_t, r_t, s_{t+1}).

This trajectory is then divided into mini-batches (size= 32), as implemented in `perform_training_updates`. These mini-batches are created by shuffling the trajectory to break temporal correlations, approximating independent and identically distributed (i.i.d.) samples. The training process then performs 5 updates per episode across 200 episodes, totaling 1000 update steps, with each update iterating over the mini-batches.

During each update, the mini-batches are used to compute gradients for both the actor and critic networks.

For the critic, the mini-batches provide state-value pairs $(s_t, V^*(s_t))$ and target returns (return$_t = V(s_t) + \hat{A}_t$), computed using GAE with OCE-transformed rewards. The critic's MSE loss, $\mathcal{L}_{\text{critic}} = (\text{v\_pred} - \text{return})^2$, is minimized using these mini-batch gradients, updating the critic's weights to better approximate $V^*(s)$. For the actor, the mini-batches supply state-action pairs (s_t, a_t) and normalized advantages ($\hat{A}_t$), which are used to compute the policy gradient loss 17. The actor's weights are updated to maximize this loss, refining the Gaussian policy. This is all summarized in Algorithm 1(28).

Algorithm 1 Training for Risk-Averse Deep Hedging

```
Input: Episodes $E = 200$, Updates $U = 5$, Time steps $T = 60$, Mini-batch size $M = 32$, Discount factor $\gamma = 0.99$, GAE parameter $\lambda = 0.95$, OCE risk parameter $\lambda_{\text{OCE}} = 0.5$
Initialize: Actor (GaussianPolicyActor) and Critic (RiskAverseCritic) networks with parameters $\theta$ and $\phi$, respectively
for episode $e = 1$ to $E$ do
    Collect trajectory $\tau = \{(s_t, a_t, r_t, s_{t+1})\}_{t=0}^{T-1}$ using $\pi_\theta$
    Transform rewards: $r_t \leftarrow \text{oce\_utility}(r_t, \lambda_{\text{OCE}})$
    Compute value estimates $V(s_t)$ for all $t$ using Critic
    Compute GAE advantages $\hat{A}_t$ for all $t$:
    Initialize $\hat{A}_T = 0$
    for $t = T - 1$ down to 0 do
        $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$
        $\hat{A}_t = \delta_t + \gamma\lambda\hat{A}_{t+1}$
    end for
    Compute target returns: $\text{return}_t = V(s_t) + \hat{A}_t$
    Shuffle trajectory $\tau$ and split into mini-batches of size $M$
    for update $u = 1$ to $U$ do
        for each mini-batch $\{(s_i, a_i, r_i, s_{i+1}, \hat{A}_i, \text{return}_i)\}$ do
            Standardize states $s_i$ to zero mean, unit variance
            Normalize advantages $\hat{A}_i$ within mini-batch
            Compute critic prediction: $\text{v\_pred}_i = V_\phi(s_i)$
            Compute critic loss: $\mathcal{L}_{\text{critic}}$
            Update critic: $\phi \leftarrow \phi - \alpha_c \nabla_\phi \mathcal{L}_{\text{critic}}$
            Compute policy log-prob: $\log \pi_\theta(a_i|s_i)$
            Compute actor loss: $\mathcal{L}_{\text{actor}}$
            Update actor: $\theta \leftarrow \theta - \alpha_a \nabla_\theta \mathcal{L}_{\text{actor}}$
        end for
    end for
end for
Output: Trained actor and critic networks
```

To ensure numerical stability and efficient training in `train_gae.py`, several normalization and clipping techniques are applied. In `perform_training_updates`, states (s_t) are standardized to have zero mean and unit variance across each mini-batch. This ensures scale consistency across features, such as portfolio values (z_t) and market observables (m_t). GAE values ($\hat{A}_t$) are normalized within each mini-batch to maintain policy gradient stability, preventing extreme values from causing unstable policy updates.

Returns (return$_t = V(s_t) + \hat{A}_t$) and advantages are scaled by 0.001 before training to prevent large gradients in the actor and critic losses, ensuring stable updates given small learning rates (`actor_lr=0.0001`, `critic_lr=1e-6`). Episode returns and baseline P&L are normalized by their maximum absolute values, ensuring inputs to `oce_utility` remain within a reasonable scale, preventing overflow in exponential utility computations.

We also incorporated clipping in `oce_utility`. This resulted in clipped P&L and reward values to $[-20.0, 20.0]$ before computing exponential utility to prevent numerical overflow. In `perform_training_updates`, gradients of the actor and critic networks are clipped to an L2 norm of `max_grad_norm=0.5`, preventing exploding gradients from the LSTM and noisy financial rewards.

