Diploma Project

INVESTIGATING THE APPLICATION OF OFFLINE REINFORCEMENT LEARNING ALGORITHMS IN DIGITAL ASSET PORTFOLIO OPTIMIZATION

Christos Iosif

University of Cyprus

Department of Computer Science

May 2025

UNIVERSITY OF CYPRUS

Department of Computer Science

Investigating the Application of Offline Reinforcement Learning Algorithms in Digital Asset Portfolio Optimization

Christos Iosif

Supervisor

Prof. Chryssis Georgiou

The Thesis was submitted in partial fulfillment of the requirements for obtaining the Computer Science degree of the Department of Computer Science of the University of Cyprus

May 2025

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Prof. Chryssis Georgiou, for his guidance, support, and insightful feedback throughout the process of this thesis. His expertise and encouragement have been instrumental in helping me achieve my academic goals and complete this work.

I also extend my thanks to Giorgos Demosthenous, a Ph.D. Student at University of Cyprus, for his exceptional guidance and support during the technical aspects of this thesis. His expertise and willingness to share knowledge were invaluable when faced with challenges. His contribution was vital for the success of this work.

Lastly, I would like to thank my friends and family for supporting me during my studies.

ABSTRACT

The increasing prominence of cryptocurrencies as an alternative asset class has significantly reshaped the financial landscape, presenting both novel opportunities and unique challenges for investors. Characterized by high volatility, non-linear dependencies, and rapidly evolving market conditions, cryptocurrencies require innovative strategies for effective portfolio management.

This thesis explores a novel approach to digital asset portfolio optimization by applying offline reinforcement learning algorithms, a machine learning paradigm that enables agents to learn optimal decision policies from fixed historical datasets, without interacting with a live environment. This approach is particularly attractive in financial domains, where extensive historical data is available but real-time experimentation is costly or impractical.

We model the portfolio optimization task as a Markov Decision Process, using historical hourly and daily price data for eleven major cryptocurrencies. This dataset is preprocessed and augmented in order to be efficiently used for offline reinforcement learning.

A comprehensive set of offline RL algorithms is evaluated, including Batch-Constrained Q-Learning (BCQ), Discrete BCQ, Conservative Q-Learning (CQL), Discrete CQL, Implicit Q-Learning (IQL), Bootstrapping Error Accumulation Reduction (BEAR), TD3 with Behavioral Cloning (TD3+BC), and the Decision Transformer (DT). Each algorithm is trained under two distinct reward functions: absolute return and Sharpe ratio. Performance is assessed using key financial metrics such as mean return, volatility, and risk-adjusted return, and is benchmarked against traditional strategies like HODL and uniformly random allocation.

Our results demonstrate that offline RL methods can match or exceed the performance of traditional strategies in digital asset markets. The analysis also underscores the sensitivity of these methods to hyperparameter tuning and the critical role of reward function design in shaping agent behavior.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

RL	Reinforcement Learning
MDP	Markov Decision Process
TD	Temporal Difference
BCQ	Batch Constrained Q-Learning
CQL	Conservative Q-Learning
IQL	Implicit Q-Learning
BEAR	Bootstrapping Error Accumulation Reduction
BC	Behavioral Cloning
DT	Decision Transformer
VAE	Variational Autoencoder
SAC	Soft Actor-Critic
AWR	Advantage-Weighted Regression
MMD	Maximum Mean Discrepancy
GPT	Generative Pre-trained Transformer
OLHCV	Open, Low, High, Close, Volume
USD	United Stated Dollar
BTC	Bitcoin
ETH	Ethereum
DeFi	Decentralized Finance
API	Application Programming Interface
AVG	Average
Std dev	Standard Deviation
OOD	Out Of Distribution

1 Introduction

1.1 Motivation

The *portfolio optimization problem* [1] has been a central focus of financial research for decades, as investors and researchers strive to maximize returns while minimizing risk. Over the past ten years, the financial landscape has undergone a significant transformation with the emergence of a new asset class: *cryptocurrencies* [2]. Among these, *Bitcoin* [2] has emerged as the most prominent, capturing the attention of both individual and institutional investors. Unlike traditional assets such as stocks, bonds, and commodities, cryptocurrencies are characterized by extreme volatility and high risk, presenting unique challenges for portfolio management. This unpredictability has made it increasingly important to develop innovative strategies that can effectively incorporate these digital assets into investment portfolios.

Over the years, numerous techniques have been proposed to address the portfolio optimization problem, ranging from traditional mean-variance optimization to more advanced approaches [3]. However, the effectiveness of these methods varies, and many struggle to adapt to the dynamic and unpredictable nature of financial markets. With the rapid advancement of machine learning and artificial intelligence, new opportunities have emerged to tackle this complex problem. Researchers have explored a wide array of techniques, including statistical models, supervised learning, evolutionary algorithms, and reinforcement learning. Despite these efforts, no single approach has achieved widespread success, particularly when applied to the unique challenges posed by cryptocurrencies.

In this thesis, we explore a promising and novel direction in the field of portfolio optimization by leveraging a machine learning technique that has not yet been applied to this task. *Offline reinforcement learning* [4] offers a unique combination of strengths, blending the flexibility and adaptive learning capabilities of traditional reinforcement learning with the ability to utilize vast amounts of historical data. This makes it particularly well-suited for financial applications, where large datasets are readily available, but real-time interaction with the environment is often impractical or costly. By testing a variety of offline reinforcement learning algorithms, we aim to uncover new insights into their potential for optimizing portfolios, especially in the context

of highly volatile assets like cryptocurrencies. Our work seeks to contribute to the ongoing evolution of portfolio management strategies, offering a fresh perspective on how cutting-edge machine learning techniques can address one of finance's most enduring challenges.

1.2 Goals of the Study

The primary objective of this study is to explore the potential of offline reinforcement learning algorithms in addressing the complex challenges of cryptocurrency portfolio optimization. To achieve this, we aim to train and evaluate a diverse set of offline RL algorithms, leveraging their ability to learn from historical data without requiring real-time interaction with the environment.

A critical aspect of this study involves evaluating different *hyperparameter* [5] combinations for each algorithm to assess their impact on performance. This systematic exploration aims to identify the most effective configurations for optimizing portfolio performance, particularly in cryptocurrency markets. By comparing the results across algorithms, we seek to determine which offline reinforcement learning methods are best suited for managing portfolios that include digital assets. Additionally, the study examines how different reward functions influence outcomes, providing deeper insights into the strengths and limitations of each approach in addressing the unique challenges posed by cryptocurrencies.

1.3 Methodology

Firstly, we selected datasets comprising time series data for eleven cryptocurrencies, segmented into hourly and daily intervals. The next step involved formulating the portfolio optimization problem as a reinforcement learning task, specifically modeling it as a *Markov Decision Process* (MDP) [6]. This required defining the state space, action space, and reward function in a way that accurately reflected the dynamics of cryptocurrency trading.

To evaluate the selected algorithms, we selected a reliable and robust framework that offered pre-built implementations of offline RL methods. Next, we realized that the dataset was incomplete, lacking certain critical elements necessary for training offline RL algorithms. To address this, we devised a creative solution to generate the missing data, ensuring the dataset was both comprehensive and suitable for training.

With the dataset complete, we normalized the data to ensure consistency and improve the stability of the training process. We trained a total of 64 different models with different hyperparameter combinations, conducted on the High-Performance Computing (HPC) infrastructure at the University of Cyprus.

Finally, after completing the experiments, we analyzed the results to evaluate the performance

of each algorithm. This analysis involved comparing the algorithms based on their ability to optimize cryptocurrency portfolios, with a focus on metrics such as Sharpe Ratio.

1.4 Document Organization

Chapter	Description
Chapter 2	This chapter introduces the foundational concepts necessary to contextualize the thesis. We begin by defining portfolio optimization and cryptocurrencies. Next, we present the core principles of classical reinforcement learning, followed by a detailed exploration of offline RL and its distinguishing characteristics. Each algorithm is examined in depth, with a focus on its theoretical underpinnings and practical implications. The chapter concludes with a discussion of related work, positioning this research within the broader landscape of existing studies.
Chapter 3	This chapter details the implementation methodology of our study. We begin by describing the dataset's characteristics and the pre-processing steps applied. Next, we present the formulation of the problem as a Markov Decision Process (MDP). Following this, we explain the techniques employed to handle missing values within the dataset.
Chapter 4	This chapter outlines the experimental methodology and evaluates the results obtained from various hyperparameter configurations and reward functions, providing a detailed analysis for each scenario.
Chapter 5	This final chapter evaluates the overall performance of the proposed algorithms, summarizes the key findings, and suggests potential directions for future research.

Table 1.1: Chapter Descriptions

2 Background Knowledge

2.1 Portfolio Optimization Definition

Portfolio optimization is the systematic process of selecting the most efficient allocation of assets from a set of possible portfolios, guided by specific objectives [1]. These objectives typically involve maximizing desirable factors, such as expected returns, while simultaneously minimizing associated costs, such as financial risk [1]. This dual focus creates a multi-objective optimization problem, where the goal is to strike an optimal balance between risk and reward. The complexity of this problem arises from the need to consider various constraints, market dynamics, and the interplay between different assets, making it a fundamental challenge in both theoretical finance and practical investment management.

2.2 Introduction to Cryptocurrencies

A cryptocurrency (commonly referred to as ”crypto”) is a form of digital currency that operates on a decentralized computer network, eliminating the need for reliance on a central authority, such as a government or financial institution, to manage or regulate it [2]. At the core of cryptocurrencies is the *blockchain* [2], a distributed digital ledger that records all transactions in a secure, transparent, and immutable manner. This ledger uses a consensus mechanism, such as Proof of Work (PoW), to validate transactions, control the creation of new units (coins), and verify the transfer of ownership. The blockchain's decentralized nature ensures that every transaction is publicly recorded and tamper-proof, providing a high level of transparency and security.

Despite the term ”cryptocurrency,” these digital assets function less like traditional currencies and more like a non-correlated asset class, distinguished by decentralization, extreme volatility, and non-linear return profiles. Their inclusion in portfolios is driven by their low historical correlation with conventional assets, offering potential diversification benefits and efficiency gains in optimized allocations.

2.3 Reinforcement Learning

Reinforcement Learning (RL) [6] is a process of learning how to map situations to actions in order to maximize a numerical reward signal. Unlike other learning methods, the learner is not explicitly instructed on which actions to take. Instead, it must explore and determine which actions produce the highest rewards through trial and error. In more complex scenarios, actions not only influence the immediate reward but also impact future situations and, consequently, all subsequent rewards. The two key features that define reinforcement learning are trial-and-error exploration and the handling of delayed rewards, which set it apart from other learning paradigms.

2.3.1 Essential Elements of Reinforcement Learning

2.3.1.1 Policy

A *policy* [6] dictates how a learning agent behaves at any given moment. Essentially, it serves as a mapping from the perceived states of the environment to the actions the agent should take in those states. In some instances, the policy may be straightforward, such as a simple function or a lookup table. In more complex cases, it might require extensive computational processes, like a search algorithm. The policy is the heart of a reinforcement learning agent because it

alone determines the agent's behavior. Policies can also be stochastic, meaning they assign probabilities to each possible action rather than prescribing a single definitive action .

2.3.1.2 Reward Signal

A *reward signal* [6] establishes the objective of a reinforcement learning problem. At each time step, the environment provides the reinforcement learning agent with a single numerical value, known as the reward. The agent's primary goal is to maximize the cumulative reward it accumulates over time. This reward signal essentially defines what constitutes positive and negative outcomes for the agent, serving as the immediate and fundamental aspect of the problem it must solve. The reward signal is also the key driver for modifying the agent's policy; if an action chosen by the policy results in a low reward, the policy may be adjusted to favor different actions in similar situations in the future. In many cases, reward signals can be stochastic, meaning they depend probabilistically on the state of the environment and the actions taken by the agent .

2.3.1.3 Value Function

While the reward signal reflects what is beneficial in the short term, a *value function* [6] defines what is advantageous over the long term. Essentially, the value of a state represents the total reward an agent can expect to accumulate in the future, starting from that state. Rewards capture the immediate, intrinsic appeal of a given state, whereas values reflect the long-term desirability of states, considering the likelihood of subsequent states and the rewards they may offer. For instance, a state might provide a low immediate reward but still hold a high value if it frequently leads to other states that yield substantial rewards. Conversely, a state with a high immediate reward might have a low value if it often results in unfavorable future states .

2.3.1.4 Environment Model

The fourth and final component of certain reinforcement learning systems is an *environment model* [6]. This model simulates the behavior of the environment or, more broadly, enables predictions about how the environment will respond. For example, given a specific state and action, the model can forecast the resulting next state and the associated reward. Models are primarily used for planning, which refers to the process of determining a course of action by anticipating potential future scenarios before they are encountered. Reinforcement learning methods that incorporate models and planning are referred to as model-based methods. These contrast with model-free methods, which rely solely on trial-and-error learning and do not use explicit models, representing an approach that is often seen as the opposite of planning.

2.3.1.5 Markov Decision Process

A *Markov Decision Process* [6] is a mathematical framework designed to model decision-making in environments where outcomes are influenced by both randomness and the choices of a decision-maker. It is defined by four key components: a set of states, a set of actions, transition probabilities that describe how the system moves from one state to another based on the chosen action, and a reward function that provides feedback for each transition. At each time step, the process is in a specific state, and the decision-maker selects an action. This action triggers a transition to a new state according to the transition probabilities, and a corresponding reward is received. In the case of a finite MDP we assume that the state, action, and reward sets are finite . The following figure (Figure 2.1) shows an illustration of the process:

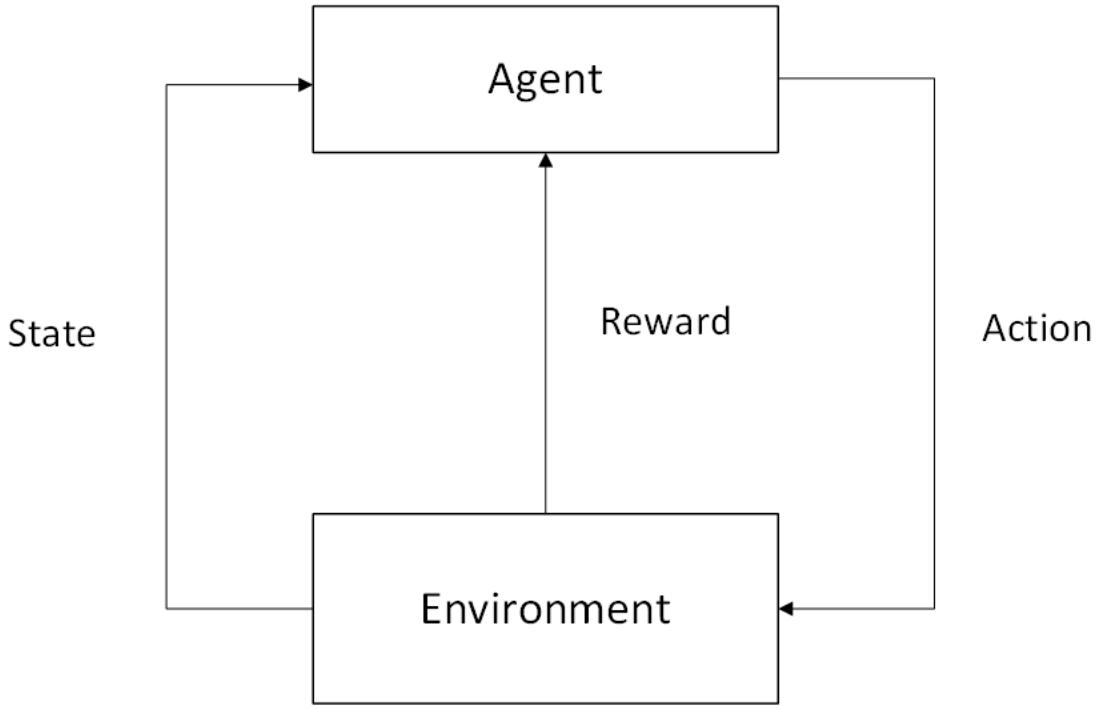


Figure 2.1: The agent–environment interaction in a Markov decision process.

2.3.2 Tabular Solution Methods

In its simplest form, reinforcement learning deals with problems where the state and action spaces are small enough for value functions to be represented as arrays or tables. In such cases, the methods can often achieve exact solutions, meaning they can precisely determine the optimal value function and the optimal policy [6].

2.3.2.1 Dynamic Programming

The term *Dynamic Programming* (DP) [6] refers to a collection of algorithms used to compute optimal policies when a perfect model of the environment is available, represented as a Markov decision process. While classical DP algorithms are of limited practical use in reinforcement learning due to their reliance on a perfect model and their high computational cost, they remain theoretically significant. DP serves as a crucial foundation for understanding many modern methods, as it provides a framework for achieving optimal decision-making. In essence, many reinforcement learning techniques can be seen as efforts to replicate the effectiveness of DP but with reduced computational demands and without requiring a perfect model of the environment [6].

2.3.2.2 Monte Carlo Methods

Unlike dynamic programming (DP), *Monte Carlo* [6] methods do not assume complete knowledge of the environment. Instead, they rely solely on experience, sample sequences of states, actions, and rewards obtained through actual or simulated interaction with the environment. Learning from actual experience is particularly remarkable because it does not require any prior knowledge of the environment's dynamics, yet it can still lead to optimal behavior. Learning from simulated experience is equally powerful. While a model is necessary in this case, it only needs to generate sample transitions rather than the full probability distributions of all possible transitions, as required by DP. In many practical scenarios, it is surprisingly straightforward to generate experience that follows the desired probability distributions, even when obtaining those distributions explicitly is impractical or infeasible [6].

2.3.2.3 Temporal-Difference Learning

Temporal Difference (TD) [6] learning combines ideas from both Monte Carlo methods and dynamic programming (DP). Similar to Monte Carlo approaches, TD methods can learn directly from raw experience without requiring a model of the environment's dynamics. Like DP, TD methods update their estimates by bootstrapping, using other learned estimates to refine predictions without waiting for a final outcome. This interplay between TD, DP, and Monte Carlo methods is a central and recurring theme in the theory of reinforcement learning, highlighting how these approaches complement and build upon one another [6]. The TD(0) algorithm is shown below in procedural form (Algorithm 1).

Algorithm 1 Tabular TD(0)

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in S^+$, arbitrarily except that $V(\text{terminal}) = 0$
for each episode **do**
    Initialize S
    for each step of episode **do**
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha \times [R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    until S is terminal

2.3.2.4 On-Policy SARSA

The process begins by focusing on learning an action-value function instead of a state-value function. Specifically, in the context of on-policy methods, the goal is to estimate $q_\pi(s, a)$ for the current behavior policy π, across all states s and actions a. This estimation can be accomplished using a Temporal Difference (TD) approach, similar to the one discussed earlier. In previous sections, we examined transitions from state to state, learning the values of states. Now, we extend this to transitions between state-action pairs, aiming to learn the values of these pairs. Formally, both cases are analogous, as they represent Markov chains with an associated reward process. From this foundation, it is straightforward to design an on-policy control algorithm based on the SARSA prediction method. As with all on-policy approaches, we continuously estimate q_π for the behavior policy π, while simultaneously refining π to become more greedy with respect to q_π. This dual process of estimation and policy improvement is central to on-policy reinforcement learning [6]. The SARSA algorithm is shown below in procedural form (Algorithm 2).

Algorithm 2 SARSA

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a)$, for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
for each episode **do**
    Initialize S
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    for each step of episode **do**
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha \times [R + \gamma Q(S', A') - Q(S, A)]$
        $S \leftarrow S'$; $A \leftarrow A'$
    until S is terminal

2.3.2.5 Off-Policy Q-Learning

In this scenario, the learned action-value function, Q, directly approximates q^*, the optimal action-value function, regardless of the policy being followed. Simply put, in off-policy learning we seek to learn a value function for a target policy π, given data due to a different behavior policy b. This significantly simplifies the algorithm's analysis and has enabled early proofs of convergence. While the policy still plays a role by determining which state-action pairs are visited and updated, the only requirement for correct convergence is that all pairs continue to be updated over time. This ensures that the algorithm progressively refines its estimates toward the optimal values [6]. The Q-learning algorithm is shown below in procedural form (Algorithm 3).

Algorithm 3 Q-Learning

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a)$, for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
for each episode **do**
    Initialize S
    for each step of episode **do**
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha \times [R + \gamma \max_a Q(S', a) - Q(S, A)]$
        $S \leftarrow S'$
    until S is terminal

2.3.3 Approximate Solution Methods

In many real-world tasks where reinforcement learning is applied, the state space is combinatorial and extremely large. For instance, the number of possible camera images far exceeds the number of atoms in the universe. In such cases, finding an optimal policy or the optimal value function is infeasible, even with infinite time and data. Instead, the goal shifts to finding a good approximate solution using limited computational resources [6].

The challenge with large state spaces is not only the memory required to store large tables but also the time and data needed to populate them accurately. In many practical tasks, almost every state encountered will be unique, having never been seen before. To make informed decisions in such states, the agent must generalize from previous experiences with states that are similar in some sense. This raises the central issue of generalization: How can experience with a limited subset of the state space be effectively generalized to produce a reliable approximation over a much larger subset?

To address this, we can combine reinforcement learning methods with existing generalization

techniques. The type of generalization required is often referred to as function approximation, as it involves using examples from a desired function (e.g., a value function) to construct an approximation of the entire function. Function approximation is essentially an instance of supervised learning, where the goal is to learn a mapping from inputs (states) to outputs (values or actions) based on observed data. This integration allows reinforcement learning algorithms to scale to problems with vast state spaces by leveraging patterns and similarities in the data [6].

2.3.4 Hyper-Parameters

Hyper-parameters [5] are parameters whose values are set before the learning process of a machine learning model begins. Unlike model parameters (e.g., weights in a neural network), hyper-parameters are not learned from the data but are instead configured by the practitioner to control the learning process. Common hyper-parameters in both RL and offline RL are:

- Actor and Critic learning rate: controls how much a model's parameters are adjusted during training.
- Discount Factor Gamma γ: determines the importance of future rewards, with values between 0 (myopic) and 1 (far-sighted).
- Batch Size: the number of experience samples (e.g., state-action-reward tuples) used in a single training update.

2.4 Offline Reinforcement Learning

The online learning model of reinforcement learning algorithms, while a core strength, also poses significant challenges to their broader adoption. Traditional RL relies on iteratively collecting experience by interacting with the environment, typically using the latest learned policy, and then using that experience to improve the policy. However, in many real-world scenarios, this online interaction is impractical. Data collection can be prohibitively expensive (e.g., in robotics, educational agents, or healthcare) or dangerous (e.g., in autonomous driving or healthcare). Moreover, even in domains where online interaction is possible, there is often a preference to leverage previously collected data—particularly in complex environments where large datasets are essential for effective generalization. These challenges underscore the need for approaches that can learn effectively from offline data, minimizing the risks and costs associated with online interaction [4].

