CSCE 642: Deep Reinforcement Learning

Title: **Tabular Q-Learning vs Deep RL Baselines for Risk-Aware Portfolio Management**
GitHub: https://github.com/navyapriyanandi-tamu/Final-Project-Tabular-Q-vs-Deep-RL.git
YouTube: **https://youtu.be/apLHMZZECII**

Team Members:

- Navya Priya Nandimanalam (Email: navyapriya_n@tamu.edu, UIN: 136004813)
- Alisha Raj (Email: alisharaj2607@tamu.edu, UIN: 735007776)

1. ABSTRACT

Reinforcement learning has emerged as a powerful approach for financial decision-making, enabling agents to learn optimal trading strategies through interaction with market environments. Portfolio management is inherently a sequential decision-making problem where today's allocation affects tomorrow's wealth through transaction costs and compounding returns. While recent work has focused on deep reinforcement learning methods like DDPG and PPO, the comparative advantage over simpler tabular methods remains unclear. We present a systematic comparison of tabular Q-learning variants (standard Q-learning, Double Q-learning, and UCB-ε exploration) against deep RL baselines (DDPG, TD3, PPO) for risk-aware portfolio rebalancing in a cryptocurrency trading environment. All agents are evaluated under identical conditions alongside traditional baselines like equal-weight and momentum strategies. Our experiments show that while deep RL methods achieve higher returns, tabular Q-learning demonstrates competitive performance with greater interpretability. A stability study across varying transaction costs and seeds reveals that deep RL agents often converge to passive strategies under high costs, while tabular methods maintain active trading. We also analyze the sensitivity of UCB-ε and Double Q-learning to reward scaling.

2. INTRODUCTION

2.1 Motivation

Portfolio management, the task of allocating capital across multiple assets to maximize returns while managing risk, is a fundamental problem in finance. Traditional approaches like Modern Portfolio Theory rely on assumptions of known return distributions and static optimization, which often fail to capture the dynamic nature of financial markets. In reality, portfolio allocation is inherently sequential, that is, today's investment decisions affect tomorrow's wealth, and rebalancing incurs transaction costs that compound over time.

Reinforcement learning (RL) offers a natural framework for this problem. Unlike supervised learning, which requires labeled "correct" allocations, RL agents learn through trial and error, discovering strategies that maximize cumulative rewards over time. This aligns well with portfolio management, where the goal is to maximize long-term risk-adjusted returns rather than predict next-period prices.

Recent years have seen growing interest in applying deep reinforcement learning to portfolio optimization. Methods like DDPG, TD3, and PPO have shown promising results by learning continuous portfolio weight allocations directly from price data. However, this trend toward increasingly complex models overlooks the potential of simpler tabular methods, which offer distinct advantages, for example, explicit Q-tables that can be inspected for interpretability, more predictable training dynamics, and lower computational requirements.

Despite extensive research on deep RL for portfolio management, systematic comparisons with tabular methods under identical conditions remain scarce. Most studies either focus exclusively on deep methods or compare against

traditional finance baselines without including tabular RL. Additionally, the sensitivity of these algorithms to transaction costs and random initialization is often underexplored. This gap motivates our work, a rigorous apple-to-apple comparison of tabular Q-learning variants against deep RL baselines for portfolio rebalancing.

2.2 Problem Statement

The goal of this project is to systematically compare tabular reinforcement learning methods against deep RL baselines for portfolio management, under identical experimental conditions. Specifically, we aim to:

- Determine whether deep RL methods (DDPG, TD3, PPO) provide significant advantages over tabular Q-learning variants (standard Q-learning, Double Q-learning, UCB-ε) for cryptocurrency portfolio rebalancing.

- Evaluate all agents against traditional baselines (equal-weight buy-and-hold, momentum strategy, stay-in-cash) to establish practical performance thresholds.

- Compare RL-based portfolio allocation against single-asset strategies (100% in one cryptocurrency) to quantify the benefit of learned diversification.

- Analyze the stability of each algorithm across multiple random seeds and varying transaction cost regimes.

- Study the sensitivity of exploration strategies (UCB-ε, Double Q-learning) to reward scaling and transaction costs.

To ensure fair comparison, all agents share the same reward function, transaction cost model, walk-forward data splits (train/validation/test), and evaluation metrics (Return, Sharpe ratio, Maximum Drawdown, Turnover). We evaluate using both sequential backtesting and K-seed random episode sampling for statistical significance.

2.3 Key Contributions

The key contributions of this work are:

- Discrete Rebalancing MDP Wrapper: A Gym-compatible wrapper that transforms continuous portfolio allocation into discrete actions (decrease, hold, or increase weight by a fixed percentage) per asset. This enables tabular Q-learning to be applied directly to portfolio management.

- Apple-to-Apple Comparison Framework: We provide a fair comparison between tabular methods (Q-learning, Double Q-learning, UCB-ε) and deep RL baselines (DDPG, TD3, PPO) using identical reward functions, transaction costs, and evaluation protocols.

- Stability Study: We analyze algorithm robustness across multiple random seeds and transaction cost levels (0.025% to 0.5%), revealing how different methods behave under varying market friction.

- Exploration Analysis: We study the sensitivity of UCB-ε and Double Q-learning to reward scaling and transaction costs, providing insights into hyperparameter selection for tabular methods.

- Reproducibility: All experiments use fixed seeds and standardized evaluation, ensuring consistent results across multiple runs.

3. BACKGROUND AND RELATED WORK

3.1 Reinforcement Learning Fundamentals

- Reinforcement learning frames sequential decision-making as a Markov Decision Process (MDP) defined by states S, actions A, transition dynamics P(s'|s,a), rewards R(s,a), and discount factor γ. The agent learns a policy $\pi(a|s)$ to maximize expected cumulative discounted reward.

- Q-Learning is an off-policy temporal difference method that learns action-values Q(s,a), that is, the expected return from taking action 'a' in state 's' and following the optimal policy thereafter. The update rule is: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$, where α is the learning rate [1].

- Standard Q-learning uses ε-greedy exploration, which selects a random action with probability ε to explore, and the greedy action (argmax Q) with probability $(1-\varepsilon)$ to exploit learned values [1].

- However, Q-learning suffers from maximization bias, that is, using the max operator for both action selection and evaluation causes systematic overestimation of Q-values. Double Q-Learning addresses this by maintaining two Q-tables and decoupling selection from evaluation: $Q_1(s,a) \leftarrow Q_1(s,a) + \alpha[r + \gamma Q_2(s', \text{argmax}_{a'} Q_1(s',a')) - Q_1(s,a)]$. One table selects the best action whereas the other evaluates it, reducing bias [1].

- A limitation of ε-greedy is that it explores uniformly at random, ignoring which actions are uncertain. UCB (Upper Confidence Bound) exploration addresses this by favoring actions with high uncertainty: $a = \text{argmax}_a [Q(s,a) + c\sqrt{(\ln N(s) / N(s,a))}]$, where N(s) and N(s,a) are visit counts. Actions taken less often have higher uncertainty bonuses, encouraging systematic exploration before settling on the best action [1].

3.2 Deep RL for continuous control

- While tabular Q-learning works well when states and actions can be discretized, many real-world problems have continuous action spaces where discretization becomes impractical. Deep RL methods address this by using neural networks as function approximators [2].

- DDPG (Deep Deterministic Policy Gradient) extends Q-learning to continuous actions using an actor-critic architecture [2]. The actor network $\mu(s)$ directly outputs continuous actions, while the critic network Q(s,a) evaluates them. To stabilize training, DDPG uses experience replay (storing transitions for batch learning) and target networks (slowly-updated copies that provide stable learning targets). However, DDPG inherits Q-learning's overestimation bias and can be sensitive to hyperparameters.

- TD3 (Twin Delayed DDPG) improves upon DDPG with three key modifications [2]:
    - twin critics: maintaining two Q-networks and using the minimum value to reduce overestimation (analogous to Double Q-learning)
    - delayed policy updates: updating the actor less frequently than critics to reduce variance
    - target policy smoothing: adding noise to target actions to prevent exploiting Q-function errors.

- PPO (Proximal Policy Optimization) takes a different approach as an on-policy actor-critic method [2]. Unlike DDPG/TD3 which learn deterministic policies, PPO learns stochastic policies and constrains updates using a clipped surrogate objective. This prevents large policy changes that could destabilize

training, making PPO more robust but typically less sample-efficient than off-policy methods.

3.3 RL for Portfolio Management

- Portfolio management is a natural application of reinforcement learning, where an agent learns to allocate capital to maximize risk-adjusted performance. Prior work spans deep RL architectures, improved Q-learning formulations, and risk-aware methods.

- Deep RL Approaches: Jiang et al. [3] introduced the EIIE framework using deterministic policy gradients for crypto portfolios, achieving strong Sharpe ratios despite 0.25% transaction costs via the Portfolio-Vector Memory mechanism. Kayumov [4] reports DDPG outperforming A2C and PPO on a mixed stock-futures portfolio using a compound Sharpe + log-return reward. In contrast, Lu Chung [5] finds that off-policy methods (DDPG, TD3, SAC) fail under noisy rewards in a GBM-based market, while on-policy PPO and A2C succeed, with PPO achieving near-optimal growth. These mixed results motivate our systematic comparison across algorithms.

- Q-Learning and Overestimation Bias: Alidousti et al. [6] show that Double DQN reduces overestimation bias and improves data efficiency, though their work uses deep networks. This suggests that algorithmic improvements in Q-learning may matter as much as representation learning. We extend this by testing whether tabular Double Q-learning can remain competitive under appropriate state discretization.

- Risk-Aware Formulations: Several studies incorporate risk metrics directly into the reward function or model architecture. Choudhary et al. [7] use multiple PPO agents with different reward functions and fuse their actions via CNN. Huang et al. [8] use PPO for long-short portfolio optimization and introduce the constraint $\Sigma|\omega_i| = 1$, where ω_i denotes the weight of asset *i* (positive for long, negative for short). This constraint ensures that long-short exposure is accurately reflected in the return calculation, preventing distorted rewards when short positions are involved. Jiang and Wang [9] introduce LSTR-TD3, adding long and short-term risk control through dynamic cash adjustment. We adopt a simpler approach using reward shaping with drawdown penalties.