2.7. Data

We pulled historical cryptocurrency spot price data from Gemini, focusing on Ethereum (ETH), Bitcoin (BTC), and Litecoin (LTC). The SEC approvals of spot ETFs for ETH and BTC highlight their regulatory maturation, while LTC's longevity and liquidity completed our basket. These cryptocurrencies exhibit substantial tail risks and shock events, making them ideal for testing our jump-diffusion modeling approach.

This basket also provides diverse correlation patterns—BTC as the market leader, ETH representing the smart contract ecosystem, and LTC as a long-standing altcoin. While we used minute-level data, the freely accessible market data alone was insufficient for training our model.

To address this, we generated synthetic data through the Bates model calibrated with the basket's historical data sets. Additionally, we incorporated 12 hand-selected deep OTM options from Deribit, specifically chosen to represent scenarios where conventional models break down due to their assumptions about normally distributed returns.

3. Results

3.1. Performance

The agent was placed through a grid-search over hyperparameters (actor/critic learning rates, γ, λ, utility type), selecting the best configuration based on performance on a held-out validation set. Conclusively, we found a constant absolute risk aversion approach through an exponential utility function, with a slightly lower discount factor (0.95), lower actor learning rate (1×10^{-4}), and lower critic learning rate (1×10^{-6}) returned the best performance (Table 1).

With the hyperparameters selected, we went on to confirm convergence in our actor loss (blue) and critic loss (orange)

Model	Actor LR	Critic LR	Gamma	Lambda	Utility	Final Reward
Model 1	0.0001	0.000001	0.95	0.5	exp	**137.06**
Model 2	0.0030	0.000001	0.95	1.0	power	136.76
Model 3	0.0001	0.003000	0.99	0.5	exp	136.39
Model 4	0.0001	0.000001	0.99	1.0	power	136.05
Model 5	0.0001	0.003000	0.99	1.0	power	135.13

Table 1. Comparison of different DRL Hyperparameters.

which stabilized over 1000 update steps. These stabilizing losses support the reliability of the trained policy. Actor loss variability reflects the response to the stochastic environment and exploration, while critic spikes align with known challenges in difficult value estimation (Buehler et al., 2022). Stability in the trend validates the policy.

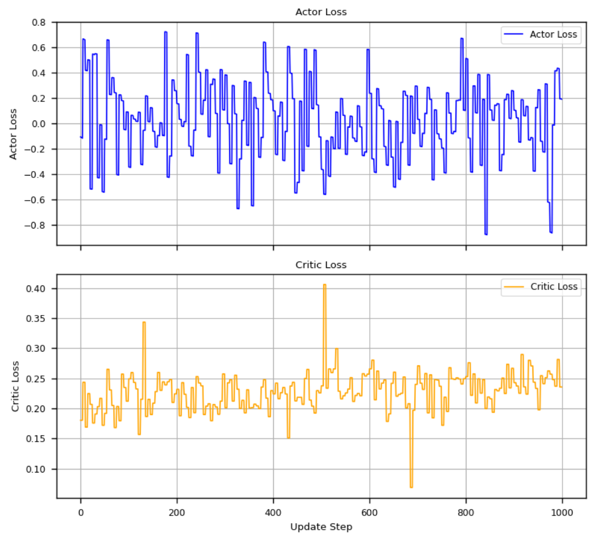

Figure 2. Actor and Critic Losses over 1000 gradient updates of the networks.

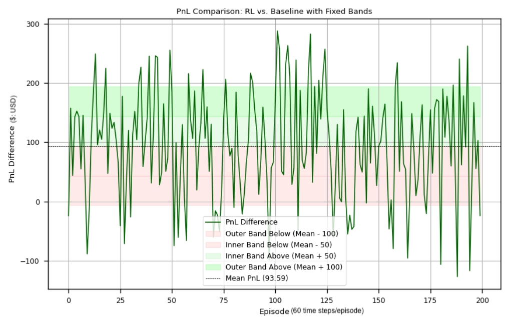

Figure 3. P&L Comparison, including bands at ±$50 and ±$100.