2.4.1 Offline RL Workflow

Offline reinforcement learning utilizes a fixed dataset D, which is collected by some (potentially unknown) behavior policy. This dataset is gathered once and remains unchanged throughout the training process, making it possible to leverage large, pre-existing datasets. During training, the algorithm does not interact with the Markov Decision Process (MDP) at all. The policy is deployed only after it has been fully trained, ensuring that all learning occurs offline without any additional environmental interaction. The following figure (Figure 2.2) shows an illustration of the process:

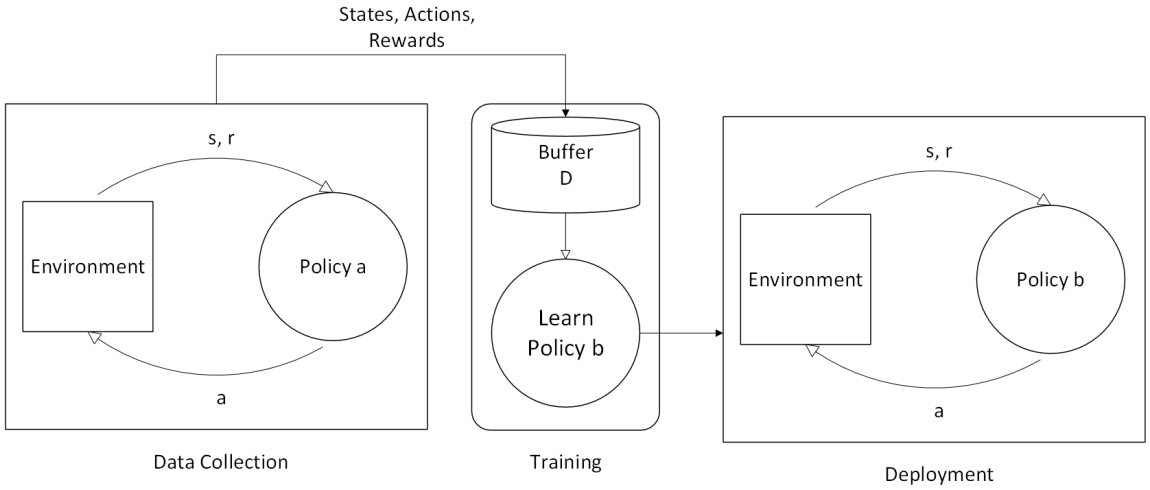


Figure 2.2: Illustration of offline RL workflow.

2.4.2 Offline RL Challenges

One of the most apparent challenges in offline reinforcement learning is its complete reliance on a static dataset D. Since the learning algorithm cannot interact with the environment, there is no opportunity to improve exploration. Exploration lies entirely outside the algorithm's scope, meaning that if D lacks transitions from high-reward regions of the state space, discovering those regions may be impossible. This limitation highlights the critical importance of the dataset's quality and coverage for successful offline RL [4].

A more subtle yet practically significant challenge is that offline reinforcement learning fundamentally revolves around counterfactual reasoning. Counterfactual queries involve asking "what if" questions, hypotheses about what might happen if the agent were to take actions different from those recorded in the dataset. This is essential because, to improve upon the behavior policy that generated D, the learned policy must deviate from the actions observed in the data. However, this requirement pushes the boundaries of many current machine learning tools, which are designed under the assumption that data is independent and identically distributed (i.i.d.). In

standard supervised learning, the goal is to perform well on data drawn from the same distribution as the training set. In contrast, offline RL aims to learn a policy that behaves differently, and ideally better, than the behavior policy reflected in D [4].

The core difficulty in addressing these counterfactual queries is *distributional shift* [4]. While the function approximator (e.g., the policy, value function) is trained on data from one distribution, it is evaluated on a different distribution. This shift arises from two factors: (1) the new policy visits states that may differ from those in D, and (2) the process of maximizing expected return inherently alters the distribution of states and actions.

2.4.3 Batch Constrained Q-Learning

Batch-Constrained Q-learning (BCQ) [7] is one of the first deep reinforcement learning algorithms specifically designed to learn effectively from fixed, offline datasets. Its core idea is to constrain the action space, ensuring that the agent's behavior remains close to the data distribution in the provided batch. By doing so, BCQ mitigates the risks of distributional shift and extrapolation errors, which are common challenges in offline reinforcement learning. This approach allows the algorithm to learn robust policies without requiring additional interaction with the environment.

2.4.3.1 Motivation

Most modern off-policy deep reinforcement learning algorithms operate in a growing batch setting, where data is collected, stored in an experience replay buffer, and used to train the agent before further data collection. However, these algorithms often fail in a pure batch setting, especially when the dataset is not representative of the true distribution under the current policy. Surprisingly, off-policy agents can perform significantly worse than the behavioral agent that generated the dataset, even when trained with the same algorithm. This failure stems from extrapolation error, a fundamental issue in off-policy RL where unseen state-action pairs are assigned unrealistic values due to a mismatch between the policy-induced data distribution and the batch data distribution [7]. As a result, learning an accurate value function for policies that select actions outside the batch becomes highly challenging.

2.4.3.2 Main Idea

Current off-policy deep reinforcement learning algorithms struggle with extrapolation error because they select actions based on learned value estimates without considering the reliability of those estimates. This can lead to overestimating the value of out-of-distribution actions that are not well-represented in the dataset. However, the value of an off-policy agent can be accurately

assessed in regions where sufficient data is available. The solution is conceptually straightforward: to prevent extrapolation error, a policy should induce a similar state-action visitation to the batch [7].

2.4.3.3 Key Components of BCQ

- **Generative Model for Action Constraints:** BCQ employs a generative model (e.g., a Variational Autoencoder) trained on the actions present in the dataset. This model captures the distribution of actions conditioned on states, enabling the generation of candidate actions that are similar to those in the batch.
- **Perturbation Model:** A perturbation model is used to make small adjustments to the actions generated by the generative model. This allows the policy to explore slight variations of the existing actions without significantly deviating from the data distribution.
- **Q-Networks:** BCQ uses two Q-networks to estimate the value of state-action pairs. The Q-networks are trained using a modified version of Clipped Double Q-learning [8], which takes the minimum of the two Q-values to reduce overestimation bias and penalize uncertainty in regions where data is scarce. The learning target for the Q-networks is a convex combination of the minimum and maximum Q-values, which helps to balance exploration and exploitation.
- **Policy Improvement:** During policy improvement, BCQ selects the action with the highest Q-value from the set of generated and perturbed actions. This ensures that the policy remains within the support of the batch data while still seeking to improve performance.

2.4.3.4 Algorithm Overview

The Batch-Constrained Q-learning (BCQ) algorithm begins by initializing a generative model $G_\omega(s)$, a perturbation model $\xi_\phi(s, a, \Phi)$, and two Q-networks Q_{θ_1} and Q_{θ_2}. During training, a batch of transitions (s, a, r, s') is sampled from the fixed dataset. Candidate actions $a_i \sim G_\omega(s')$ are generated and perturbed using $\xi_\phi(s', a_i, \Phi)$. The target Q-value y is computed as:

$$y = r + \gamma \max_{a_i} \left[\lambda \min_{j=1,2} Q_{\theta'_j}(s', a_i) + (1 - \lambda) \max_{j=1,2} Q_{\theta'_j}(s', a_i) \right] \tag{2.1}$$

The Q-networks are updated using this target, and the perturbation model is updated to maximize the Q-value of the perturbed actions. The generative model is also updated to better approximate the action distribution in the batch. During policy execution, at each state s, n actions are generated from $G_\omega(s)$, perturbed, and the action with the highest Q-value is selected[7]. The BCQ algorithm is shown below (Algorithm 4):

Algorithm 4 BCQ

Input: Batch $\mathcal{B}$, horizon T, target network update rate τ, mini-batch size N, max perturbation Φ, number of sampled actions n, minimum weighting λ.
Initialize Q-networks $Q_{\theta_1}, Q_{\theta_2}$, perturbation network ξ_ϕ, and VAE $G_\omega = \{E_{\omega_1}, D_{\omega_2}\}$, with random parameters θ_1, θ_2, ϕ, ω, and target networks $Q_{\theta'_1}, Q_{\theta'_2}$, $\xi_{\phi'}$ with $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$.
for $t = 1$ **to** T **do**
    Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
    $\mu, \sigma = E_{\omega_1}(s, a), \quad \tilde{a} = D_{\omega_2}(s, z), \quad z \sim \mathcal{N}(\mu, \sigma)$ ▷ Sample VAE
    $\omega \leftarrow \arg\min_\omega \sum (a - \tilde{a})^2 + D_{\text{KL}}(\mathcal{N}(\mu, \sigma)||\mathcal{N}(0, 1))$ ▷ Update VAE
    Sample n actions: $\{a_i \sim G_\omega(s')\}_{i=1}^n$
    Perturb each action: $\{a_i = a_i + \xi_\phi(s', a_i, \Phi)\}_{i=1}^n$
    Set value target y (Eqn. 2.1)
    $\theta \leftarrow \arg\min_\theta \sum (y - Q_\theta(s, a))^2$ ▷ Update critic
    $\phi \leftarrow \arg\max_\phi \sum Q_{\theta_1}(s, a + \xi_\phi(s, a, \Phi)), a \sim G_\omega(s)$ ▷ Update perturbation model
    Update target networks: $\theta'_i \leftarrow \tau\theta + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$

2.4.3.5 Hyper-parameters

These are the hyper-parameters we experimented with during evaluation:

- Number of sampled actions n and action flexibility Φ: The selection of n involves a trade-off between imitation learning and reinforcement learning approaches. When $\Phi = 0$ and the number of sampled actions $n = 1$, the policy mimics behavioral cloning. Conversely, as $\Phi \to a_{max} - a_{min}$ and $n \to \infty$, the algorithm converges toward Q-learning.
- Actor Learning Rate.
- Critic Learning Rate.
- Discount Factor Gamma.
- Number of Critics: How many Q-Networks.
- Weight factor λ (Equation 2.1): enables control over how heavily uncertainty at future time steps is penalized.
- Target Network Synchronization Coefficiency: Controls how slowly the target networks are updated relative to the main networks.
- Batch Size.
- KL regularization term for Conditional VAE.

2.4.4 Discrete BCQ

2.4.4.1 Algorithm Overview

The *Discrete BCQ* [9] algorithm is a streamlined adaptation of the original BCQ algorithm, tailored specifically for discrete action spaces. By eliminating the complexities associated with continuous actions, the discrete version retains the core principles of BCQ while simplifying its implementation.

In the original BCQ, a state-conditioned generative model G_ω is trained to approximate the behavioral policy π_b, and actions are sampled from this model before being perturbed within a bounded range. For discrete actions, this process is simplified by directly computing the probabilities of all possible actions using $G_\omega(a|s) \approx \pi_b(a|s)$. A threshold τ is then applied to filter out low-probability actions, ensuring that only actions with sufficient likelihood under the behavioral policy are considered. This threshold is adaptively scaled by the maximum action probability from G_ω, allowing the algorithm to focus on actions that are both high-value and consistent with the data distribution:

$$\pi(s) = \underset{a|G_\omega(a|s)/\max_{\hat{a}} G_\omega(\hat{a}|s)>\tau}{\arg\max} Q_\theta(s,a). \tag{2.2}$$

The policy is derived by selecting the highest-valued action among those that meet the threshold criterion, effectively constraining the arg max operation to in-distribution actions. The Q-network is trained using a modified Bellman update, where the max operation is replaced with actions chosen by this constrained policy:

$$\mathcal{L}(\theta) = l_\kappa\left(r + \gamma \max_{a'|G_\omega(a'|s')/\max_{\hat{a}} G_\omega(\hat{a}|s')>\tau} Q_{\theta'}(s',a') - Q_\theta(s,a)\right). \tag{2.3}$$

To further mitigate overestimation bias, the discrete BCQ employs *Double DQN* [10], leveraging the current Q-network Q_θ for action selection and the target Q-network $Q_{\theta'}$ for value evaluation.

The generative model G_ω, which serves as a behavioral cloning network, is trained using standard supervised learning with a cross-entropy loss. This ensures an accurate approximation of the behavioral policy, enabling the algorithm to effectively constrain action selection to the support of the batch data. The result is a robust and efficient batch reinforcement learning method that outperforms existing algorithms in discrete-action settings. The algorithm is summarized below (Algorithm 5):

Algorithm 5 Discrete BCQ

Require: Batch $\mathcal{B}$, number of iterations T, target update rate τ, mini-batch size N, threshold τ

Initialize Q-network Q_θ, generative model G_ω and target network $Q_{\theta'}$ with $\theta' \leftarrow \theta$

for $t = 1$ **to** T **do**

  Sample mini-batch M of N transitions (s, a, r, s') from $\mathcal{B}$

  $a' = \arg\max_{a'|G_\omega(a'|s')/\max_{\hat{a}} G_\omega(\hat{a}|s') > \tau} Q_\theta(s', a')$

  $\theta \leftarrow \text{argmin}_\theta \sum_{(s,a,r,s')\in M} l_\kappa \left(r + \gamma Q_{\theta'}(s', a') - Q_\theta(s, a)\right)$

  $\omega \leftarrow \text{argmin}_\omega - \sum_{(s,a)\in M} \log G_\omega(a|s)$

  if t mod target_update_rate $= 0$ **then**

    $\theta' \leftarrow \theta$

2.4.4.2 Hyper-parameters

These are the hyper-parameters we experimented with during evaluation:

- Learning Rate.
- Discount Factor Gamma.
- Number of Critics: How many Q-Networks.
- Target Network Update Interval: How often updating the parameters of the Q-Networks.
- Batch Size.
- Action flexibility threshold represented as τ in Equation 2.2
- Regularization term for imitation function.

2.4.5 Conservative Q-Learning

2.4.5.1 Motivation

Standard value-based off-policy reinforcement learning algorithms exhibit significant limitations when applied to offline settings. The fundamental challenges stem from bootstrapping with out-of-distribution actions and overfitting to limited data. During policy evaluation, the algorithms propagate increasingly erroneous estimates when encountering actions not represented in the dataset. Furthermore, without environmental interaction, they tend to develop pathologically optimistic value estimates, particularly for states and actions near the boundaries of the batch data distribution. This systematic overestimation in value functions critically undermines policy improvement, as the agent becomes drawn to unreliable, overvalued actions. These inherent limitations of offline learning create a pressing need for more robust value estimation approaches that can maintain accuracy without online interaction [11].

2.4.5.2 Main Idea

The algorithm tackles value overestimation in offline RL by learning a conservative Q-function that produces lower-bound estimates. Instead of strictly constraining all state-action values, the method specifically targets the policy's expected value, maintaining useful estimates while preventing overoptimism. The approach modifies standard Q-learning through a dual optimization process: first minimizing Q-values for selected state-action pairs, then maximizing values for dataset examples. This balanced formulation preserves realistic value estimates while enabling effective learning from the available offline data [11].

2.4.5.3 Algorithm Overview

The core innovation of CQL lies in its modified Q-function update, which incorporates a regularization term to prevent overestimation of Q-values due to out-of-distribution actions [11]. The Q-function update is given by the Equation 2.4:

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \alpha \cdot \left(\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}\left[Q(\mathbf{s},\mathbf{a})\right] - \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\hat{\pi}_\beta(\mathbf{a}|\mathbf{s})}\left[Q(\mathbf{s},\mathbf{a})\right]\right) + \frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[\left(Q(\mathbf{s},\mathbf{a}) - \hat{B}^\pi\hat{Q}^k(\mathbf{s},\mathbf{a})\right)^2\right]. \tag{2.4}$$

The first term, $\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]$, minimizes Q-values under a chosen distribution $\mu(\mathbf{a}|\mathbf{s})$. This ensures that the Q-function does not overestimate values for actions not well-covered in the dataset [11]. The second term, $\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\hat{\pi}_\beta(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]$, maximizes Q-values under the empirical behavior policy $\hat{\pi}_\beta$. This prevents excessive underestimation for actions that are in the dataset [11]. CQL can be implemented with minimal modifications to existing deep RL algorithms, typically requiring fewer than 20 additional lines of code [11]. For example, in actor-critic methods like *SAC* [12], the Q-loss is augmented with the CQL regularization term, while the policy update remains unchanged. It is also important to note that the algorithm works with discrete action spaces if built on top of *QR-DQN* [13] instead of SAC.

2.4.5.4 Hyper-parameters

These are the hyper-parameters we experimented with during evaluation:

- Conservative Weight α: Scales the strength of the CQL regularization term in the Q-function loss (Equation 2.4).
- Actor Learning Rate.
- Critic Learning Rate.
- Discount Factor Gamma.
- Number of Critics: How many Q-Networks.

- Target Network Synchronization Coefficiency: Controls how slowly the target networks are updated relative to the main networks.
- Batch Size.
- Number of sampled actions.

2.4.6 Implicit Q-Learning

2.4.6.1 Motivation

Offline reinforcement learning faces a fundamental challenge: improving over the behavior policy in a dataset while avoiding errors caused by evaluating unseen actions, which can lead to overestimation or instability due to distributional shift [14]. Existing methods either constrain the policy to stay close to the data like BCQ [7], which limits improvement, or regularize value functions like CQL [11], resulting in additional complexity. *Implicit Q-Learning* (IQL) [14] aims to bypass this trade-off entirely by never querying out-of-sample actions during training, enabling safer and more scalable offline RL without sacrificing performance [14].

2.4.6.2 Main Idea

Prior offline reinforcement learning methods rely on in-distribution constraints to mitigate the risks of value function extrapolation, but these constraints alone often fail to fully prevent estimation errors when evaluating unseen actions. This work challenges the necessity of such constraints by introducing a method that learns an optimal policy entirely from in-sample data, eliminating the need to query the value of any out-of-distribution actions during training. The core innovation lies in approximating the optimal in-support value function by estimating an upper expectile of the action-value distribution for each state, a technique enabled by *expectile regression*. This allows the algorithm to implicitly prioritize high-value actions within the dataset while avoiding explicit maximization over unseen actions. The approach alternates between fitting this value function and performing Bellman updates on the Q-function, using only observed actions in the targets. Finally, the policy is extracted via *advantage-weighted regression* (AWR) [15][16][17][18], which naturally constrains the learned policy to the dataset support without explicit regularization. Unlike prior methods, this framework achieves multi-step dynamic programming, critical for tasks requiring trajectory stitching, while maintaining the stability of purely in-sample learning [14].

2.4.6.3 Expectile Regression

Expectile regression [19] is a statistical method that generalizes ordinary least squares by asymmetrically weighting positive and negative residuals. For a given asymmetry parameter $\tau \in$

$(0, 1)$, the τ-expectile m_τ of a random variable X minimizes the asymmetric squared loss:

$$m_\tau = \arg\min_m \mathbb{E}\left[L_2^\tau(X - m)\right] \tag{2.5}$$

where the loss function L_2^τ is given by:

$$L_2^\tau(u) = |\tau - \mathbb{I}(u < 0)|u^2 \tag{2.6}$$

When $\tau = 0.5$, this reduces to standard mean regression, while $\tau > 0.5$ places more weight on over-prediction errors, yielding higher estimates that approximate conditional upper quantiles. Expectile regression was introduced as a least-squares analogue to *quantile regression* [20], offering computational advantages while preserving similar tail-sensitive properties.

2.4.6.4 Algorithm Overview

The method employs expectile regression to approximate the maximum Q-value over in-distribution actions, enabling multi-step dynamic programming. The algorithm proceeds in two phases:

1. **Value Function Learning:** IQL first learns a state value function $V_\psi(s)$ using an asymmetric L_2 loss:

   $$L_V(\psi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[L_2^\tau\left(Q_{\hat{\theta}}(s, a) - V_\psi(s)\right)\right], \tag{2.7}$$

   where $L_2^\tau(u) = |\tau - \mathbb{I}(u < 0)|u^2$ is the expectile loss and $\tau \in (0.5, 1)$ controls optimism. The Q-function is then updated via TD learning:

   $$L_Q(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\left[\left(r(s, a) + \gamma V_\psi(s') - Q_\theta(s, a)\right)^2\right]. \tag{2.8}$$

2. **Policy Extraction:** The policy $\pi_\phi(a|s)$ is obtained through advantage-weighted regression (AWR):

   $$L_\pi(\phi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\exp\left(\beta\left(Q_{\hat{\theta}}(s, a) - V_\psi(s)\right)\right)\log \pi_\phi(a|s)\right], \tag{2.9}$$

   where β is an inverse temperature parameter that trades off between policy improvement and behavioral cloning.

This approach avoids evaluating unseen actions while still performing dynamic programming, achieving state-of-the-art results on offline RL benchmarks [21]. The algorithm is shown below (Algorithm 6):

Algorithm 6 IQL

Initialize parameters $\psi, \theta, \hat{\theta}, \phi$.
TD learning (IQL):
for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \nabla_\psi L_V(\psi)$
    $\theta \leftarrow \theta - \lambda_Q \nabla_\theta L_Q(\theta)$
    $\hat{\theta} \leftarrow (1-\alpha)\hat{\theta} + \alpha\theta$
Policy extraction (AWR):
for each gradient step **do**
    $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi L_\pi(\phi)$

2.4.6.5 Hyper-parameters

These are the hyper-parameters we experimented with during evaluation:

- Expectile τ: Expectile value for value function training (Equation 2.7).
- Actor Learning Rate.
- Critic Learning Rate.
- Discount Factor Gamma.
- Number of Critics: How many Q-Networks.
- Target Network Synchronization Coefficiency: Controls how slowly the target networks are updated relative to the main networks.
- Batch Size.

2.4.7 Bootstrapping Error Accumulation Reduction

2.4.7.1 Motivation

The motivation behind the *BEAR* (Bootstrapping Error Accumulation Reduction) [22] algorithm stems from the challenges faced in off-policy reinforcement learning, particularly when learning from static datasets. In many real-world applications, such as autonomous driving or robotics, collecting new on-policy data is expensive or impractical, making it crucial to leverage existing off-policy data effectively. However, traditional off-policy methods often struggle with instability and poor performance due to bootstrapping errors caused by evaluating actions outside the training data distribution. These errors accumulate during training, leading to divergent or suboptimal policies. BEAR aims to address this limitation by developing a more robust approach

to off-policy learning that can handle diverse and imperfect datasets, such as those generated by random, suboptimal, or expert policies.

2.4.7.2 Main Idea

The main idea of BEAR revolves around mitigating bootstrapping errors by constraining the policy to actions within the support of the training data distribution. Instead of strictly matching the behavior policy's density, BEAR focuses on ensuring that the learned policy only selects actions that are well-represented in the dataset. This approach strikes a balance between avoiding out-of-distribution actions, which introduce errors, and maintaining the flexibility to improve upon the behavior policy. By doing so, BEAR achieves more stable and reliable performance across a wide range of datasets, whether they contain random, mediocre, or expert demonstrations, without requiring additional interaction with the environment.

2.4.7.3 Algorithm Overview

The authors propose a practical actor-critic algorithm that employs distribution-constrained backups to mitigate the accumulation of bootstrapping errors. The core idea is to restrict policy search to those policies that share the same support as the training distribution, thereby preventing inadvertent error propagation.

The algorithm consists of two key components:

1. **Multiple Q-functions with a conservative update rule**: Similar to BCQ [7], the method utilizes K Q-functions and selects the minimum Q-value for policy improvement.
2. **A support constraint**: This ensures that the policy search is confined to policies that align with the support of the behavior policy.