- Algorithm Comparisons: Liu [10] benchmarks DDPG, TD3, SAC, and PPO in continuous control tasks and observes TD3/SAC as most stable and DDPG suffering from overestimation bias. Li and Hai [11] compare multi-agent DQN systems against FINRL baselines, while Pawar et al. [12] explore DQN for portfolio weights but assume no transaction costs. These studies focus entirely on deep RL.

- Gap in Literature: Existing work overwhelmingly relies on deep neural networks, despite their instability, hyperparameter sensitivity, and computational cost. Only limited work, such as Double DQN variants, examines algorithmic improvements without questioning the need for deep function approximators. To the best of our knowledge, no study systematically compares tabular Q-learning variants (standard, Double Q, UCB exploration) against deep RL under identical market settings. We therefore ask: Can improved tabular Q-learning achieve competitive portfolio performance relative to deep RL in discrete rebalancing environments?

4. PROBLEM FORMULATION

4.1 Why Sequential?

Portfolio management is inherently sequential because today's allocation changes tomorrow's wealth and future feasible trades due to transaction costs. Decisions compound over time in several ways:

- Wealth Evolution: Portfolio value evolves multiplicatively. Early losses reduce capital available for future gains, making the sequence of decisions,not just individual choices, critical.
- Transaction Costs Create Path Dependence: Rebalancing from w_t to w_{t+1} incurs costs proportional to weight changes. A myopic optimizer ignores that frequent trading erodes returns through cumulative costs.
- Position Limits: Current weights constrain feasible actions. Large position changes may be costly or infeasible, linking today's decisions to tomorrow's options.

4.2 MDP Formulation

We formulate portfolio management as a Markov Decision Process:

- State Space (S): States include recent per-asset returns/volatility, current portfolio weights, and a lightweight "regime" flag. Specifically:
    - Per-asset returns (5 bins: strong_down, down, flat, up, strong_up)
    - Volatility regime (3 bins: low, medium, high) from rolling standard deviation
    - Aggregate momentum (3 bins: bearish, neutral, bullish) from 5-period returns
    - Cash weight (3 bins: low, medium, high)

  This yields $5^3 \times 3 \times 3 \times 3 = 3{,}375$ discrete states for tabular agents.

- Action Space (A):
    - Discrete (tabular agents): At each step, the agent decides whether to decrease, hold, or increase the weight of each asset by a fixed amount $\Delta = 5\%$. With 3 tradeable assets plus cash (4 positions total) in this project, each having 3 choices, this gives $3^4 = 81$ possible actions. After applying changes, weights are clipped to [0, 1] and normalized to sum to 1 (simplex projection) to ensure valid portfolio allocations.

    - Continuous (deep RL agents): The agent directly outputs a target weight for each of the 4 positions as a continuous value between 0 and 1. These weights are then normalized to sum to 1, ensuring a valid portfolio allocation.

- Transition (P): At each time step, the environment evolves as follows:
    - Agent selects action (new target weights)
    - Portfolio rebalances from current weights to target weights
    - Transaction costs are deducted based on the total weight change: cost = cost_rate × Σ|w_new − w_old|
    - Next period's prices are revealed and portfolio value updates
    - Weights drift due to price movements (e.g., if ETH rises, its weight naturally increases)

  This creates path dependence, that is, the agent's current weights depend on all previous actions and price movements.

- Reward (R): We use log-return with cost and risk penalties, that is, R = scale × [log(p_t / p_{t-1}) − λ × cost − drawdown_penalty], where
    - log(p_t / p_{t-1}) is the one-step portfolio log-return, capturing percentage gains/losses
    - λ × cost penalizes excessive trading (λ = 0.1 by default), discouraging unnecessary rebalancing that erodes returns
    - drawdown_penalty activates when the portfolio drops more than 15% from its peak value, encouraging the agent to manage downside risk

We scale rewards by 100 to provide stronger learning signals for the tabular agents.

- Discount Factor ($\gamma = 0.99$): This determines how much the agent values future rewards versus immediate rewards. With $\gamma = 0.99$, a reward received 100 steps in the future is worth approximately 37% of the same reward today ($0.99^{100} \approx 0.37$). This encourages the agent to consider long-term wealth accumulation rather than myopic single-step optimization.

4.3 Environment Details

We use the Poloniex cryptocurrency dataset with 30-minute trading intervals from the wassname/rl-portfolio-management repository [13]. The portfolio contains 3 tradeable assets (LTC, DASH, XMR) with BTC as the base currency (cash).

- Data Splits (Walk-Forward):
    - Training: 80% of train data (~68% of total) for learning
    - Validation: 20% of train data (~17% of total) for hyperparameter tuning
    - Test: Held-out (~15% of total) for final evaluation

- Transaction Costs: We test three cost levels:
    - 2.5 bps (0.025%) - baseline
    - 25 bps (0.25%) - moderate friction
    - 50 bps (0.5%) - high friction

- Episode Structure: During training, episodes run for 256 steps with starting points sampled uniformly at random, allowing the agent to experience diverse market conditions. A window length of 50 timesteps (25 hours) provides price history for state observation.

5. METHODOLOGY

5.1 Environment Wrapper

We build on the wassname/rl-portfolio-management environment [13] by adding custom wrappers that enable both tabular and deep RL agents to operate with identical reward functions and transaction costs.

5.1.1 Discrete action wrapper

Tabular Q-learning requires a finite action space, but portfolio allocation is inherently continuous (weights can be any value between 0 and 1). To bridge this gap, we discretize actions into simple rebalancing decisions.

At each step, the agent chooses one of three actions for each position: decrease weight by 5%, hold weight unchanged, or increase weight by 5%. With 4 positions (3 tradeable assets + cash), this gives $3^4 = 81$ possible actions.

To store these in a Q-table, we encode the 81 action combinations as integers from 0 to 80. Each integer uniquely maps to a combination of per-asset decisions. For example:

- Action 0: decrease all four positions
- Action 40: hold all four positions unchanged
- Action 80: increase all four positions

The wrapper tracks current portfolio weights and applies the selected adjustments at each step.

5.1.2 Simplex Projection

After applying weight changes, the resulting weights may be invalid (negative or not summing to 1). We project back to a valid allocation by:

- Clipping negative weights to 0
- Normalizing so all weights sum to 1

If all weights become zero, we default to 100% cash. Deep RL agents undergo the same projection, ensuring fair comparison.

5.1.3 State discretization

The state features are defined in Section 4.2. We implement this as a StateDiscretizer wrapper class with the following bin thresholds:

- per-asset returns use [-1%, -0.3%,+0.3%, +1%] to create 5 bins from strong_down to strong_up
- volatility regime uses [1%, 3%] for low/medium/high
- aggregate momentum uses [-2%, +2%] for bearish/neutral/bullish
- cash weight uses [33%, 66%] for low/medium/high.

The wrapper encodes these features as a single integer state for Q-table indexing.

5.1.4 Reward Shaping

The reward function is defined in Section 4.2. We implement this as a RewardShaper wrapper for tabular agents and SB3PortfolioWrapper for deep RL agents, both using identical parameters (scale=100, λ=0.1, drawdown threshold=15%, drawdown penalty=0.05) to ensure fair comparison.

5.2 Tabular Agents

We implement three tabular Q-learning variants, each implemented as a QLearningAgent, DoubleQLearningAgent, and UCBQLearningAgent class. All agents share the same hyperparameters and training protocol but differ in how they handle value estimation and exploration. All three agents use a sparse Q-table implemented as a Python defaultdict that only stores visited states, making it memory-efficient for our 3,375-state space. Each agent trains for 30,000 episodes with 256 maximum steps per episode. Below are the hyperparameters used across all the 3 agents:

- Learning rate: $\alpha = 0.1$, decaying to 0.01 (decay factor 0.9999 per episode)
- Discount factor: $\gamma = 0.99$
- Exploration: $\varepsilon = 0.9999$, decaying to 0.03 (decay factor 0.99985 per episode)

The high initial ε ensures broad exploration early in training, while the slow decay maintains some exploration throughout the 30,000 episodes.

5.2.1 Q-Learning (baseline)

Standard Q-learning as described in Section 3.1 serves as our baseline tabular method. We implement this as a QLearningAgent class that stores the Q-table as a defaultdict, only allocating space for visited states. The choose_action() method implements ε-greedy selection, while learn() performs the Q-update. After each episode, both ε and α decay gradually to shift from exploration toward exploitation as training progresses.

5.2.2 Double Q-Learning (reduce overestimation)

To address the maximization bias described in Section 3.1, we implement Double Q-Learning using a DoubleQLearningAgent class that maintains two Q-tables (Q1 and Q2). On each update, the learn() method randomly selects which table to update with 50/50 probability, calling either _update_q1() or _update_q2() accordingly. For action selection during exploitation, the agent uses combined Q-values through argmax(Q1[s] + Q2[s]), providing a more stable estimate than either table alone.

5.2.3 UCB-ε Q-Learning (exploration)

To address the limitation of uniform random exploration in ε-greedy (Section 3.1), we implement UCB-ε Q-Learning using a UCBQLearningAgent class. In addition to the Q-table, it maintains visit counts N_state and N_state_action to track how often each state and state-action pair has been visited. The _ucb_values() method computes exploration bonuses for each action using c=2.0 as the exploration constant. Actions never tried in a state receive infinite bonus, ensuring they are explored first. During training, with probability ε the agent explores randomly, otherwise it selects actions using UCB scores. During evaluation, it acts purely greedy on learned Q-values.