The impressive performance of the agent's policy can be confirmed by looking at the P&L graph in figure 3 and the relative utility gain graph in figure 4. The P&L difference (green) fluctuates around a mean of 93.59, with most values

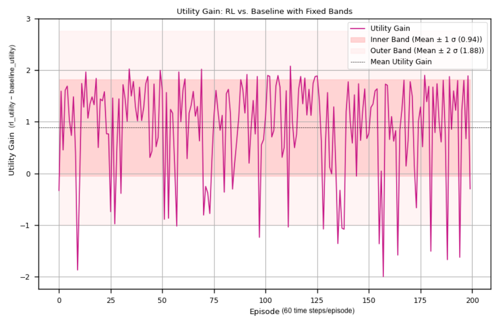

Figure 4. Utility gains with bounds ±1 and ±2 standard deviations centered around the mean.

between -50 and +100 (inner bands). The RL agent occasionally outperforms the baseline significantly (e.g., +200) and rarely underperforms by more than -100. The positive mean P&L difference indicates the RL agent generally achieves higher profits than the baseline, confirming outperformance on average.

Additionally, the utility gain (magenta) fluctuates around a mean of 0.94, mostly within ±1 (inner band) and rarely exceeding ±1.88 (outer band). Positive utility gain reflects the OCE-adjusted performance advantage. Given the RL agent's positive utility gain over the baseline, which is consistent with the OCE objective, we can confirm an improvement of 89.41% in the average reward and 81.62% in the average utility.

3.2. Hedging Policy Comparison

One of the key advantages of deep hedging is that it does not rely on analytical Greeks or model assumptions. The agent implicitly learns the appropriate sensitivities to various risk factors without needing to calculate traditional Greeks.

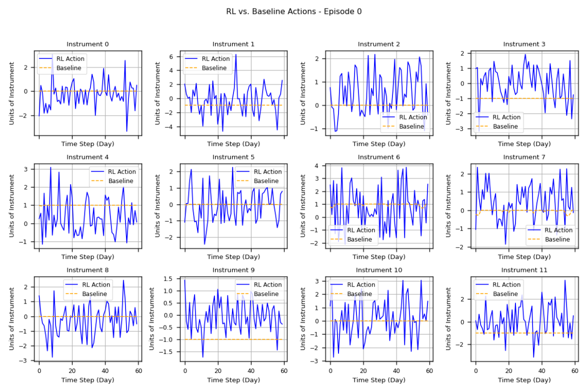

Figure 5. RL Agent and Delta-Hedging Agent actions across the 4 different products for the 3 underlying assets.

As in 5, even for the most volatile position in an instrument, the baseline delta-hedging agent exhibited minimal reaction to cross-correlations, stochastic volatility, and transaction costs compared to the RL agent. Its lack of adaptability can be explained by the underlying theory. Instrument 0, for example, is a call option on BTC with a strike price $K = 91585$, while the spot price S starts at 51873.09 and fluctuates around 51000–52000. This makes the option deep out-of-the-money (OTM) because $S \ll K$. The Black-Scholes delta for a call option is given by $\Delta = N(d_1)$, where:

$$d_1 = \frac{\ln(S/K) + (r + \sigma^2/2)\tau}{\sigma\sqrt{\tau}}. \tag{18}$$

Using typical values $S = 51873.09$, $K = 91585$, risk-free rate $r = 0.02$, volatility $\sigma = 0.3$, and time to maturity $\tau = 0.1$, we compute:

$$d_1 = \frac{-0.569 + 0.0065}{0.0948} \approx -5.93. \tag{19}$$

For a deep OTM call option ($S \ll K$), the probability of finishing in-the-money is very low, resulting in $N(d_1) = \text{CDF}(-5.93) \approx 0$. This concludes that $\Delta \approx 0$, aligning with the observed baseline action of 0.0, even as S, σ, and τ fluctuate over episodes.

The RL agent is highly reactive because it is trained to optimize P&L directly, considering not just Δ but also higher-order effects, jumps, cross-correlations, and stochastic volatility. It can take larger, more dynamic positions (e.g., -2.599 to 2.088 for Instrument 0), even for a deep OTM option, because it learns to hedge beyond the Black-Scholes framework.

We can see the difference in hedging actions across time steps and instruments in plot 6 and 7, which show how the RL agent's strategy diverges from the delta-neutral baseline. At Episode 0, the surface shows a wide range of hedging differences ratios (Max: 1.40, Min: -0.43), with significant peaks and valleys, indicating an unrefined policy with high variance (Std: 1.34). The mean (0.28) is positive but modest, suggesting the initial policy has some hedging capability but is inconsistent.