Both components modify the policy improvement step in standard actor-critic algorithms. Additionally, the authors note that policy improvement can also be performed using the mean of the K Q-functions, which in their experiments yields comparable performance.

Let $\{Q_1, \ldots, Q_K\}$ denote the set of Q-functions. The policy is then updated to maximize a conservative Q-value estimate within the constrained policy set:

$$\pi_\phi(s) := \max_{\pi \in \Pi_\epsilon} \mathbb{E}_{a \sim \pi(\cdot|s)} \left[\min_{j=1,\ldots,K} \hat{Q}_j(s,a) \right] \tag{2.10}$$

In practice, the behavior policy is typically unknown, necessitating an approximate method to constrain the learned policy to its support. To address this, the authors introduce a differentiable constraint that approximately enforces this restriction and solve the resulting constrained optimization problem using dual gradient descent.

The method employs *Maximum Mean Discrepancy* (MMD) [23] as a measure between the unknown behavior policy π_β and the actor policy π_θ. MMD is chosen because it can be estimated using only samples from the distributions, without requiring explicit density estimates. Given samples $\{x_1, \ldots, x_n\} \sim P$ and $\{y_1, \ldots, y_m\} \sim Q$, the empirical MMD between P and Q is computed as:

$$\text{MMD}^2(\{x_1, \cdots, x_n\}, \{y_1, \cdots, y_m\}) = \frac{1}{n^2}\sum_{i,i'} k(x_i, x_{i'}) - \frac{2}{nm}\sum_{i,j} k(x_i, y_j) + \frac{1}{m^2}\sum_{j,j'} k(y_j, y_{j'}). \tag{2.11}$$

where $k(\cdot, \cdot)$ denotes any universal kernel. Experiments show that both Laplacian and Gaussian kernels perform well. Notably, MMD does not depend on the explicit densities of the distributions and can be optimized directly through samples. Putting everything together, the optimization problem in the policy improvement step is:

$$\pi_\phi := \max_{\pi \in \Delta_{|S|}} \mathbb{E}_{s\sim\mathcal{D}} \mathbb{E}_{a\sim\pi(\cdot|s)} \left[\min_{j=1,\ldots,K} \hat{Q}_j(s, a) \right] \quad \text{s.t.} \quad \mathbb{E}_{s\sim\mathcal{D}} \left[\text{MMD}(\mathcal{D}(s), \pi(\cdot|s)) \right] \leq \varepsilon \tag{2.12}$$

where ε is an approximately chosen threshold. The algorithm is summarized below (Algorithm 7):

Algorithm 7 BEAR

Require: Dataset $\mathcal{D}$, target network update rate τ, mini-batch size N, sampled actions for MMD n, minimum λ

1: Initialize Q-ensemble $\{Q_{\theta_i}\}_{i=1}^K$, actor π_ϕ, Lagrange multiplier α, target networks $\{Q_{\theta'_i}\}_{i=1}^K$, and a target actor $\pi_{\phi'}$, with $\phi' \leftarrow \phi$, $\theta'_i \leftarrow \theta_i$
2: **for** $t \in \{1, \ldots, N\}$ **do**
3: Sample mini-batch of transitions $(s, a, r, s') \sim \mathcal{D}$
4: **Q-update:**
5: Sample p action samples, $\{a_i \sim \pi_{\phi'}(\cdot|s')\}_{i=1}^p$
6: Define $y(s, a) := \max_{a_i}[\lambda \min_{j=1,\ldots,K} Q_{\theta'_j}(s', a_i) + (1-\lambda)\max_{j=1,\ldots,K} Q_{\theta'_j}(s', a_i)]$
7: $\forall i, \theta_i \leftarrow \arg\min_{\theta_i}(Q_{\theta_i}(s, a) - (r + \gamma y(s, a)))^2$
8: **Policy-update:**
9: Sample actions $\{\hat{a}_i \sim \pi_\phi(\cdot|s)\}_{i=1}^m$ and $\{a_j \sim \mathcal{D}(s)\}_{j=1}^n$, n preferably an intermediate integer (1-10)
10: Update ϕ, α by minimizing Equation 2.12 by using dual gradient descent with Lagrange multiplier α
11: **Update Target Networks:** $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$; $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$

2.4.7.4 Hyper-Parameters

These are the hyper-parameters we experimented with during evaluation:

- Number of MMD Action Samples.
- MMD Kernel: Gaussian or Laplacian.
- Gaussian kernel sigma.
- Number of steps to warmup the policy function.
- Actor Learning Rate.
- Critic Learning Rate.
- Discount Factor Gamma.
- Number of Critics: How many Q-Networks.
- Target Network Synchronization Coefficiency: Controls how slowly the target networks are updated relative to the main networks.
- Batch Size.

2.4.8 Temporal Difference Learning 3 with Behavioral Cloning

2.4.8.1 Motivation

The motivation behind the *Temporal Difference Learning 3 with Behavioral Cloning* (TD3+BC) [24] algorithm stems from the observation that many existing offline reinforcement learning methods introduce significant complexity, including additional hyper-parameters, secondary components like generative models, and extensive modifications to underlying RL algorithms. These complexities not only increase computational costs [24] but also make the algorithms harder to implement, tune, and reproduce [25][26][27]. The authors sought to address these challenges by exploring whether a minimalist approach could achieve comparable performance to state-of-the-art offline RL algorithms. Their goal was to reduce the barriers to entry for offline RL by developing a method that is simple to implement, computationally efficient, and requires minimal changes to existing online RL frameworks, while still effectively mitigating the extrapolation error problem inherent in offline settings.

2.4.8.2 Main Idea

The main idea of the TD3+BC algorithm is to combine the strengths of reinforcement learning and behavior cloning in a straightforward manner. By adding a behavior cloning term to the policy update step of the TD3 [28] algorithm, the method ensures that the learned policy remains

close to the actions present in the dataset, thereby reducing the risk of extrapolation errors. Additionally, the algorithm normalizes the state features in the dataset to improve stability during training.

2.4.8.3 Behavioral Cloning

An alternative method for training policies involves imitating an expert or predefined behavior. *Behavior Cloning* (BC) [29] is a form of imitation learning that uses supervised learning to train a policy to replicate the actions observed in a given dataset. In contrast to reinforcement learning, the effectiveness of this approach heavily relies on the quality and performance of the data collection process.

2.4.8.4 Algorithm Overview

The authors present their approach to offline reinforcement learning by building upon the TD3 algorithm with two key modifications. The first change introduces a behavior cloning regularization term into TD3's standard policy update step:

$$\pi = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{s\sim\mathcal{D}}\left[Q(s,\pi(s))\right] \rightarrow \pi = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\lambda\, Q(s,\pi(s)) - (\pi(s)-a)^2\right] \quad (2.13)$$

This modification encourages the learned policy to stay close to the actions present in dataset D, addressing the extrapolation error problem common in offline RL. The behavior cloning term acts as a constraint, preventing the policy from deviating too far from the demonstrated actions while still allowing for policy improvement through reinforcement learning. The second modification involves normalizing the state features across the dataset to have zero mean and unit variance:

$$s_i = \frac{s_i - \mu_i}{\sigma_i + \epsilon} \quad (2.14)$$

Where s_i is the i-th feature of the state s, ϵ is a small normalization constant and, μ_i and σ_i are the mean and standard deviation, respectively, of the i-th feature across the dataset. This simple preprocessing step enhances training stability, particularly important in the offline setting where the dataset remains fixed.

2.4.8.5 Hyper-Parameters

These are the hyper-parameters we experimented with during evaluation:

- α: Balances RL (Q-maximization) and BC (imitation) in policy updates.
- Actor Learning Rate.
- Critic Learning Rate.

- Discount Factor Gamma.
- Number of Critics: How many Q-Networks.
- Target Network Synchronization Coefficiency: Controls how slowly the target networks are updated relative to the main networks.
- Batch Size.
- Policy Update Frequency: How often updating the actor.
- Std dev for target noise.
- Clipping range for target noise.

2.4.9 Decision Transformer

2.4.9.1 Motivation

The motivation behind the *Decision Transformer* (DT) [30] stems from the desire to simplify and scale reinforcement learning by re-framing it as a sequence modeling problem. Traditional RL methods often rely on complex techniques like temporal difference learning or policy gradients, which can be unstable and require careful tuning. By leveraging the success of transformer architectures in language [31] and vision tasks [32], the authors propose a paradigm shift: instead of using dynamic programming or value functions, they treat RL as a conditional sequence generation task. This approach avoids challenges like bootstrapping and discounting future rewards, while capitalizing on the transformer's ability to model long-range dependencies and perform implicit credit assignment.

2.4.9.2 Main Idea

The Decision Transformer models RL trajectories as sequences of states, actions, and returns-to-go (cumulative future rewards), using a causally masked transformer to autoregressively predict actions. At test time, the model generates actions conditioned on a desired return, effectively "prompting" the policy to achieve specific performance levels. Unlike traditional RL methods, it does not explicitly learn value functions or policy gradients; instead, it relies on the transformer's ability to capture patterns in historical data and generalize to new scenarios. The simplicity of this framework enables it to match or surpass state-of-the-art offline RL methods while avoiding many of their pitfalls like distributional shift and value overestimation [4].

2.4.9.3 Transformers

The transformer architecture, introduced by Vaswani et al. [33], was designed to efficiently process sequential data. It relies on stacked self-attention layers with residual connections. In

each layer, the model takes *n* input embeddings (each representing a token) and produces *n* output embeddings while preserving dimensionality. For every token, linear transformations generate a key, query, and value vector. The output for a given token is computed by weighting all value vectors based on how closely their corresponding keys match its query—this is done using a softmax-normalized dot product:

$$z_i = \sum_{j=1}^{n} \text{softmax}\left(\{(q_i, k_j)\}_{j'=1}^{n}\right)_j \cdot v_j \tag{2.15}$$

This mechanism allows the model to dynamically assign importance (”credit”) to different parts of the input by measuring similarity between queries and keys. The authors adopt the GPT architecture [34], which adds a causal self-attention mask to the transformer. This modification restricts each token to attending only to previous tokens in the sequence, enabling auto-regressive generation (predicting one token at a time). Further architectural details can be found in the original papers.

2.4.9.4 Trajectory representation

The design of the trajectory representation prioritizes two key goals: it must allow transformers to capture meaningful patterns in the data, and it must support conditional action generation during inference. A major challenge lies in handling rewards—since the model should base its actions on future desired outcomes rather than past rewards, simply using raw rewards is insufficient. To address this, the authors replace rewards with returns-to-go, the sum of future rewards from the current timestep onward:

$$\widehat{R}_t = \sum_{t'=t}^{T} r_{t'} \tag{2.16}$$

This leads to the following trajectory representation:

$$\tau = \left(\widehat{R}_1, s_1, a_1, \widehat{R}_2, s_2, a_2, \ldots, \widehat{R}_T, s_T, a_T\right) \tag{2.17}$$

2.4.9.5 Hyper-Parameters

These are the hyper-parameters we experimented with during evaluation:

- Context Size: The number of past timesteps (states, actions, returns-to-go) fed into the model for auto-regressive prediction.
- Learning Rate.

- Discount Factor Gamma.
- Number of Layers: The depth of the Transformer model.
- Number of Attention Heads.
- Batch Size.
- Warm-up Steps: Gradually increase the learning rate from 0 to its peak value over N steps.

2.4.10 Algorithm Classification

Offline reinforcement learning algorithms can be broadly categorized based on their core methodological approaches. Policy Constraint methods, such as BCQ, BEAR, and TD3+BC, address distributional shift by explicitly or implicitly regularizing the learned policy to remain close to the behavior policy underlying the dataset. Pessimistic Value Learning methods, including CQL and IQL, mitigate overestimation of out-of-distribution (OOD) actions by conservatively modifying the Q-function through penalties or implicit constraints. Finally, Transformer-Based Methods like Decision Transformer, reformulate RL as a sequence modeling task, leveraging autoregressive prediction to bypass traditional value or policy optimization entirely. This work focuses on analyzing these three key paradigms.

2.5 Related Work

While offline reinforcement learning has demonstrated success across various domains, its application to cryptocurrency portfolio optimization remains unexplored in the existing literature. A thorough review reveals no prior work specifically evaluating offline RL algorithms for this use case. However, several relevant studies have investigated related approaches in adjacent domains.

Recent advances have applied deep reinforcement learning techniques to traditional stock portfolio optimization, achieving promising results in simulated market environments. Notably, Yang et al. [35] introduce a novel ensemble strategy that integrates three actor-critic [36] algorithms: Proximal Policy Optimization (PPO) [37][38], Advantage Actor-Critic (A2C) [39], and Deep Deterministic Policy Gradient (DDPG) [40], to dynamically optimize trading strategies in complex and non-stationary markets.

Similarly, a growing body of work has begun exploring RL methods for cryptocurrency trading strategies. Lucarelli at al. [41] used a multi-agent approach with local agents (DQN techniques [10]), one for each asset of the portfolio, and a global agent as a reward function that describes the global portfolio environment.

A significant contribution by Cui et al. [42] introduced a Conditional Value-at-Risk (CVaR)-based deep reinforcement learning framework for portfolio optimization. Their approach leverages Proximal Policy Optimization (PPO) [38] as the core algorithm, incorporating CVaR directly into the reward function to explicitly optimize for tail risk management in volatile market conditions.

3 Methodology and Implementation

3.1 Dataset Characteristics

This study utilizes hourly OHLCV (Open, High, Low, Close, Volume) data for six cryptocurrency trading pairs (AAVE/USDT, BTC/USDT, DOGE/USDT, DOT/USDT, ETH/USDT, and UNI/USDT) collected via the Binance API. OHLCV is a standardized financial data format where each timestamp includes the opening price (Open), highest and lowest prices during the interval (High, Low), closing price (Close), and trading volume (Volume).

The selected cryptocurrencies represent a diverse range of market capitalizations, use cases, and volatility profiles: Bitcoin (BTC) and Ethereum (ETH) as large-cap benchmarks; AAVE, DOT, and UNI as mid-cap DeFi tokens; and DOGE as a high-volatility meme coin. This variety ensures the portfolio optimization framework is tested under heterogeneous market conditions. All pairs are quoted against Tether (USDT), a stablecoin pegged 1:1 to the US dollar, isolating price movements to the crypto assets themselves rather than countercurrency fluctuations.

The datasets span from October 15, 2020 up to September 30, 2024. Though both hourly and daily resolutions were available, hourly data was chosen to maximize training samples, enabling the model to capture finer temporal patterns despite increased computational costs. The resulting high-frequency dataset provides a robust foundation for offline reinforcement learning algorithms to learn nuanced, time-dependent strategies for portfolio allocation.

3.2 Dataset Pre-Processing

The dataset was partitioned into training and test sets with an 85:15 split ratio, a common practice for datasets of this scale. While shuffling is typically recommended in machine learning to mitigate bias and enhance generalization [43], it is critical to preserve the temporal order in time-series data. Shuffling would disrupt the inherent sequential dependencies, thereby invalidating the modeling of temporal dynamics.

For feature scaling, we applied standardization using the StandardScaler from scikit-learn [44]. The scaler was fitted exclusively on the training set features, followed by a transformation of both the training and test sets. Importantly, the test set was only transformed (not refit) to prevent data leakage, ensuring that model evaluation reflects real-world generalization performance.

Standardization was chosen over alternative normalization techniques (e.g., Min-Max scaling) due to the nature of cryptocurrency price data. Unlike bounded datasets, cryptocurrencies lack a fixed maximum value, making Min-Max scaling infeasible. Standardization centers the data to zero mean and unit variance, preserving relative value distributions while maintaining comparability across features.

3.3 MDP Model for Portfolio Optimization

In order to model the dynamic and highly volatile crypto market we use a Markov Decision Process as follows:

- State $s = [o_1, h_1, l_1, c_1, v_1, \ldots, o_n, h_n, l_n, c_n, v_n]$: a vector that includes open prices, high prices, low prices, close prices and volumes for n cryptocurrencies.
- Action a: a vector of n discrete actions, one for each crypto. An action is either *buy*, *sell*, or *hold*. For the *buy* and *sell* options, we use a fixed amount worth of asset (e.g. \$500).
- Reward $r(s, a)$: The reward for taking action a at state s. For our case the reward can be Sharpe Ratio or the returns.

An illustration of the model is shown in Figure 3.1.

3.4 Dataset Augmentation

A significant challenge in our experimental setup was the absence of two critical components required for offline reinforcement learning: actions and rewards at each timestep. In offline RL, the learning process relies on logged experience tuples of the form (s_t, a_t, r_t, s_{t+1}), where s_t is the state, a_t the action, r_t the reward, and s_{t+1} the subsequent state. However, our dataset

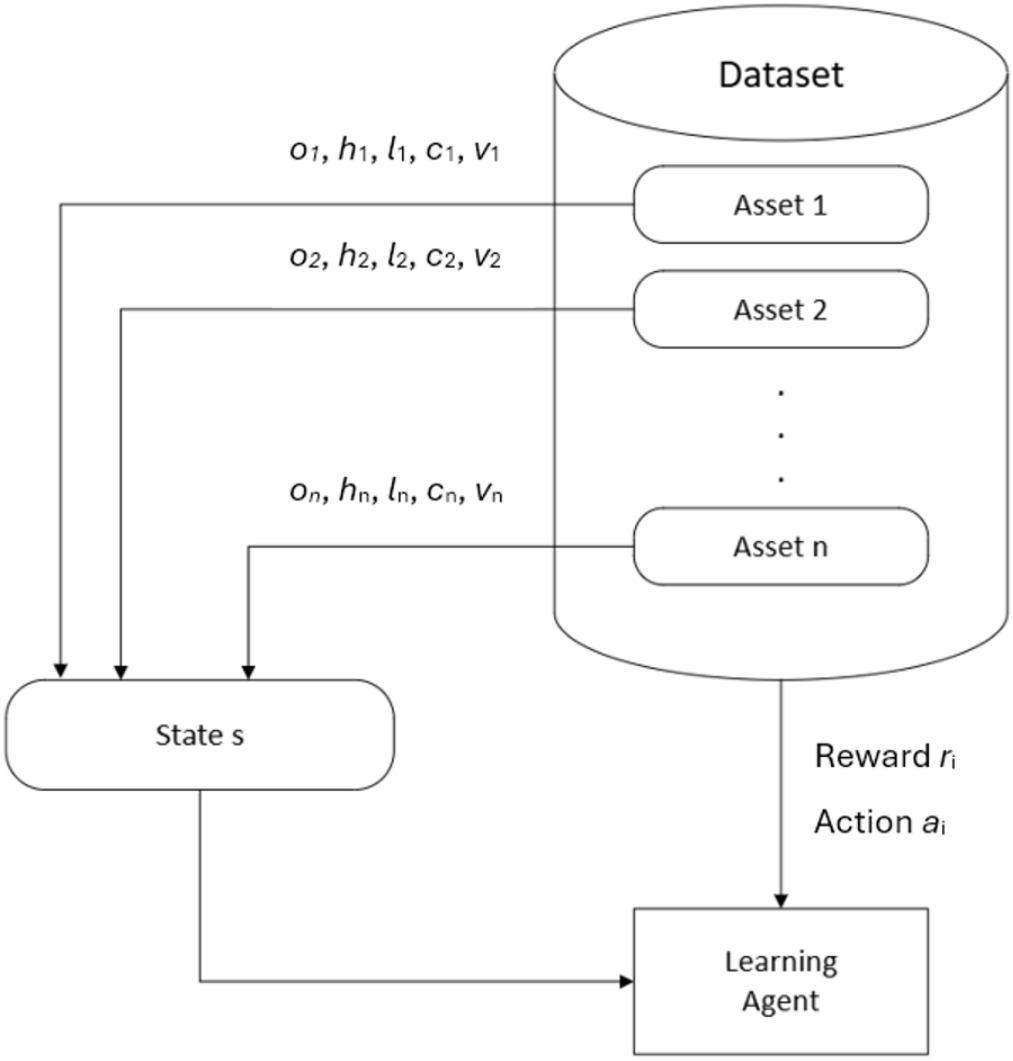


Figure 3.1: Illustration of MDP model for portfolio optimization.

consisted solely of raw time-series observations, generated passively rather than by an explicit *behavior policy* [6].

This limitation is analogous to supervised learning, where models require labeled input-output pairs for training. Without actions and rewards, the dataset lacks the necessary structure to learn a policy or value function directly. To address this, we augmented the data by deriving implicit actions and rewards from the time-series dynamics, enabling the application of offline RL algorithms.

3.4.1 Actions Generation

To generate actions for each timestep, we employed a computationally efficient method, crucial for processing large datasets without introducing significant overhead. First, the percentage change in the closing price between consecutive time steps was calculated as:

$$q_t = \frac{p_t - p_{t-1}}{p_{t-1}}, \tag{3.1}$$

where q_t represents the price change at timestep t and p_t denotes the closing price at t. These price changes were then mapped to discrete actions (buy, sell, or hold) based on a fixed threshold ϵ. Specifically, the action a_t for each timestep was determined as follows:

$$a_t = \begin{cases} \text{buy} & \text{when } q_t > \epsilon \\ \text{sell} & \text{when } q_t < -\epsilon \\ \text{hold} & \text{when } -\epsilon \leq q_t \leq \epsilon \end{cases} \tag{3.2}$$

This approach encodes market reactions to price movements: buy and sell actions are triggered only when the price change exceeds a meaningful threshold, while insignificant fluctuations result in a hold signal. The threshold ϵ must be small to avoid overlooking subtle but statistically relevant trends, yet large enough to filter out negligible noise. For our experiments, we set $\epsilon = 0.01$ (1%), balancing sensitivity to market movements with robustness to micro-volatility. This choice ensures that actions reflect economically meaningful decisions rather than stochastic price variations.

3.4.2 Rewards Generation

The reward signal was derived from a rolling Sharpe ratio of equally-weighted portfolio returns to evaluate investment performance while accounting for risk. First, percentage returns were computed for each asset by taking the first difference of closing prices normalized by the previous price:

$$r_t = \frac{p_t - p_{t-1}}{p_{t-1}}. \tag{3.3}$$

These individual asset returns were combined into a returns matrix, representing the multivariate time series of all assets. A simple equal-weight portfolio strategy was implemented by assigning uniform weights to all assets, with portfolio returns calculated as the dot product of the returns matrix and the weight vector. The rewards were then generated as rolling Sharpe ratios: for each timestep t after an initial window period, the reward was computed as the Sharpe ratio of the portfolio's most recent rolling window returns. This approach provides a risk-adjusted

performance metric that penalizes volatile returns, encouraging the agent to learn strategies that achieve consistent gains rather than erratic performance. The rolling window mechanism ensures local sensitivity to recent market conditions while maintaining computational efficiency through vectorized operations. During our experiments, we used a rolling window of 168 time steps, corresponding to one week of hourly data. This choice balances provides statistical robustness, since a week-long window provides sufficient data points to compute a stable Sharpe ratio, mitigating the impact of short-term noise.