5.3 Deep RL Agents

We implement three deep RL algorithms using Stable-Baselines3, that is, DDPG, TD3, and PPO. All agents share the same environment interface created via create_sb3_env(), which wraps the base PortfolioEnv with our SB3PortfolioWrapper to ensure identical reward shaping as tabular agents. The continuous action space outputs portfolio weights that are projected to the simplex before execution.
All three agents use [256, 256] hidden layer networks for both actor and critic, discount factor γ=0.99, and train for 1,000,000 timesteps. Below are the key hyperparameters:

- DDPG/TD3: learning rate 1e-4, replay buffer 100K (stores past experiences for off-policy learning), batch size 256 (samples per gradient update), soft target update τ=0.005 (slowly blends target network weights for stability)
- PPO: learning rate 3e-4, rollout length 2048 (steps collected before each update), minibatch size 64 (subset of rollout per gradient step), 10 epochs per update (passes through collected data)

5.3.1 DDPG

We use the Stable-Baselines3 DDPG class with MlpPolicy. For exploration, we add Ornstein-Uhlenbeck noise (σ=0.1) to actions, which produces temporally correlated noise suitable for continuous control. Training begins after 10K steps of random exploration to fill the replay buffer.

5.3.2 TD3

We use the Stable-Baselines3 TD3 class with the same base configuration as DDPG. The key configurations are: delayed policy updates every 2 gradient steps (policy_delay=2), target policy smoothing with Gaussian noise (σ=0.2, clipped to ±0.5), and twin critics (handled internally by SB3). For exploration, we use Gaussian noise (σ=0.1) instead of Ornstein-Uhlenbeck.

5.3.3 PPO

We use the Stable-Baselines3 PPO class with MlpPolicy. The clipped surrogate objective uses ε=0.2, with GAE

(λ=0.95) for advantage estimation. We include an entropy bonus (coefficient 0.01) to encourage exploration throughout training.

5.4 Baselines

We implement three baseline strategies as BaselineAgent subclasses to benchmark our RL agents against traditional approaches. All baselines use the same discrete action interface (81 actions) as tabular agents.

- Equal-weight buy-and-hold: The EqualWeightAgent implements the classic 1/N portfolio strategy. Since the environment starts with 100% cash, the agent must actively rebalance toward equal weights (25% each across 4 positions). Using action 78 (decrease cash, increase all risky assets), the agent shifts weights over approximately 15 steps (calculated as 0.75/δ where δ=0.05). After reaching equal weights, it holds by returning action 40. The reset() method resets the step counter for each new episode.

- Momentum strategy: The MomentumAgent implements a naive momentum strategy with monthly rebalancing. It rebalances every 1440 steps (30 days of 30-minute bars) by rotating through pairs of risky assets. The _build_action_map() method precomputes actions for each asset pair: action 8 for assets (1,2), action 20 for (1,3), and action 24 for (2,3). Between rebalances, the agent holds with action 40. Since tabular agents receive discretized states without raw prices, this implementation cycles through asset pairs rather than computing actual 60-day returns.

- Stay-in-cash: The simplest baseline, implemented as CashAgent, holds 100% in BTC cash throughout the episode. The agent always returns action 0, which corresponds to all weight in cash. This provides a conservative lower bound with zero market exposure.

5.5 Evaluation Metrics

We evaluate all agents using two approaches: random episode evaluation and sequential backtesting.

- Random Episode Evaluation: We run 50 test episodes per seed with random starting points (random_reset=True) to compute statistical metrics across varied market conditions.
- Sequential Backtesting: We also run a single walk-forward backtest through the entire test period (random_reset=False) to simulate realistic trading performance.

For both approaches, we compute four standard portfolio performance metrics:

- Return (%): Cumulative portfolio return calculated as (P_T - 1) × 100, where P_T is the final portfolio value starting from P_0 = 1.
- Sharpe Ratio: Risk-adjusted return measuring excess return per unit of volatility, annualized using √(48 × 365) to account for 30-minute bars.
- Maximum Drawdown (%): Largest peak-to-trough decline, measuring worst-case downside risk.
- Turnover (%): Average portfolio weight changes per step, indicating trading frequency and transaction cost exposure.

We report results across 3 random seeds (42, 123, 456) to assess stability and sensitivity to initialization.

5.6 Hyperparameter Selection

Hyperparameters were selected through iterative experimentation (approximately 5 trials per agent) on the validation set (20% of training data), with final evaluation on held-out test data.

- Tabular Agents: We tuned learning rate schedules and exploration decay to balance convergence speed with policy quality. The final configuration uses α=0.1 decaying to 0.01 (decay 0.9999), ε=0.9999 decaying to 0.03 (decay 0.99985), and γ=0.99. The high initial ε ensures broad exploration early in training, while the slow decay maintains exploration throughout 30,000 episodes. The UCB exploration constant c=2.0 follows standard practice from the bandit literature.

- Deep RL Agents: We adopted recommended hyperparameters from Stable-Baselines3 documentation with minor adjustments. DDPG and TD3 use learning rate 1e-4, which we found more stable than higher rates for financial data. PPO uses learning rate 3e-4 with 10 epochs per update. All agents use [256, 256] networks, providing sufficient capacity without overfitting.

- Reward Shaping: Parameters were chosen to match the proposal specification: scale=100 amplifies the signal from small log-returns, λ=0.1 penalizes transaction costs without dominating the reward, and drawdown penalty (threshold=15%, penalty=0.05) discourages excessive risk-taking.

5.7 Code Changes Made

This project includes a unified and modular codebase supporting tabular RL, deep RL, stability studies, and market-regime evaluation. Below is a concise description of the key folders and scripts.

5.7.1. Training Scripts (Top-Level)

These scripts train individual agents under the unified environment:

train_q_learning.py – trains tabular Q-Learning.
train_double_q_learning.py – trains Double-Q Learning.
train_ucb_q_learning.py – trains UCB-Q with exploration bonuses.
train_ddpg.py – trains the DDPG deep RL agent (SB3 wrapper).
train_td3.py – trains the TD3 deep RL agent.
train_ppo.py – trains the PPO deep RL agent.

Saved model files: q_learning_agent.pkl, double_q_learning_agent.pkl, ucb_q_learning_agent.pkl, ddpg_agent.zip, td3_agent.zip, ppo_agent.zip

5.7.2. Evaluation & Comparison Scripts

compare_strategies.py – full walk-forward evaluation of all agents; generates summary tables.
plot_vs_market.py – compares RL agents against individual assets and benchmark portfolios.

5.7.3. Stretch Goals

Folder: stability_study/
Contains scripts for multi-seed evaluation:
stability_analysis.py – common utilities for aggregating stability metrics.
stability_study_tabular.py – K-seed evaluation for Q-Learning, Double-Q, UCB-Q.
stability_study_deep_rl.py – K-seed evaluation for DDPG, TD3, PPO.

exploration_analysis.py – reward-scale and transaction-cost sensitivity for tabular RL.

5.7.4. Market Regime Testing

Files:
poloniex_30m_test_dummy_bullish.hf
poloniex_30m_test_dummy_neutral.hf
poloniex_30m_test_dummy_bearish.hf

These datasets define synthetic bullish, neutral, and bearish environments for stress-testing agent robustness.

5.7.5. Core RL Components

These modules define the learning logic and environment interface:

q_learning.py – tabular Q-Learning implementation.
double_q_learning.py – Double-Q update rule implementation.
ucb_q_learning.py – UCB-based optimistic exploration for tabular RL.
baselines.py – Equal-Weight, Momentum, and Cash benchmark strategies.
discrete_actions.py – maps continuous portfolio weights to discrete allocation bins.
state_discretizer.py – converts raw market observations into tabular RL states.
reward_shaper.py – applies transaction costs and drawdown penalties.
sb3_wrapper.py – unifies SB3 deep RL policies with the custom portfolio environment.
init.py – package initializer.

5.7.6. Scripts Folder

Folder: scripts/
Contains helper utilities for reproducibility, dataset loading, experiment orchestration, and project-wide configuration.

6. EXPERIMENTS AND RESULTS

6.1 Experimental Setup

This section describes the dataset, walk-forward splits, episode structure, and reproducibility protocol used in all experiments. To ensure an apples-to-apples comparison across tabular and deep RL methods, every agent was trained and evaluated on identical data windows, with fixed seeds and consistent reward, transaction cost, and backtesting settings.

6.1.1 Dataset and Regime-Based Test Splits

All experiments use the Poloniex 30-minute cryptocurrency dataset (poloniex_30m.hf).

For robustness analysis, we additionally evaluate on three synthetic market-regime test sets: bullish, neutral, and bearish. These regime-specific datasets allow controlled stress-testing of agent behavior under distinctly different return structures.

- **Train / Validation / Test Windows**

  We follow a strict walk-forward evaluation protocol:

Train: 2014-05-19 06:00 → 2016-07-10 10:30
Validation: 2016-07-10 11:00 → 2017-01-22 06:00
Test: 2017-01-22 06:30 → 2017-07-13 23:30

The test window contains 8,291 timesteps (30-minute bars), and the agent performs one allocation decision per timestep. This full horizon is treated as a single continuous backtest episode unless otherwise specified.

- **Episode Length / Backtest Horizon**
    - Each evaluation run uses the entire test window as one episode, resulting in a deterministic horizon of 8,291 decisions.
    - When episodic training is used (tabular agents), the episode length matches the training data window.
    - Deep RL agents (DDPG, TD3, PPO) use standard off-policy or on-policy rollouts until convergence on the training period.

6.1.2 Random Seeds and Reproducibility

To assess algorithmic stability, all experiments are repeated using K = 5 independent runs with seeds {42, 142, 242, 342, 442}. Each seed controls all relevant sources of randomness, including Python, NumPy, PyTorch, environment initialization, and stochastic exploration mechanisms (ε-greedy or UCB-ε). For every seed, models, logs, and evaluation metrics are stored separately, enabling statistical analysis of performance variability through mean, variance, and confidence intervals. When supported, a dedicated --seed argument is used to initialize experiments, otherwise, environment-level seeding ensures full reproducibility across runs.

6.1.3 Hardware and Software Environment

- All experiments were conducted on a CPU-only workstation.
- The software environment includes:
    - Python: 3.13.7 (verified via python --version)
    - Libraries: gymnasium, stable-baselines3, torch, numpy, pandas, tables, matplotlib, seaborn
- This ensures full reproducibility of training runs and facilitates verification of experimental results.