By Episode 199, the range narrows (Max: 1.06, Min: -0.59), and the standard deviation increases slightly (1.43), indicating the policy is exploring a broader range of hedging strategies, including over-hedging and speculative positions. The mean drops to 0.21, reflecting a shift toward more balanced hedging in response to market dynamics. The surface evolves from a jagged, exploratory shape in Episode 0 to a more structured, smoother shape in Episode 199, reflecting a more consistent and adaptive hedging strategy that is supported by the positive rewards in figure 3.

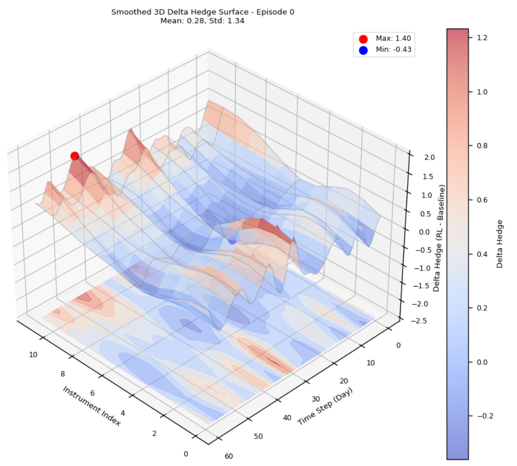

Figure 6. The difference in RL and Baseline Actions across instruments and time steps ending at Episode 0.

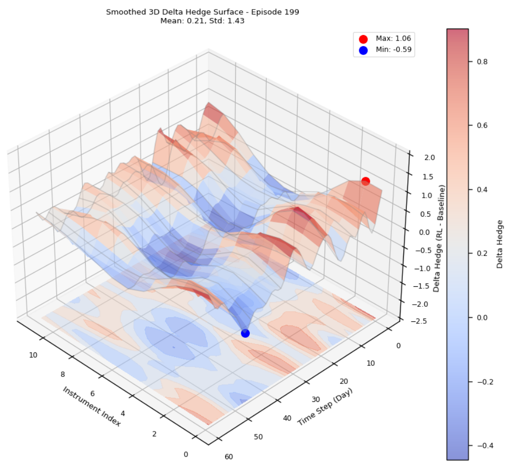

Figure 7. The difference in RL and Baseline Actions across instruments and time steps ending at Episode 199.

3.3. Risk Assessment

To further evaluate the risk profiles of the RL agent and the baseline delta-hedging agent, we compute three risk metrics on a test episode: Maximum Drawdown (MaxDD), Value at Risk (VaR), and Conditional Value at Risk (CVaR). Let $L = -\text{P\&L}$ be the episode-wise loss, with CDF $F_L(l) = P(L \leq l)$. The α-level Value at Risk (VaR) is:

$$\text{VaR}_\alpha = F_L^{-1}(\alpha),$$

and the Conditional VaR (CVaR) at level α is:

$$\text{CVaR}_\alpha = \mathbb{E}[L \mid L \geq \text{VaR}_\alpha].$$

Maximum drawdown (MaxDD) measures the largest peak-to-trough decline in cumulative P&L over an episode. For P&L r_t at step t, the cumulative P&L at time t is $\sum_{i=0}^{t} r_i$, and:

$$\text{MaxDD} = \max_{0 \le t \le T} \left[\max_{0 \le s \le t} \left(\sum_{i=0}^{s} r_i \right) - \sum_{i=0}^{t} r_i \right],$$

where $T = 59$ (60 steps per episode).

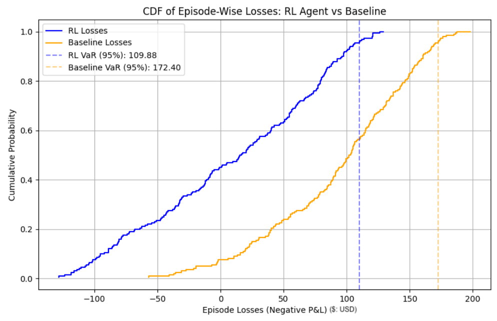

Figure 8. CDF of episode-wise losses for the RL agent and baseline delta-hedging agent, with 95% VaR highlighted.

The RL agent's risk management is evidenced by the cumulative distribution function (CDF) of episode-wise losses (negative P&L), shown in Figure 8. The blue line (RL) reaches higher cumulative probabilities at more negative values than the orange line (baseline). That implies the RL agent has a larger fraction of episodes in profitable territory (negative losses) compared to the baseline.