To evaluate the sensitivity of our reinforcement learning agent to different reward formulations, we implemented a simpler secondary reward function based solely on raw portfolio returns without risk adjustment. This approach follows the same initial steps as our primary method: percentage returns are computed for each asset, which are then combined into a multivariate returns matrix. An equal-weight portfolio strategy is again enforced, with portfolio-wide returns calculated via the dot product of the returns matrix and weight vector. The key distinction lies in the reward definition: instead of using a rolling Sharpe ratio, this variant directly uses the instantaneous portfolio return as the reward signal at each timestep. While computationally more efficient, this simplification trades off risk-awareness for simplicity. Its inclusion serves as an ablation study to isolate the impact of risk-adjusted rewards, particularly relevant given the high volatility inherent to cryptocurrency markets.

3.5 Python Implementation

For our experiments, we utilized *d3rlpy* [45], a Python framework specifically designed for data-driven deep reinforcement learning. This library provides an efficient implementation of the algorithms we aimed to evaluate, along with useful utilities for handling reinforcement learning datasets.

3.5.1 Dataset Handling with MDPDataset

In supervised learning, training typically involves iterating over input data X and corresponding labels Y. However, reinforcement learning requires mini-batches consisting of state-action-reward-next-state tuples (s_t, a_t, r_t, s_{t+1}) along with episode termination flags. Manually converting raw observations, actions, rewards, and terminal flags into these tuples can be tedious and error-prone. To streamline this process, d3rlpy provides the MDPDataset class (Listing 3.1), which automates dataset construction and management, allowing us to focus on algorithm evaluation rather than data pre-processing.

```
train_dataset = d3rlpy.dataset.MDPDataset(
       observations=train_observations,
```

```
    actions=actions,
    rewards=rewards,
    terminals=terminals,
    action_space=d3rlpy.constants.ActionSpace.DISCRETE if action_type == 0
        else d3rlpy.constants.ActionSpace.CONTINUOUS
)
```

Listing 3.1: Dataset construction using MDPDataset class.

3.5.2 Action Encoding for Discrete and Continuous Algorithms

We encoded actions as integers:

- 0 for hold
- 1 for buy
- 2 for sell

Since our portfolio consists of n assets, each action is initially represented as a vector of length n, with one entry per asset.

- Discrete Algorithms (Discrete BCQ, Discrete CQL): To handle discrete action spaces, we converted the action vector into a single discrete value using a base-3 encoding. For example, the action vector $a = (1, 2, 1, 0, 1, 2)$ translates to:

$$a = 1 \times 3^5 + 2 \times 3^4 + 1 \times 3^3 + 0 \times 3^2 + 1 \times 3^1 + 2 \times 3^0 = 437 \tag{3.4}$$

  With 6 assets, this results in $3^6 = 729$ possible discrete actions. We chose base-3 encoding because it is mathematically correct since collisions are not possible and computationally optimal since it requires the least amount of memory to store the value compared to other techniques (for example concatenating the values as a string).

  The following two code snippets show the encoding and decoding in python:

```
combined_action = sum(action * (3 ** i) for i, action in
    enumerate(reversed(action_set)))
```

Listing 3.2: Encoding from base-3 to base-10.

```
for idx, asset in enumerate(assets):
    actions[asset] = (action // (3 ** (len(assets) - 1 - idx))) % 3
```

Listing 3.3: Decoding from base-10 to base-3.

- Continuous Algorithms (BCQ, CQL, BEAR, IQL, TD3+BC, DT): Since continuous algorithms cannot process discrete actions directly but can keep the vector format, we introduced a pseudo-continuous action space by mapping discrete actions to random floats within defined ranges:
  - Hold: Random value in $[-0.34, 0.34]$
  - Buy: Random value in $[0.34, 1]$
  - Sell: Random value in $[-1, -0.34]$

  The action space was normalized to the range $[-1, 1]$, a standard convention in continuous reinforcement learning [12][28].

3.5.3 Training Process

The training protocol consisted of 20 epochs per algorithm, with each epoch comprising 34694 time-steps, resulting in a total of $20 \times 34694 = 693880$ time-steps. This training duration follows standard reinforcement learning requirements for datasets of comparable size, ensuring sufficient learning while mitigating overfitting risks.

After loading the dataset into MDPDataset, we create an algorithm object (Listing 3.4) and start training using the fit() method provided by the d3rlpy API (Listing 3.5). More details about the hyper-parameter selection can be found in Section 4.1.4.

```
algo = d3rlpy.algos.BCQConfig(
           batch_size=512,
           gamma=0.98,
           actor_learning_rate=0.0001,
           critic_learning_rate=0.0001,
           n_critics=2,
           tau=0.003,
           beta=0.1,
           action_flexibility=0.2,
           n_action_samples=40,
           lam=0.7,
           update_actor_interval=1,
       ).create(device=torch.cuda.is_available())
```

Listing 3.4: An example of creating a BCQ algorithm object with specific hyper-parameters. Also if available, the algorithm will run on CUDA.

```
algo.fit(dataset=train_dataset,
```

```
        n_steps=n_train_timesteps*args.epochs,
        n_steps_per_epoch=n_train_timesteps)
```

Listing 3.5: Initiating the training of the algorithm, while specifying the dataset and the number of timesteps and epochs.

4 Algorithm Evaluation and Analysis

4.1 Evaluation Framework

Unlike supervised learning where evaluation simply compares predictions to ground truth labels, reinforcement learning requires a dynamic testing environment to properly assess an agent's sequential decision-making capabilities. Our evaluation framework simulates real-world trading conditions by processing the agent's actions (buy/sell/hold) through a virtual trading system that tracks portfolio value, budget constraints, and asset allocations over time. The environment maintains three parallel portfolios: one controlled by our trained RL agent, a benchmark portfolio using a simple HODL strategy (a simple strategy that allocates assets in the beginning and then holds), and a random-action portfolio for baseline comparison. At each timestep, the system executes trades based on the agent's decisions while respecting real-world constraints like available budget and asset units. Key performance metrics including portfolio value trajectory and risk-adjusted returns are computed and compared against benchmarks. This approach provides a comprehensive assessment of the agent's ability to maximize returns while managing risk, crucial for financial applications where the consequences of actions unfold over time and depend on market dynamics. The testing environment also logs all transactions and portfolio states, enabling detailed analysis of the agent's strategy.

4.1.1 Metrics

The metrics used for evaluating the performance of the agent are the following:

- **Sharpe Ratio:** The Sharpe Ratio measures risk-adjusted returns by comparing excess returns above the risk-free rate to their volatility. It is calculated as:

$$\text{Sharpe Ratio} = \frac{\mathbb{E}[R_p - R_f]}{\sigma_p}, \tag{4.1}$$

  where R_p are the portfolio returns, R_f is the risk free rate, and σ_p is the standard deviation of the returns. The risk-free rate R_f represents the return of a theoretically "safe" investment (like US Treasury bills). By subtracting this from portfolio returns, we isolate the extra compensation earned for taking risk. A ratio above 1 indicates the strategy generates excess returns that adequately compensate for its volatility, while a ratio below 1 suggests the risk may not be justified.

- **Average of Daily Returns:** The arithmetic mean of daily portfolio value changes. This baseline metric measures the strategy's raw earning potential but should be interpreted alongside risk metrics.

- **Average of Monthly Returns:** The mean monthly portfolio growth, providing insight into medium-term performance while smoothing out daily market noise. Particularly relevant for assessing the strategy's robustness across market cycles.

- **Standard Deviation of Daily Returns:** The volatility of daily returns, quantifying the strategy's risk profile. High values indicate greater unpredictability in short-term performance.

- **Standard Deviation of Monthly Returns:** The dispersion of monthly returns, revealing the strategy's consistency over longer periods. Lower values suggest more reliable performance regardless of market conditions.

- **Cumulative Return:** Cumulative return is the total change in the value of an investment over a specific period of time, expressed as a percentage of the original investment. It measures the overall gain or loss, including all income and capital gains. It is calculated as:

$$R = \frac{V_f - V_i}{V_i} \times 100, \tag{4.2}$$

  where R is the cumulative return percentage, and V_i and V_f represent the initial and final portfolio values, respectively.

This evaluation approach allows us to assess both the absolute performance (through return averages) and the quality of returns (via volatility measures and risk-adjusted metrics). The combination of daily and monthly perspectives provides complementary views of the strategy's behavior across different time horizons, while the Sharpe ratio serves as our primary metric for comparing risk-adjusted performance against alternative strategies.

In evaluating our reinforcement learning agent, we deliberately focused on financial performance metrics rather than algorithmic training statistics such as actor or critic loss values. This methodological choice reflects our primary objective of assessing real-world trading performance rather than model convergence characteristics. While training losses provide insight into the learning process, they serve as indirect proxies that may not correlate with actual trading effectiveness.

4.1.2 Experimental Setup

The evaluation framework was designed to provide sufficient capital for meaningful trading activity while maintaining realistic constraints. All three portfolios (RL agent, HODL, and random strategy) were initialized with $10,000 in total asset value, equally distributed across the six assets in the portfolio. To ensure adequate liquidity for trading decisions, the agent was allocated an operating budget of $1,000,000. Each trading action (buy or sell) involved $500 worth of the specified asset, establishing a consistent trade size that was small relative to the total budget (0.05% of available funds per transaction). This configuration ensured that the agent's trading capacity would not be artificially constrained by capital limitations during the evaluation period, while still maintaining reasonable position sizes that reflect practical trading scenarios. The $500 trade size represents a balanced value, large enough to impact portfolio performance while small enough to allow for gradual position adjustments.

4.1.3 Hardware Specifications

The experiments were conducted on the UCY High Performance Computing (HPC) cluster on COMPUTE partition. Each node has the following CPU model: Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz [46].

Regarding the memory, we ran each algorithm with 600 MB of allocated memory.

4.1.4 Experiment Methodology

We began by initializing all algorithms with the default parameter values recommended by both the original authors and the d3rlpy framework [45]. Through an iterative refinement process, we adjusted core hyper-parameters, such as the learning rate and batch size, until achieving stable agent performance, halting further adjustments once no additional improvements were observed.

With these optimized base hyper-parameters fixed, we conducted eight experimental trials per algorithm. Each trial tested four distinct hyperparameter configurations (specific to each algorithm's unique design) under two reward functions: Sharpe Ratio and Returns (using identical

configurations for both). This resulted in a total of 64 experiments. Ideally, we would test every possible hyperparameter configuration, both common and unique, but the resulting combinatorial explosion makes this approach impractical given the time constraints of this thesis. Thus, even slight parameter changes can lead to different results upon reproduction.

Our analysis proceeded in four stages:

- **Hyperparameter Sensitivity:** Evaluating how each configuration influenced agent performance and identifying the optimal setup for each algorithm.
- **Benchmark Comparison:** Contrasting the best-performing configuration against two baseline strategies, HODL and Random, for every algorithm.
- **Cross-Algorithm Evaluation:** Comparing the top-performing configurations across all algorithms.
- **Reward Function Analysis:** Assessing the impact of the two reward functions (Sharpe Ratio and Returns) on overall results.

4.2 BCQ Results

The following hyper-parameters remained constant:

- Actor learning rate: 0.0001
- Critic learning rate: 0.0001
- Number of critics: 2
- Discount factor γ: 0.98
- Batch size: 512
- Target network synchronization coefficiency: 0.003
- Beta: 0.1

The experiments were performed with the following configurations:

- **Configuration 1:** Sampled Actions: 40, Action Flexibility: 0.2, Weight Factor λ: 0.7
- **Configuration 2:** Sampled Actions: 40, Action Flexibility: 0.5, Weight Factor λ: 0.7
- **Configuration 3:** Sampled Actions: 60, Action Flexibility: 0.2, Weight Factor λ: 0.7
- **Configuration 4:** Sampled Actions: 40, Action Flexibility: 0.2, Weight Factor λ: 0.4

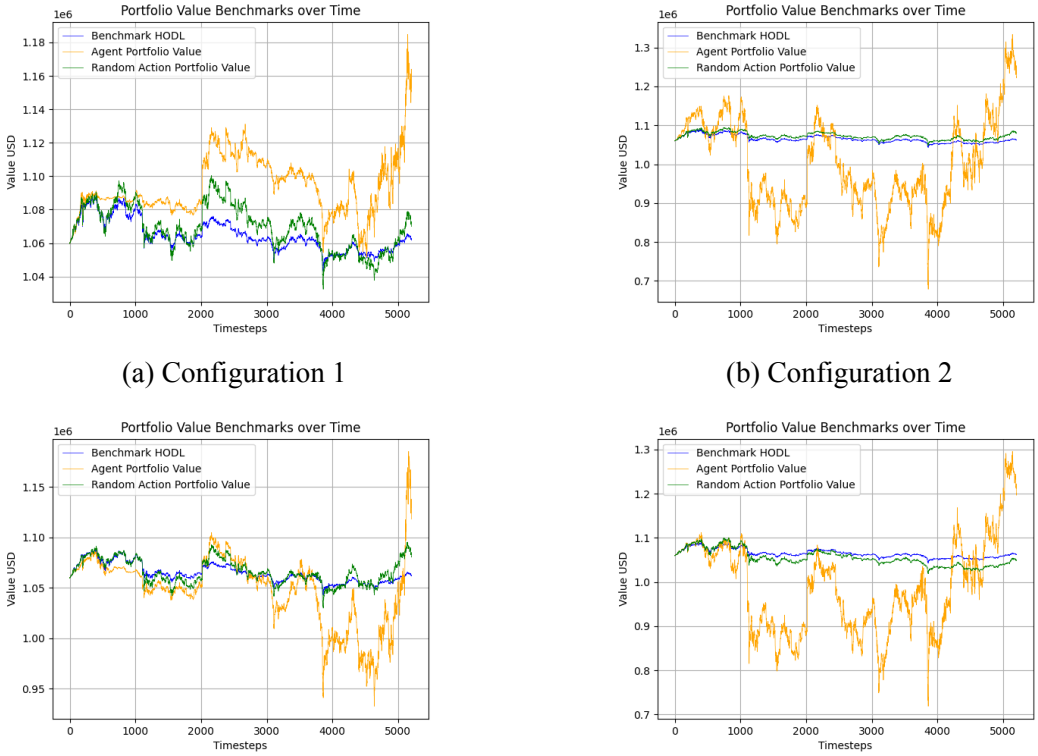
4.2.1 Sharpe Ratio Reward Function Results

Metrics	Config 1	Config 2	Config 3	Config 4
Sharpe Ratio	0.0095	0.0069	0.0035	0.0063
Avg Daily Return	0.035	0.163	0.030	0.139
Avg Monthly Return	0.851	4.873	-0.053	4.126
Std Dev Daily Return	0.608	3.688	1.012	3.131
Std Dev Monthly Return	1.951	17.498	2.700	14.080
Cumulative Return	8.733	15.656	5.490	13.306

Table 4.1: BCQ agent performance comparison across different configurations on Sharpe Ratio reward function.

Metrics	Config 1	HODL	Random
Sharpe Ratio	0.0095	-0.0089	-0.0032
Avg Daily Return	0.035	0.003	0.006
Avg Monthly Return	0.851	0.054	0.149
Std Dev Daily Return	0.608	0.205	0.368
Std Dev Monthly Return	1.951	1.185	1.991
Cumulative Return	8.733	0.171	0.953

Table 4.2: Best performing BCQ configuration compared to HODL and Random strategies on Sharpe Ratio reward function.



(a) Configuration 1

(b) Configuration 2

(c) Configuration 3

(d) Configuration 4

Figure 4.1: BCQ agent performance comparison across different configurations and strategies on Sharpe Ratio reward function.

4.2.2 Analysis

Based on the results on Tables 4.1, 4.2 and the plots in Figure 4.1:

The results indicate notable trade-offs between risk and return across the four configurations. Configuration 1 achieves the highest Sharpe Ratio, suggesting better risk-adjusted returns compared to the others, despite its lower average and cumulative returns. This is likely due to its significantly lower volatility compared to Configuration 2, which, while yielding higher average and cumulative returns, suffers from higher risk, reflected in its lower Sharpe Ratio. Configuration 3 performs the worst in terms of Sharpe Ratio and average monthly return, possibly due to over-sampling actions without sufficient flexibility, leading to higher volatility. Configuration 4 shows a middle-ground performance, with improved cumulative returns over Configuration 1 but lower risk-adjusted returns, likely due to the reduced weight factor weakening the impact of the Q-network constraints. Overall, Configuration 1 appears most balanced for risk-averse strategies, while Configuration 2 may suit risk-tolerant investors seeking higher absolute returns despite higher volatility.

The trained agent in Configuration 1 significantly outperforms both the HODL and random

strategies across all metrics, demonstrating superior risk-adjusted returns and cumulative performance. With a positive Sharpe Ratio of 0.0095, the agent achieves better risk-adjusted returns compared to the negative ratios of HODL and random, indicating more consistent profitability relative to volatility. The agent also exhibits substantially higher average daily and monthly returns compared to HODL and random, along with a much larger cumulative return. However, the agent's higher standard deviations in daily and monthly returns suggest greater volatility than HODL, though comparable to the random strategy. Overall, the agent's ability to generate higher returns with a favorable Sharpe Ratio makes it a more effective strategy than passive holding or random decision-making, despite its increased risk exposure.

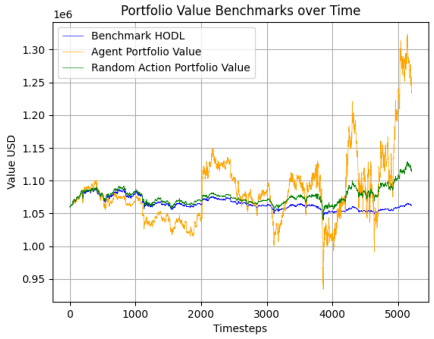
4.2.3 Returns Reward Function Results

Metrics	**Config 1**	**Config 2**	**Config 3**	**Config 4**
Sharpe Ratio	0.0083	0.0060	-0.0101	-0.0044
Avg Daily Return	0.101	0.118	-0.112	-0.064
Avg Monthly Return	3.299	4.032	-4.245	-2.501
Std Dev Daily Return	1.769	3.190	2.525	2.704
Std Dev Monthly Return	6.134	15.549	10.339	11.760
Cumulative Return	16.574	12.181	-30.253	-19.824

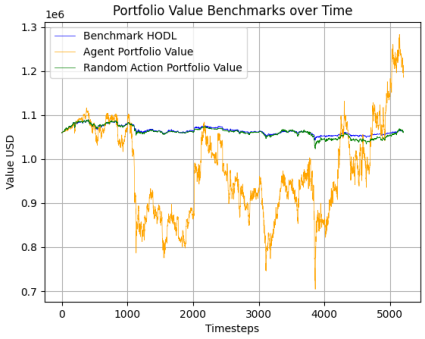
Table 4.3: BCQ agent performance comparison across different configurations on Returns reward function.

Metrics	**Config 1**	**HODL**	**Random**
Sharpe Ratio	0.0083	-0.0089	0.0062
Avg Daily Return	0.101	0.003	0.026
Avg Monthly Return	3.299	0.054	0.784
Std Dev Daily Return	1.769	0.205	0.337
Std Dev Monthly Return	6.134	1.185	1.533
Cumulative Return	16.574	0.171	5.085

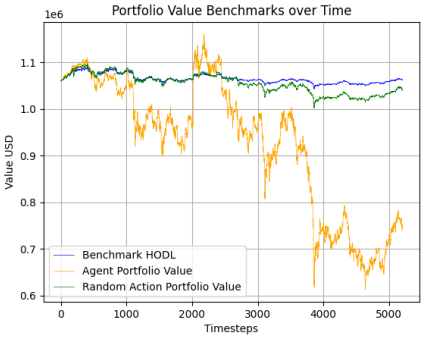
Table 4.4: Best performing BCQ configuration compared to HODL and Random strategies on Returns reward function.



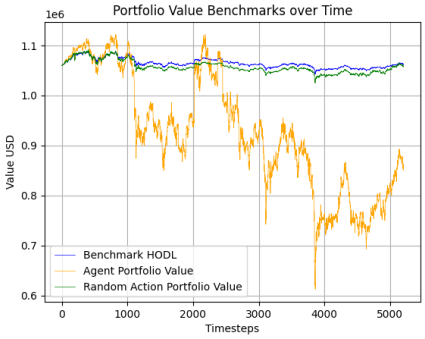
(a) Configuration 1



(b) Configuration 2



(c) Configuration 3



(d) Configuration 4

Figure 4.2: BCQ agent performance comparison across different configurations and strategies on Returns reward function.

4.2.4 Analysis

Based on the results on Tables 4.3, 4.4 and the plots in Figure 4.2:

The results demonstrate significant performance variations across the four BCQ configurations. Configuration 1 achieves the highest Sharpe Ratio and cumulative return, suggesting a balanced trade-off between exploration and exploitation with moderate action sampling and conservative flexibility. Configuration 2 yields higher average daily and monthly returns but suffers from elevated volatility, likely due to its higher action flexibility destabilizing returns. Configuration 3 performs poorly, with negative Sharpe Ratio and cumulative return, indicating that increasing sampled actions without adjusting flexibility or λ harms stability. Configuration 4 shows marginally better risk-adjusted metrics than Configuration 3 but still under-performs Configuration 1, highlighting the importance of a higher λ in mitigating overestimation bias. Overall, Configuration 1 strikes the best balance, while higher flexibility or reduced λ introduces instability, and excessive action sampling degrades performance sharply.

The agent in Configuration 1 significantly outperforms both the HODL and Random strategies across all metrics, demonstrating superior risk-adjusted returns and cumulative performance.

With a Sharpe Ratio of 0.0083, it surpasses both HODL and Random, indicating better return per unit of risk. The agent also achieves substantially higher average daily and monthly returns compared to HODL and Random. However, the agent exhibits higher volatility, as seen in its larger standard deviations for daily and monthly returns, suggesting more aggressive trading behavior. Despite this, its cumulative return vastly exceeds both benchmarks, highlighting its ability to generate higher profits over time.

4.3 Discrete BCQ Results

The following hyper-parameters remained constant:

- Learning rate: 0.0003
- Number of critics: 2
- Discount factor γ: 0.99
- Batch size: 128
- Target network update interval: 1000

The experiments were performed with the following configurations:

- **Configuration 1:** Regularization term β: 0.01, Action Flexibility: 0.3
- **Configuration 2:** Regularization term β: 0.01, Action Flexibility: 0.1
- **Configuration 3:** Regularization term β: 0.01, Action Flexibility: 0.5
- **Configuration 4:** Regularization term β: 0.05, Action Flexibility: 0.3

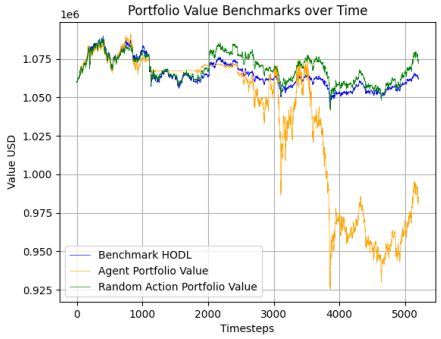
4.3.1 Sharpe Ratio Reward Function Results

Metrics	Config 1	Config 2	Config 3	Config 4
Sharpe Ratio	-0.0125	-0.0188	-0.0071	0.0006
Avg Daily Return	-0.025	-0.010	-0.102	0.019
Avg Monthly Return	-1.247	-0.332	-3.563	0.483
Std Dev Daily Return	0.596	0.201	3.106	1.805
Std Dev Monthly Return	1.872	0.949	14.593	6.836
Cumulative Return	-7.530	-2.317	-30.606	-0.287

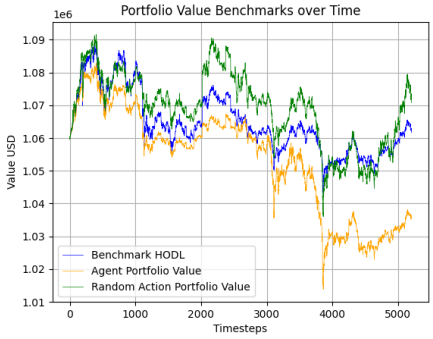
Table 4.5: Discrete BCQ agent performance comparison across different configurations on Sharpe Ratio reward function.