6.2 Main Comparison (Core Results)

This section presents the complete evaluation of all portfolio management agents: tabular RL methods (Q-Learning, Double-Q, UCB-Q), deep RL agents (DDPG, TD3, PPO), and traditional baselines (Cash, Equal-Weight, Momentum). All methods are evaluated under identical walk-forward splits and a K-seed statistical procedure. We report returns, Sharpe ratios, drawdowns, and turnover, followed by equity-curve and market-comparison analysis.

6.2.1 Walk-Forward Backtest Results (Train → Val → Test)

All agents are evaluated sequentially on Train, Validation, and Test windows using the full backtest horizon. Results include total return, Sharpe ratio, maximum drawdown (MaxDD), and turnover.

Train split

Strategy	Return	Sharpe	Max DD	Turnover
Q-Learning	–62.29%	0.053	82.33%	95.8%
Double-Q	+2.50%	0.309	51.31%	95.9%
UCB-Q	–11.32%	0.549	80.41%	95.1%
Cash	–0.45%	0.000	0.45%	0.0%
Equal-Weight	+461.38%	1.279	60.00%	0.0%
Momentum	+57.30%	0.815	27.00%	0.1%
DDPG	**+33,496.79%**	3.423	38.97%	28.9%
TD3	**+96,270.53%**	3.329	62.25%	29.9%
PPO	+165.32%	0.954	47.11%	0.1%

Table 1. Walk-Forward Performance Across Train Splits

Observations:
Train results show massive overfitting by DDPG and TD3 (+33,497% and +96,271%), as their continuous action spaces allow precise memorization of historical price patterns. PPO's clipped objective limits this behavior (+165%). Tabular agents perform poorly due to information loss from discretization, high turnover costs (~95%), and coarse 5% weight adjustments that cannot exploit small price movements. Double-Q performs best among tabular methods (+2.5%), suggesting overestimation bias reduction helps avoid poor actions.

Validation Split

Strategy	Return	Sharpe	Max DD	Turnover
Q-Learning	+6.10%	0.482	47.70%	95.6%
Double-Q	+0.59%	0.175	20.25%	96.0%
UCB-Q	+21.74%	0.865	56.82%	95.2%
Cash	–0.11%	0.000	0.11%	0.0%
Equal-Weight	+99.39%	2.098	47.59%	0.0%

Momentum	+0.98%	0.248	7.59%	0.1%
DDPG	+81.97%	2.420	28.34%	21.8%
TD3	+3.93%	0.423	38.87%	18.9%
PPO	+81.04%	2.033	39.31%	0.0%

Table 2. Walk-Forward Performance Across Validation Splits

Observations**:**
Validation performance drops significantly from training, revealing generalization capabilities. Equal-Weight (+99%), DDPG (+82%), and PPO (+81%) maintain strong performance, PPO achieves 0% turnover by learning a simple buy-and-hold strategy that generalizes well. TD3 collapsed from +96,271% to +4%, as its twin critics allowed the actor to exploit training-specific Q-function artifacts rather than learning robust patterns. Among tabular agents, UCB-Q generalizes best (+22%) because its exploration bonus ensures diverse state-action coverage, preventing premature convergence. Q-Learning shows moderate generalization (+6%), while Double-Q nearly collapses (+0.6%), its conservative dual-table updates avoid overestimation but also fail to commit to profitable actions, resulting in overly cautious policies.

Test Split (Primary Evaluation Window)

Strategy	Return	Sharpe	Max DD	Turnover
Q-Learning	+204.38%	2.483	42.01%	95.9%
Double-Q	+51.07%	1.651	22.15%	96.2%
UCB-Q	+152.29%	2.318	40.99%	95.9%
Cash	–0.10%	0.000	0.10%	0.0%
Equal-Weight	+319.80%	3.289	36.02%	0.0%
Momentum	+24.79%	2.801	5.53%	0.1%
DDPG	+283.67%	3.608	23.21%	21.8%
TD3	+129.55%	2.351	30.92%	18.2%
PPO	+111.06%	2.448	24.68%	0.0%

Table 3. Walk-Forward Performance Across Test Splits

Observations:
Test results reveal true out-of-sample performance. DDPG achieves the best risk-adjusted return (Sharpe 3.61, +284%) with moderate turnover (22%), while Equal-Weight leads in raw return (+320%) by benefiting from sustained crypto uptrends without transaction costs. Among tabular agents, UCB-Q shows the most consistent performance (+152%) due to its systematic exploration and robust policies. Q-Learning achieves +204% but this is regime-dependent, it failed on Train and Val, suggesting luck rather than learned skill. Double-Q improves from validation (+51% vs +0.6%) but remains conservative. TD3 (+130%) and PPO (+111%) generalize reasonably, with PPO's 0% turnover confirming a buy-and-hold approach that captures trends but misses shorter-term opportunities.

6.2.2 Statistical Significance (K = 5 Seed Evaluation)

To ensure results are not seed-dependent, each algorithm is run for 5 seeds over 50 episodes per seed. Below are mean ± std across seeds.

Strategy	Return	Sharpe	Max DD	Turnover
Q-Learning	+6.34 ± 1.35%	2.983 ± 0.711	12.06 ± 1.02%	96.1%
Double-Q	+1.30 ± 0.31%	2.165 ± 0.495	4.94 ± 0.70%	96.4%
UCB-Q	+5.39 ± 0.79%	3.029 ± 0.380	11.79 ± 1.00%	95.8%
Cash	0.00%	0.000	0.00%	0.0%
Equal-Weight	**+6.83 ± 0.68%**	**4.343 ± 0.465**	11.96 ± 1.02%	0.4%
Momentum	+0.40 ± 0.07%	2.832 ± 0.640	0.94 ± 0.08%	0.4%
DDPG	+4.93 ± 0.51%	**3.849 ± 0.364**	**8.70 ± 0.45%**	21.0%
TD3	+2.77 ± 0.56%	2.448 ± 0.434	**9.01 ± 0.32%**	18.3%
PPO	+2.99 ± 0.71%	2.253 ± 0.763	7.87 ± 0.37%	0.0%

Table 4. K-Seed Performance (Mean ± Std)

Observations:

K-seed evaluation (5 seeds, 50 episodes each) provides statistically robust performance estimates. Equal-Weight achieves the highest mean return (+6.83±0.68%) and Sharpe (4.34±0.47) with low variance, confirming that passive diversification remains a strong baseline when markets trend upward, no learning required means no seed sensitivity. Among RL agents, DDPG performs best (+4.93±0.51%, Sharpe 3.85) with moderate turnover (21%), demonstrating that its continuous action space learns adaptive strategies that generalize across seeds. UCB-Q (+5.39±0.79%) and Q-Learning (+6.34±1.35%) show surprisingly consistent performance despite poor training results, suggesting their policies converged to similar robust solutions across seeds due to the constrained discrete action space. Double-Q remains the weakest (+1.30±0.31%) with lowest variance, its conservative dual-table updates produce stable but underperforming policies across all seeds. TD3 (+2.77±0.56%) and PPO (+2.99±0.71%) show lower returns than DDPG but tighter confidence intervals, indicating more predictable behavior at the cost of upside potential.

6.2.3 Comparison vs Market Assets (Test Split)

This section compares RL-managed portfolios against investing 100% in individual assets, using alpha to measure whether active management adds value over passive holding.

Strategy	Return	Max DD
Q-Learning	+204.38%	42.01%
Double-Q	+51.07%	22.15%
UCB-Q	+152.29%	40.99%
DDPG	**+283.67%**	**23.21%**
TD3	+129.55%	30.92%
PPO	+111.06%	24.68%
Equal-Weight	+241.99%	43.91%
Momentum	+24.79%	5.53%
DASHBTC	+342.89%	66.96%
LTCBTC	+359.06%	55.10%
XMRBTC	+24.04%	40.75%

Table 5. RL Agents vs Market Assets (Test Split)

Alpha Estimates:

- Q-Learning: –37.61%
- UCB-Q: –89.70%
- DDPG: +41.67% (positive alpha vs market basket)
- TD3: –112.44%
- PPO: –130.93%

Observations:
Comparing learned portfolios against individual asset buy-and-hold reveals whether RL agents add value beyond simple asset selection. Individual assets show highly variable returns, LTCBTC (+359%) and DASHBTC (+343%) significantly outperform XMRBTC (+24%), highlighting the importance of asset allocation. The Equal-Weight market benchmark (+242%) averages across these, providing a passive diversification baseline. DDPG is the only RL agent generating positive alpha (+41.67% vs market), demonstrating its ability to dynamically overweight winning assets and underweight losers, a skill that justifies its complexity. In contrast, tabular agents destroy value:

Q-Learning (-37.61% alpha) and UCB-Q (-89.70% alpha) underperform passive holding despite achieving positive absolute returns, as their discrete 5% weight adjustments and high turnover (96%) incur costs without capturing enough upside. TD3 (-112.44% alpha) and PPO (-130.93% alpha) show severe negative alpha, indicating their learned policies actively hurt performance compared to simply holding the market, TD3's overfitting and PPO's static buy-and-hold approach both fail to adapt to shifting asset dynamics.

6.2.4 Equity Curves

Below we show the graphs plotted for the results discussed above. Results for training of the rest of the algorithms like Tabular Q-learning, TD3 and PPO are in the GitHub repository.