Coincidingly, the blue line reaches 100% of its episodes at a lower maximum loss than the orange line, suggesting that the RL agent avoids the large worst-case losses that the baseline incurs more frequently. The RL agent's tail is thinner, with a 95% Value at Risk (VaR) of 109.88 compared to the baseline's 172.40, a 56.89% improvement. This behavior aligns with the RL agent's risk-averse training objective (OCE utility with $\lambda = 0.5$), which prioritizes minimizing large losses over small ones, as the exponential utility heavily penalizes tail events.

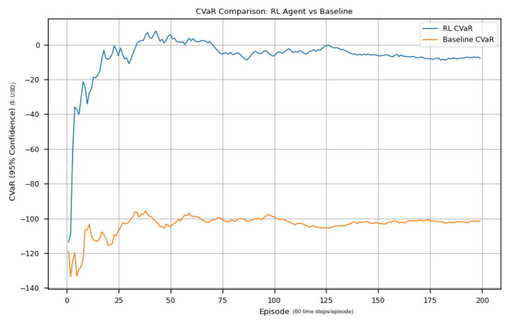

Figure 9. Comparison of RL CVaR and Baseline CVaR at 95% confidence level over the test episodes.

In figure 9 we can see the agent learning to reduce worst-case losses over time. Whereas figure 8 is a holistic overview of all episode outcomes, the CVaR figure shows how the tail risk measure—the worst 5% of outcomes—changes over episodes. The RL agent's CVaR decreases from 110 to near 0, indicating minimal expected loss in the worst 5% of scenarios, while the baseline's CVaR fluctuates between 100 and 140, ending around 110. This further confirms the RL agent's risk considerations, facilitating smaller worst-case scenario losses on average.

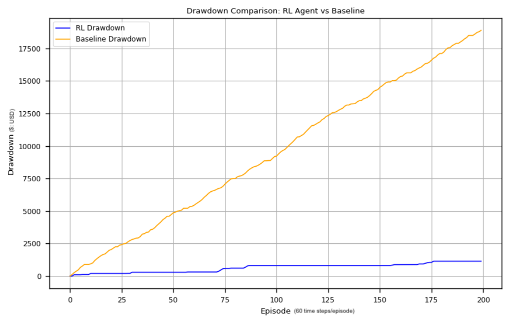

Figure 10. Drawdown chart showing how much each strategy's cumulative P&L falls below its historical maximum at each episode.

Lastly, figure 10, illustrates the evolution of MaxDD over 200 training episodes. The RL agent's MaxDD remains low and stable, while the baseline's MaxDD increases steadily to approximately 17,500, reflecting that the baseline's cumulative P&L consistently drops below its historical peak.

These trends demonstrate that the OCE-adjusted training process enables the RL agent to learn a policy that significantly reduces tail risk, even during training, compared to the static delta-hedging baseline. Concretely, the agent returned average improvements of 56.89%, 74.82%, and 76.79% across Var, CVaR, and MDD, respectively.

While it is unconventional in reinforcement learning to split data into training, validation, and testing sets, the variety of market conditions necessitated these separations. This allowed us to calibrate the Bates model on different sections of historical data, ensuring the agent generalized well to unseen market conditions, as shown in figure 11.

4. Discussion

4.1. Advances

These results advance industry-wide hedging research in several key ways. First, we answer our initial topic question by establishing that deep hedging can be effectively

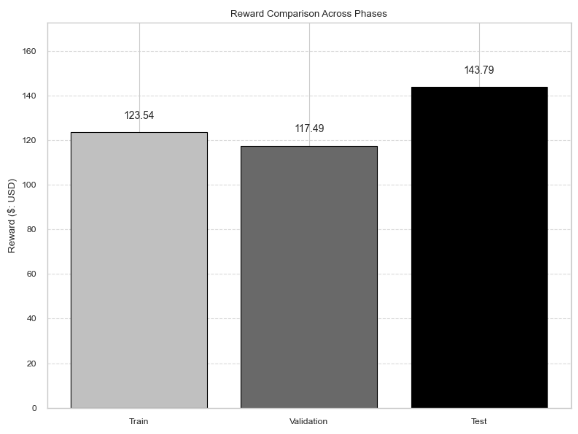

Figure 11. Model rewards across training, validation, and testing sets.

calibrated to cryptocurrency markets, which serve as excellent proxies for extreme market conditions. Second, by incorporating cross-correlations, in addition to transaction costs and liquidity constraints, our framework addresses real trading limitations that are often ignored in theoretical models. Third, our approach demonstrates that neural networks can learn optimal hedging strategies that adapt to cross-asset correlations and sudden market jumps without relying on distributional assumptions.