Metrics	Config 4	HODL	Random
Sharpe Ratio	0.0006	-0.0089	-0.0053
Avg Daily Return	0.019	0.003	0.005
Avg Monthly Return	0.483	0.054	0.101
Std Dev Daily Return	1.805	0.205	0.313
Std Dev Monthly Return	6.836	1.185	1.525
Cumulative Return	-0.287	0.171	0.358

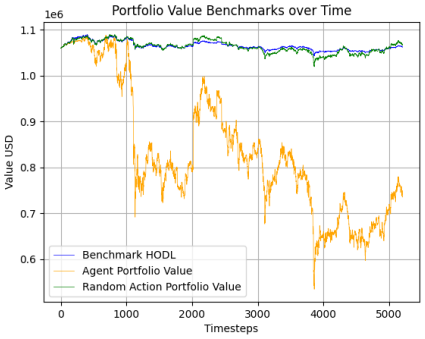
Table 4.6: Best performing Discrete BCQ configuration compared to HODL and Random strategies on Sharpe Ratio reward function.



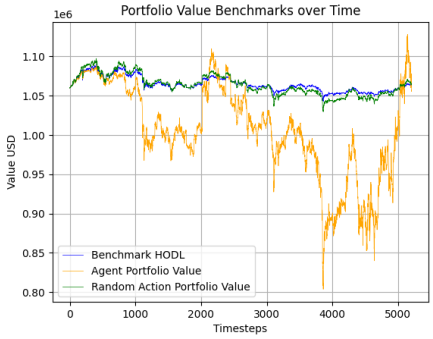
(a) Configuration 1



(b) Configuration 2



(c) Configuration 3



(d) Configuration 4

Figure 4.3: Discrete BCQ agent performance comparison across different configurations and strategies on Sharpe Ratio reward function.

4.3.2 Analysis

Based on the results on Tables 4.5, 4.6 and the plots in Figure 4.3:

The results of the Discrete BCQ agent across the four configurations reveal notable variations in performance, particularly in terms of risk-adjusted returns and volatility. Configuration 4 out-

performs the others, achieving a marginally positive Sharpe Ratio and the highest average daily and monthly returns. This suggests that stronger regularization helps stabilize returns, though the agent still exhibits high volatility. In contrast, Configuration 3 yields the worst performance, with the lowest Sharpe Ratio, significant negative returns, and extreme volatility, indicating that excessive action flexibility may lead to erratic decisions. Configurations 1 and 2 show intermediate results, with Configuration 2 (lower flexibility = 0.1) demonstrating better risk management but still negative returns. Overall, the results highlight a trade-off between regularization and flexibility: higher β (Configuration 4) improves stability, while excessive flexibility (Configuration 3) exacerbates risk. However, even the best configuration achieves only negligible positive returns, suggesting further tuning or alternative approaches may be needed for robust performance.

The agent in Configuration 4 exhibits a marginally better risk-adjusted performance than both the HODL and Random strategies, as indicated by its slightly positive Sharpe Ratio compared to the negative ratios of HODL and Random. However, despite higher average daily and monthly returns relative to the benchmarks, the agent suffers from significantly greater volatility, with daily and monthly standard deviations far exceeding those of HODL and Random. This high volatility likely contributed to its poor cumulative return, under-performing both HODL and Random. While the agent demonstrates an ability to generate higher returns in the short term, its excessive risk-taking leads to substantial draw downs, resulting in long-term under-performance compared to passive and random strategies.

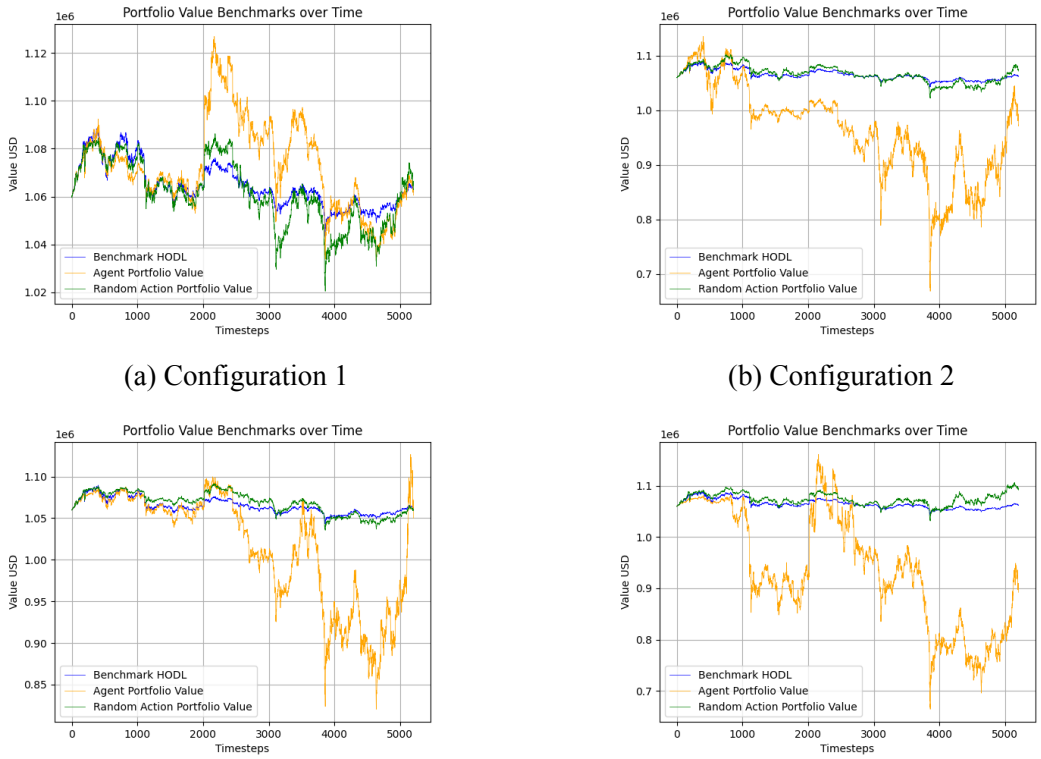
4.3.3 Returns Reward Function Results

Metrics	**Config 1**	**Config 2**	**Config 3**	**Config 4**
Sharpe Ratio	-0.0044	-0.0016	-0.0002	-0.0043
Avg Daily Return	0.002	0.000	0.013	-0.054
Avg Monthly Return	0.102	-0.794	-1.333	-2.490
Std Dev Daily Return	0.446	2.108	1.442	2.478
Std Dev Monthly Return	2.616	5.175	3.397	13.530
Cumulative Return	0.055	-8.203	-0.942	-15.850

Table 4.7: Discrete BCQ agent performance comparison across different configurations on Returns reward function.

Metrics	Config 3	HODL	Random
Sharpe Ratio	-0.0002	-0.0089	-0.0084
Avg Daily Return	0.013	0.003	0.001
Avg Monthly Return	-1.333	0.054	-0.001
Std Dev Daily Return	1.442	0.205	0.255
Std Dev Monthly Return	3.397	1.185	1.414
Cumulative Return	-0.942	0.171	-0.090

Table 4.8: Best performing Discrete BCQ configuration compared to HODL and Random strategies on Returns reward function.



(a) Configuration 1

(b) Configuration 2

(c) Configuration 3

(d) Configuration 4

Figure 4.4: Discrete BCQ agent performance comparison across different configurations and strategies on Returns reward function.

4.3.4 Analysis

Based on the results on Tables 4.7, 4.8 and the plots in Figure 4.4:

The results of the Discrete BCQ offline RL agent across the four configurations reveal notable differences in performance. Configuration 3, with a higher action flexibility and low regular-

ization, achieved the best Sharpe Ratio and the highest average daily return, suggesting a better risk-adjusted performance compared to the others. However, its average monthly return and cumulative return were still negative, indicating overall losses. Configuration 2, with the lowest action flexibility, showed moderate risk but poor cumulative returns, while Configuration 1, with balanced settings, had the least volatility in daily returns and the best cumulative return, albeit still marginal. Configuration 4, with higher regularization, exhibited the worst performance across most metrics, including the lowest average daily return and highest volatility, suggesting that excessive regularization harms performance. Overall, lower action flexibility and moderate regularization appear more stable, whereas higher flexibility may improve short-term returns but not necessarily long-term profitability.

The agent in Configuration 3 exhibits mixed performance compared to the HODL and Random benchmarks. While it achieves a marginally better Sharpe Ratio than both HODL and Random, its higher standard deviations indicate significantly greater volatility. The agent's average daily return outperforms HODL and Random, but its negative average monthly return and cumulative return are substantially worse than HODL and slightly worse than Random. This suggests that while the agent may generate short-term gains, its high-risk strategy leads to poor consistency and long-term under-performance compared to passive holding. The Random strategy, though slightly worse in Sharpe Ratio and cumulative return, demonstrates lower volatility, making it a less risky alternative. Overall, Configuration 3's aggressive approach does not translate into sustainable returns, highlighting potential flaws in its decision-making or risk management.

4.4 CQL Results

The following hyper-parameters remained constant:

- Actor learning rate: 0.001
- Critic learning rate: 0.0001
- Number of critics: 2
- Discount factor γ: 0.999
- Batch size: 256
- Target network synchronization coefficiency: 0.08

The experiments were performed with the following configurations:

- **Configuration 1:** Conservative Weight α: 6, Sampled Actions: 25
- **Configuration 2:** Conservative Weight α: 1, Sampled Actions: 25
- **Configuration 3:** Conservative Weight α: 10, Sampled Actions: 25

- **Configuration 4:** Conservative Weight α: 6, Sampled Actions: 40

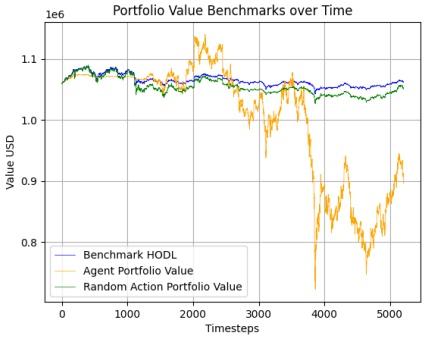
4.4.1 Sharpe Ratio Reward Function Results

Metrics	Config 1	Config 2	Config 3	Config 4
Sharpe Ratio	-0.0071	-0.0030	-0.0008	-0.0056
Avg Daily Return	-0.053	-0.004	-0.001	-0.024
Avg Monthly Return	-2.394	-1.179	-0.572	-1.624
Std Dev Daily Return	1.667	2.431	3.075	1.416
Std Dev Monthly Return	5.111	8.710	13.969	6.050
Cumulative Return	-15.306	-13.203	-12.614	-9.702

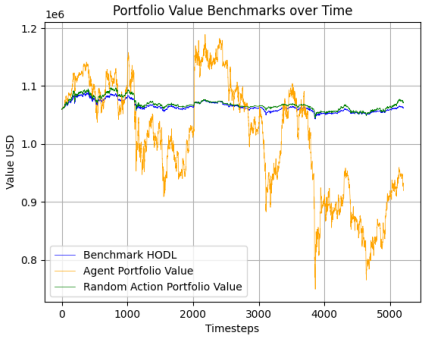
Table 4.9: CQL agent performance comparison across different configurations on Sharpe Ratio reward function.

Metrics	Config 3	HODL	Random
Sharpe Ratio	-0.0008	-0.0089	-0.0027
Avg Daily Return	-0.001	0.003	0.008
Avg Monthly Return	-0.572	0.054	0.219
Std Dev Daily Return	3.075	0.205	0.245
Std Dev Monthly Return	13.969	1.185	1.113
Cumulative Return	-12.614	0.171	1.658

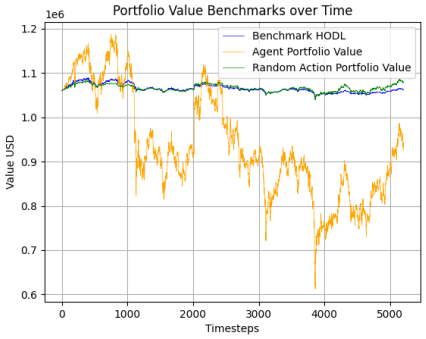
Table 4.10: Best performing CQL configuration compared to HODL and Random strategies on Sharpe Ratio reward function.



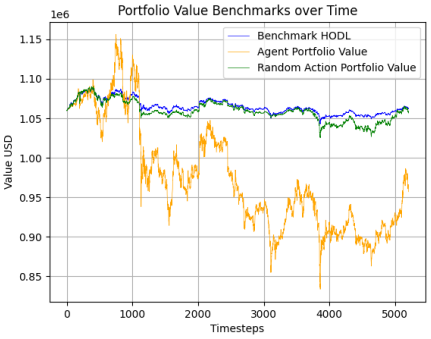
(a) Configuration 1



(b) Configuration 2



(c) Configuration 3



(d) Configuration 4

Figure 4.5: CQL agent performance comparison across different configurations and strategies on Sharpe Ratio reward function.

4.4.2 Analysis

Based on the results on Tables 4.9, 4.10 and the plots in Figure 4.5:

The results indicate that Configuration 3 achieves the best performance across most metrics, including the highest Sharpe Ratio, the least negative average daily and monthly returns, and the second-best cumulative return. However, it also exhibits the highest volatility in daily and monthly returns, suggesting that a higher conservative weight reduces overestimation bias but increases variance. Configuration 2 performs moderately, with a slightly worse Sharpe Ratio but lower volatility than Configuration 3, indicating that a lower α may not sufficiently constrain Q-values. Configuration 1 and Configuration 4 show mixed results: Configuration 4 improves cumulative return over Configuration 1 due to more sampled actions reducing variance, but both under-perform Configuration 3. Overall, a higher α (Configuration 3) appears most effective for risk-adjusted returns, though at the cost of increased volatility, while increasing sampled actions (Configuration 4) helps stabilize returns but does not match the performance of an optimally tuned α.

The agent in Configuration 3 under-performs compared to both the HODL and Random strate-

gies across most metrics. While its Sharpe Ratio is slightly better than HODL and Random, it still indicates poor risk-adjusted returns. The agent suffers from significantly higher volatility and delivers negative average daily and monthly returns, contrasting with the positive returns of both benchmarks. Most notably, its cumulative return is drastically worse than HODL and Random, suggesting severe value erosion over time. While the agent may marginally mitigate risk compared to HODL, its excessive volatility and consistent losses make it inferior to passive or even random strategies in this evaluation.

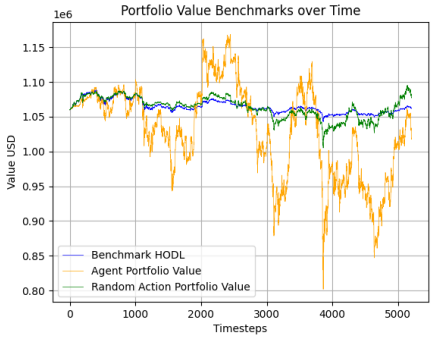
4.4.3 Returns Reward Function Results

Metrics	Config 1	Config 2	Config 3	Config 4
Sharpe Ratio	-0.0003	0.0102	-0.0124	-0.0079
Avg Daily Return	0.011	0.058	-0.189	-0.056
Avg Monthly Return	-0.121	1.029	-6.806	-1.400
Std Dev Daily Return	1.967	0.980	2.656	1.548
Std Dev Monthly Return	6.267	1.631	10.137	8.632
Cumulative Return	-3.855	13.483	-39.827	-12.205

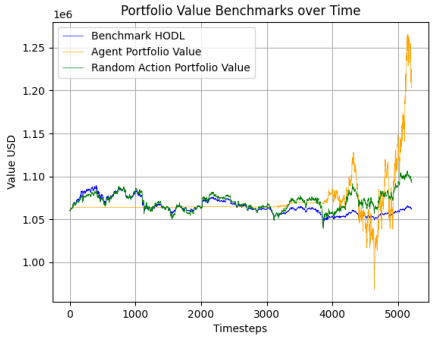
Table 4.11: CQL agent performance comparison across different configurations on Returns reward function.

Metrics	Config 2	HODL	Random
Sharpe Ratio	0.0102	-0.0089	0.0018
Avg Daily Return	0.058	0.003	0.019
Avg Monthly Return	1.029	0.054	0.618
Std Dev Daily Return	0.980	0.205	0.348
Std Dev Monthly Return	1.631	1.185	1.440
Cumulative Return	13.483	0.171	3.135

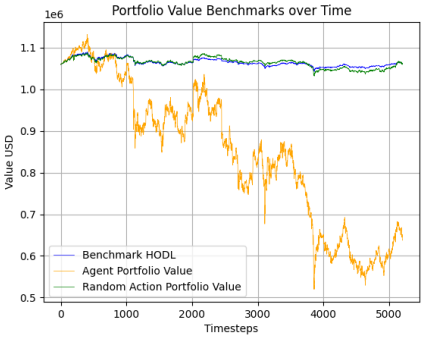
Table 4.12: Best performing CQL configuration compared to HODL and Random strategies on Returns reward function.



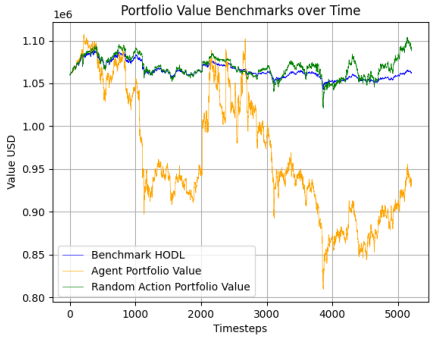
(a) Configuration 1



(b) Configuration 2



(c) Configuration 3



(d) Configuration 4

Figure 4.6: CQL agent performance comparison across different configurations and strategies on Returns reward function.

4.4.4 Analysis

Based on the results on Tables 4.11, 4.12 and the plots in Figure 4.6:

The experimental results reveal significant performance differences across the four CQL configurations. Configuration 2 outperforms the others, achieving the highest Sharpe ratio, average daily and monthly returns, and cumulative return, while maintaining the lowest daily and monthly return volatility. This suggests that a lower conservative weight strikes an effective balance between policy improvement and conservatism, avoiding the over-regularization seen in Configurations 1, 3, and 4. In contrast, Configuration 3 performs the worst, with highly negative returns and high volatility, indicating that excessive conservatism severely degrades performance. Configuration 4 shows slightly better results than Configurations 1 and 3 but still under-performs compared to Configuration 2, implying that increasing sampled actions alone does not compensate for suboptimal α. Configuration 1 exhibits moderate performance but suffers from negative Sharpe ratios and cumulative returns, reinforcing that $\alpha = 6$ may still be too restrictive. Overall, these results highlight the critical role of α in CQL's trade-off between conservatism and policy optimization, with $\alpha = 1$ emerging as the most effective choice in this

setting.

The agent in Configuration 2 outperforms both the HODL and Random strategies across most metrics, demonstrating superior risk-adjusted returns and cumulative performance. With a Sharpe Ratio of 0.0102, it surpasses HODL and Random, indicating better return per unit of risk. The agent also achieves significantly higher average daily and monthly returns, as well as a substantially higher cumulative return. However, the agent exhibits higher volatility compared to HODL and Random. While the Random strategy shows some competitive risk-adjusted returns, the agent's significantly higher cumulative and average returns suggest it is more effective at capitalizing on market opportunities, albeit with greater risk. HODL, as expected, under-performs due to its passive nature.

4.5 Discrete CQL Results

The following hyper-parameters remained constant:

- Learning rate: 0.01
- Number of critics: 2
- Discount factor γ: 0.99
- Batch size: 32
- Target network update interval: 3000

The experiments were performed with the following configurations:

- **Configuration 1:** α value: 2
- **Configuration 2:** α value: 1
- **Configuration 3:** α value: 5
- **Configuration 4:** α value: 3

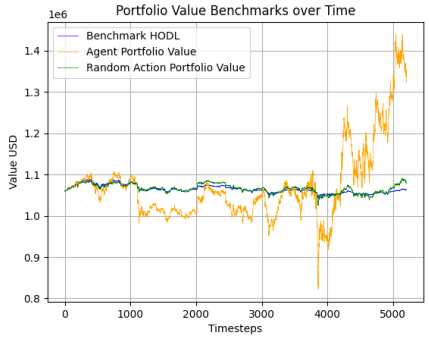
4.5.1 Sharpe Ratio Reward Function Results

Metrics	Config 1	Config 2	Config 3	Config 4
Sharpe Ratio	0.0102	0.0028	0.0118	0.0120
Avg Daily Return	0.159	0.039	0.057	0.062
Avg Monthly Return	4.888	1.193	1.901	2.063
Std Dev Daily Return	2.239	0.888	0.570	0.625
Std Dev Monthly Return	9.086	3.104	2.450	2.703
Cumulative Return	25.384	4.472	10.630	11.530

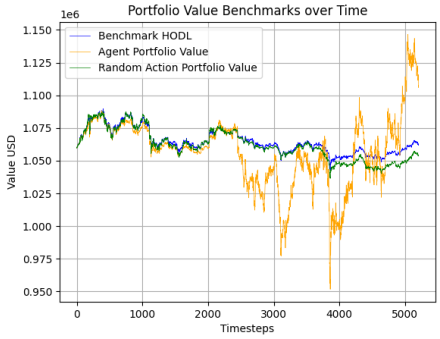
Table 4.13: Discrete CQL agent performance comparison across different configurations on Sharpe Ratio reward function.

Metrics	Config 4	HODL	Random
Sharpe Ratio	0.0120	-0.0089	0.0031
Avg Daily Return	0.062	0.003	0.018
Avg Monthly Return	2.063	0.054	0.607
Std Dev Daily Return	0.625	0.205	0.321
Std Dev Monthly Return	2.703	1.185	1.477
Cumulative Return	11.530	0.171	3.630

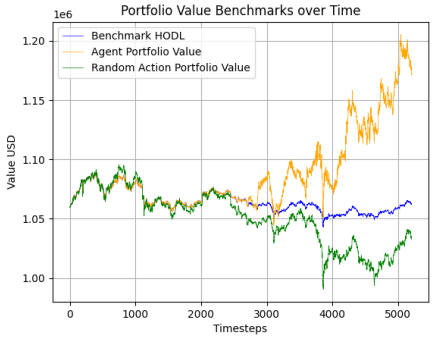
Table 4.14: Best performing Discrete CQL configuration compared to HODL and Random strategies on Sharpe Ratio reward function.