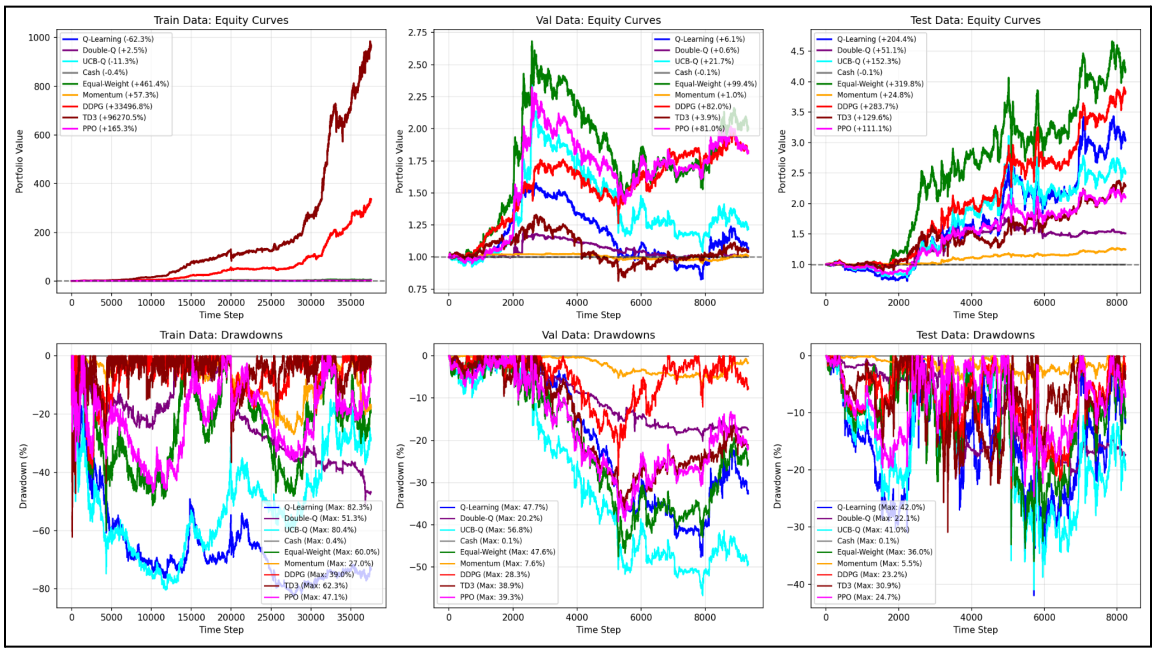

Figure 1: Walk-Forward Equity Curves (Train/Val/Test)

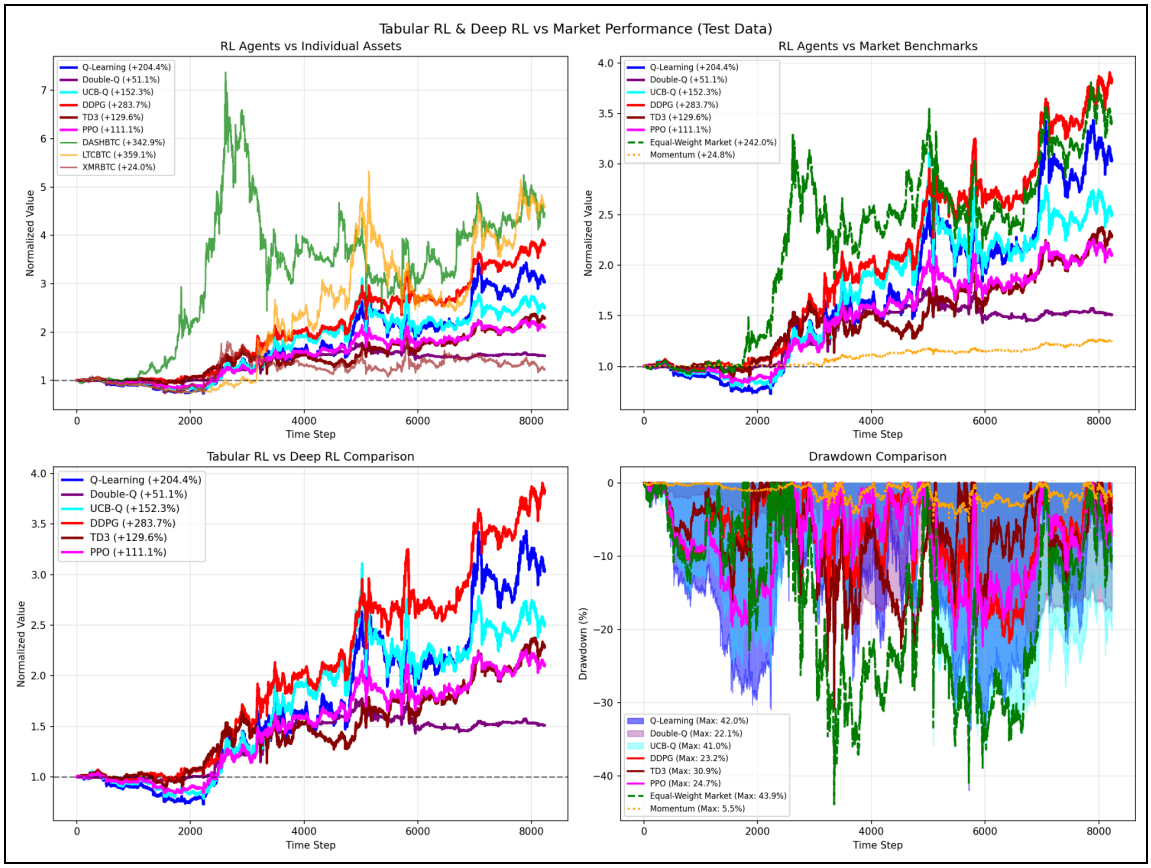

Figure 2: Q-Learning vs Market Assets

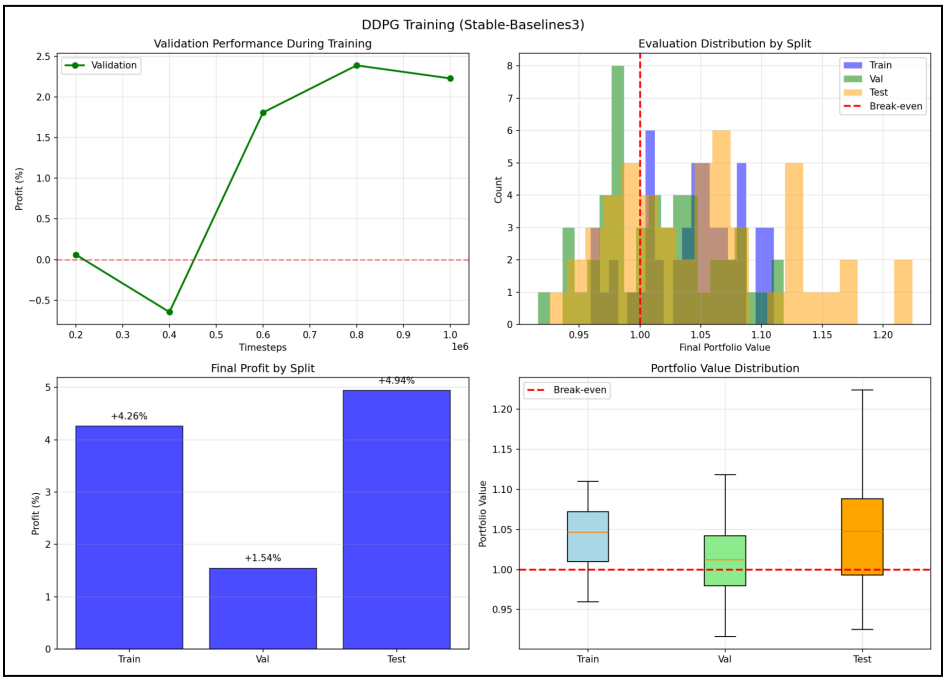

Figure 3: DDPG Training Curve

Observations:

Across train, validation, and test periods, the equity curves display a clear separation between deep RL, tabular RL, and traditional baselines. Among the deep RL agents, DDPG consistently demonstrates the strongest overall performance, characterized by smooth growth, moderate volatility, and comparatively low drawdowns, its continuous action space and experience replay enable precise weight adjustments that adapt to changing market conditions. TD3 shows extremely strong growth in the Train split but fails to generalize, producing weaker and more volatile curves on Val/Test, its twin critics overfit to training-specific Q-function artifacts rather than learning robust patterns. PPO delivers stable but more conservative performance, generating smoother curves but lower overall returns than DDPG, its clipped objective and 0% turnover indicate a learned buy-and-hold strategy that avoids transaction costs but misses shorter-term opportunities.

Within tabular RL, Q-Learning achieves the highest test return (+204%) but with high variance across seeds, while UCB-Q (+152%) shows more consistent performance due to its systematic exploration building robust policies during training. However, both exhibit sharp fluctuations and deep interim drawdowns (~40%), reflecting sensitivity to state discretization noise and coarse 5% weight adjustments that cannot react precisely to price movements. Double-Q produces the most stable tabular results with lowest drawdown (22%) but sacrifices returns (+51%), its conservative dual-table updates avoid overestimation but also prevent commitment to profitable actions.

When compared to market benchmarks, DDPG is the only agent generating positive alpha (+41.67%), successfully overweighting winners and underweighting losers through its adaptive policy. Equal-Weight frequently matches or exceeds RL agents in raw returns (+320% on Test) because sustained crypto uptrends reward passive diversification without transaction costs. Tabular agents show negative alpha despite positive absolute returns, as their high turnover (~96%) erodes gains through transaction costs. Momentum performs well in trending regimes but weakens in neutral or volatile periods due to its fixed rebalancing schedule that cannot adapt to regime shifts.

Drawdown plots reinforce the performance hierarchy: tabular methods experience the deepest and most frequent drawdowns (Q-Learning 42%, UCB-Q 41%) due to their discrete action constraints and high trading frequency, PPO (25%) and TD3 (31%) sit in the middle with more controlled risk, and DDPG (23%) consistently maintains the lowest peak drawdowns among learned models, its continuous policy smoothly adjusts positions rather than making abrupt discrete changes.

DDPG's training diagnostics further validate its stability: validation performance improves steadily during training without the sharp degradation seen in TD3, and the distribution of final portfolio values across Train/Val/Test clusters above break-even with low variance. This consistency stems from DDPG's balanced exploration via Ornstein-Uhlenbeck noise and stable target network updates.

Overall, the equity curves, drawdown behavior, and diagnostics show that deep RL, especially DDPG, produces the most stable, high-growth, low-risk portfolios through adaptive continuous control, while TD3 and PPO provide intermediate performance limited by overfitting and conservatism respectively. Tabular RL is fragile, noisy, and highly regime-dependent due to information loss from discretization and high turnover costs. Although Equal-Weight is surprisingly competitive in trending markets, it lacks the adaptability and risk control achieved by deep RL methods during regime shifts.