4.2. Mathematical Assumptions

For practitioners, our framework offers a viable alternative to traditional hedging methods, particularly for managing tail risks in volatile markets. However, our model relies on several mathematical assumptions that underpin the theoretical framework and training process. First, we assume a MDP with continuous state and action spaces, where the state $s_t = (z_t, m_t)$ evolves according to the Bates model's stochastic dynamics (Equation 10). The Bates model assumes stochastic volatility (via Heston) and jumps (via Merton), with cross-correlations between assets modeled through a multivariate structure. While this captures key features of cryptocurrency markets, it assumes that the market dynamics are fully described by these processes, potentially overlooking higher-order effects like regime shifts or non-stationary correlations, which are prevalent in crypto markets.

Second, the risk-averse value function $V^*(s)$ is defined using the Optimized Certainty Equivalent (OCE) utility with an exponential form, $U(x) = -\exp(-\lambda x)$, where $\lambda = 0.5$ (Equation 13). This assumes that the agent's risk aversion is constant and can be modeled by a single parameter, which may not fully capture the dynamic risk preferences of a real hedger. Additionally, the proof of convergence for V^* (Section 2.5) assumes a compact action space and bounded rewards, enforced by the liquidity penalty. In practice, market liquidity constraints may vary, and unbounded P&L fluctuations could challenge the theoretical convergence guarantees.

Third, the policy gradient with Generalized Advantage Estimation (GAE) assumes that the advantage function $\hat{A}_t^{\mathrm{GAE}}$ provides a reliable estimate of action quality (Equation 16). This relies on the critic's ability to approximate $V^*(s)$ accurately, which may be affected by the high-dimensional state space ($d_m \geq 6$) and the LSTM-based actor's ability to capture sequential dependencies. The Gaussian policy assumption further implies that actions follow a multivariate normal distribution, which may not fully represent the complex, potentially multi-modal action distributions required for optimal hedging in volatile markets.

Finally, the training process assumes that shuffling trajectories into mini-batches approximates independent and identically distributed (i.i.d.) samples, enhancing gradient stability (Section 2.6). While this improves convergence in on-policy RL, it may disrupt temporal dependencies in the data, potentially affecting the LSTM's ability to model long-term market trends.

4.3. Limitations and Further Studies

While our RL-based deep hedging model demonstrates superior risk management compared to the baseline delta-hedging approach (Figures 8, 9, and 10) several additional limitations warrant consideration to those posed in Section 4.2. First, the training process, spanning 200 episodes with 5 updates per episode (Section 2.6), is computationally intensive due to the high-dimensional state space and the LSTM-based actor-critic architecture. Each episode involves 60 time steps, and the Bates model's stochastic simulations add significant overhead, making real-time deployment and further developments in this field challenging without substantial computational resources.

Second, the baseline delta-hedging agent, while a useful benchmark, is overly simplistic. Its reliance on Black-Scholes delta (Section 2.4) does not account for stochastic volatility or jumps, which the Bates model captures. A more sophisticated baseline, such as a local volatility model or a machine learning-based hedging strategy, could provide a stronger comparison in future studies.

Third, the reward function (Equation 12) penalizes large positions but does not account for slippage or market impact, which could significantly impact P&L in high-frequency trading scenarios. These limitations suggest that while our model advances deep OTM hedging, there are natural next steps for practical deployment and additional research.

5. References

Bolfake, A., Mousavi, S. N., and Mashayekhi, S. (2023). Deep learning for option pricing under heston and bates models. *Journal of Mathematics and Modeling in Finance*, 3(1).

Buehler, H., Gonon, L., Teichmann, J., and Wood, B. (2018). Deep hedging. Working paper.

Buehler, H., Murray, P., and Wood, B. (2022). Deep bellman hedging. Revision 2.02, Working paper.

Cao, J., Chen, J., Hull, J., and Poulos, Z. (2019). Deep hedging of derivatives using reinforcement learning. *Journal of Financial Data Science*, 3(1):10–27.

Carbonneau, A. and Godin, F. (2020). Equal risk pricing of derivatives with deep hedging. *Quantitative Finance*, pages 593–608.

Mikkilä, O. and Kanniainen, J. (2023). Empirical deep hedging. *Quantitative Finance*, 23(1):111–122.

Neagu, A., Godin, F., Simard, C., and Kosseim, L. (2024). Deep hedging with market impact. Unpublished manuscript.

Noguer i Alonso, M. and Haida, A. (2024). Deep learning approach for multi-asset option pricing. Working paper.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.