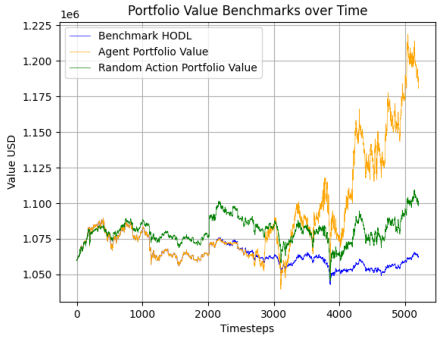
(a) Configuration 1



(b) Configuration 2



(c) Configuration 3



(d) Configuration 4

Figure 4.7: Discrete CQL agent performance comparison across different configurations and strategies on Sharpe Ratio reward function.

4.5.2 Analysis

Based on the results on Tables 4.13, 4.14 and the plots in Figure 4.7:

The results of the Discrete CQL offline RL agent with different α values reveal notable performance variations. Configuration 4 achieved the highest Sharpe Ratio, indicating the best risk-adjusted returns, closely followed by Configuration 3 and Configuration 1. Configuration 2 performed the worst across all metrics, suggesting that a lower α value may inadequately penalize out-of-distribution actions, leading to suboptimal policies. Notably, Configurations 3 and 4 exhibited lower daily and monthly return volatility compared to Configuration 1, implying better stability despite slightly lower average returns. Configuration 1, while yielding the highest average daily and monthly returns, also had the highest volatility, which likely contributed to its lower Sharpe Ratio compared to Configurations 3 and 4. The cumulative returns further reinforce this trend, with Configuration 1 leading but at the cost of higher risk, while Configurations 3 and 4 offered a more balanced trade-off between return and risk. Overall, an intermediate α value (3 or 5) appears optimal for maximizing risk-adjusted performance in this setting.

The agent in Configuration 4 significantly outperforms both the HODL and Random strategies

across all key metrics. With a Sharpe Ratio of 0.0120, it demonstrates better risk-adjusted returns compared to HODL and Random. The agent also achieves substantially higher average daily and monthly returns than HODL and Random, though with greater volatility, as evidenced by higher standard deviations in daily and monthly returns. Most notably, the agent's cumulative return vastly surpasses both benchmarks, highlighting its superior profitability over the tested period. While the agent's strategy is riskier than the passive HODL approach or a random strategy, its significantly higher returns justify the additional volatility, making it the most effective strategy among the three.

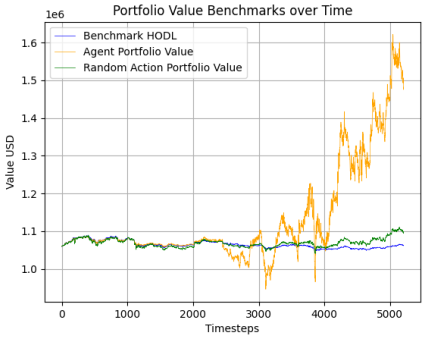
4.5.3 Returns Reward Function Results

Metrics	Config 1	Config 2	Config 3	Config 4
Sharpe Ratio	0.0151	-0.0024	0.0137	0.0116
Avg Daily Return	0.202	-0.031	0.178	0.119
Avg Monthly Return	6.686	-2.759	5.824	3.884
Std Dev Daily Return	2.021	2.146	1.909	1.437
Std Dev Monthly Return	9.935	4.696	8.964	6.114
Cumulative Return	39.746	-11.597	33.297	21.465

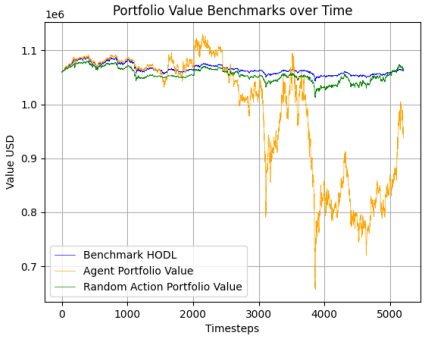
Table 4.15: Discrete CQL agent performance comparison across different configurations on Returns reward function.

Metrics	Config 1	HODL	Random
Sharpe Ratio	0.0151	-0.0089	0.0022
Avg Daily Return	0.202	0.003	0.019
Avg Monthly Return	6.686	0.054	0.655
Std Dev Daily Return	2.021	0.205	0.344
Std Dev Monthly Return	9.935	1.185	1.454
Cumulative Return	39.746	0.171	3.308

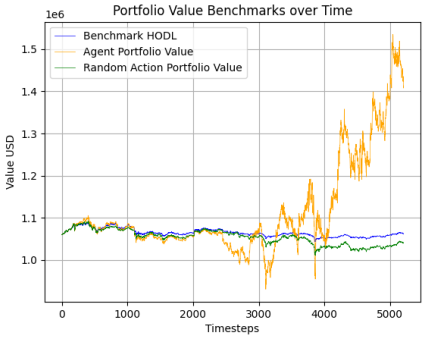
Table 4.16: Best performing Discrete CQL configuration compared to HODL and Random strategies on Returns reward function.



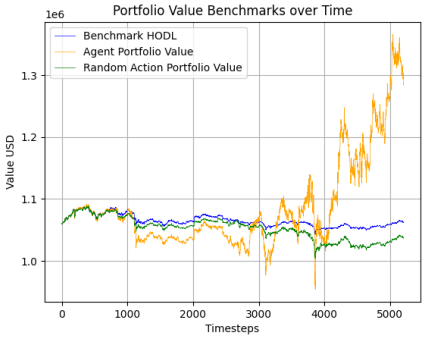
(a) Configuration 1



(b) Configuration 2



(c) Configuration 3



(d) Configuration 4

Figure 4.8: Discrete CQL agent performance comparison across different configurations and strategies on Returns reward function.

4.5.4 Analysis

Based on the results on Tables 4.15, 4.16 and the plots in Figure 4.8:

The results of the Discrete CQL offline RL agent across the four configurations reveal notable differences in performance based on the chosen α values. Configuration achieves the highest Sharpe Ratio, average daily and monthly returns, as well as the highest cumulative return, suggesting it strikes an effective balance between risk and reward. In contrast, Configuration 2 performs poorly, with negative Sharpe Ratio, average daily, and monthly returns, along with a significant cumulative loss, indicating insufficient conservatism leading to unfavorable outcomes. Configuration 3 shows competitive performance, with the second-highest Sharpe Ratio and returns, but slightly lower than Configuration 1, suggesting that excessive conservatism may slightly hinder returns. Configuration 4 demonstrates moderate performance, with lower returns but the lowest standard deviations, implying better stability at the cost of reduced profitability. Overall, Configuration 1 appears optimal, while Configuration 2 is clearly suboptimal.

The agent in Configuration 1 significantly outperforms both the HODL and Random strategies across all metrics, demonstrating superior risk-adjusted returns and cumulative performance.

With a Sharpe Ratio of 0.0151, it surpasses HODL and Random, indicating better return per unit of risk. The agent also achieves substantially higher average daily and monthly returns compared to HODL and Random, albeit with greater volatility, as seen in the higher standard deviations of daily and monthly returns. Most notably, the agent's cumulative return vastly exceeds both benchmarks, highlighting its ability to generate significant gains over time. While the agent's strategy introduces higher risk, its returns justify the increased volatility, making it a more effective approach than passive holding or random decision-making.

4.6 IQL Results

The following hyper-parameters remained constant:

- Actor learning rate: 0.0003
- Critic learning rate: 0.0003
- Number of critics: 2
- Discount factor γ: 0.99
- Batch size: 256
- Target network synchronization coefficiency: 0.005

The experiments were performed with the following configurations:

- **Configuration 1:** Expectile τ: 0.9
- **Configuration 2:** Expectile τ: 0.5
- **Configuration 3:** Expectile τ: 0.7
- **Configuration 4:** Expectile τ: 0.95

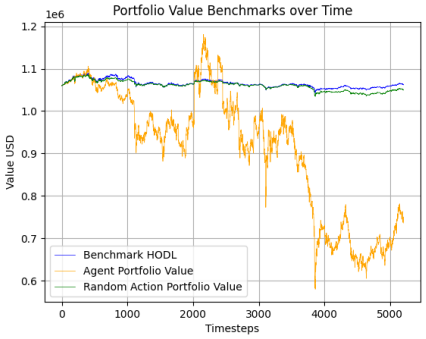
4.6.1 Sharpe Ratio Reward Function Results

Metrics	Config 1	Config 2	Config 3	Config 4
Sharpe Ratio	-0.0084	0.0020	-0.0102	-0.0134
Avg Daily Return	-0.134	0.070	-0.168	-0.170
Avg Monthly Return	-4.743	1.856	-5.474	-6.526
Std Dev Daily Return	2.748	3.394	2.948	2.260
Std Dev Monthly Return	13.411	15.277	14.770	7.345
Cumulative Return	-30.569	-5.298	-37.070	-36.813

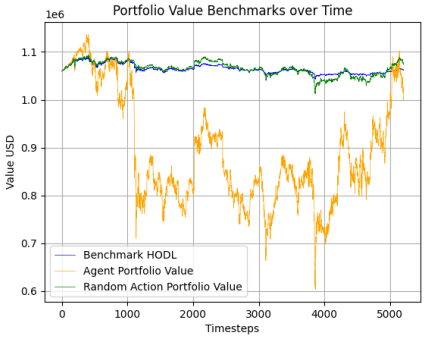
Table 4.17: IQL agent performance comparison across different configurations on Sharpe Ratio reward function.

Metrics	Config 2	HODL	Random
Sharpe Ratio	0.0020	-0.0089	-0.0022
Avg Daily Return	0.070	0.003	0.011
Avg Monthly Return	1.856	0.054	0.246
Std Dev Daily Return	3.394	0.205	0.373
Std Dev Monthly Return	15.277	1.185	1.660
Cumulative Return	-5.298	0.171	1.332

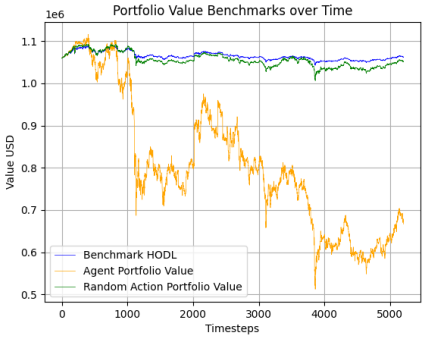
Table 4.18: Best performing IQL configuration compared to HODL and Random strategies on Sharpe Ratio reward function.



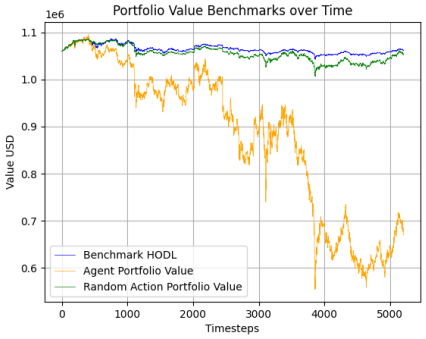
(a) Configuration 1



(b) Configuration 2



(c) Configuration 3



(d) Configuration 4

Figure 4.9: IQL agent performance comparison across different configurations and strategies on Sharpe Ratio reward function.

4.6.2 Analysis

Based on the results on Tables 4.17, 4.18 and the plots in Figure 4.9:

The results of the IQL offline RL agent across the four configurations reveal notable differences in performance based on the expectile parameter. Configuration 2 stands out as the only setup with positive Sharpe Ratio and average daily and monthly returns, suggesting it strikes a better balance between risk and return compared to the others. In contrast, Configurations 1, 3, and 4 all yield negative Sharpe Ratios and cumulative returns, with Configuration 4 exhibiting the worst average monthly return despite having the lowest standard deviation in monthly returns, indicating overly conservative or misaligned value estimation. Configuration 3 shows the highest volatility in daily returns and poor cumulative returns, while Configuration 1 performs marginally better but still suffers from negative returns. Overall, the results suggest that a moderate expectile value aligns best with the Sharpe Ratio reward function, whereas higher τ values lead to suboptimal risk-adjusted returns.

The agent in Configuration 2 demonstrates a mixed performance compared to the HODL and Random strategies. While it achieves a marginally positive Sharpe Ratio, outperforming both

HODL and Random, its higher average daily and monthly returns suggest better short-term gains. However, the agent exhibits significantly higher volatility, with daily and monthly standard deviations far exceeding those of HODL and Random, indicating greater risk. Notably, the agent's cumulative return under-performs both benchmarks, suggesting that its aggressive strategy may lead to substantial losses over time despite short-term advantages. Overall, while the agent shows potential for higher returns, its risk-adjusted performance and long-term sustainability are questionable compared to simpler strategies.

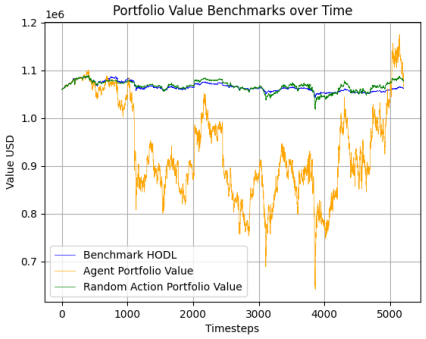
4.6.3 Returns Reward Function Results

Metrics	Config 1	Config 2	Config 3	Config 4
Sharpe Ratio	0.0034	-0.0036	-0.0092	-0.0024
Avg Daily Return	0.076	-0.067	-0.166	-0.037
Avg Monthly Return	2.481	-2.798	-6.323	-3.137
Std Dev Daily Return	3.266	3.488	3.291	3.143
Std Dev Monthly Return	14.220	17.323	15.964	13.976
Cumulative Return	0.947	-24.420	-39.887	-18.660

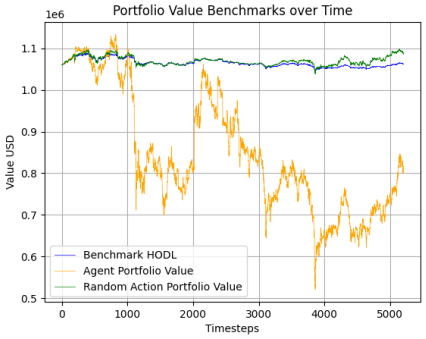
Table 4.19: IQL agent performance comparison across different configurations on Returns reward function.

Metrics	Config 1	HODL	Random
Sharpe Ratio	0.0034	-0.0089	-0.0018
Avg Daily Return	0.076	0.003	0.012
Avg Monthly Return	2.481	0.054	0.344
Std Dev Daily Return	3.266	0.205	0.346
Std Dev Monthly Return	14.220	1.185	1.245
Cumulative Return	0.947	0.171	1.617

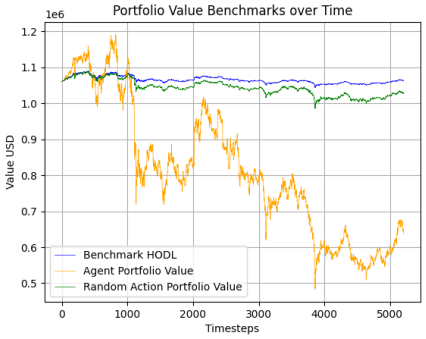
Table 4.20: Best performing IQL configuration compared to HODL and Random strategies on Returns reward function.



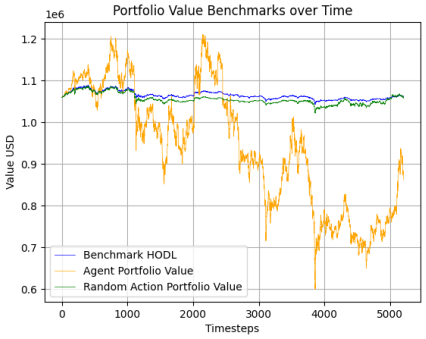
(a) Configuration 1



(b) Configuration 2



(c) Configuration 3



(d) Configuration 4

Figure 4.10: IQL agent performance comparison across different configurations and strategies on Returns reward function.

4.6.4 Analysis

Based on the results on Tables 4.19, 4.20 and the plots in Figure 4.10:

The results indicate that Configuration 1 performs significantly better than the other configurations across most metrics, achieving the highest Sharpe Ratio, positive average daily and monthly returns, and the highest cumulative return. This suggests that a higher expectile value effectively balances risk and return, as evidenced by its lower standard deviation in monthly returns compared to Configurations 2 and 3. In contrast, Configurations 2 and 3 yield negative returns and Sharpe Ratios, with Configuration 3 performing the worst in terms of average and cumulative returns, likely due to insufficient optimism in value estimation. Configuration 4 shows slightly better risk-adjusted returns than Configurations 2 and 3 but still under-performs compared to Configuration 1, suggesting that an overly aggressive expectile value may lead to suboptimal policy extraction. Overall, Configuration 1 strikes the best trade-off, demonstrating robustness in both return generation and risk management.

The agent in Configuration 1 demonstrates a mixed performance compared to the HODL and Random strategies. While it achieves a higher Sharpe Ratio than both HODL and Random, sug-

gesting marginally better risk-adjusted returns, its high volatility indicates significantly greater risk. The agent also outperforms HODL in average daily and monthly returns, but its cumulative return lags behind the Random strategy, which benefited from higher volatility and chance. Overall, the agent's aggressive approach yields higher returns than HODL but with substantially more risk, while the Random strategy, despite its lack of sophistication, achieved the highest cumulative return, likely due to favorable market fluctuations.

4.7 BEAR Results

The following hyper-parameters remained constant:

- Actor learning rate: 0.004
- Critic learning rate: 0.0003
- Number of critics: 2
- Discount factor γ: 0.99
- Batch size: 128
- Target network synchronization coefficiency: 0.005
- Warm-up steps: 8000
- Gaussian kernel sigma: 1

The experiments were performed with the following configurations:

- **Configuration 1:** Action Samples: 6, Kernel: Laplacian
- **Configuration 2:** Action Samples: 6, Kernel: Gaussian
- **Configuration 3:** Action Samples: 3, Kernel: Laplacian
- **Configuration 4:** Action Samples: 10, Kernel: Gaussian

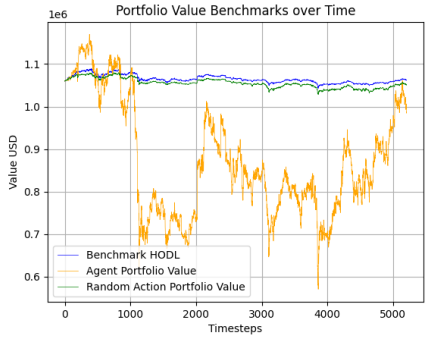
4.7.1 Sharpe Ratio Reward Function Results

Metrics	Config 1	Config 2	Config 3	Config 4
Sharpe Ratio	0.0022	0.0053	-0.0017	0.0161
Avg Daily Return	0.060	0.126	0.002	0.336
Avg Monthly Return	2.379	4.681	-0.640	10.416
Std Dev Daily Return	3.955	3.818	2.147	3.414
Std Dev Monthly Return	21.062	20.007	6.994	17.554
Cumulative Return	-6.894	7.161	-8.096	69.339

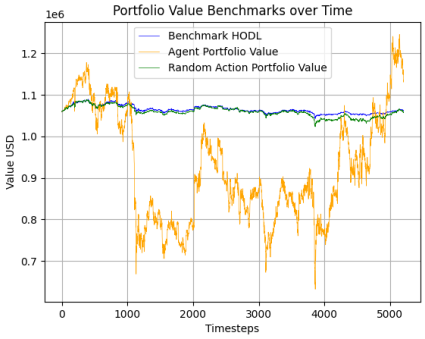
Table 4.21: BEAR agent performance comparison across different configurations on Sharpe Ratio reward function.

Metrics	Config 4	HODL	Random
Sharpe Ratio	0.0161	-0.0089	-0.0079
Avg Daily Return	0.336	0.003	0.000
Avg Monthly Return	10.416	0.054	-0.195
Std Dev Daily Return	3.414	0.205	0.346
Std Dev Monthly Return	17.554	1.185	1.516
Cumulative Return	69.339	0.171	-0.982

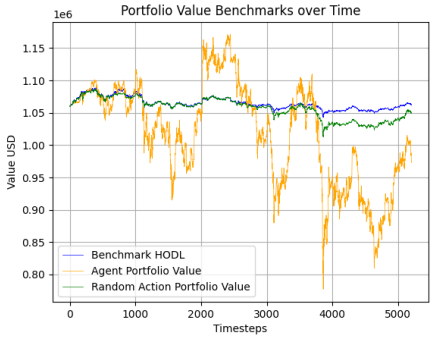
Table 4.22: Best performing BEAR configuration compared to HODL and Random strategies on Sharpe Ratio reward function.



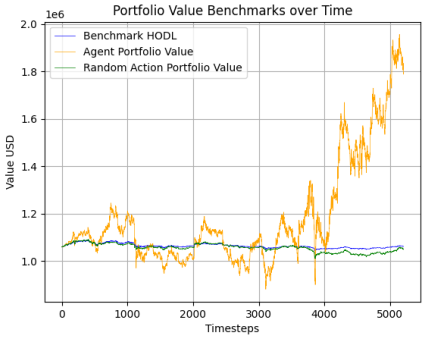
(a) Configuration 1



(b) Configuration 2



(c) Configuration 3



(d) Configuration 4

Figure 4.11: BEAR agent performance comparison across different configurations and strategies on Sharpe Ratio reward function.

4.7.2 Analysis

Based on the results on Tables 4.21, 4.22 and the plots in Figure 4.11:

The results demonstrate significant performance variations across the four configurations. Configuration 4 outperforms the others, achieving the highest Sharpe Ratio, cumulative return, and average returns, suggesting that a higher number of action samples combined with a Gaussian kernel improves stability and profitability. Configuration 2 also performs reasonably well, with positive Sharpe Ratio and cumulative return, indicating that the Gaussian kernel may be more effective than the Laplacian for this task. In contrast, Configuration 1 yields mixed results, with a near-zero Sharpe Ratio and negative cumulative return, while Configuration 3 performs the worst, with a negative Sharpe Ratio and the lowest returns, suggesting insufficient exploration due to fewer action samples. Notably, Configurations 1 and 2 exhibit higher volatility compared to Configuration 3, but the latter's lower risk does not compensate for its poor returns. Overall, the results highlight the importance of kernel selection and sufficient action sampling, with Configuration 4 emerging as the most effective.

The agent in Configuration 4 significantly outperforms both the HODL and Random strategies

across all metrics. With a positive Sharpe Ratio, it demonstrates better risk-adjusted returns compared to the negative ratios of HODL and Random, indicating superior performance relative to volatility. The agent also achieves substantially higher average daily and monthly returns compared to HODL and Random, highlighting its ability to generate consistent profits. While its daily and monthly return volatility is higher than the benchmarks, this is justified by its significantly higher cumulative return versus HODL and Random. Overall, the agent's strategy delivers stronger returns with acceptable risk, whereas HODL and Random strategies either under-perform or yield losses.