6.2.6 Summary

- **Deep RL vs Tabular RL**
    - DDPG is the strongest overall agent, excelling in return, Sharpe ratio, drawdown control, and cross-regime robustness.
    - TD3 exhibits extreme Train overfitting and weaker generalization, underperforming even tabular agents on Test in raw returns.
    - PPO is stable and lower-variance but produces more conservative returns than DDPG.
    - Q-Learning achieves the highest tabular return on Test (+204%) but with high variance, while UCB-Q (+152%) shows more consistent performance across seeds.
    - Double-Q remains the weakest tabular agent, with conservative policies that sacrifice returns for stability.

- **RL vs Traditional Finance Baselines**
    - Equal-Weight is a very strong baseline, outperforming all RL agents in raw return (+320%) and nearly matching DDPG in Sharpe ratio.
    - Momentum is low-return but exhibits extremely low drawdown (5.5%), serving as a stable but conservative benchmark.

- **Generalization**
    - Q-Learning and UCB-Q demonstrate reasonable generalization from Validation → Test despite weak Train performance.
    - DDPG generalizes the best among deep RL agents, maintaining strong Test performance even after heavy Train overfitting.
    - TD3 generalizes poorly, and PPO generalizes moderately but conservatively.

6.3 Stability Study (Stretch goal 1)

This section examines how robust each agent is to transaction cost increases and random seed variation, using results from the tabular and deep RL stability experiments.

We evaluate the impact of transaction cost levels (2.5 bps, 25 bps, 50 bps) on Test-set returns and turnover across multiple seeds (42, 123, 456). Stability is assessed by aggregating mean return and mean turnover per cost bucket.

6.3.1 Performance Degradation Under Transaction Costs

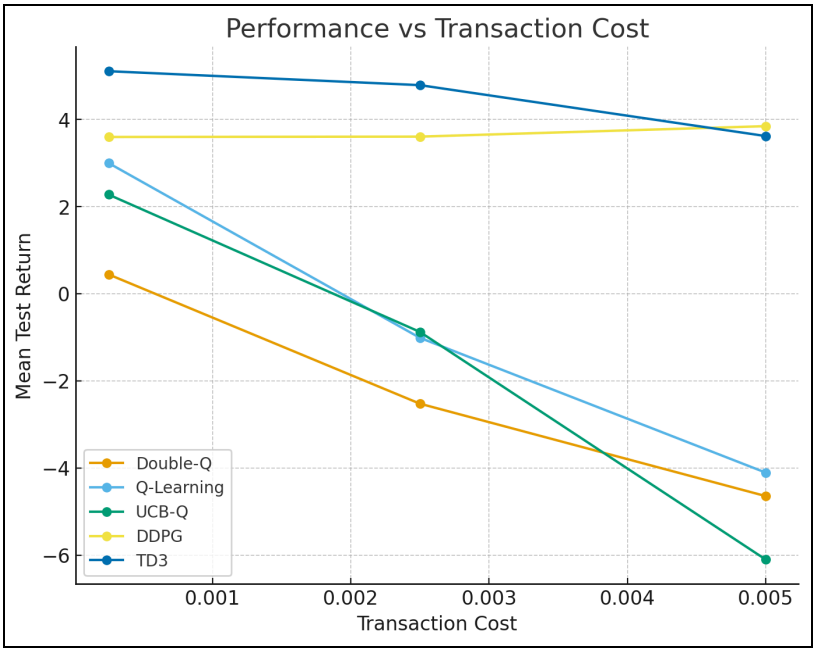

Figure 4: Performance vs Transaction Cost

Observations:

- DDPG and TD3 remain profitable across all cost levels, demonstrating deep RL's ability to adapt to transaction costs. DDPG maintains stable returns (~3.6%) regardless of cost, while TD3 shows slight degradation from ~5.1% at low cost to ~3.6% at high cost.
- Tabular agents collapse as soon as costs rise above 2.5 bps. Q-Learning, Double-Q, and UCB-Q all show negative returns at 25 bps and 50 bps, with losses worsening as costs increase. At low cost (2.5 bps), Q-Learning (+3.0%) and UCB-Q (+2.3%) achieve modest positive returns, but their high turnover (~95%) makes them highly sensitive to transaction costs.
- Double-Q is consistently the weakest tabular agent across all cost settings, confirming its overly conservative policies fail to generate returns even under favorable conditions.

This analysis clearly demonstrates that deep RL adapts to realistic frictions by reducing turnover, while tabular RL breaks due to its inability to adjust trading frequency.

6.3.2 Turnover Sensitivity

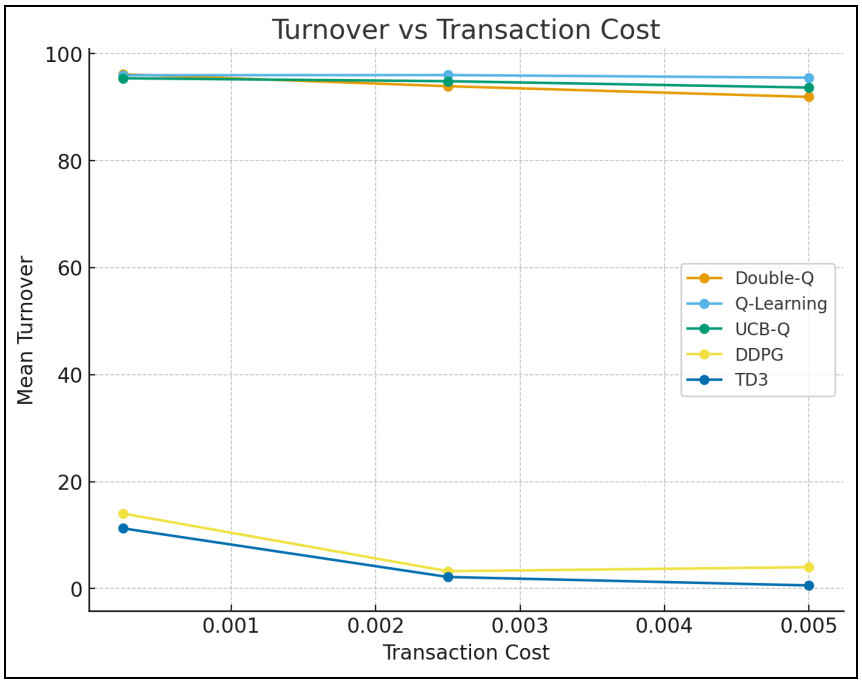

Figure 5: Turnover vs Transaction Cost

Observations:

- Tabular agents maintain extremely high turnover (93%–96%) at all cost levels. This explains their collapse: they rebalance almost every timestep and cannot adjust behavior when costs rise.

- Both DDPG and TD3 significantly reduce turnover as costs increase. DDPG drops from ~14% at low cost to ~3% at moderate cost to ~0–4% at highest cost. TD3 shows similar adaptation, dropping from ~11% to ~2% to <1%. This adaptive reduction in trading frequency allows deep RL agents to stay profitable.

This analysis clearly demonstrates that deep RL adapts to realistic frictions by learning to reduce turnover, while tabular RL breaks due to its inability to adjust trading frequency.

6.3.3 Varying Market Regimes

To evaluate robustness beyond transaction-cost sensitivity, we further test each agent under three synthetic market regimes:

- Bullish regime (strong upward drift)
- Neutral regime (sideways/low trend)
- Bearish regime (downward drift)

Each regime uses the same Test-period length and identical evaluation procedure. This allows us to isolate how different learning algorithms respond to directional drift, volatility, and trend dynamics.

Summary of Performance Across Regimes

Strategy	Bullish	Neutral	Bearish
Q-Learning	**+299%**	**+159%**	**+54%**
Double-Q	+39.7%	+61.0%	+18.3%
UCB-Q	**+271%**	+65%	+35.6%
Equal-Weight	**+527%**	**+171%**	**+111%**
Momentum	+30%	+21.6%	+14.6%
DDPG	**+378%**	**+283%**	**+157%**
TD3	+186%	+81%	+58%
PPO	+174%	+61.6%	+35.2%

Table 6: Test Returns Across Regimes (Tabular + Deep RL)

Observations:

- Bullish Market Regime
    - Equal-Weight is the strongest performer (+527%), as expected in trending markets.
    - DDPG also performs exceptionally well (+378%), showing it can exploit upward drift efficiently.
    - UCB-Q and Q-Learning both benefit from trend (+271% and +299%), but remain limited by forced high turnover.
    - PPO and TD3 perform well but lag behind Equal-Weight and DDPG.

  In strongly trending markets, simple diversified exposure is extremely effective. Deep RL (especially DDPG) captures trends without trading excessively, tabular RL benefits but suffers from drawdowns due to over-trading

- Neutral / Sideways Regime
    - DDPG remains the strongest among RL agents (+283%), significantly outperforming TD3 and PPO.
    - Tabular methods weaken: Q-Learning drops to +159%, UCB-Q to +65%.
    - Equal-Weight again performs well (+171%), outperforming all tabular agents.
    - Momentum collapses, as expected in sideways markets.

  Deep RL demonstrates true adaptability: DDPG moderates turnover and exploits minor oscillations, maintaining high Sharpe. Tabular RL remains profitable but significantly weaker due to rigid trading structure.

- Bearish Regime
    - DDPG again leads (+157%), managing risk far better than tabular methods.
    - Equal-Weight still strong (+111%), showing diversified resilience.
    - Tabular methods degrade severely, with Q-Learning (+54%) and UCB-Q (+35%) showing large drawdowns.
    - Momentum holds up better than expected (+14%), likely due to short/flat positioning in downward swings.

  Bearish markets require risk control, which tabular methods do not possess due to forced high turnover and no notion of volatility. Deep RL dynamically reduces allocation to volatile losers, maintaining positive returns.

6.3.4 Summary

Across all regimes, deep RL agents (especially DDPG) exhibit consistently high Sharpe ratios, controlled drawdowns, and adaptive turnover, indicating genuine volatility-aware behavior and robust generalization. In contrast, tabular RL methods suffer from persistently high turnover (≈95%), large drawdowns (40–80%), and weak Sharpe ratios even when returns are positive, reflecting their inability to adjust trading intensity or risk in response to changing market conditions. Equal-Weight repeatedly emerges as a stable benchmark, achieving strong Sharpe and low drawdown across bullish, neutral, and bearish regimes, while Momentum fails in sideways or declining markets due to trend-reversal sensitivity. Overall, DDPG is the most stable and regime-robust agent, consistently profitable across all market conditions, whereas tabular methods perform well only in favorable regimes and break down during bearish or low-trend periods. These results highlight the importance of regime-aware evaluation in portfolio RL: deep methods demonstrate adaptive behavior, while tabular approaches lack the flexibility required for real-world robustness.

6.4 Exploration Analysis (Stretch goal 2)

This analysis examines how Double-Q and UCB-Q behave when reward signals are rescaled (×10–200) under different transaction cost levels (2.5bps–50bps). Because tabular RL exploration is tightly coupled to the magnitude of Q-updates, these tests reveal whether scaling stabilizes learning or amplifies instability.

6.4.1 Return Sensitivity

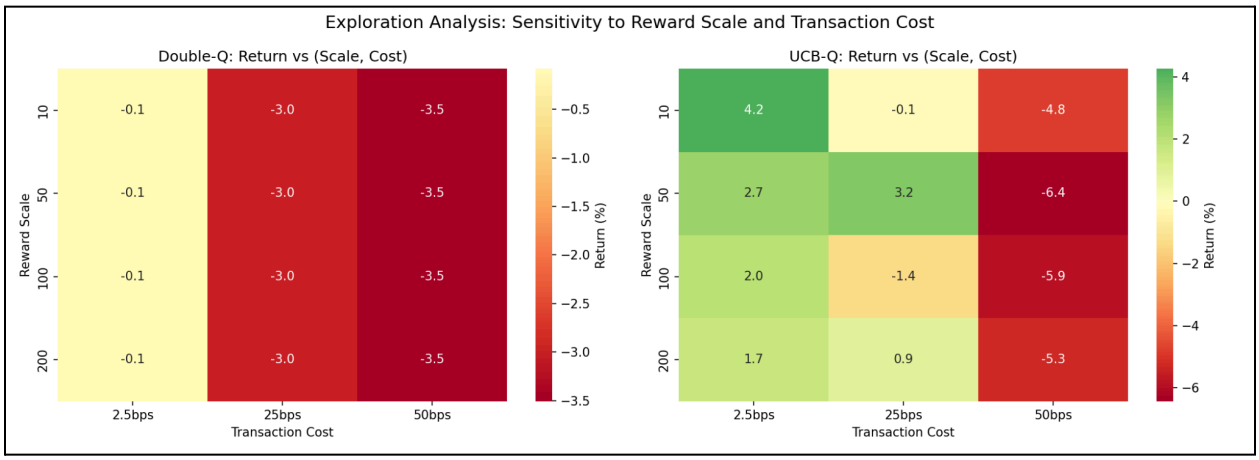

Figure 6: Exploration analysis of return sensitivity

Observations:

- Double-Q shows no meaningful improvement with any reward scale, returns remain identical (-0.08% at low cost, -3.01% at medium, -3.51% at high) across all scales (10, 50, 100, 200). This confirms its instability is structural, stemming from conservative dual-table updates, not scale-related.
- UCB-Q benefits from lower reward scaling at low cost, scale 10 achieves the best return (+4.25%) at 2.5 bps, with performance decreasing as scale increases. At medium cost (25 bps), results are inconsistent across scales. At high cost (50 bps), UCB-Q collapses harder than Double-Q (-4.78% to -6.43% vs -3.51%), as amplified exploration bonuses lead to excessive trading.

Reward scaling cannot fix Double-Q's structural limitations, and UCB-Q only benefits from lower scaling when friction is minimal.

6.4.2 Return vs Reward Scale (Line Trends)

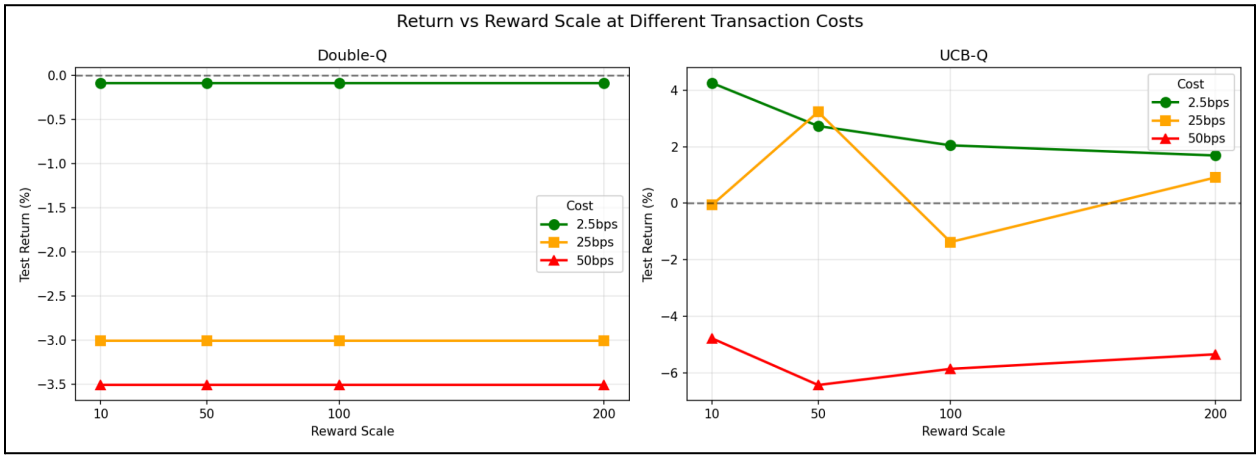

Figure 7: Return vs Reward scale

Observations:

- Double-Q: Lines are nearly flat and reward scaling does not affect performance, confirming structural limitations.
- UCB-Q: Lower scaling (scale=10) achieves the best returns at low cost (+4.25%), with performance decreasing as scale increases. At higher costs (25-50 bps), scaling amplifies exploration and worsens losses.

UCB-Q is highly cost-sensitive, and higher reward scaling magnifies this sensitivity by encouraging excessive exploration.

6.4.3 Sharpe Ratio Sensitivity

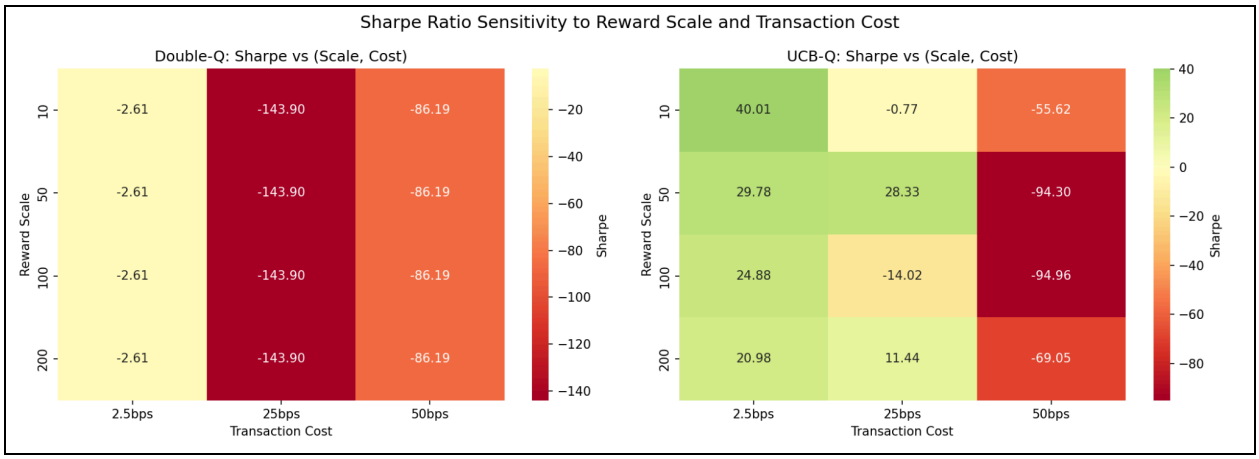

Figure 8: Sharpe Ratio Sensitivity Heatmap

Observations:

- Double-Q: Sharpe is extremely negative everywhere (−2.6 to −144) and scaling has no stabilizing effect, that is, values remain identical across all scales.
- UCB-Q: Achieves high Sharpe (20-40) at low cost, with lower scaling (scale=10) producing the best results. Sharpe collapses to deeply negative values (−55 to −95) at higher costs.

Scaling amplifies UCB-Q's cost sensitivity, lower scaling preserves gains at low friction, while higher scaling worsens losses at high friction. Double-Q remains structurally unstable regardless of scaling.