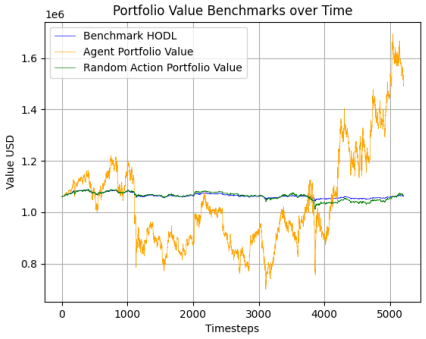
4.7.3 Returns Reward Function Results

Metrics	Config 1	Config 2	Config 3	Config 4
Sharpe Ratio	0.0113	-0.0019	0.0031	0.0109
Avg Daily Return	0.292	-0.011	0.077	0.213
Avg Monthly Return	9.382	-2.112	3.566	6.949
Std Dev Daily Return	4.133	2.840	4.163	3.125
Std Dev Monthly Return	21.393	10.897	22.616	15.036
Cumulative Return	41.400	-15.002	-4.497	34.450

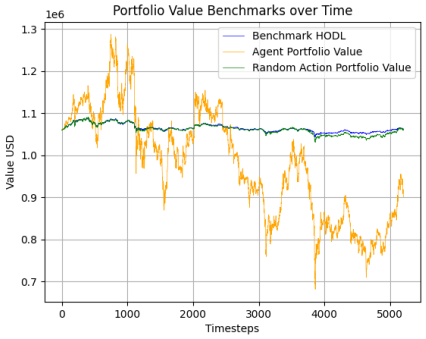
Table 4.23: BEAR agent performance comparison across different configurations on Returns reward function.

Metrics	Config 1	HODL	Random
Sharpe Ratio	0.0113	-0.0089	-0.0052
Avg Daily Return	0.292	0.003	0.006
Avg Monthly Return	9.382	0.054	0.052
Std Dev Daily Return	4.133	0.205	0.329
Std Dev Monthly Return	21.393	1.185	1.378
Cumulative Return	41.400	0.171	0.238

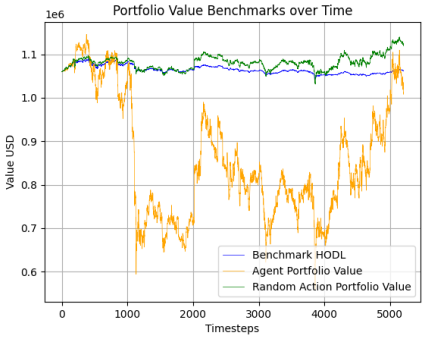
Table 4.24: Best performing BEAR configuration compared to HODL and Random strategies on Returns reward function.



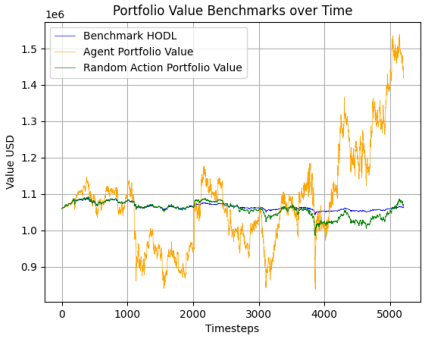
(a) Configuration 1



(b) Configuration 2



(c) Configuration 3



(d) Configuration 4

Figure 4.12: BEAR agent performance comparison across different configurations and strategies on Returns reward function.

4.7.4 Analysis

Based on the results on Tables 4.23, 4.24 and the plots in Figure 4.12:

The results demonstrate notable performance differences across the four BEAR offline RL configurations. Configuration 1 achieves the highest Sharpe Ratio, average daily and monthly returns, and cumulative return, suggesting robust performance with balanced risk-adjusted rewards. However, it also exhibits the highest volatility, indicating greater risk. Configuration 2 performs poorly, with negative Sharpe Ratio, average returns, and cumulative return, but shows the lowest volatility, implying overly conservative actions. Configuration 3 yields modest returns but struggles with cumulative return and high volatility, likely due to insufficient action sampling. Configuration 4 strikes a balance, delivering strong Sharpe Ratio, solid returns, and cumulative return, with moderate volatility, suggesting that increasing action samples with a Gaussian kernel can mitigate risk while maintaining performance. Overall, Configurations 1 and 4 are the most effective, with Configuration 1 favoring returns despite higher risk, and Configuration 4 offering a more stable alternative. The Gaussian kernel appears less effective with fewer samples but improves with more, while the Laplacian kernel benefits from intermediate

sampling.

The agent in Configuration 1 significantly outperforms both the HODL and Random strategies across all metrics, demonstrating superior risk-adjusted returns and cumulative performance. With a positive Sharpe Ratio compared to negative values for HODL and Random, the agent achieves better returns per unit of risk. It also exhibits substantially higher average daily and monthly returns versus the near-zero returns of the benchmarks. However, the agent's strategy comes with significantly higher volatility, as seen in the elevated standard deviations of daily and monthly returns, suggesting a more aggressive approach. Despite this, the agent's cumulative return vastly surpasses HODL and Random, indicating strong overall performance, albeit with greater risk. In contrast, the passive and random strategies deliver minimal returns with lower volatility, reflecting their lack of active decision-making.

4.8 TD3+BC Results

The following hyper-parameters remained constant:

- Actor learning rate: 0.0001
- Critic learning rate: 0.0001
- Number of critics: 2
- Discount factor γ: 0.99
- Batch size: 256
- Target network synchronization coefficiency: 0.005
- Actor update interval: 2

The experiments were performed with the following configurations:

- **Configuration 1:** Std dev for target noise: 0.1, Clipping range for target noise: 0.3, α value: 0.5
- **Configuration 2:** Std dev for target noise: 0.05, Clipping range for target noise: 0.2, α value: 0.8
- **Configuration 3:** Std dev for target noise: 0.2, Clipping range for target noise: 0.4, α value: 0.2
- **Configuration 4:** Std dev for target noise: 0.2, Clipping range for target noise: 0.5, α value: 2.5

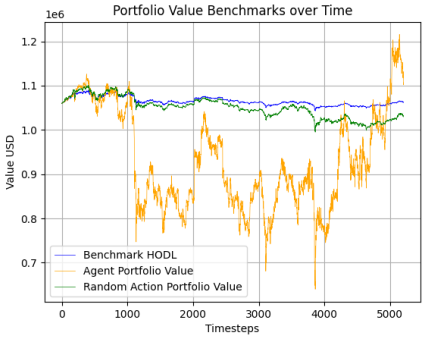
4.8.1 Sharpe Ratio Reward Function Results

Metrics	Config 1	Config 2	Config 3	Config 4
Sharpe Ratio	0.0044	-0.0106	-0.0066	0.0009
Avg Daily Return	0.100	-0.138	-0.089	0.012
Avg Monthly Return	3.449	-5.315	-4.117	0.403
Std Dev Daily Return	3.506	2.755	2.700	3.630
Std Dev Monthly Return	16.390	12.537	10.426	18.469
Cumulative Return	4.378	-35.804	-26.510	-9.529

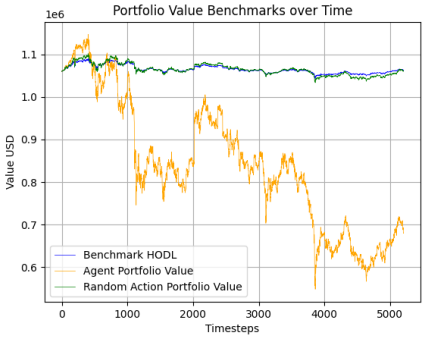
Table 4.25: TD3+BC agent performance comparison across different configurations on Sharpe Ratio reward function.

Metrics	Config 1	HODL	Random
Sharpe Ratio	0.0044	-0.0089	-0.011
Avg Daily Return	0.100	0.003	-0.007
Avg Monthly Return	3.449	0.054	-0.421
Std Dev Daily Return	3.506	0.205	0.383
Std Dev Monthly Return	16.390	1.185	1.924
Cumulative Return	4.378	0.171	-2.876

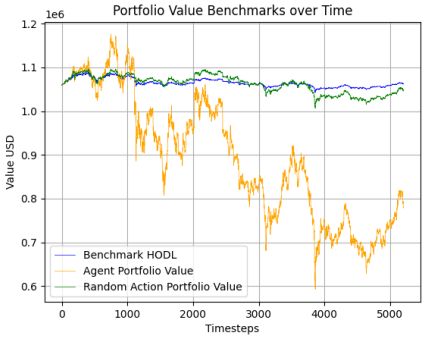
Table 4.26: Best performing TD3+BC configuration compared to HODL and Random strategies on Sharpe Ratio reward function.



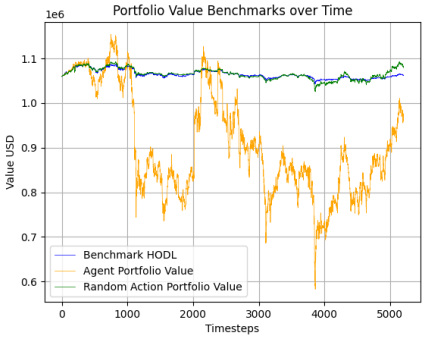
(a) Configuration 1



(b) Configuration 2



(c) Configuration 3



(d) Configuration 4

Figure 4.13: TD3+BC agent performance comparison across different configurations and strategies on Sharpe Ratio reward function.

4.8.2 Analysis

Based on the results on Tables 4.25, 4.26 and the plots in Figure 4.13:

The results indicate notable performance differences across the four configurations of the TD3+BC agent. Configuration 1 achieves the highest Sharpe Ratio and cumulative return, suggesting a balanced trade-off between exploration (via target noise) and policy regularization (via α). Configuration 2 performs poorly, with negative Sharpe Ratio and cumulative return, likely due to excessive conservatism from high α and insufficient exploration from low noise. Configuration 3 shows marginally better but still subpar results compared to Configuration 1, with higher volatility, indicating that increased noise without proper regularization harms stability. Configuration 4 yields near-neutral Sharpe Ratio but suffers from high return volatility and negative cumulative returns, suggesting that overly aggressive policy constraints destabilize performance. Overall, Configuration 1 strikes the best balance, while extreme settings (Configurations 2 and 4) degrade performance.

The agent in Configuration 1 significantly outperforms both the HODL and Random strategies across all metrics. With a positive Sharpe Ratio of 0.0044, it demonstrates better risk-adjusted

returns compared to HODL and Random, which both yield negative ratios. The agent also achieves substantially higher average daily and monthly returns versus HODL and Random, highlighting its ability to generate consistent profits. While its daily and monthly return volatility is higher than the benchmarks, this is justified by its superior cumulative return of 4.378%, vastly exceeding HODL and Random. Overall, the agent's strategy delivers stronger returns with acceptable risk, whereas HODL preserves capital with minimal growth, and Random leads to significant losses.

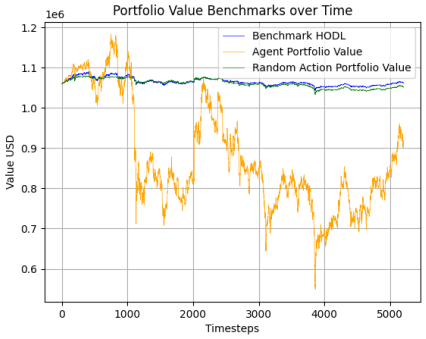
4.8.3 Returns Reward Function Results

Metrics	Config 1	Config 2	Config 3	Config 4
Sharpe Ratio	-0.0003	0.0032	-0.0065	-0.0132
Avg Daily Return	-0.001	0.092	-0.126	-0.228
Avg Monthly Return	-0.561	2.810	-4.506	-7.238
Std Dev Daily Return	3.651	3.489	3.283	3.001
Std Dev Monthly Return	17.996	16.543	15.704	14.329
Cumulative Return	-14.991	-0.184	-31.004	-44.863

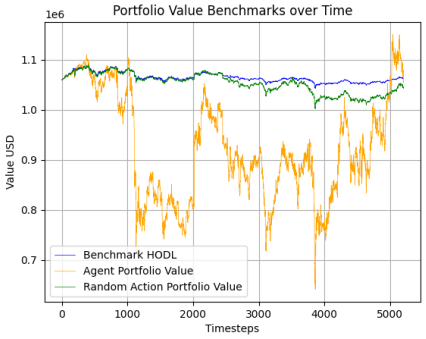
Table 4.27: TD3+BC agent performance comparison across different configurations on Returns reward function.

Metrics	Config 2	HODL	Random
Sharpe Ratio	0.0032	-0.0089	-0.0097
Avg Daily Return	0.092	0.003	-0.002
Avg Monthly Return	2.810	0.054	-0.247
Std Dev Daily Return	3.489	0.205	0.334
Std Dev Monthly Return	16.543	1.185	1.490
Cumulative Return	-0.184	0.171	-1.624

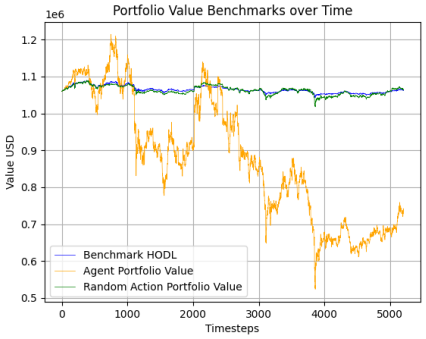
Table 4.28: Best performing TD3+BC configuration compared to HODL and Random strategies on Returns reward function.



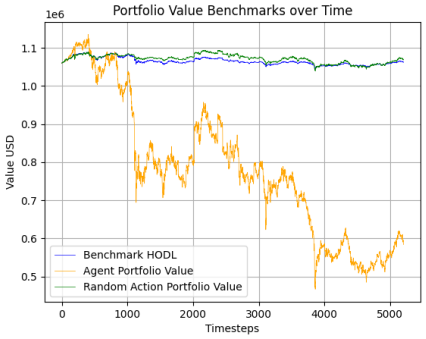
(a) Configuration 1



(b) Configuration 2



(c) Configuration 3



(d) Configuration 4

Figure 4.14: TD3+BC agent performance comparison across different configurations and strategies on Returns reward function.

4.8.4 Analysis

Based on the results on Tables 4.27, 4.28 and the plots in Figure 4.14:

The results of the TD3+BC agent across the four configurations reveal notable differences in performance. Configuration 2 outperforms the others, achieving the highest Sharpe Ratio, positive average daily and monthly returns, and the lowest cumulative loss. This suggests that moderate noise and a stronger behavioral cloning component (higher α) help stabilize learning and improve returns. In contrast, Configurations 1, 3, and 4 exhibit negative Sharpe Ratios and cumulative returns, with Configuration 4 performing the worst, indicating that excessive BC weighting or aggressive target noise destabilizes the policy. Interestingly, higher noise settings reduce return volatility, but this comes at the cost of significantly poorer returns, likely due to overly conservative actions.

The agent in Configuration 2 demonstrates a mixed performance compared to the HODL and Random strategies. While it achieves a marginally positive Sharpe Ratio and higher average daily and monthly returns than both benchmarks, its cumulative return under-performs HODL but significantly surpasses the Random strategy. The agent exhibits much higher volatility,

with daily and monthly return standard deviations far exceeding those of HODL and Random, indicating greater risk. Although the agent generates higher returns on average, its risk-adjusted performance is only slightly better than the benchmarks, which both show negative values. This suggests that while the agent is more active and potentially profitable in the short term, its high volatility and negative cumulative return raise concerns about long-term consistency compared to the passive HODL strategy. The Random strategy performs the worst across all metrics, as expected.

4.9 DT Results

The following hyper-parameters remained constant:

- Learning rate: 0.0001
- Discount factor γ: 0.99
- Batch size: 32
- Warm-up steps: 10000

The experiments were performed with the following configurations:

- **Configuration 1:** Context Size: 20, Attention Heads: 1, Layers: 3
- **Configuration 2:** Context Size: 50, Attention Heads: 2, Layers: 3
- **Configuration 3:** Context Size: 128, Attention Heads: 1, Layers: 3
- **Configuration 4:** Context Size: 128, Attention Heads: 4, Layers: 4

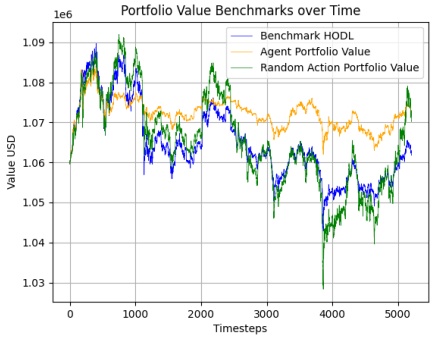
4.9.1 Sharpe Ratio Reward Function Results

Metrics	Config 1	Config 2	Config 3	Config 4
Sharpe Ratio	-0.0091	-0.0037	-0.006	-0.0106
Avg Daily Return	0.007	0.006	0.008	-0.011
Avg Monthly Return	0.179	0.166	0.218	-0.453
Std Dev Daily Return	0.119	0.668	0.193	0.461
Std Dev Monthly Return	0.614	2.541	0.924	1.773
Cumulative Return	1.147	-1.102	0.991	-3.748

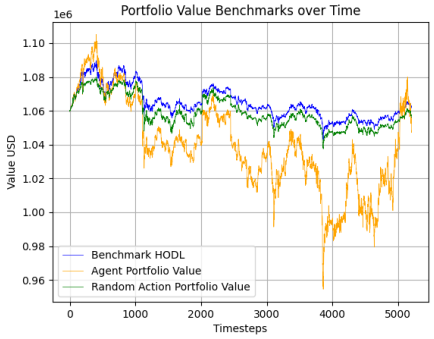
Table 4.29: DT agent performance comparison across different configurations on Sharpe Ratio reward function.

Metrics	Config 1	HODL	Random
Sharpe Ratio	-0.0091	-0.0089	-0.0041
Avg Daily Return	0.007	0.003	0.008
Avg Monthly Return	0.179	0.054	0.150
Std Dev Daily Return	0.119	0.205	0.291
Std Dev Monthly Return	0.614	1.185	1.473
Cumulative Return	1.147	0.171	0.970

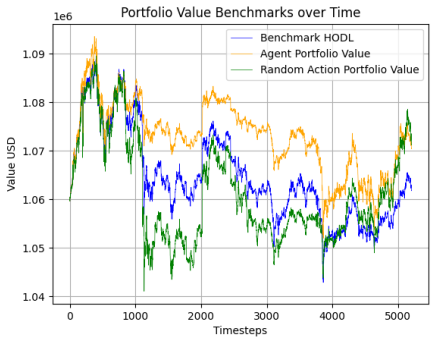
Table 4.30: Best performing DT configuration compared to HODL and Random strategies on Sharpe Ratio reward function.



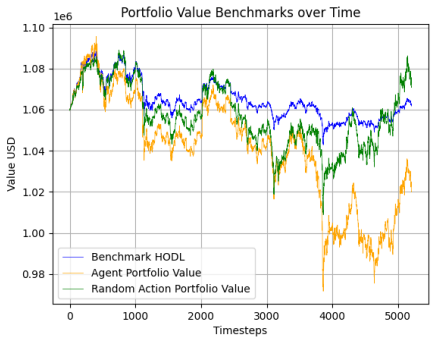
(a) Configuration 1



(b) Configuration 2



(c) Configuration 3



(d) Configuration 4

Figure 4.15: DT agent performance comparison across different configurations and strategies on Sharpe Ratio reward function.

4.9.2 Analysis

Based on the results on Tables 4.29, 4.30 and the plots in Figure 4.15:

The results indicate that none of the Decision Transformer configurations achieved a positive Sharpe Ratio, suggesting poor risk-adjusted performance across all experiments. Configuration

2 performed slightly better in terms of Sharpe Ratio and had lower volatility in daily returns compared to Configurations 1 and 3, though its cumulative return was negative. Configuration 3 achieved the highest average daily and monthly returns but still suffered from suboptimal risk-adjusted returns. Configuration 4, with the largest context size, most attention heads, and deepest architecture, performed the worst, exhibiting the lowest Sharpe Ratio, negative average returns, and the highest cumulative loss, likely due to overfitting or instability from increased complexity. Overall, the results suggest that increasing model capacity does not necessarily improve performance, and a moderate context size with additional attention heads may offer a better balance, though further tuning is needed to achieve positive risk-adjusted returns. The high standard deviations in returns, particularly for Configurations 2 and 4, indicate significant volatility, which aligns with their poor Sharpe Ratios.

The agent in Configuration 1 demonstrates mixed performance compared to the HODL and Random strategies. While it achieves a slightly lower Sharpe Ratio than HODL and significantly under-performs the Random strategy, it delivers higher average daily and monthly returns than HODL but slightly lower than the Random strategy. Notably, the agent exhibits lower volatility in daily and monthly returns compared to both HODL and Random, suggesting better risk management. Additionally, the agent's cumulative return significantly outperforms HODL and slightly exceeds the Random strategy, indicating stronger long-term growth despite its suboptimal risk-adjusted returns. Overall, the agent balances return and risk better than HODL but lags behind the Random strategy in Sharpe Ratio, highlighting a trade-off between volatility and absolute returns.

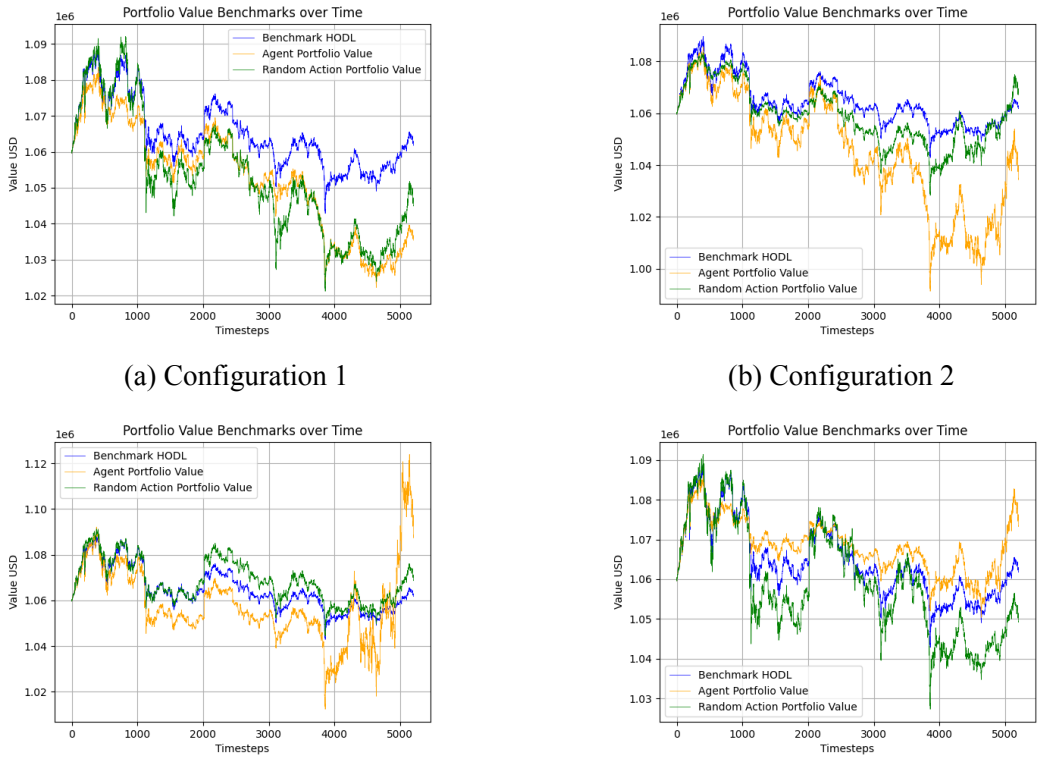
4.9.3 Returns Reward Function Results

Metrics	Config 1	Config 2	Config 3	Config 4
Sharpe Ratio	-0.0197	-0.0098	0.0008	-0.0058
Avg Daily Return	-0.008	-0.005	0.021	0.008
Avg Monthly Return	-0.337	-0.183	0.781	0.196
Std Dev Daily Return	0.195	0.401	0.521	0.172
Std Dev Monthly Return	1.020	1.646	2.100	0.789
Cumulative Return	-2.321	-2.363	2.704	1.268

Table 4.31: DT agent performance comparison across different configurations on Returns reward function.

Metrics	Config 3	HODL	Random
Sharpe Ratio	0.0008	-0.0089	-0.0053
Avg Daily Return	0.021	0.003	0.006
Avg Monthly Return	0.781	0.054	0.161
Std Dev Daily Return	0.521	0.205	0.245
Std Dev Monthly Return	2.100	1.185	1.398
Cumulative Return	2.704	0.171	0.834

Table 4.32: Best performing DT configuration compared to HODL and Random strategies on Returns reward function.



(a) Configuration 1

(b) Configuration 2

(c) Configuration 3

(d) Configuration 4

Figure 4.16: DT agent performance comparison across different configurations and strategies on Returns reward function.

4.9.4 Analysis

Based on the results on Tables 4.31, 4.32 and the plots in Figure 4.16:

The results of the Decision Transformer agent across the four configurations reveal notable differences in performance. Configuration 3 stands out as the best-performing setup, achieving

the highest Sharpe Ratio, average daily and monthly returns, and cumulative return. However, it also exhibits the highest volatility, with the largest standard deviations in daily and monthly returns, suggesting a trade-off between risk and reward. In contrast, Configuration 1 performs the worst, with negative Sharpe Ratios, returns, and cumulative returns, though it has lower volatility than Configurations 2 and 3. Configuration 2 shows slight improvements but still under-performs compared to Configuration 3. Configuration 4 delivers moderate performance, with positive but lower returns than Configuration 3 and significantly reduced volatility, indicating that increasing model complexity does not necessarily translate to better returns. Overall, Configuration 3 strikes the best balance for the returns-based reward function, while Configuration 4 may be preferable if lower risk is prioritized. The results suggest that context size plays a critical role in performance, while the impact of attention heads and layers is less straightforward.

The agent in Configuration 3 significantly outperforms both the HODL and random strategies across all metrics, demonstrating superior risk-adjusted returns and cumulative performance. With a Sharpe Ratio of 0.0008, it surpasses HODL and random, indicating better return per unit of risk. The agent also achieves substantially higher average daily and monthly returns compared to HODL and random, albeit with greater volatility, as reflected in higher standard deviations. Most notably, the agent's cumulative return dwarfs both benchmarks, suggesting strong compounding growth over time. While the agent's strategy is riskier, its higher returns justify the additional volatility, making it a more effective approach than passive holding or random trading.

4.10 Comparison Between Algorithms

4.10.1 Sharpe Ratio Reward Function

Metrics	BCQ	D-BCQ	CQL	D-CQL	IQL	BEAR	TD3+BC	DT
Sharpe Ratio	0.0095	0.0006	-0.0008	0.0120	0.0020	0.0161	0.0044	-0.0091
Avg Daily Return	0.035	0.019	-0.001	0.062	0.070	0.336	0.100	0.007
Avg Monthly Return	0.851	0.483	-0.572	2.063	1.856	10.416	3.449	0.179
Std Dev Daily Return	0.608	1.805	3.075	0.625	3.394	3.414	3.506	0.119
Std Dev Monthly Return	1.951	6.836	13.969	2.703	15.277	17.554	16.390	0.614
Cumulative Return	8.733	-0.287	-12.614	11.530	-5.298	69.339	4.378	1.147

Table 4.33: Comparison of best performing configuration of each algorithm on Sharpe Ratio reward function.

4.10.1.1 Analysis

Based on the results on Table 4.33:

The results reveal significant variations in performance across the eight offline RL algorithms for portfolio optimization using the Sharpe Ratio reward function. BEAR emerges as the strongest performer, achieving the highest Sharpe Ratio, average daily and monthly returns, and cumulative return, but it also exhibits the highest volatility, with daily and monthly standard deviations of 3.414 and 17.554, respectively. This suggests BEAR takes on substantial risk for its out-sized returns. Discrete CQL strikes a better risk-return balance, with a competitive Sharpe Ratio, moderate returns, and significantly lower volatility, making it a more stable alternative. BCQ and TD3+BC deliver modest but relatively stable returns, while IQL shows higher average returns but suffers from extreme volatility and negative cumulative returns, indicating poor consistency. Discrete BCQ and CQL under-perform, with near-zero or negative Sharpe Ratios, while DT is the most conservative, with minimal returns and the lowest volatility but a negative Sharpe Ratio, suggesting excessive risk aversion. Overall, BEAR is the highest-risk, highest-reward option, whereas Discrete CQL offers a more balanced approach, and DT is the safest but least profitable.

4.10.2 Returns Reward Function

Metrics	BCQ	D-BCQ	CQL	D-CQL	IQL	BEAR	TD3+BC	DT
Sharpe Ratio	0.0083	-0.0002	0.0102	0.0151	0.0034	0.0113	0.0032	0.0008
Avg Daily Return	0.101	0.013	0.058	0.202	0.076	0.292	0.092	0.021
Avg Monthly Return	3.299	-1.333	1.029	6.686	2.481	9.382	2.810	0.781
Std Dev Daily Return	1.769	1.442	0.980	2.021	3.266	4.133	3.489	0.521
Std Dev Monthly Return	6.134	3.397	1.631	9.935	14.220	21.393	16.543	2.100
Cumulative Return	16.574	-0.942	13.483	39.746	0.947	41.400	-0.184	2.704

Table 4.34: Comparison of best performing configuration of each algorithm on Returns reward function.

4.10.2.1 Analysis

Based on the results on Table 4.34:

Discrete CQL and BEAR emerge as the top performers in terms of cumulative returns and average monthly returns, suggesting their effectiveness in capturing profitable opportunities in the crypto market. However, BEAR's high volatility indicates aggressive risk-taking, whereas

Discrete CQL balances return and risk more effectively. CQL also shows promise with a competitive Sharpe Ratio and the lowest daily volatility, making it suitable for risk-averse strategies. In contrast, BCQ and IQL deliver moderate but stable returns, while TD3+BC and Discrete BCQ under-perform, with negative cumulative returns in some cases. DT exhibits the lowest volatility but also the weakest returns, highlighting a trade-off between safety and profitability. Overall, D-CQL stands out as the most balanced algorithm, while BEAR's high-risk, high-reward approach may appeal to aggressive investors. The results underscore the importance of aligning algorithm choice with risk tolerance and return objectives in dynamic crypto markets.

4.10.3 Reward Function Comparison

The comparison between the Sharpe ratio and returns reward functions reveals notable differences in algorithm performance. When optimizing for the Sharpe ratio, BEAR achieves the highest Sharpe ratio and cumulative return, but with high volatility. In contrast, when optimizing for raw returns, BEAR still performs well but with a lower Sharpe ratio and slightly reduced cumulative return, suggesting that the Sharpe ratio reward function better balances risk and return for this algorithm. Discrete CQL shows consistent improvement with the returns reward function, achieving a higher Sharpe ratio and significantly better cumulative return. Notably, DT performs poorly with the Sharpe ratio reward but marginally improves with returns reward, indicating its sensitivity to the reward function. Overall, the Sharpe ratio reward tends to produce more stable returns, while the returns reward can lead to higher absolute performance but with greater volatility. BEAR stands out as the strongest performer across both reward functions, though its advantage is more pronounced under Sharpe ratio optimization. The differences between the two reward functions are illustrated in the following bar charts (Figure 4.17) and heatmap (Figure 4.18).

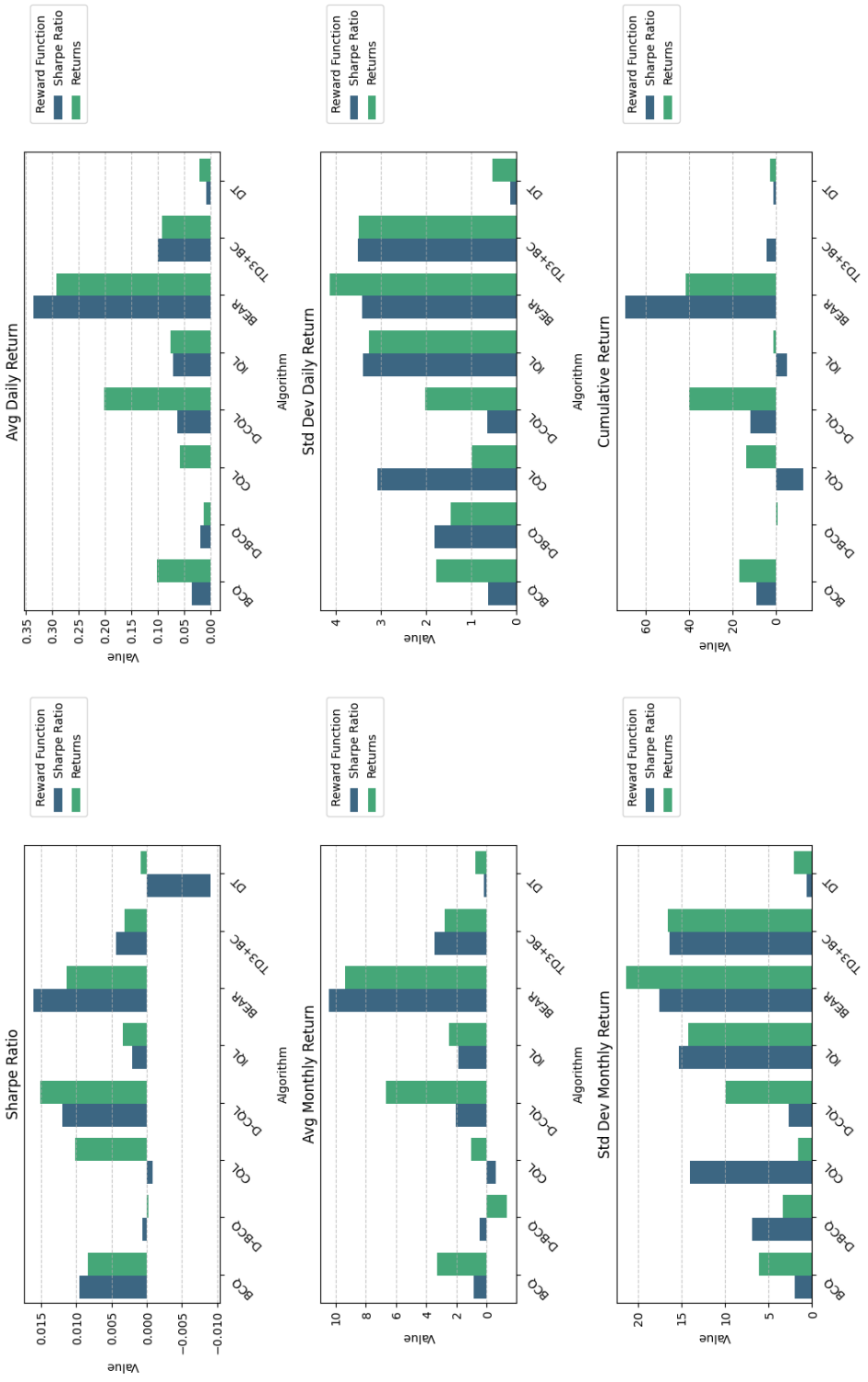


Figure 4.17: Bar charts comparing the two reward functions for each metric and algorithm

Figure 4.18: Heatmap comparing the performance of algorithms under the two reward functions. Color encoding: red indicates higher values under the Returns reward, blue indicates higher values under the Sharpe Ratio reward.

5 Discussion

5.1 Conclusion

In summary, most of the evaluated reinforcement learning algorithms struggled to discover an optimal trading strategy capable of delivering strong risk-adjusted returns. Among the tested approaches, BEAR and Discrete CQL demonstrated the most promising performance, while others failed to achieve consistent profitability. This limitation is evident from the fact that none of the trained agents attained a Sharpe Ratio exceeding 1.0, the threshold generally considered acceptable for a viable trading strategy.

The primary constraint likely stems from the limited feature space of the dataset. While OHLCV data provides a foundational representation of market dynamics, cryptocurrencies are influenced by a multitude of exogenous factors beyond raw price movements. A richer dataset incorporating the following features could significantly enhance model performance:

- **Technical indicators** to capture momentum and volatility regimes
- **On-chain metrics** to gauge network health
- **Sentiment and social media signals** (e.g., Twitter/Reddit sentiment, Google Trends) to quantify market psychology
- **Macro-financial factors** (e.g., Bitcoin dominance, correlation with traditional assets) to assess broader market conditions

Another critical factor hindering the discovery of an optimal strategy arises from the mismatch between discrete action spaces and the continuous nature of most reinforcement learning algorithms, compounded by the unique volatility of cryptocurrency markets. While our experiments employed a simplified discrete action space (e.g., fixed buy/sell amounts of $500), the majority of offline RL algorithms (excluding Discrete BCQ and Discrete CQL) are inherently designed for continuous action spaces, which better accommodate nuanced position sizing and dynamic risk management.

Cryptocurrency markets demand far greater flexibility than rigid, fixed-quantity actions can provide. For instance, during extreme volatility events (e.g., flash crashes or rapid bullish rallies), the ability to liquidate 100% of holdings or scale positions proportionally to confidence levels is essential to mitigate losses or capitalize on momentum. A discrete framework forces the agent

into suboptimal decisions, such as holding a fixed position during a market downturn due to insufficient granularity in risk-off actions.

5.2 Future Work

The findings of this study reveal critical limitations and opportunities for advancing RL-based cryptocurrency trading strategies.

Enriching the Dataset: A natural next step involves recreating the experiments with an enriched dataset that extends beyond basic OHLCV data to incorporate technical indicators, on-chain metrics, and sentiment features. Furthermore, since we observed improvements from transitioning from daily to hourly data, a next step would be to try even larger volumes of data.

Changing the Action Space: Transitioning from discrete to purely continuous action spaces, could enable more nuanced position sizing and dynamic risk management, particularly vital in cryptocurrency markets where the ability to partially liquidate holdings or scale positions proportionally to market conditions can significantly impact risk-adjusted returns.

Improving Evaluation Framework: Migrating to a *Gymnasium Environment* [47] would provide more rigorous tracking of agent performance through metrics like cumulative reward. Such an environment could also incorporate realistic market friction by modeling transaction costs, slippage, and liquidity constraints, factors often overlooked in academic simulations but critical for real-world deployment.

Hybrid Approach: Future research should explore hybrid architectures combining offline RL with online fine-tuning mechanisms to adapt to sudden market regime shifts, as well as ensemble methods that dynamically weight sub-policies based on prevailing volatility conditions.

Alternative Reward Functions: Our experiments demonstrated that the choice of reward function significantly impacts performance, underscoring its critical role in reinforcement learning. Future work should explore more sophisticated reward functions, such as the Omega Ratio, Sortino Ratio, Drawdown-Based Rewards, or CVaR, to better capture the nuances of portfolio optimization. Additionally, combining these metrics (e.g., via a weighted sum) could provide a powerful way to balance competing objectives like returns, risk, and turnover. This approach would allow for more flexible and adaptive optimization strategies tailored to specific investor preferences.

BIBLIOGRAPHY

[1] H. Markowitz, "Portfolio selection," *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," https://www.bitcoin.org/bitcoin.pdf, 2008, (cit. on pp. 1, 7).

[3] A. Gunjan and S. Bhattacharyya, "A brief review of portfolio optimization techniques," *Artificial Intelligence Review*, vol. 56, pp. 1–40, 09 2022.

[4] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020. [Online]. Available: https://arxiv.org/abs/2005.01643

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[6] Sutton and Barto, *Reinforcement learning: An introduction*. The MIT Press, 2018.

[7] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," 2019. [Online]. Available: https://arxiv.org/abs/1812.02900

[8] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018. [Online]. Available: https://arxiv.org/abs/1802.09477

[9] S. Fujimoto, E. Conti, M. Ghavamzadeh, and J. Pineau, "Benchmarking batch deep reinforcement learning algorithms," 2019. [Online]. Available: https://arxiv.org/abs/1910.01708

[10] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015. [Online]. Available: https://arxiv.org/abs/1509.06461

[11] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," 2020. [Online]. Available: https://arxiv.org/abs/2006.04779

[12] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," 2017. [Online]. Available: https://arxiv.org/abs/1702.08165

[13] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11791

[14] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit q-learning," 2021. [Online]. Available: https://arxiv.org/abs/2110.06169

[15] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 745–750. [Online]. Available: https://doi.org/10.1145/1273496.1273590

[16] Q. Wang, J. Xiong, L. Han, p. sun, H. Liu, and T. Zhang, "Exponentially weighted imitation learning for batched historical data," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/4aec1b3435c52abbdf8334ea0e7141e0-Paper.pdf

[17] X. B. Peng, A. Kumar, G. Zhang, and S. Levine, "Advantage-weighted regression: Simple and scalable off-policy reinforcement learning," 2019. [Online]. Available: https://arxiv.org/abs/1910.00177

[18] A. Nair, A. Gupta, M. Dalal, and S. Levine, "Awac: Accelerating online reinforcement learning with offline datasets," 2021. [Online]. Available: https://arxiv.org/abs/2006.09359

[19] R. Koenker and G. Bassett Jr, "Regression quantiles," *Econometrica: journal of the Econometric Society*, pp. 33–50, 1978.

[20] R. Koenker and K. F. Hallock, "Quantile regression," *Journal of Economic Perspectives*, vol. 15, no. 4, p. 143–156, December 2001. [Online]. Available: https://www.aeaweb.org/articles?id=10.1257/jep.15.4.143

[21] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," 2021. [Online]. Available: https://arxiv.org/abs/2004.07219

[22] A. Kumar, J. Fu, G. Tucker, and S. Levine, "Stabilizing off-policy q-learning via bootstrapping error reduction," 2019. [Online]. Available: https://arxiv.org/abs/1906.00949

[23] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *J. Mach. Learn. Res.*, vol. 13, no. 1, p. 723–773, Mar. 2012.

[24] S. Fujimoto and S. S. Gu, "A minimalist approach to offline reinforcement learning," 2021. [Online]. Available: https://arxiv.org/abs/2106.06860

[25] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," 2019. [Online]. Available: https://arxiv.org/abs/1709.06560

[26] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in deep policy gradients: A case study on ppo and trpo," 2020. [Online]. Available: https://arxiv.org/abs/2005.12729

[27] H. Furuta, T. Kozuno, T. Matsushima, Y. Matsuo, and S. S. Gu, "Co-adaptation of algorithmic and implementational innovations in inference-based deep reinforcement learning," 2021. [Online]. Available: https://arxiv.org/abs/2103.17258

[28] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018. [Online]. Available: https://arxiv.org/abs/1802.09477

[29] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Computation*, vol. 3, no. 1, pp. 88–97, 1991.

[30] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," 2021. [Online]. Available: https://arxiv.org/abs/2106.01345

[31] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[32] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8821–8831. [Online]. Available: https://proceedings.mlr.press/v139/ramesh21a.html

[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[34] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf, 2018, unpublished manuscript.

[35] H. Yang, X.-Y. Liu, S. Zhong, and A. Walid, "Deep reinforcement learning for automated stock trading: An ensemble strategy," September 2020, available at SSRN: https://ssrn.com/abstract=3690996 or http://dx.doi.org/10.2139/ssrn.3690996.

[36] V. Konda and V. Gao, "Actor-critic algorithms," *Neural Information Processing Systems*, 01 2000.

[37] Z. Liang, H. Chen, J. Zhu, K. Jiang, and Y. Li, “Adversarial deep reinforcement learning in portfolio management,” 2018. [Online]. Available: https://arxiv.org/abs/1808.09940

[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[39] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” 2016. [Online]. Available: https://arxiv.org/abs/1602.01783

[40] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” 2019. [Online]. Available: https://arxiv.org/abs/1509.02971

[41] G. Lucarelli and M. Borrotti, “A deep q-learning portfolio management framework for the cryptocurrency market,” *Neural Computing and Applications*, vol. 32, pp. 17 229–17 244, 2020. [Online]. Available: https://doi.org/10.1007/s00521-020-05359-8

[42] T. Cui, S. Ding, H. Jin, and Y. Zhang, “Portfolio constructions in cryptocurrency market: A cvar-based deep reinforcement learning approach,” *Economic Modelling*, vol. 119, p. 106078, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0264999322003157

[43] T. Nagler, L. Schneider, B. Bischl, and M. Feurer, “Reshuffling resampling splits can improve generalization of hyperparameter optimization,” 2024. [Online]. Available: https://arxiv.org/abs/2405.15393

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[45] T. Seno and M. Imai, “d3rlpy: An offline deep reinforcement learning library,” 2022. [Online]. Available: https://arxiv.org/abs/2111.03788

[46] Intel Corporation, “Intel Xeon Gold 6226R Processor Specifications,” https://www.intel.com/content/www/us/en/products/sku/199347/intel-xeon-gold-6226r-processor-22m-cache-2-90-ghz/specifications.html, 2020, accessed: 2025-05-17.

[47] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis, “Gymnasium: A standard interface for reinforcement learning environments,” 2024. [Online]. Available: https://arxiv.org/abs/2407.17032

APPENDICES

APPENDIX I

Guide on Reproducing the Experiments

Reproducing the experiments on the HPC is a seamless procedure:

1. Open the **Portfolio_Optimization.py** file with a text editor, go to the main function and manually set the hyper-parameters of each algorithm (like the example in Listing 3.4). We opted for a manual approach since the hyper-parameters are too many to pass them as command line arguments.
2. Open the **batch** file with a text editor. Go to lines 6 and 7 and change the error and output paths to your own:

```
6. #SBATCH --error=/home/ciosif01/output/error.out
7. #SBATCH --output=/home/ciosif01/output/output.out
```

   Also in line 9 change the email address to your own to receive updates regarding your submitted jobs:

```
9. #SBATCH --mail-user=ciosif01@ucy.ac.cy
```

   Finally in line 19, you can change basic parameters like the reward function and number of training epochs:

```
19. python3 Portfolio_Optimization.py -a $1 -w 168 -bd 1000000 -b
    500 -s 500 -r sharpe -e 20
```

   To view all available arguments execute in a terminal the following command:

```
$ python3 Portfolio_Optimization.py --help
```

3. To submit the jobs run the **batch_script.sh** by running the following:

```
$ ./batch_script.sh
```

   This is going to train and evaluate all eight algorithms with the specified hyper-parameter combination.
4. The trained models will appear under the automatically created directory /d3rlpy_logs. The metrics and graphs will appear under the /stats_logs directory.