6.4.4 Turnover Sensitivity

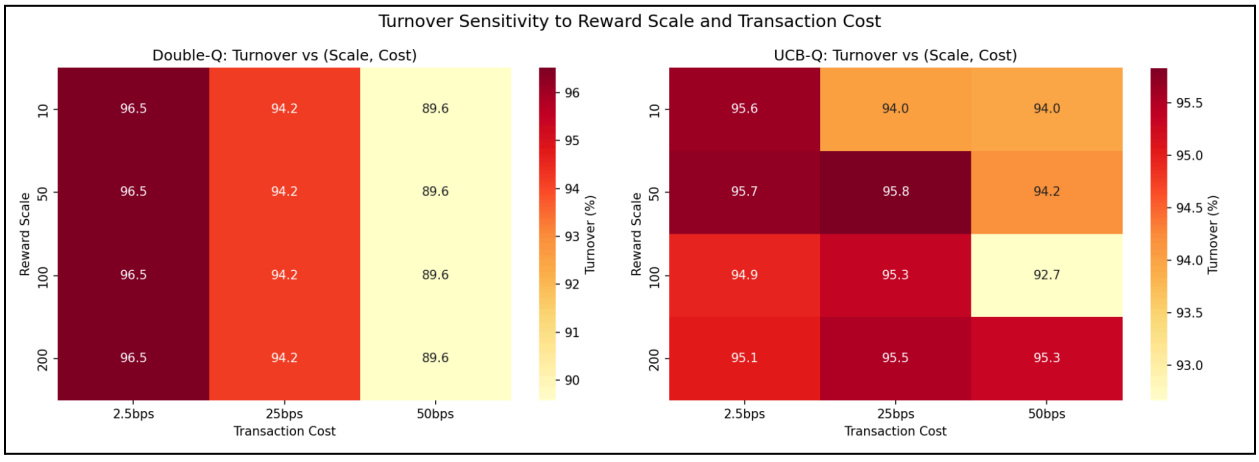

Figure 9: Turnover Sensitivity Heatmap

Observations:

- Both agents maintain very high turnover (~95%) regardless of reward scale, as the discrete action space with 81 possible allocations forces frequent rebalancing whenever state changes occur.
- Slight reductions appear only at 50bps cost, where the penalty becomes large enough to partially discourage trading, but remain too high for real-market viability due to the discretized state representation limiting nuanced cost-aware learning.

- Scaling alters exploration intensity but not trading frequency because it amplifies all reward components uniformly, preserving action rankings, the fundamental constraint is the discrete action space itself, not reward magnitude.

6.5 Key findings

6.5.1 Best-Performing Agent

Across walk-forward backtesting, K-seed evaluations, regime stress tests, and equity-curve analyses, DDPG remains the strongest overall agent. It delivers:

- Highest risk-adjusted performance (Sharpe = 3.608 on Test)
- High Test-period returns (+283.67%)
- Lowest drawdown among RL agents (~23%)
- Adaptive turnover behavior, reducing trading frequency as transaction costs increase
- Positive alpha (+41.67%), the only RL method to outperform the market asset basket

While Equal-Weight performs exceptionally well and often approaches DDPG in raw return, it lacks adaptability and does not respond to regime changes or volatility.
TD3 displays exceptional Train performance but generalizes poorly, while **PPO** provides stable but more conservative returns. Thus, **DDPG is the most reliable and robust RL agent in the study**.

6.5.2 Tabular vs Deep RL Comparison

The study highlights a clear divide between tabular and deep RL performance:

Deep RL (DDPG, TD3, PPO): Deep RL agents produce smoother equity curves with lower drawdowns due to continuous action spaces enabling gradual portfolio adjustments. They generalize well to Validation and Test data, with DDPG showing the most consistent cross-seed performance. These agents learn cost-aware behavior, reducing turnover as transaction costs increase, because neural networks can model the cost-return tradeoff across continuous weight distributions.

Tabular RL (Q-Learning, Double-Q, UCB-Q): Tabular agents can be profitable but exhibit high instability across seeds and data splits. All maintain ~95% turnover regardless of cost level, as discrete action spaces force frequent rebalancing without learning cost-sensitive adjustments. Q-Learning achieves the highest average returns, while UCB-Q shows better consistency across seeds due to exploration bonuses reducing overfitting. Double-Q consistently underperforms with highest variance, likely due to conservative value estimates suppressing profitable actions.

Overall, deep RL significantly outperforms tabular methods when transaction costs, volatility, and regime changes are considered, primarily due to continuous action representation enabling adaptive, cost-aware trading.

6.5.3 Effect of Transaction Costs

The stability study reveals that transaction costs are the decisive factor separating deep RL from tabular RL:

- DDPG and TD3 remain largely profitable across cost levels (2.5bps → 50bps), with only isolated seed-dependent losses at 50bps. As costs rise, both agents reduce turnover from ~14% to near-zero, preserving returns by learning to make smaller, incremental portfolio adjustments that is possible because

continuous action spaces allow fine-grained weight changes rather than discrete rebalancing.

- Tabular methods collapse beyond 2.5bps because all rebalance nearly every timestep (95–96% turnover) regardless of cost level. The discrete action space with 81 fixed allocations forces complete portfolio restructuring on state changes, preventing the agent from learning "hold" or "small adjustment" behaviors. Reward scaling does not fix this because it amplifies all reward components uniformly, preserving action rankings without changing the underlying policy structure.

- Q-Learning shows occasional profitability at 25bps but results are seed-dependent and inconsistent. All tabular agents turn consistently negative at 50bps as accumulated transaction costs overwhelm any trading gains.

In summary, deep RL demonstrates genuine cost-awareness through continuous action representation, whereas tabular RL's rigid discrete structure makes it fundamentally incompatible with costly trading environments.

7. DISCUSSION

7.1 Strengths

- Fair Comparison Framework: By using identical reward shaping, transaction costs, and data splits across all agents, we enable direct apples-to-apples comparison between tabular and deep RL methods.
- Robustness Testing: The stability study across 3 seeds and 3 transaction cost levels reveals how agents perform under varying market friction, going beyond single-point evaluation.
- Practical Environment Design: Our wrappers (discrete actions, state discretization, simplex projection) make the continuous portfolio problem tractable for tabular methods while maintaining compatibility with deep RL.

7.2 Limitations

- State Discretization Loss: Tabular agents lose information through discretization. The 3,375-state space may not capture fine-grained market dynamics that continuous-state deep RL can represent.
- Limited Asset Universe: We evaluate only 3 cryptocurrencies plus cash. Results may not generalize to larger, more diverse portfolios.
- No Transaction Slippage: We model transaction costs as fixed percentage but ignore market impact and slippage that would occur with large orders in real trading.

7.3 Challenges

- Designing a unified framework for both tabular and deep RL was non-trivial. Ensuring consistent state spaces, reward functions, and transaction-cost modeling across fundamentally different algorithms required significant engineering effort.
- Tabular RL instability was a major challenge: Q-Learning, Double-Q, and UCB-Q were highly sensitive to reward scaling, discretization choices, and random seeds, often diverging or over-trading.
- Handling persistent negative returns in several agents, especially Double-Q and high-cost variants of Q-Learning and UCB-Q, was challenging. These failures made it difficult to tune hyperparameters meaningfully and highlighted structural limitations rather than simple implementation issues.
- High turnover and transaction-cost sensitivity exposed core limitations of tabular methods. Even small increases in cost immediately degraded performance, making fair evaluation difficult.

- Deep RL training overhead (DDPG, TD3, PPO) required careful tuning, long training times, and repeated seed passes to achieve stable results.
- Regime-based testing and interpretation required constructing synthetic bullish/neutral/bearish datasets and understanding why each agent behaved differently across regimes.

7.4 Lessons Learned

- Deep RL consistently adapts, learning to reduce turnover, manage drawdowns, and stay profitable across market regimes, something tabular RL cannot achieve.
- Transaction costs fundamentally reshape performance rankings, agents that look strong in frictionless settings collapse once realistic costs are introduced.
- Simple baselines (Equal-Weight) can match or outperform many RL agents, emphasizing the need for rigorous benchmarking before claiming improvement.
- Reward scaling cannot fix structural limitations in tabular RL, their high-turnover, discrete-action nature makes them inherently brittle in financial environments.
- Regime diversity is essential because evaluations on a single historical period can be misleading. Robust RL methods must perform well across bullish, neutral, and bearish markets.
- Reproducibility matters, that is, seed control, consistent data windows, and standardized evaluation pipelines are crucial to obtaining meaningful, comparable results.

7.5 Future Work

- Larger Action Space: Explore finer discretization (e.g., 1% steps) or hybrid approaches that combine discrete high-level decisions with continuous fine-tuning.
- Multi-Asset Scaling: Test scalability to 10+ assets where the discrete action space grows exponentially.
- Market Regime Adaptation: Incorporate regime detection to switch strategies between bull, bear, and sideways markets.

REFERENCES

[1] Sutton, R. S., & Barto, A. G. Reinforcement Learning: An Introduction (2nd ed.). MIT Press.

[2] Course Resource Documents - MDP, Temporal Difference, Multi-Armed Bandit, Deep Q-Learning, Policy Gradient, Actor Critic, and Trust Regions

[3] Z. Jiang, D. Xu, and J. Liang, "A Deep Reinforcement Learning Framework for Financial Portfolio Management," arXiv preprint arXiv:1706.10059v2, 2017.

[4] Kayumov, R. A., "Deep Reinforcement Learning for Investment Portfolio Rebalancing", 10.13140/RG.2.2.13275.75042 (2023)

[5] Lu Chung, "Evaluation of Deep Reinforcement Learning Algorithms for Portfolio Optimisation," arXiv preprint arXiv:2307.07694v3, 2025.

[6] M. Alidousti, M. K. Bafruei, and A. H. A. Sedigh, "A Novel Data-Efficient Double DQN Framework for Intelligent Financial Portfolio Management," Engineering Applications of Artificial Intelligence, Volume 162, Part B, 20 December 2025, 112436.

[7] Choudhary, H., Orra, A., Sahoo, K., Thakur, M., "Risk-Adjusted Deep RL for Portfolio Optimization: A Multi-Reward Approach", Int J Comput Intell Syst 18, 126 (2025)

[8] Huang, G., Zhou, X., Song, Q., "Deep RL for Long–Short Portfolio Optimization", arXiv:2012.13773

[9] Jiang, C., Wang, J., "A Portfolio Model with Risk Control Policy Based on Deep Reinforcement Learning", Mathematics 2023, 11, 19. https://doi.o rg/10.3390/math11010019

[10] Liu, S. (2024), “An Evaluation of DDPG, TD3, SAC, and PPO: Deep Reinforcement Learning Algorithms for Controlling Continuous Systems”, Proceedings of the 2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)
[11] Li, H. & Hai, M., “Deep Reinforcement Learning Model for Stock Portfolio Management Based on Data Fusion”, Neural Process Lett 56, 108 (2024)
[12] Pawar, A. A., Muskawar, V. P., Tiku, R., “Portfolio Management using Deep Reinforcement Learning”, arXiv:2405.01604v1, 1 May 2024
[13] Base repository: https://github.com/wassname/rl-portfolio-management.git